Accounting [2]

The UNICOS operating system supports two types of accounting: Cray Research system accounting and standard UNIX accounting. Both types of accounting are described in this chapter.

2.1 Cray Research System Accounting (CSA)

Cray Research system accounting (CSA) is designed to meet the unique accounting requirements of Cray Research sites. Like the standard UNIX accounting package, CSA provides methods to collect per-process resource utilization data, record connect sessions, monitor disk usage, and charge fees to specific logins. CSA also provides other facilities that are not available from the standard accounting package. These include the following:

- Per-job accounting
- Accounting for socket usage
- Device accounting
- Daemon accounting (for the Network Queuing System (NQS) and the UNICOS tape subsystem)
- Disk accounting by account ID
- Arbitrary accounting periods
- Flexible system billing unit (SBU) system
- One file containing all data for an accounting period
- Off-line archiving of accounting data

Sites may run either the standard UNICOS accounting programs or the CSA package by invoking the appropriate shell scripts and programs. Both packages are installed with the UNICOS 10.0 release.

UNICOS system features in the CSA package include configurable parameters located in a single file, `/etc/config/acct_config`, and a set of user-defined exits that allows sites to tailor the daily run of accounting to their specific needs.

2.1.1 Concepts and Terminology

The following concepts and terms are important in CSA:

Term	Description
Daily accounting	Unlike the standard daily accounting, CSA's accounting can be run as many times as necessary during a day. However, this feature is still referred to as *daily accounting*.
Periodic accounting	Accounting similar to the standard UNICOS monthly accounting. CSA, however, lets system administrators specify the time periods for which "monthly" or cumulative accounting is to be run. Thus, periodic accounting can be run more than once a month.
Recycled data	By default, accounting data for active sessions is recycled until the session terminates. CSA reports only data for terminated sessions unless `csarun`(8) is invoked with the `-A` option. `csarun` places recycled data into data files in the `/usr/adm/acct/day` directory. These data files are suffixed with `0`; for example, per-process accounting data for active sessions from previous accounting periods is in the `/usr/adm/acct/day/pacct0` file.
Session	CSA organizes accounting data by sessions and boot times and then places the data into a session record file. For non-NQS jobs, a *session* consists of all accounting data for a given job ID during a single boot period. A *session* for an NQS job consists of the accounting data for all job IDs associated with the job's NQS sequence number/machine name identifier. NQS jobs may span multiple boot periods. If a job is restarted, it has the same job ID associated with it during all boot periods in which it runs. Rerun NQS jobs have multiple job IDs. CSA treats all phases of an NQS job as being in the same session.

Uptime period or boot period	A period delineated by the system boot times found in `/etc/csainfo`. The `csaboots`(8) command writes to this file during system boot.

2.1.2 Files and Directories Overview

This section provides a brief overview of the CSA file and directory structure. A more complete description of the files and directories can be found in Section 2.1.7, page 23.

2.1.2.1 Structures of the `acct` and `tmp` Directories

The directory structure of `/usr/adm/acct` is set up so that it is easy to find CSA data files and reports. The `/tmp` structure is also used while `csarun`(8) is running. Figure 1 illustrates the directory structure for both directories.

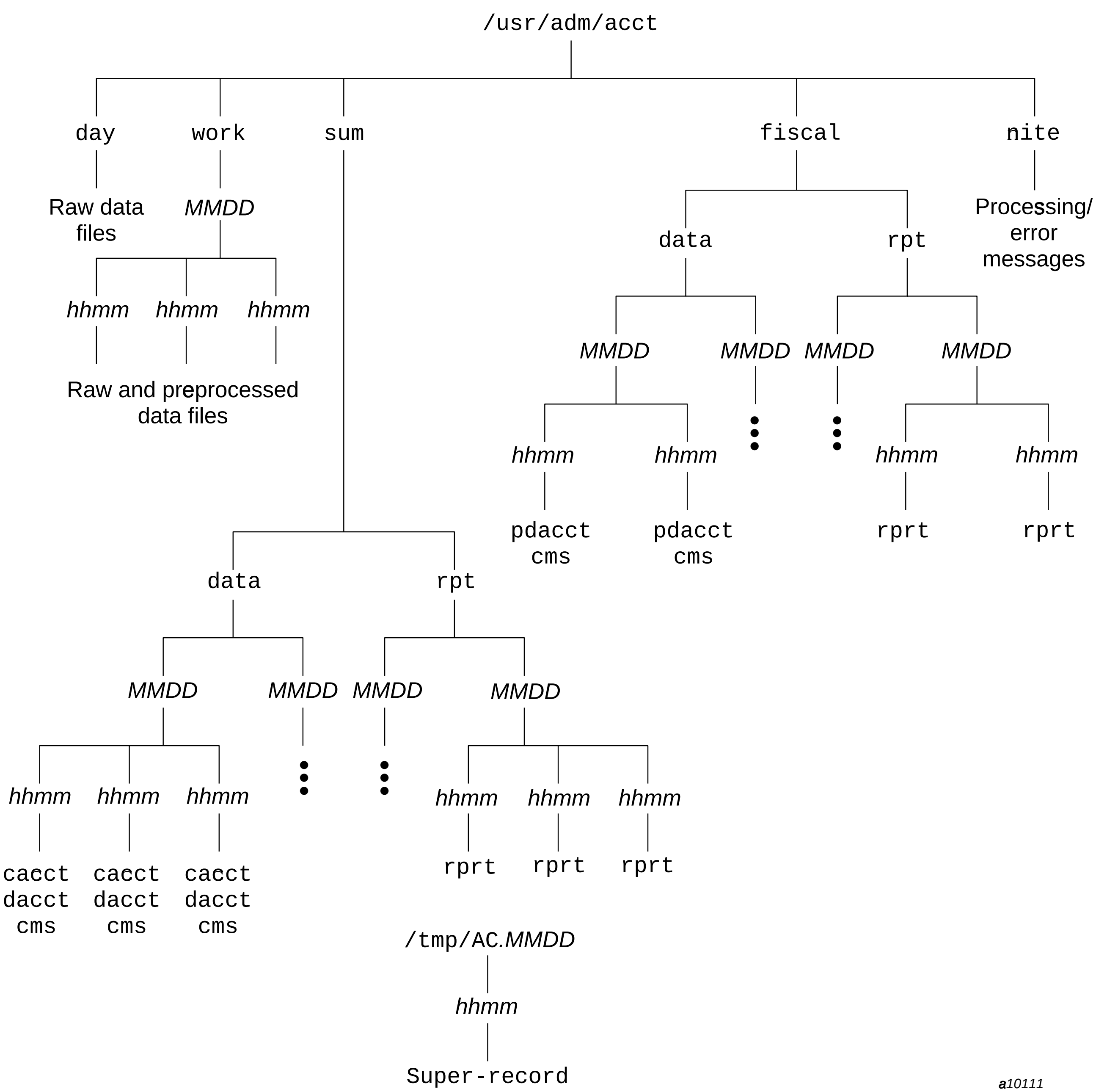

Figure 1. `/usr/adm/acct` and `tmp` directory structures

Note: As distributed, only the directory `/usr/adm/acct/day` is readable by all users. Within the `day` directory, only the `pacct*` files are readable by all users. This allows any user to examine the `pacct*` files by using the `acctcom`(1) command. All other directories and files within `/usr/adm/acct` are accessible only by `root` and users in the group `adm`.

Warning: `acctcom`(1) on a Cray ML-Safe configuration of the UNICOS system is considered to be a covert channel. You may want to consider restricting access to this command to the `adm` group.

The following abbreviations have these meanings:

Abbreviation	Definition
MMDD	Month, day
hhmm	Hour, minute

2.1.2.2 Shell Scripts and C Binaries

The `/usr/lib/acct` directory contains virtually all of the programs and scripts used by both the standard accounting and CSA packages. The only CSA program not located here is `/etc/csaboots` (see `csaboots`(8)), which records boot times at system startup. Programs used only by CSA begin with the characters `csa`.

2.1.2.3 Unprocessed Data Files

Both CSA and the standard accounting package expect most unprocessed accounting files to be located in the `/usr/adm/acct/day` directory. The use of this directory simplifies tracking of the current accounting files. The following table shows the location of the raw data files.

Accounting file	Description
`/usr/adm/acct/day/dtmp`	Disk accounting data
`/usr/adm/acct/day/nqacct*`	NQS daemon accounting data
`/usr/adm/acct/day/pacct*`	Per-process accounting data
`/usr/adm/acct/day/tpacct*`	Tape daemon accounting data
`/usr/adm/acct/day/soacct*`	Socket accounting data
`/etc/csainfo`	Boot times

/etc/wtmp	Connect time accounting data

Warning: On a Cray ML-Safe configuration of the UNICOS system, /etc/wtmp is considered a covert channel. You may want to consider restricting access to this file to the adm group.

Accounting files in /usr/adm/acct/day whose names include the suffix 0 contain data from sessions that did not complete during the previous accounting periods.

During CSA data processing, sites may select to archive the raw and/or processed data off-line. Section 2.1.5, page 16, describes how to do this. By default, all raw data files are deleted after use and are not archived.

2.1.2.4 Data Files Being Processed

At the start of a daily accounting run, CSA moves the raw data files from /usr/adm/acct/day to the appropriate /usr/adm/acct/work/*MMDD/hhmm* directory. The files in the work directory are as follows:

File	Description
Ever.tmp	Data verification work file
Pctime*	Preprocessed connect time data
Pnqacct*	Preprocessed NQS data
Puptime*	Uptimes
Rctime0	Connect data to be recycled in the next accounting run
Rnqacct0	NQS data to be recycled in the next accounting run
Rpacct0	Per-process accounting data to be recycled in the next accounting run
Rtpacct0	Tape data to be recycled in the next accounting run
Ruptime0	Uptimes to be recycled in the next accounting run
Wctime*	Verified raw connect time data
Wdisktacct	Disk accounting data (cacct.h format)

`Wdtmp`	Disk accounting data from `diskusg`(8) or `acctdusg`(8)
`Wnqacct*`	Raw NQS accounting data
`Wpacct*`	Raw per-process accounting data
`Wsoacct*`	Raw socket accounting data
`Wtpacct*`	Raw tape accounting data
`Wwtmp`	Raw connect time data

2.1.2.5 Processed Data Files

CSA outputs the following data files:

File / Description

`/tmp/AC.`*MMDD*`/`*hhmm*`/Super-record`

Session record file; this file is usually deleted after it has been used by CSA.

`/usr/adm/acct/fiscal/data/`*MMDD*`/`*hhmm*`/pdacct`

Consolidated periodic data.

`/usr/adm/acct/fiscal/data/`*MMDD*`/`*hhmm*`/cms`

Periodic command usage data.

`/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm*`/cacct`

Consolidated daily data; this file is deleted by `csaperiod`(8) if the `-r` option is specified.

`/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm*`/cms`

Daily command usage data; this file is deleted by `csaperiod`(8) if the `-r` option is specified.

`/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm*`/dacct`

Daily disk usage data; this file is deleted by `csaperiod`(8) if the `-r` option is specified.

2.1.2.6 Reports

CSA generates daily and periodic reports. The locations of these reports are as follows:

File	Description
`/usr/adm/acct/fiscal/rpt/`*MMDD*`/`*hhmm*`/rprt`	Periodic accounting report
`/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm*`/rprt`	Daily accounting report

2.1.3 Daily Operation Overview

When the UNICOS operating system is run in multiuser mode, accounting behaves in a manner similar to the following process. However, because sites may customize CSA, the following may not reflect the actual process at a particular site:

1. System boot time is written to `/etc/csainfo`. Each time the system is booted, the boot time is written to `/etc/csainfo` by the `/etc/csaboots` command, which is invoked by `rc` (see `brc`(8)) during system startup.
2. Process accounting is enabled. When the system is switched to multiuser mode, the `/usr/lib/acct/startup` (see `acctsh`(8)) script is called by `/etc/rc` and performs the following functions:
   a. Writes an `acctg on` record to `/etc/wtmp`; the `acctwtmp` program is used to write this record.
   b. Enables process accounting with the command line `/usr/lib/acct/turnacct on`; `turnacct`(8) calls the `accton` program with the argument `/usr/adm/acct/day/pacct`.
   c. Removes lock files and saved `pacct` and `wtmp` files. `/usr/lib/acct/remove` is invoked to clean up saved `pacct` and `wtmp` files in `/usr/adm/acct/sum`. Unlike the standard accounting

package, CSA does not leave files in this directory. In addition, the lock files are removed from /usr/adm/acct/nite.

3. By default, daemon accounting for NQS, tape, and sockets is handled by the /usr/lib/acct/startup script. However, in order to run NQS and tape daemon accounting, you must modify the appropriate subsystem. Section 2.1.4, page 11, describes this process in detail.

4. The amount of disk space used by each user is determined periodically. /usr/lib/acct/dodisk (see dodisk(8)) is run periodically by cron to generate a snapshot of the amount of disk space being used by each user. dodisk should be run at most once for each time /usr/lib/acct/csarun (see csarun(8)) is run. Multiple invocations of dodisk during the same accounting period write over previous dodisk output.

5. A fee file is created. Sites desiring to charge fees to certain users can do so by invoking /usr/lib/acct/chargefee (see chargefee(8)). Each accounting period's fee file (/usr/adm/acct/day/fee) is merged into the consolidated accounting records by /usr/lib/acct/csaperiod (see csaperiod(8)).

6. Daily accounting is run. At specified times during the day, csarun is executed by cron to process the current accounting data. The output from csarun is a consolidated daily accounting file and an ASCII report.

7. Periodic accounting is run. At a specific time during the day, or on certain days of the month, /usr/lib/acct/csaperiod (see csaperiod(8)) is executed by cron to process consolidated accounting data from previous accounting periods. The output from csaperiod is a consolidated periodic accounting file and an ASCII report.

8. Accounting is disabled. When the system is shut down gracefully, the script /usr/lib/acct/shutacct (see shutacct(8)) is executed by /etc/shutdown (see shutdown(8)). shutacct writes an "acctg off" record to /etc/wtmp. It then calls /usr/lib/acct/turnacct and /usr/lib/acct/turndacct to disable per-process and daemon accounting (see turnacct(8) and turndacct(8)).

2.1.4 Setting up CSA

The following is a brief description of setting up CSA. Site-specific modifications are discussed in detail in Section 2.1.10, page 39. As described in this section, CSA is run by a person with super-user permissions. CSA also can be run by users who have `acct` permissions and are in the `adm` group. See Section 2.1.10.7, page 54, for the necessary modifications.

1. Change the default system billing unit (SBU) weighting factors, if necessary. By default, no SBUs are calculated. If your site wants to report SBUs, you must modify the configuration file `/etc/config/acct_config`.

2. Modify any necessary parameters in the `/etc/config/acct_config` file, which contains configurable parameters for the accounting system. Ensure that parameters, such as `MEMINT`, reflect the needs of your site.

3. If you want daemon accounting, you must enable daemon accounting at system startup time by performing the following steps:

   a. Ensure that the variables in `/etc/config/acct_config` for the subsystems for which you want to enable daemon accounting are set to `on`. Set the `NQS_START`, `TAPE_START`, and `SOCKET_START` parameters to `on` to enable NQS, online tapes, and socket accounting, respectively.

   b. If necessary, enable accounting from the daemon's side. Specifically, NQS and tape accounting must also be enabled by the associated daemon. Use the `qmgr`(8) `set accounting on` command to turn on NQS accounting. To enable tape daemon accounting, execute `tpdaemon`(8) with the `-c` option. Socket accounting does not require any additional processing.

4. Prior to setting up the following `cron` jobs, ensure that the `/etc/checklist` file exists. By default, `dodisk`(8) performs disk accounting on the special files listed in `checklist`. For most installations, entries similar to the following should be made in `/usr/spool/cron/crontabs/root` so that `cron`(8) automatically runs daily accounting:

```
0 4 * * 1-6 /usr/lib/acct/csarun 2> /usr/adm/acct/nite/fd2log
0 3 * * 1-6 /usr/lib/acct/dodisk -a -v 2> /usr/adm/acct/nite/dk2log
```

   `csarun`(8) should be executed at such a time that `dodisk` has sufficient time to complete. If `dodisk` does not complete before `csarun` executes, disk accounting information may be missing or incomplete.

   `dodisk` must be invoked with either the `-a` or the `-A` option. If it is not, `csaperiod`(8) aborts when it attempts to merge the disk usage information with other accounting data.

5. Periodically check the size of the `acct` files. Entries similar to the following should be made in `/usr/spool/cron/crontabs/root`:

```
0 * * * * /usr/lib/acct/ckdacct nqs tape socket
0 * * * * /usr/lib/acct/ckpacct
```

cron(8) should periodically execute the ckpacct(8) and ckdacct(8) shell scripts. If the pacct file grows larger than 500 blocks (default), ckpacct calls the command /usr/lib/acct/turnacct switch to start a new pacct file. ckpacct also makes sure that there are at least 500 free blocks on the file system containing /usr/adm/acct (/usr by default). If there are not enough blocks, per-process accounting is turned off. The next time ckpacct is executed, it turns per-process accounting back on if there are enough free blocks.

ckdacct performs an analogous function for daemon accounting. If a daemon's accounting file is larger than 500 blocks (default), the command /usr/lib/acct/turndacct switch is executed in order to start a new accounting file. In addition, ckdacct also checks the amount of free blocks on the ACCT_FS file system (/usr by default).

Ensure that the ACCT_FS and MIN_BLKS variables have been set correctly in the /etc/config/acct_config configuration file. ACCT_FS is the file system containing /usr/adm/acct; the default is /usr. MIN_BLKS is the minimum number of free blocks needed in the ACCT_FS file system. The default is 500.

It is very important that ckpacct and ckdacct be run periodically so that an administrator is notified when the accounting file system (/usr by default) runs out of disk space. After the file system is cleaned up, the next invocation of ckpacct and ckdacct enables per-process and daemon accounting. You can manually reenable accounting by invoking turnacct(8) and turndacct(8) with the on operand.

If ckpacct and ckdacct are not run periodically, and the accounting file system runs out of space, an error message is written to the console stating that a write error occurred and that accounting is disabled. If you do not free disk space as soon as possible, a vast amount of accounting data can be lost unnecessarily. Additionally, lost accounting data can cause csarun(8) to abort or report erroneous information.

6. To run periodic accounting, an entry similar to the following should be made in /usr/spool/cron/crontabs/root. This command generates a periodic report on all consolidated data files found in /usr/adm/acct/sum/data/* and then deletes those data files:

```
15 5 1 * * /usr/lib/acct/csaperiod -r 2> /usr/adm/acct/nite/pd2log
```

This entry is executed at such a time that csarun(8) has sufficient time to complete. This example results in the creation of a monthly accounting file and report on the first day of each month. These files contain information about the previous month's accounting.

7. Update the holidays file. The /usr/lib/acct/holidays file contains the prime/nonprime time table for the accounting system, which should be edited to reflect your site's holiday schedule for the year.

   By default, the holidays file is located in the /usr/lib/acct directory. You can change this location by modifying the *HOLIDAY_FILE* variable in /etc/config/acct_config. If necessary, modify the *NUM_HOLIDAYS* variable (also located in /etc/config/acct_config), which sets the upper limit on the number of holidays that can be defined in *HOLIDAY_FILE*. The format of this file is composed of the following types of entries:

   - Comment lines: These lines may appear anywhere in the file as long as the first character in the line is an asterisk (*).

   - Version line: This line must be the first uncommented line in the file and must only appear once. It denotes that the new holidays file format is being used. This line should not be changed by the site.

   - Year designation line: This line must be the second uncommented line in the file and must only appear once. The line consists of two fields. The first field is the keyword YEAR. The second field must be either the current year or the wildcard character, asterisk (*). If the year is wildcarded, the current year is automatically substituted for the year. The following are examples of two valid entries:

     ```
     YEAR        1997
     YEAR        *
     ```

   - Prime/nonprime time designation lines: These must be uncommented lines 3, 4, and 5 in the file. The format of these lines is as follows:

     period   *prime_time_start*   *nonprime_time_start*

     The variable *period* is one of the following: WEEKDAY, SATURDAY, or SUNDAY. The *period* can be in either upper or lowercase.

     The prime and nonprime start time can be one of two formats:

     – Both start times are 4-digit numeric values between 0000 and 2359. The *nonprime_time_start* value must be greater than the *prime_time_start* value. For example, it is incorrect to have prime time start at 07:30 A.M. and nonprime time start at 1 minute after midnight. Therefore, the following entry is wrong and can cause incorrect accounting values to be reported.

```
WEEKDAY  0730  0001
```

It is correct to specify prime time to start at 07:30 A.M. and nonprime time to start at 5:30 P.M. on weekdays. You would enter the following in the holiday file:

```
WEEKDAY  0730  1730
```

– Start times specify that the entire period is to be either all prime time or all nonprime time. To specify that the entire period is to be considered prime time, set *prime_time_start* to `ALL` and *nonprime_time_start* to `NONE`. If the period is to be considered all nonprime time, set *prime_time_start* to `NONE` and *nonprime_time_start* to `ALL`. For example, to specify Monday through Friday as all prime time, you would enter the following:

```
WEEKDAY  ALL  NONE
```

To specify all of Sunday to be nonprime time, you would enter the following:

```
SUNDAY  NONE  ALL
```

- Company holidays lines: These entries follow the year designation line and have the following general format:

  day-of-year Month Day Description of Holiday

  The *day-of-year* field is a number in the range 1 through 366, indicating the day for a given holiday (leading white space is ignored). The other three fields are commentary and are not currently used by other programs. Each holiday is considered all nonprime time.

  If the `holidays` file does not exist or there is an error in the year designation line, the default values for all lines are used.

  If there is an error in a prime/nonprime time designation line, the entry for the erroneous line is set to a default value. All other lines in the `holidays` file are ignored and default values are used.

  If there is an error in a company holidays line, all holidays are ignored.

The default values are as follows:

`YEAR`	The current year.
`WEEKDAY`	Monday through Friday is all prime time.
`SATURDAY`	Saturday is all nonprime time.
`SUNDAY`	Sunday is all nonprime time.

No holidays are specified

2.1.5 The `csarun` Command

The `/usr/lib/acct/csarun` command is the primary daily accounting shell script. It processes connect, disk, per-process, and daemon accounting files and is normally initiated by `cron`(8) during nonprime hours.

`csarun`(8) also contains four user-exit points allowing sites to tailor the daily run of accounting to their specific needs (see Section 2.1.10.3, page 51 for information on setting up user exits callable from `csarun` and Section 2.2.3.1, page 83, for information on setting up a user exit callable from `runacct`).

The `csarun` command does not damage files in the event of errors. It contains a series of protection mechanisms that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that `csarun` can be restarted with minimal intervention.

2.1.5.1 Daily Invocation

The `csarun` command is invoked periodically by `cron`(8). It is very important that you ensure that the previous invocation of `csarun` completed successfully before invoking `csarun` for a new accounting period. If this is not done, information about unfinished sessions will be inaccurate.

Data for a new accounting period can also be interactively processed by executing the following:

```
nohup csarun 2> /usr/adm/acct/nite/fd2log &
```

Before executing `csarun` in this manner, ensure that the previous invocation completed successfully. To do this, look at the files `active` and `statefile` in `/usr/adm/acct/nite`. Both files should specify that the last invocation completed successfully.

2.1.5.2 Error and Status Messages

The csarun error and status messages are placed in the /usr/adm/acct/nite directory. The progress of a run is tracked by writing descriptive messages to the file active. Diagnostic output during the execution of csarun is written to fd2log. The lock and lock1 files prevent concurrent invocations of csarun; csarun will abort if these two files exist when it is invoked. The clastdate file contains the month, day, and time of the last two executions of csarun.

Errors and warning messages from programs called by csarun are written to files that have names beginning with E and ending with the current date and time. For example, Ebld.11121400 is an error file from csabuild(8) for a csarun invocation on November 12, at 14:00.

If csarun detects an error, it sends an informational message to the operator with msgi(1), sends mail to root and adm, removes the locks, saves the diagnostic files, and terminates execution. When csarun detects an error, it will send mail either to MAIL_LIST if it is a fatal error, or to WMAIL_LIST if it is a warning message, as defined in the configuration file /etc/config/acct_config.

2.1.5.3 States

Processing is broken down into separate reentrant states so that csarun can be restarted. As each state completes, /usr/adm/acct/nite/statefile is updated to reflect the next state. When csarun reaches the CLEANUP state, it removes various data files and the locks, and then terminates.

The following describes the events that occur in each state. *MMDD* refers to the month and day csarun was invoked. *hhmm* refers to the hour and minute of invocation.

State	Description
SETUP	The current accounting files are switched via turnacct(8) and turndacct(8). These files are then moved to the /usr/adm/acct/work/*MMDD*/*hhmm* directory. File names are prefaced with W. /etc/wtmp and /etc/csainfo are also moved to this directory.
WTMPFIX	The wtmp file in the work directory is checked for accuracy by wtmpfix (see fwtmp(8)). Some date changes cause csaline(8) to fail, so wtmpfix attempts to adjust the time stamps in the wtmp file if a date change record appears.

If `wtmpfix` is unable to fix the `wtmp` file, the `wtmp` file must be manually repaired. This is described in Section 2.1.6.1, page 20.

`VERIFY` By default, per-process and NQS accounting files are checked for valid data. In addition, tape and socket accounting files are verified. Records with invalid data are removed. Names of bad data files are prefixed with `BAD.` in the `/usr/adm/acct/work/*` directory. The corrected files do not have this prefix.

`PREPROC` The NQS and connect time (`wtmp`) accounting files are run through preprocessors. File names of preprocessed files are prefixed with a `P` in the `/usr/adm/acct/work/`*MMDD*`/`*hhmm* directory.

`ARCHIVE1` First user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.archive1` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. You might use this user exit to archive the accounting files in `${WORK}`.

`BUILD` The per-process, NQS, tape, socket, and connect accounting data is organized into a session record file.

`ARCHIVE2` Second user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.archive2` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. You might use this exit to archive the session record file.

`CMS` Produces a command summary file in `cacct.h` format. The `cacct` file is put into the `/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm* directory for use by `csaperiod`(8).

`REPORT` Generates the daily accounting report and puts it into `/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm*`/rprt`. A consolidated data file, `/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm*`/cacct`, is also produced from the session record file. In addition, accounting data for unfinished sessions is recycled.

`DREP` Generates a daemon usage report based on the session file. This report is appended to the daily accounting report, `/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm*`/rprt`.

`FEF` Third user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.fef` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. `csarun` variables are available, without being exported, to the user exit script. You might use this exit to convert the session record file to a format suitable for a front-end system.

`USEREXIT` Fourth user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.user` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. `csarun` variables are available, without being exported, to the user exit script. You might use this exit to run local accounting programs.

`CLEANUP` Cleans up temporary files, removes the locks, and then exits.

2.1.5.4 Restarting csarun

If `csarun`(8) is executed without arguments, the previous invocation is assumed to have completed successfully.

The following operands are required with `csarun` if it is being restarted:

`csarun [`*MMDD* `[`*hhmm* `[`*state*`]]]`

MMDD is month and day, *hhmm* is hour and minute, and *state* is the `csarun` entry state.

To restart `csarun`, follow these steps:

1. Remove all lock files by using the following command line:

   ```
   rm -f /usr/adm/acct/nite/lock*
   ```

2. Execute the appropriate `csarun` restart command, using the following examples as guides:

   a. To restart `csarun` using the time and state specified in `clastdate` and `statefile`, execute the following command:

      ```
      nohup csarun 0601 2> /usr/adm/acct/nite/fd2log &
      ```

      In this example, `csarun` will be rerun for June 1, using the time and state specified in `clastdate` and `statefile`.

b. To restart `csarun` using the state specified in `statefile`, execute the following command:

```
nohup csarun 0601 0400 2> /usr/adm/acct/nite/fd2log &
```

In this example, `csarun` will be rerun for the June 1 invocation that started at 4:00 A.M., using the state found in `statefile`.

c. To restart `csarun` using the specified date, time, and state, execute the following command:

```
nohup csarun 0601 0400 BUILD 2> /usr/adm/acct/nite/fd2log &
```

In this example, `csarun` will be restarted for the June 1 invocation that started at 4:00 A.M., beginning with state `BUILD`.

Before `csarun` is restarted, the appropriate directories must be restored. If the directories are not restored, further processing is impossible. These directories are as follows:

`/usr/adm/acct/work/`*MMDD*`/`*hhmm*
`/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm*
`/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm*
`/tmp/AC.`*MMDD*`/`*hhmm*

If you are restarting at state `ARCHIVE2`, `CMS`, `REPORT`, `DREP`, or `FEF`, the session file must already exist in `/tmp/AC.`*MMDD*`/`*hhmm*. If the file does not exist, `csarun` will automatically restart at the `BUILD` state. Depending on the tasks performed during the site-specific `USEREXIT` state, the session file may or may not need to exist.

2.1.6 Verifying and Correcting Data Files

This section describes how to remove bad data from various accounting files.

2.1.6.1 Fixing `wtmp` Errors

The `wtmp` files generally cause the highest number of errors in the day-to-day operation of the accounting subsystem. When the date is changed, and the UNICOS system is in multiuser mode, a set of date change records is written into the `/etc/wtmp` file. The `wtmpfix` (see `fwtmp`(8)) program is designed to adjust the time stamps in the `wtmp` records when a date change is encountered.

Some combinations of date changes and reboots, however, slip by `wtmpfix` and cause `csaline`(8) to fail. The following example shows how to repair a `wtmp` file:

```
$ cd /usr/adm/acct/work/MMDD/hhmm
$ /usr/lib/acct/fwtmp < Wwtmp > xwtmp
$ ed xwtmp
    (delete corrupted records)
$ /usr/lib/acct/fwtmp -ic < xwtmp > Wwtmp
    (restart csarun at the WTMPFIX state)
```

If the `wtmp` file is beyond repair, create a null `Wwtmp` file. This prevents any charging of connect time.

2.1.6.2 Verifying Data Files

You can verify data files with the `csaedit`(8), `csapacct`(8), and `csaverify`(8) commands. `csaedit` and `csapacct` verify and delete bad data records, while `csaverify` only flags bad records. By default, `csaedit` and `csaverify` are invoked in `csarun` to verify the data files.

Note that these commands may allow files that contain bad data, such as very large values, to be successfully verified.

2.1.6.3 Editing Data Files

You can use the `csaedit`(8) and `csapacct`(8) commands to verify and remove records from various accounting files. The following example shows how you can use `csapacct` to verify and remove bad records from a per-process (`pacct`) accounting file.

In this example, `csapacct` is invoked with verbose mode enabled (valid data records are written to the file `pacct.NEW`):

```
/usr/lib/acct/csapacct -v pacct pacct.NEW
```

The output produced by this command line is as follows:

```
Bad record - starting byte offset is 077740 (32736)
   invalid pacct record - bad base parent process id 97867
Found the next magic word at byte offset 0100130, ignored 120 bytes


Found 394 BASE records
Found 4 EOJ records
Found 1 MTASK (multitasking) records
Found 0 ERROR records
Found 0 IO records
Found 0 SDS records                       # not on CRAY EL systems
Found 0 MPP records                       # not on CRAY EL systems
Found 0 PERFORMANCE records
Outputted records for 398 processes
Ignored 120 bytes from the input file
```

You can use `csaedit` and `csapacct` in conjunction with `csaverify`, by first running `csaverify` and noting the byte offsets of the first bad record. Next, execute `csaedit` or `csapacct` and remove the record at the specified offset. The following example shows how you can verify and then edit a bad `pacct` accounting file:

1. The `pacct` file is verified with the following command line, and the following output is received:

```
$  /usr/lib/acct/csaverify -P pacct

/usr/lib/acct/csaverify: pacct: invalid pacct record - bad base parent process id 97867
  byte offset: start = 077740 (32736)  word offset: start = 07774 (4092)
/usr/lib/acct/csaverify: pacct: invalid pacct record - bad magic word 03514000
  byte offset: start = 0100070 (32824)  word offset: start = 010007 (4103)
```

2. The record found at byte offset 32736 is deleted as follows (valid records are written to `pacct.NEW`):

   ```
   /usr/lib/acct/csapacct -o 32736 pacct pacct.NEW
   ```

3. The new `pacct` file is reverified as follows to ensure that all the bad records have been deleted:

   ```
   /usr/lib/acct/csaverify -P pacct.NEW
   ```

You can use csaedit to produce an abbreviated ASCII version of some of the daemon accounting files and acctcom(1) to generate a similar ASCII version of pacct files.

2.1.7 Files and Directories

This section describes the files and directories used by CSA.

2.1.7.1 /usr/adm/acct Directory

The /usr/adm/acct directory contains the following directories:

Directory	Description
day	Current accounting files
fiscal	Periodic accounting data and reports
nite	Processing messages and errors
sum	Daily accounting data and reports
work	Temporary work area

The /usr/adm/acct/day directory contains the current accounting files, as shown in the following list. Files with names ending with 0 contain data for uncompleted sessions from previous days.

File	Description
dtmp	Disk accounting data (ASCII) created by dodisk(8)
nqacct*	NQS daemon accounting data
pacct*	Per-process accounting data
soacct*	Socket accounting data
tpacct*	Tape daemon accounting data

The /usr/adm/acct/fiscal/data/*MMDD*/*hhmm* directory contains processed, periodic, binary accounting data in the form of the following files:

File	Description
cms	Periodic command usage data in command summary (cms) record format

`pdacct`	Consolidated periodic data generated on *MMDD* at *hhmm*

The `/usr/adm/acct/fiscal/rpt/`*MMDD*`/`*hhmm* directory contains the periodic accounting report, `rprt`, that was generated on *MMDD* at *hhmm*.

The `/usr/adm/acct/nite` directory contains error messages and status information about the accounting runs in the following files:

File	Description
`active`	Progress and status of `csarun`
`active`*MMDDhhmm*	Progress and status of `csarun` after an error has been detected
`clastdate`	Last two times `csarun` was executed; in *MMDD hhmm* format
`disktacct`	Disk accounting records in `cacct.h` format; created by `dodisk`(8)
`dk2log`	Diagnostic output created during execution of `dodisk`
`E*`*MMDDhhmm*	Error/warning messages for an accounting run done on *MMDD* at *hhmm*
`fd2log`	Diagnostic output created during execution of `csarun`
`lineuse`	tty line usage report
`lock, lock1`	Controls simultaneous invocations of `csarun`
`pd2log`	Diagnostic output created during execution of `csaperiod`
`pdact`	Progress and status of `csaperiod`
`pdact`*MMDDhhmm*	Progress and status of `csaperiod` after an error has been detected
`reboots`	The start and ending dates from `wtmp` and a listing of reboots
`statefile`	Current state during `csarun` execution
`tmpwtmp`	The `wtmp` file corrected by `wtmpfix` (see `fwtmp`(8))

`wtmperror`	`wtmpfix` error messages

The `/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm* directory contains daily, binary accounting data in the following files:

File	Description
`cacct`	Consolidated daily data generated on *MMDD* at *hhmm* in `cacct.h` format
`cms`	Command usage data in command summary (cms) record format
`dacct`	Disk usage data in `cacct.h` format

The `/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm* directory contains the daily accounting report, `rprt`, which was generated on *MMDD* at *hhmm*.

The `/usr/adm/acct/work/`*MMDD*`/`*hhmm* directory is used as a work area during the processing of the accounting data. It contains the following files:

File	Description
`BAD.Wnqacct*`	Unprocessed NQS accounting data containing bad records (verified by `csaedit`)
`BAD.Wpacct*`	Unprocessed per-process accounting data containing bad records (verified by `csaedit`)
`BAD.Wtpacct*`	Unprocessed tape accounting data containing bad records (verified by `csaedit`)
`Ever.tmp`	Data verification work file
`Pctime*`	Preprocessed connect time data
`Pnqacct*`	Preprocessed NQS data
`Puptime*`	Uptimes
`Rctime0`	Recycled connect data to be used in the next accounting period
`Rnqacct0`	Recycled NQS data to be used in the next accounting period
`Rpacct0`	Recycled per-process accounting data to be used in the next accounting run
`Rtpacct0`	Recycled tape data to be used in the next accounting period

`Ruptime0`	Recycled uptimes to be used in the next accounting period
`Wctime*`	Verified, unprocessed connect time data
`Wdisktacct`	Disk accounting data (`cacct.h` format) created by `acctdisk`(8)
`Wdtmp`	Disk accounting report (ASCII) created by `diskusg`(8) or `acctdusg`(8)
`Wnqacct*`	Unprocessed NQS accounting data
`Wpacct*`	Unprocessed per-process accounting data
`Wtpacct*`	Unprocessed tape accounting data
`Wsoacct*`	Unprocessed socket accounting data
`Wwtmp*`	Unprocessed connect time data

The `/tmp/AC.`*MMDD*/*hhmm* directory contains the session record file, `Super-record`, which is generated on *MMDD* at *hhmm*.

The `/usr/lib/acct` directory contains the following programs and shell scripts used by CSA:

Program/script	Description
`csaaddc`	Merges consolidated (`cacct`) accounting files
`csabuild`	Creates a session file
`csacon`	Creates a consolidated (`cacct`) accounting file
`csaconvert`	Converts UNICOS 8.0, 8.3, 9.0, 9.1, 9.2, and 9.3 accounting file(s), both System V and CSA, to a 10.0 format
`csacrep`	Generates consolidated accounting reports
`csadrep`	Reports daemon usage based on the session file
`csaedit`	Verifies, deletes records, and prints various data files
`csafef`	Template to convert session files to an IBM front-end format
`csafef2`	Template to summarize session file records by the tuple user name, job ID, and account ID.
`csagcon`	Consolidates accounting data for `session` and `pacct` files

`csagfef`	Formats consolidated accounting data
`csaibm`	Template to convert session files to an IBM front-end format
`csajrep`	Generates job reports from a session file
`csaline`	Preprocesses connect time data (`/etc/wtmp`)
`csanqs`	Preprocesses NQS accounting data
`csapacct`	Verifies and deletes records from a `pacct` file
`csaperiod`	Performs periodic accounting
`csaperm`	Changes the group ID and permissions on the accounting files
`csarecy`	Recycles session data for unfinished sessions
`csarun`	Performs daily accounting
`csaswitch`	Enables or disables kernel and daemon accounting
`csaverify`	Verifies various data files
`getconfig`	Extracts values from the configuration file

The `/usr/lib/acct` directory may also contain the following programs if your site uses the accounting user exits:

Program/script	Description
`csa.archive1`	Site-generated user exit for `csarun`
`csa.archive2`	Site-generated user exit for `csarun`
`csa.fef`	Site-generated user exit for `csarun`
`csa.user`	Site-generated user exit for `csarun`

2.1.7.2 `/etc` Directory

The `/etc` directory contains uptime and connect time data in the following files:

File	Description
`csaboots`	Captures system boot times
`csainfo`	Output file of `csaboots`

`wtmp`	Current connect time data

2.1.7.3 `/etc/config` Directory

The `/etc/config` directory is the location of the `acct_config` file that contains the configurable parameters used by the accounting commands. These parameters can be changed by using the UNICOS installation and configuration menu system (the *menu system*). To see the `acct_config` file parameters, use the following menu selection:

```
UNICOS 8.0 Installation / Configuration Menu System
    ->Configure System
        ->Accounting Configuration
```

The main menu for accounting configuration is as follows:

```
      Mainframe Dependent Parameters ==>
      Accounting Start Parameters ==>
      Block Device SBUs ==>
      Character Device SBUs ==>
      Connect Time SBU ==>
      Multitasking CPU SBUs=>
      NQS SBUs ==>
      Pacct File SBUs ==>
      Tape SBUs ==>
      Miscellaneous Settings ==>
      Parameters for CSARUN and CSAPERIOD ==>
      Site Defined Settings ==>

      Import accounting configuration ...
      Activate accounting configuration ...
      Reload default accounting configuration ...
```

Online help for the `acct_config` parameters is available through the menu system.

The main menu for accounting configuration displays a table of `acct_config` parameters and the current values.

The `Import accounting configuration ...` option gets the local site accounting configuration.

The `Activate accounting configuration ...` option rewrites the `/etc/config/acct_config` file with the current values selected in the menus.

The `Reload default accounting configuration ...` option reloads the default values for the `acct_config` file from the released `/usr/src/skl/etc/config/acct_config` file.

2.1.8 CSA Data Processing

The flow of data among the various CSA programs is explained in this section and is illustrated in Figure 2.

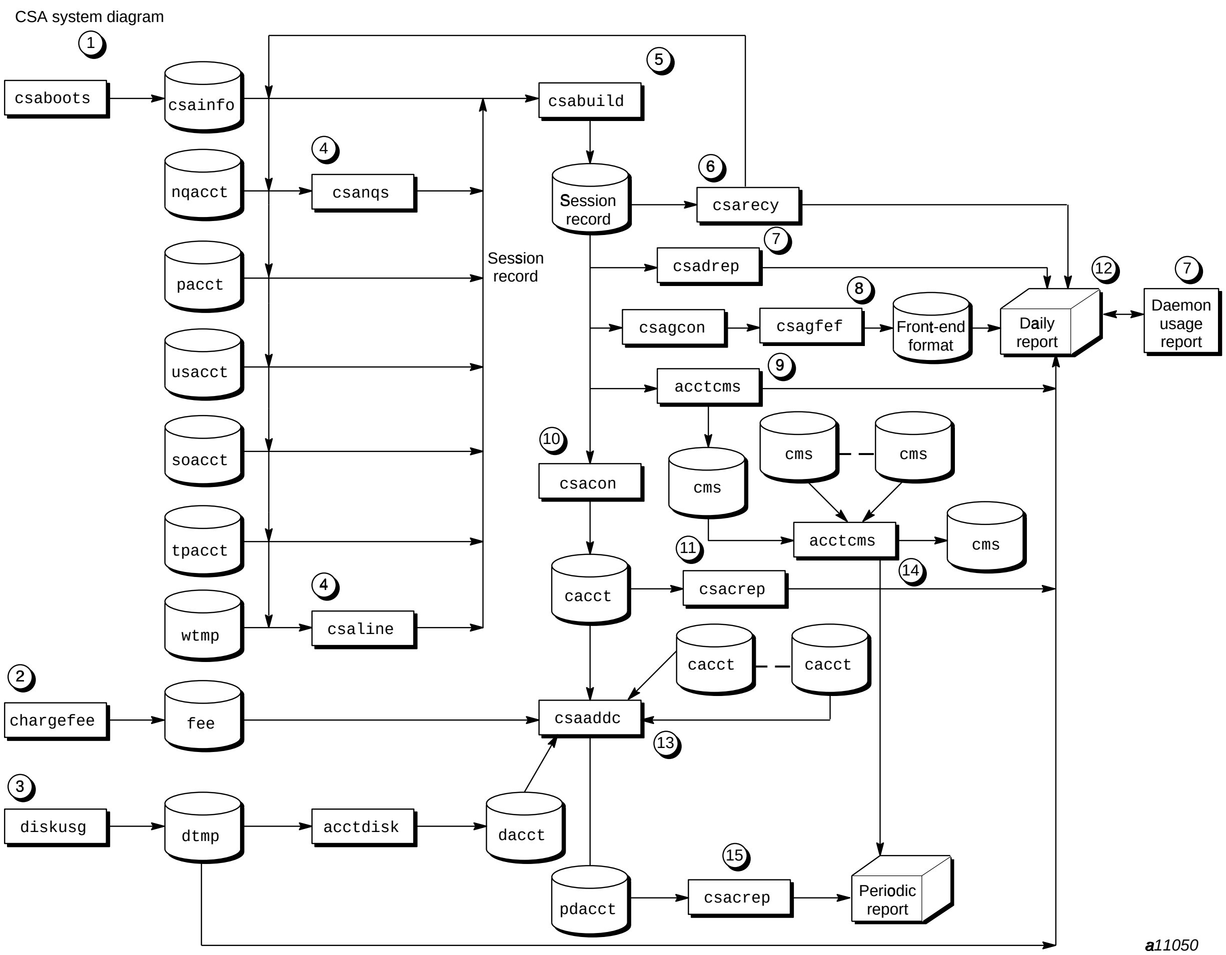

Figure 2. CSA program data flow

1. Generate raw accounting files. Various daemons and system processes write to the raw accounting files. These accounting files include `pacct`, `nqacct`, `soacct`, `usacct`, `tpacct`, `wtmp`, and `csainfo`.

2. Create a fee file. Sites that want to charge fees to certain users can do so with the `chargefee`(8) command. `chargefee` creates a fee file that is processed by `csaaddc`(8).

3. Produce disk usage statistics. The `dodisk`(8) shell script allows sites to take snapshots of disk usage. `dodisk` does not report dynamic usage; it only

reports the disk usage at the time the command was run. Disk usage is processed by `csaaddc`.

4. Preprocess selected raw accounting files. Generally, a data file that must be preprocessed contains multiple records for a session. These records are scattered throughout the file, and the processing of the records often depends upon other events that are logged in the accounting file (for example, system reboot). The preprocessor collapses information about a session into one output record.

   NQS and connect time accounting data are preprocessed by `csanqs`(8) and `csaline`(8), respectively.

5. Organize the accounting data. `csabuild`(8) organizes the raw and preprocessed accounting data by sessions and boot times. With the exception of disk usage statistics and fees, the `csabuild` output file contains all of the accounting data available about each session.

   Sometimes data for terminated sessions is continually recycled. This can occur when accounting data is lost. To prevent data from recycling forever, edit `csarun` so that `csabuild` is executed with the `-o` *nday* option, which causes all sessions older than *nday* days to terminate. Select an appropriate *nday* value (see the `csabuild`(8) man page for more information).

6. Recycle information about unfinished sessions. Accounting data about uncompleted sessions is saved and processed again during the next accounting period. This information is recycled until the session completes or until manual intervention occurs. Accounting data for unfinished sessions is reported during each accounting period.

7. Generate the daemon usage report, which is appended to the daily report. `csadrep`(8) outputs information about interactive, NQS, tape, and socket usage.

8. Convert the session record file to a front-end format. Sites that process UNICOS accounting data on a front-end system can convert the session file to a format suitable for use on the front end by using the `csafef`(8), `csafef2`(8), or `csaibm`(8) command. These programs are templates, and you must modify them to suit your site's requirements. It is suggested that you use the user exit in the `FEF` section of `csarun` (see Section 2.1.5, page 16 and Section 2.1.10.3, page 51) to convert the session record file to your front-end format.

9. Generate command usage data. The information output by `acctcms`(8) is reported in the daily and periodic reports.

10. Consolidate the session record file. Session files are too large to retain on disk for any amount of time. Consequently, CSA consolidates the data and keeps the condensed version on disk. The accounting reports are based on the consolidated data. Data consolidation is done by `csacon`(8).

11. Generate an accounting report based on the consolidated data. `csacrep`(8) outputs the report.

12. Create the daily accounting report. The daily accounting report includes the following:

    - Connect time statistics (step 4)
    - Disk usage statistics (step 3)
    - Unfinished session information (step 6)
    - Command summary data (step 9)
    - Consolidated accounting report (step 11)
    - Last login information
    - Daemon usage report (step 7)

13. Generate periodic accounting data. Periodic accounting data is an accumulation of the consolidated data created in step 10. `csaaddc`(8) merges condensed data files together. The resulting file contains accounting information for numerous accounting periods.

14. Generate periodic command usage data. `acctcms`(8) merges command usage data from multiple accounting periods. The usage information was created in step 9. Both an ASCII and a binary file are created.

15. Produce a periodic accounting report. `csacrep`(8) is used to generate a report based on a periodic accounting file.

Steps 4 through 12 are performed during each accounting period by `csarun`(8). Periodic accounting (steps 13 through 15) is initiated by the `csaperiod`(8) command. Daily and periodic accounting, as well as fee and disk usage generation (steps 2 through 3), can be scheduled by `cron`(8) to execute regularly. See Section 2.1.4, page 11, for more information.

2.1.9 Data Recycling

A system administrator must correctly maintain recycled data in order to ensure accurate accounting reports. The following sections discuss data recycling and describe how an administrator can purge unwanted recycled accounting data.

Data recycling allows CSA to properly bill sessions that are active during multiple accounting periods. By default, `csarun`(8) reports data only for sessions that terminate during the current accounting period. Through data recycling, CSA preserves data for active sessions until the sessions terminate.

In the `Super-record` file, `csabuild`(8) flags each session as being either active or terminated. `csarecy`(8) reads the `Super-record` file and recycles data for the active sessions. `csacon`(8) consolidates the data for the terminated sessions, which `csaperiod`(8) uses later. `csabuild`, `csarecy`, and `csacon` are all invoked by `csarun`.

`csarun` puts recycled data in the `/usr/adm/acct/day` directory. Data files with names suffixed with `0` contain recycled data. For example, `ctime0`, `nqacct0`, `pacct0`, `tpacct0`, `usacct0`, and `uptime0` are generally the recycled data files that are found in `/usr/adm/acct/day`.

Normally, an administrator should not have to manually purge the recycled accounting data. This purge should only be necessary if accounting data is missing. Missing data can cause sessions to recycle forever and consume valuable CPU cycles and disk space.

2.1.9.1 How Sessions Are Terminated

Interactive sessions, `cron` jobs, and `at` jobs terminate when the last process in the job exits. Normally, the last process to terminate is the login shell. The kernel writes an end-of-job (`EOJ`) record to the `pacct` file when the session terminates.

When the NQS daemon delivers an NQS request's output, the request terminates. The daemon then writes an `NQ_DISP` record type to the `NQS` accounting file, while the kernel writes an `EOJ` record to the `pacct` file.

Unlike interactive sessions, NQS requests can have multiple `EOJ` records associated with them. In addition to the request's `EOJ` record, there can be `EOJ` records for pipe clients, net clients, and checkpointed portions of the request. The pipe client and net client perform NQS processing on behalf of the request.

The `csabuild` command flags sessions in the `Super-record` file as being terminated if they meet one of the following conditions:

- The session is an interactive, cron, or at job, and there is an EOJ record for the job in the pacct file.
- The session is an NQS request, and there is both an EOJ record for the request in the pacct file and an NQ_DISP record type in the NQS accounting file.
- The session is an interactive, cron, or at job and is active at the time of a system crash.
- The session is manually terminated by the administrator using one of the methods described in Section 2.1.9.3, page 34.

2.1.9.2 Why Recycled Sessions Should Be Scrutinized

Recycling unnecessary data can consume large amounts of disk space and CPU time. The session file and recycled data can occupy a vast amount of disk space on the file systems containing /tmp and /usr/adm/acct/day. Sites that archive data also require additional offline media. Wasted CPU cycles are used by csarun to reexamine and recycle the data. Therefore, to conserve disk space and CPU cycles, unnecessary recycled data should be purged from the accounting system.

Any of the following situations can cause CSA erroneously to recycle terminated sessions:

- Kernel or daemon accounting is turned off. At boot time, the rc command must execute /usr/lib/acct/startup in order to start kernel and daemon accounting.

  The kernel, ckpacct(8) command, or ckdacct(8) command can turn off accounting when there is not enough space on the file system containing /usr/adm/acct/day.
- Accounting files are corrupt. Accounting data can be lost or corrupted during a system or disk crash.
- Boot times are not recorded in /etc/csainfo. The csaboots command must be invoked by rc to write a boot time record to /etc/csainfo.
- Recycled data is erroneously deleted in a previous accounting period.

2.1.9.3 How to Remove Recycled Data

Before choosing to delete recycled data, you should understand the repercussions, as described in Section 2.1.9.4, page 36. Data removal can affect billing and can alter the contents of the consolidated data file, which is used by csaperiod.

You can remove recycled data from CSA in the following ways:

- Interactively execute the csarecy -A command. Administrators can select the active sessions that are to be recycled by running csarecy with the -A option. Users are not billed for the resources used in the sessions terminated in this manner. Deleted data is also not included in the consolidated data file.

  The following example is one way to execute csarecy -A (which generates two accounting reports and two consolidated files):

  1. Run csarun at the regularly scheduled time.

  2. Edit a copy of /usr/lib/acct/csarun. Change the -r option on the csarecy invocation line to -A. Also, do not redirect standard output to ${CRPT}/recyrpt. The result should be similar to the following:

     ```
     csarecy -A -s ${SESSION_FILE} \
     -N ${WORK}/Rnqacct -P ${WORK}/Rpacct \
     -T ${WORK}/Rtpacct -U ${WORK}/Ruptime \
     -C ${WORK}/Rctime -u ${WORK}/Rusacct \
     2> ${NITE}/Erec.${DTIME}
     ```

     Since both the -A and -r options write output to stdout, the -r option is not invoked and stdout is not redirected to a file. As a result, the recycled job report is not generated.

  3. Execute the jstat command, as follows, to display a list of currently active jobs:

     ```
     jstat > jstat.out
     ```

  4. Execute the qstat command to display a list of NQS requests. The qstat command is used for seeing whether there are requests that are not currently running. This includes requests that are checkpointed, held, queued, or waiting.

     In order to list all NQS requests, execute the qstat command, as follows, using a login that has either NQS manager or NQS operator privilege:

     ```
     qstat -a > qstat.out
     ```

5. Interactively run the modified version of csarun. If you execute csarun soon after the first step is complete, this invocation of csarun completes quickly because not very much data exists.

   For each active session, csarecy asks you if you want to preserve the session. Preserve the active and nonrunning NQS sessions found in the third and fourth steps. All other sessions are candidates for removal.

- Execute csabuild with the -o *ndays* option, which terminates all active sessions older than the specified number of days. Resource usage for these terminated sessions is reported by csarun, and users are billed for the sessions. The consolidated data file also includes this resource usage.

  To execute csabuild with the -o option, edit /usr/lib/acct/csarun. Add the -o *ndays* option to the csabuild invocation line. Specify for *ndays* an appropriate value for your site.

  Recycled data for currently active sessions will be removed if you specify an inappropriate value for *ndays*.

- Execute csarun with the -A option. It reports resource usage for both active and terminated sessions, so users are billed for recycled sessions. This data is also included in the consolidated data file.

  None of the data for the active sessions, including the currently active sessions, is recycled. No recycled data files are generated in the /usr/adm/acct/day directory.

- Remove the recycled data files from the /usr/adm/acct/day directory. You can delete data for all of the recycled sessions, both terminated and active, by executing the following command:

  ```
  rm /usr/adm/acct/day/*[a-z]0
  ```

  The next time csarun is executed, it will not find data for any recycled sessions. Thus, users are not billed for the resources used in the recycled sessions, and this data is not included in the consolidated data file. csarun recycles the data for currently active sessions.

2.1.9.4 Adverse Effects of Removing Recycled Data

CSA assumes that all necessary accounting information is available to it, which means that CSA expects kernel and daemon accounting to be enabled and recycled data not to have been mistakenly removed. If some data is unavailable, CSA may provide erroneous billing information. Sites should be aware of the following facts before removing data:

- Users may or may not be billed for terminated recycled sessions. Administrators must understand which of the previously described methods cause the user to be billed for the terminated recycled sessions. It is up to the site to decide whether or not it is valid for the user to be billed for these sessions.

  For those methods that cause the user to be billed, both `csarun` and `csaperiod` report the resource usage.

- It may be impossible to reconstruct a terminated recycled session. If a recycled session is terminated by the administrator, but the session actually terminates in a later accounting period, information about the session is lost. If a user questions the resource billing, it may be extremely difficult or impossible for the administrator to correctly reassemble all accounting information for the session in question.

- Manually terminated recycled sessions be improperly billed in a future billing period. If the accounting data for the first portion of a session has been deleted, CSA may be unable to correctly identify the remaining portion of the job. Errors may occur, such as NQS requests being flagged as interactive sessions, or NQS requests being billed at the wrong queue rate. This is explained in detail in Section 2.1.9.5, page 38.

- CSA programs may detect data inconsistencies. When accounting data is missing, CSA programs may detect errors and abort.

The following table summarizes the effects of using the methods described in Section 2.1.9.3, page 34.

Table 1. Possible effects of removing recycled data

Method	Underbilling?	Incorrect billing?	Consolidated data file
`csarecy -A`	Yes. Users are not billed for the portion of the session that was terminated by `csarecy -A`.	Possible. Manually terminated recycled sessions may be billed improperly in a future billing period.	Does not include data for sessions terminated by `csarecy -A`.
`csabuild -o`	No. Users are billed for the portion of the session that was terminated by `csabuild -o`.	Possible. Manually terminated recycled sessions may be billed improperly in a future billing period.	Includes data for sessions terminated by `csabuild -o`.
`csarun -A`	No. All active and recycled sessions are billed.	Possible. All active and recycled sessions that eventually terminate may be billed improperly in a future billing period, because no data is recycled.	Includes data for all active and recycled sessions.
`rm`	Yes. All users are not billed for the portion of the session that was recycled.	Possible. All recycled sessions that eventually terminate may be billed improperly in a future billing period.	Does not include data for any recycled session.

By default, the consolidated data file contains data only for terminated sessions. Manual termination of recycled data may cause some of the recycled data to be included in the consolidated file. These cases are noted in the previous table.

2.1.9.5 NQS Requests and Recycled Data

In order for CSA to identify all NQS requests, data must be properly recycled. When an administrator manually purges recycled data for an NQS request, errors such as the following can occur:

- CSA flags the NQS request as an interactive session. This causes the request to be billed at interactive rates.
- The request is billed at the wrong queue rate.
- The wrong queue wait time is associated with the request.

These errors occur because valuable NQS accounting information was purged by the administrator. Only a few NQS accounting records are written by the NQS daemon, and all of the records are needed for CSA to properly bill NQS requests.

NQS accounting records are only written under the following circumstances:

- The NQS daemon receives a request.
- A request is routed to a queue.
- A request executes. This includes executing a request for the first time, and restarting and rerunning a request.
- A request terminates. A request can terminate because it is completed, requeued, preempted, held, checkpointed, or rerun by the operator.
- Output is delivered.

Thus, for long running requests that span days, there can be days when no NQS data is written. Consequently, it is extremely important that accounting data be recycled. If the site administrator manually terminates recycled sessions, care must be taken to be sure that only nonexistent NQS requests are terminated.

2.1.10 Tailoring CSA

This section describes the following actions in CSA:

- Setting up SBUs
- Setting up daemon accounting
- Setting up user exits
- Modifying the front-end formatting templates
- Modifying the charging of NQS jobs based on NQS termination status
- Tailoring CSA shell scripts
- Using `at`(1) instead of `cron`(8) to periodically execute `csarun`
- Allowing users without super-user permissions to execute CSA
- Using an alternate configuration file

2.1.10.1 System Billing Units (SBUs)

A *system billing unit* (SBU) is a unit of measure that reflects use of machine resources. You can alter the weighting factors associated with each field in each accounting record to obtain an SBU value suitable for your site. SBUs are defined in the accounting configuration file, `/etc/config/acct_config`. By default, all SBUs are set to `0.0`.

The source code for the default SBU calculations is located in `/usr/src/cmd/acct/lib/acct/sbu.c`. For sites that do not have source code, the default algorithms are also defined in `/usr/src/cmd/acct/lib/acct/user_sbu.c`. By modifying `/usr/src/cmd/acct/lib/acct/user_sbu.c`, compiling, and relinking the accounting programs, your site can use local SBU calculations.

Accounting allows different periods of time to be designated either prime or nonprime time (the time periods are specified in `/usr/lib/acct/holidays`).

Following is an example of how the prime/nonprime algorithm works:

Assume a user uses 10 seconds of CPU time, and executes for 100 seconds of prime wall-clock time, and pauses for 100 seconds of nonprime wall-clock time. Therefore, elapsed time is 200 seconds (100+100). If

prime = *prime time* / *elapsed time*
nonprime = *nonprime time* / *elapsed time*
cputime[PRIME] = *prime* * *CPU time*
cputime[NONPRIME] = *nonprime* * *CPU time*

then

cputime[PRIME] `== 5 seconds`
cputime[NONPRIME] `== 5 seconds`

Under CSA, an SBU value is associated with each record in the Session record file when that file is assembled by `csabuild`(8). Final summation of the SBU values is done by `csacon`(8) during the creation of the `cacct` record file.

Billing for SBU values is intended to be a combination of all the SBU values from each record associated with a job, as follows:

*Total SBU = (NQS queue SBU value) * (sum of all pacct record SBUs*
+ sum of all tape record SBUs
+ sum of all ctmp record SBUs)

This allows a site to bill different NQS queues at differing rates. Again, if the available formulas are insufficient to achieve the site's requirements, a site can modify the calculations found in the sbu library routine, /usr/src/cmd/acct/lib/acct/user_sbu.c.

2.1.10.1.1 pacct SBUs

The SBUs for pacct data are separated into prime and nonprime values. Prime and nonprime use is calculated by a ratio of elapsed time. If you do not want to make a distinction between prime and nonprime time, set the nonprime time SBUs and the prime time SBUs to the same value. Prime time is defined in /usr/lib/acct/holidays. By default, Saturday and Sunday are considered nonprime time.

The following is a list of prime time pacct SBU weights. Descriptions and factor units for the nonprime time SBU weights are similar to those listed here. SBU weights are defined in /etc/config/acct_config.

Value	Description
P_BASIC	Prime-time weight factor. P_BASIC is multiplied by the sum of prime time SBU values to get the final SBU factor for the pacct base record.
P_TIME	General-time weight factor. P_TIME is multiplied by the time SBUs (made up of P_STIME, P_ITIME, P_SCTIME, and P_INTTIME) to get the time contribution to the pacct base record SBU value.
P_STIME	System CPU-time weight factor. The unit used for this weight is *billing units* per second. P_STIME is multiplied by the system CPU time to get the system CPU factor.
P_UTIME	User CPU-time weight factor. The unit used for this weight is *billing units* per second. P_UTIME is multiplied by the user CPU time to get the user CPU factor. User time is the sum of user times after weighting for multitasking. Multitasking may affect the user CPU cost if the MUTIME_WEIGHT parameters have been set to values other than 1.0. See the following explanation of these values.
P_ITIME	This is the weight factor for the time spent waiting in the kernel for I/O while the process is

locked in memory. The unit used for this weight is *billing units* per second. P_ITIME is multiplied by the I/O wait time.

P_SCTIME
: Weight factor for system call time. The unit used for this weight is *billing units* per second.

P_INTTIME
: Weight factor for interrupt time. The unit used for this weight is *billing units* per second.

P_MEM
: General-memory-integral weight factor. P_MEM is multiplied by the memory SBUs (made up of P_XMEM and P_IMEM) to get the memory contribution to the pacct base record SBU value.

P_XMEM
: CPU-time-memory-integral weight factor. The unit used for this weight is *billing units* per Kword-minute. P_XMEM is multiplied by the memory integral (see Section 2.1.12.1, page 62). This value is affected by your site's choice of MEMINT (in the accounting configuration file /etc/config/acct_config).

P_IMEM
: The weight factor used with the I/O wait time memory integral. This integral includes the I/O wait time while the process is locked in memory. The unit used for this weight is *billing units* per Kword-minute. P_IMEM is multiplied by the I/O-wait-time memory integral.

P_IO
: General-I/O weight factor. P_IO is multiplied by the I/O SBUs (made up of P_BYTEIO, P_PHYIO, and P_LOGIO) to get the I/O contribution to the pacct base record SBU value.

P_BYTEIO
: Characters-transferred weight factor. The unit used for this weight is *billing units* per character transferred. P_BYTEIO is multiplied by the bytes of I/O transferred.

  If tape or device I/O is to be charged at a rate other than P_BYTEIO, the tape and device weight factors need to be adjusted accordingly. See Section 2.1.11.1, page 56 (field ac_io), for more information.

P_PHYIO
: Physical-I/O-request weight factor. The unit used for this weight is *billing units* per physical I/O request. P_PHYIO is multiplied by the number of physical I/O requests made. Physical requests are the number of driver requests made.

P_LOGIO
: Logical-I/O-request weight factor. The unit used for this weight is *billing units* per logical I/O request. P_LOGIO is multiplied by the number of logical I/O requests made. The number of logical I/O requests is the total number of read(2), write(2), reada(2), and writea(2) system calls. The number of strides, multiplied by the number of requests processed by each listio(2) call, is also added to the total.

2.1.10.1.2 Multitasking SBUs

The MUTIME_WEIGHT *i* variables define the weighting factors that are used to charge user CPU time for multitasking programs. It is used in conjunction with the ac_mutime array (see /usr/include/sys/acct.h), which defines the amount of CPU time the multitasking program spent with *i* + 1 CPUs connected.

MUTIME_WEIGHT *i* defines the marginal cost of getting the *i* + 1 CPU at one instant. If these values are set to less than 1.0, there is an incentive for multitasking. If the values are set to 1.0, multitasking programs are charged for user CPU time just as all other programs.

For more information on multitasking incentives, see Section 2.1.12, page 62.

2.1.10.1.3 SDS SBUs

(On all Cray Research systems except the CRAY EL series) Secondary data storage (SDS) system billing units are calculated from the statistics on SDS use in the pacct file. The SBU factors are defined in /etc/config/acct_config.

The values are as follows:

<u>Value</u> <u>Description</u>

NP_SDSMEM or P_SDSMEM

SDS-memory-integral weight factor. The memory integral is based on residency time and not on execution time. P_SDSMEM or NP_SDSMEM is multiplied by the SDS memory integral. The unit used for this weight is *billing units* per Mword-second.

NP_SDSLOGIO or P_SDSLOGIO

SDS-logical-I/O-request weight factor. P_SDSLOGIO or NP_SDSLOGIO is multiplied by the number of SDS logical I/O requests. The unit used for this weight is *billing units* per logical I/O request.

NP_SDSBYTEIO or P_SDSBYTEIO

SDS-characters-transferred weight factor. P_SDSBYTEIO or NP_SDSBYTEIO is multiplied by the number of SDS characters transferred. The unit used for this weight is *billing units* per character transferred.

The SBU values should be very small. On Cray Research systems, it is possible to submit a very large number of requests to SDS in a short time; therefore, to prevent these numbers from dominating the SBU values, small weight factors must be used. Values of 0 result in no charge.

2.1.10.1.4 MPP SBUs

Massively parallel processing (MPP) system billing units are calculated from the statistics on MPP use in the pacct file. The SBU factors are defined in /etc/config/acct_config.

The P_MPPPE or NP_MPPPE SBUs are the MPP processing elements (PEs) weight factors, prime and nonprime charges. The prime time billing units for PEs is calculated using the following equation:

$$\texttt{P\_MPPPE billing units = P\_MPPPE *} \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP PEs used} * \textit{MPP time used})$$

The nonprime time billing units for PEs is calculated using the following equation:

$$\texttt{NP\_MPPPE billing units = NP\_MPPPE *} \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP PEs used} * \textit{MPP time used})$$

The unit used for these weights is billing units per PE-second.

The P_MPPBB or NP_MPPBB SBUs are the MPP barrier bits weight factors, prime and nonprime charges.[1] The prime time billing units for barrier bits is calculated using the following equation:

$$\texttt{P\_MPPBB billing units = P\_MPPBB *} \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP barrier bits used} * \textit{MPP time used})$$

The nonprime time billing units for barrier bits is calculated using the following equation:

$$\texttt{NP\_MPPBB billing units = NP\_MPPBB *} \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP barrier bits used} * \textit{MPP time used})$$

The unit used for these weights is *billing units* per barrier bit-second.

The P_MPPTIME or NP_MPPTIME SBUs are the MPP time weight factors, prime and nonprime charges. The prime time billing units for MPP time is calculated using the following equation:

$$\texttt{P\_MPPTIME billing units = P\_MPPTIME *} \sum_{0}^{\#\ of\ sessions} (\textit{MPP time used})$$

The nonprime time billing units for MPP time is calculated using the following equation:

$$\texttt{NP\_MPPTIME billing units = NP\_MPPTIME *} \sum_{0}^{\#\ of\ sessions} (\textit{MPP time used})$$

The unit used for these weights is *billing units* per second.

The SBU values should be very small, which will prevent these numbers from dominating the SBU values. Values of 0 result in no charge.

2.1.10.1.5 Connect Time SBUs

There are SBUs for both prime- and nonprime-time connect data. The SBU values should reflect the system billing units per second of connect time. The weight factors, CON_PRIME and CON_NONPRIME, are defined in /etc/config/acct_config.

2.1.10.1.6 NQS SBUs

The /etc/config/acct_config file contains the configurable parameters that pertain to NQS SBUs.

[1] Deferred implementation.

The NQS_NUM_QUEUES parameter sets the number of queues for which you want to set SBUs (the value must be set to at least 1). Each NQS_QUEUE *x* variable in the configuration file has a queue name and an SBU pair associated with it (the total number of queue/SBU pairs must equal NQS_NUM_QUEUES). The queue/SBU pairs define weights for the queues. If an SBU value is less than 1.0, there is an incentive to run jobs in the associated queue; if the value is 1.0, jobs are charged as though they are non-NQS jobs; and if the SBU is 0.0, there is no charge for jobs running in the associated queue. SBUs for queues not found in the configuration file are automatically set to 1.0.

The NQS_NUM_MACHINES parameter sets the number of originating machines for which you want to set SBUs (the value must be at least 1). Each NQS_MACHINE *x* variable in the configuration file has an originating machine and an SBU pair associated with it (the total number of machine/SBU pairs must equal NQS_NUM_MACHINES). SBUs for originating machines not specified in /etc/config/acct_config are automatically set to 1.0.

The queue and machine SBUs are multiplied together to give an NQS multiplier. If the SBUs are set to less than 1.0, there is an incentive to run jobs in these queues or from these machines. SBUs of 1.0 indicate that jobs in the queues or from associated hosts are billed normally.

2.1.10.1.7 Socket SBUs

Currently, there is no way to charge for socket accounting. The socket accounting records produced are only processed in order to make the data available to the site-supplied user exits.

2.1.10.1.8 Tape SBUs

There is a set of weighting factors for each group of tape devices. By default, there are only two groups, tape and cart. The TAPE_SBU*i* parameters in /etc/config/acct_config define the weighting factors for each group. There are SBUs associated with the following:

- Number of mounts
- Device reservation time (seconds)
- Number of bytes read
- Number of bytes written

2.1.10.1.9 Device SBUs

Device accounting system billing units are calculated from the device statistics in the pacct file. SBUs can be set for both block and character devices in /etc/config/acct_config. The fields in the acct_config file that affect SBU factors for each device are as follows:

SBU factor	Description
Logical I/O Sbu	Weight given to each logical I/O request.
Characters Xfer Sbu	Weight given to the amount of data transferred.
Device Name	Device type name (see Section 2.1.14, page 65).

The Logical I/O Sbu factor is multiplied by the number of system calls that initiated I/O on a device type. The Characters Xfer Sbu factor is multiplied by the number of bytes of data transferred to a device type.

The SBUs for block devices are labeled BLOCK_DEVICE *x*, where *x* is a number from 0 to MAXBDEVNO-1. Character devices are labeled CHAR_DEVICE *x*, where *x* is a number from 0 to MAXCDEVNO-1. The numeric suffixes for the CHAR_DEVICE *x* variables must match the minor numbers in /dev, which are defined in /usr/src/uts/c1/cf/devsw.c in the cdevsw[] array.

MAXBDEVNO and MAXCDEVNO are located in the /usr/include/sys/param.h include file and have default values of 10 and 35, respectively.

Device accounting is also discussed in Section 2.1.14, page 65.

The SBU values should be very small. On Cray Research systems, it is possible to perform a very large number of I/O requests very quickly; therefore, to prevent these numbers from dominating the SBU values, a small weight factor must be used. A value of 0 results in no charge.

2.1.10.1.10 Example SBU Settings

The following section provides an example showing how you could set up the SBU system. This example is restricted to pacct base records (you should also consider pacct multitasking, pacct I/O (device accounting), and all the daemon records). In this example, it is assumed that an SBU is equal to one dollar of charge.

The formula for calculating the whole pacct base record SBU value is as follows:

```
PSBU = ((P_TIME * (P_STIME * stime + P_UTIME * utime + P_ITIME *
iowtime)) + (P_MEM * (P_XMEM * cpumem + P_IMEM * iowmem)) +
(P_IO * (P_BYTEIO * bytes + P_PHYIO * phy + P_LOGIO * log)));

NSBU = ((NP_TIME * (NP_STIME * stime + NP_UTIME * utime + NP_ITIME
* iowtime)) + (NP_MEM * (NP_XMEM * cpumem + NP_IMEM * iowmem)) +
(NP_IO*(NP_BYTEIO * bytes + NP_PHYIO * phy + NP_LOGIO * log)));

SBU = P_BASIC * PSBU + NP_BASIC * NSBU;
```

The variables in this formula are as follows:

Variable	Description
stime	System CPU time in seconds.
utime	User CPU time in seconds. User CPU time is the sum of user times after weighting for multitasking.
iowtime	Time (in seconds) spent waiting in the kernel for I/O while the process is locked in memory.
cpumem	Memory integral (see Section 2.1.12.1, page 62).
iowmem	I/O-wait-time memory integral.
bytes	Number of bytes of data transferred.
phy	Number of physical I/O requests made.
log	Number of logical I/O requests made.

All time is considered prime time. Therefore, the nonprime time SBUs should be set to the same values as their prime time counterparts.

In order to produce a billing that is somewhat repeatable, this example omits various values, such as physical I/O (set P_PHYIO to 0.0), that depend on the state of the machine at run time. In particular, system time varies greatly due to system load and will cause this example to be nonrepeatable. Information on which fields generate repeatable values is contained in Section 2.1.11.1, page 56.

In this example, users are charged for each logical request (P_LOGIO) and the total data moved (P_BYTEIO). This provides users with an incentive to use larger I/O requests, which may be more efficient. Processes that perform I/O that locks them into memory are penalized (P_IMEM), because this may result in memory fragmentation.

In this example, users are charged the following amounts for time (the accounting record fields associated with the charge are also identified):

- $100 per hour of user CPU time. This is equal to $100 per 3600 seconds, which is $0.02777777 per second (P_UTIME). To produce repeatable billing, system time must be excluded. Thus, P_STIME is set to 0.0.
- $25 for each megaword of memory per hour of CPU time. The memory integrals are in units of Kword-minutes, so the weighting factor is $25/(60 minutes * 2^{10} Kwords) or 0.0004069010 (P_XMEM).
- $3 for each hour spent waiting on I/O while locked into memory. The wait time is in units of seconds. so the weighting factor is $3/3600 seconds or .0008333333 (P_ITIME).
- $25 for I/O wait time (locked in memory) per hour. This is the same value as the memory charge because the process is using memory during this time in the same way it would when executing. The weighting factor is $25/(60 seconds * 2^{10} Kwords) or 0.0004069010 (P_IMEM).
- A DD-49 disk drive can perform I/O at a maximum rate of 9.6 Mbytes per second. Assume that the original cost of the drive was $125,000, and it will be paid for in 2 years. Also assume that it is busy 5% of the time (63072000 seconds * 5% = 3153600 seconds). The amount of I/O that can be completed in 2 years is 31745177026560 bytes (9.6 Mbytes/second * 3153600 seconds). Thus, you would charge $125,000/31745177026560 bytes or $0.00000000393760 per byte, which is approximately $0.33/10 Mwords (P_BYTEIO).
- $0 for physical I/O requests. This charge makes the billing more repeatable. The byte I/O charge covers this activity (P_PHYIO).
- $0.01 per thousand logical I/O requests. This charge encourages the user to perform larger I/O requests by charging less for a lower number of larger I/O requests (instead of a lot of small I/O requests). The weighting factor is computed as $0.01/1000 I/O requests or 0.00001 (P_LOGIO).

Therefore, in this example, the pacct base record charges are as follows (the nonprime time SBUs are set to the same value as their prime time counterparts):

Weight factor	Charge
P_BASIC	1.0

P_TIME	1.0
P_UTIME	0.02777777777777
P_STIME	0.0
P_ITIME	0.00083333333333
P_MEM	1.0
P_XMEM	0.00040690104166
P_IMEM	0.00040690104166
P_IO	1.0
P_BYTEIO	0.00000000393760
P_PHYIO	0.0
P_LOGIO	0.00001000000000

P_BASIC, P_TIME, P_MEM, and P_IO are used to weight different factors of the equation; you can use these depending on how your other groups of weighting factors are picked. For example, you could change the P_IO and P_BYTEIO factors as follows and receive the same results:

Weight factor	Charge
P_BASIC	1.0
P_TIME	1.0
P_UTIME	0.02777777777777
P_STIME	0.0
P_ITIME	0.00083333333333
P_MEM	1.0
P_XMEM	0.00040690104166
P_IMEM	0.00040690104166
P_IO	0.00001
P_BYTEIO	0.000393760
P_PHYIO	0.0

`P_LOGIO`	1.0

2.1.10.2 Daemon Accounting

Accounting information is available from NQS, online tapes, and sockets. Data is written to the `nqacct`, `tpacct`, and `soacct` files, respectively, in the `/usr/adm/acct/day` directory.

In most cases, daemon accounting must be enabled by both the CSA subsystem and the daemon. Section 2.1.4, page 11, describes how to enable daemon accounting at system startup time. You can also enable daemon accounting after the system has booted.

You can enable accounting for a specified daemon with the `turndacct`(8) command. For example, to start tape accounting, you would execute the following:

```
/usr/lib/acct/turndacct on tape
```

The NQS and online tape daemon also must enable accounting. Use the `qmgr set accounting on` command to turn on NQS accounting. Tape daemon accounting is enabled when `tpdaemon`(8) is executed with the `-c` option.

Daemon accounting is disabled by `shutacct`(8) at system shutdown (see Section 2.1.4, page 11). It can also be disabled at any time by the `turndacct`(8) command when used with the `off` operand. For example, to disable NQS accounting, execute the following command:

```
/usr/lib/acct/turndacct off nqs
```

New daemon accounting files can be started when `turndacct` is invoked with the `switch` operand. No data is lost when files are switched. For example, to start a new NQS accounting file, execute the following command:

```
/usr/lib/acct/turndacct switch nqs
```

2.1.10.3 Setting up User Exits

CSA accommodates the following user exits, which can be called from certain `csarun` states:

`csarun` state	User exit
`ARCHIVE1`	`/usr/lib/acct/csa.archive1`
`ARCHIVE2`	`/usr/lib/acct/csa.archive2`

FEF	/usr/lib/acct/csa.fef
USEREXIT	/usr/lib/acct/csa.user

These exits allow an administrator to tailor the csarun procedure to the individual site's needs by creating scripts to perform additional site-specific processing during daily accounting.

While executing, csarun checks in the ARCHIVE1, ARCHIVE2, FEF, and USEREXIT states for a shell script with the appropriate name.

If the script exists, it is executed via the shell . (dot) command. If the script does not exist, the user exit is ignored. The . (dot) command will not execute a compiled program, but the user exit script can. csarun variables are available, without being exported, to the user exit script. csarun checks the return status from the user exit and, if it is nonzero, the execution of csarun is terminated.

If CSA is run by a user without super-user permissions, the user exits must be both readable and executable by this user (see page Section 2.1.10.7, page 54).

2.1.10.4 Charging for NQS Jobs

By default, SBUs are calculated for all NQS jobs regardless of the job's NQS termination code. If you do not want to bill portions of an NQS request, set the appropriate NQS_TERM_ *xxxx* variable (termination code) in /etc/config/acct_config to 0, which sets the SBU for this portion to 0.0. By default, all portions of a request are billed.

The following table describes the termination codes:

Code	Description
NQS_TERM_EXIT	Generated when the request finishes running and is no longer in a queued state. At NQS shutdown time, requests that specified both the -nc (no checkpoint) and -nr (no rerun) options for qsub also have NQS_TERM_EXIT records written. In addition, this record is written for requests that specified the -nr option for qsub and were running at the time of a system crash.
NQS_TERM_REQUEUE	Written for running requests that are checkpointed and then requeued when NQS shuts down.
NQS_TERM_PREEMPT	Written when a request is preempted with the qmgr preempt request command.

NQS_TERM_HOLD	Written for a request that is checkpointed with the qmgr hold request command. The hold request command differs from the checkpoint done at daemon shutdown time because a "hold" keeps the job from being scheduled until a qmgr release command is executed.
NQS_TERM_OPRERUN	Written when a request is rerun with the qmgr rerun request command. At NQS shutdown time, jobs that cannot be checkpointed and do not have the -nr (no rerun) option for qsub specified have this type of termination record written. The requests are requeued with this status.

2.1.10.5 Tailoring CSA Shell Scripts and Commands

Modify the following variables in /etc/config/acct_config if necessary:

Variable	Description
ACCT_FS	File system on which /usr/adm/acct resides. The default is /usr.
MAIL_LIST	List of users to whom mail is sent if fatal errors are detected in the accounting shell scripts. The default is root and adm.
WMAIL_LIST	List of users to whom mail is sent if warning errors are detected by the csarun script at cleanup time. The default is root and adm.
MIN_BLKS	Minimum number of free blocks needed in ${ACCT_FS} to run csarun or csaperiod. The default is 500 free blocks.

2.1.10.6 Using at to Execute csarun

You can use the at(1) command instead of cron(8) to execute csarun periodically. If your Cray Research system is down when csarun is scheduled to run via cron, csarun will not be executed until the next scheduled time. On the other hand, at jobs execute when the machine reboots if their scheduled execution time was during a down period.

You can execute csarun with at in several ways. For instance, a separate script can be written to execute csarun and then resubmit the job at a specified time. Also, an at invocation of csarun could be placed in a user exit script, /usr/lib/acct/csa.user, that is executed from the USEREXIT section of csarun. See Section 2.1.10.3, page 51, for more information.

2.1.10.7 Allowing Nonsuper Users to Execute CSA

Your site may want to allow users without super-user permissions to run CSA accounting. CSA can be run by users who are in the group adm and have permission bit acct set in their UDB entries.

Note: If root has run CSA, you must execute the shell script /usr/lib/acct/csaperm (see csaperm(8)) to change the group ID and file permissions of all accounting files in /usr/adm/acct so they can be accessed by a nonsuper user running CSA.

The following steps describe the process of setting up CSA so it is executed automatically on a daily basis by a user without super-user permissions. In this example, the user without super-user permissions is adm:

1. Ensure that user adm is a member of group adm and has the permission bit acct set in its UDB entry (see udbgen(8)).

2. As root, execute the shell script csaperm to change the group ID and file permissions of all accounting files in /usr/adm/acct so they can be accessed by a nonsuper user.

3. Ensure that, if they exist, the user exits /usr/lib/acct/csa.archive1, /usr/lib/acct/csa.archive2, /usr/lib/acct/csa.fef, and /usr/lib/acct/csa.user have the group ID adm and are both readable and executable by group adm.

4. Follow steps 1 through 5 of Section 2.1.4, page 11, to set up system billing units, record system boot times, and turn off accounting before system shutdown.

5. Include an entry similar to the following in /usr/spool/cron/crontabs/root so that cron(8) automatically runs dodisk(8):

   ```
   0 3 * * 1-6 /usr/lib/acct/dodisk -a -v 2> /usr/adm/acct/nite/dk2log
   ```

   dodisk must be executed by root, because no other user has the correct permissions to read /dev/dsk/*.

6. Include entries similar to the following in `/usr/spool/cron/crontabs/adm` so that user `adm` automatically runs daily accounting by using `cron`:

   ```
   0 4 * * 1-6 /usr/lib/acct/csarun 2> /usr/adm/acct/nite/fd2log
   0 * * * * /usr/lib/acct/ckdacct nqs tape
   0 * * * * /usr/lib/acct/ckpacct
   ```

   `csarun`(8) should be executed at a time that allows `dodisk` to complete. If `dodisk` does not complete before `csarun` executes, disk accounting information may be missing or incomplete.

7. To run periodic accounting, place an entry similar to the following in `/usr/spool/cron/crontabs/adm` (this command generates a periodic report on all consolidated data files found in `/usr/adm/acct/sum/data/*` and then deletes those data files):

   ```
   15 5 1 * * /usr/lib/acct/csaperiod -r 2>/usr/adm/acct/nite/pd2log
   ```

8. Update the holidays file as described in Section 2.1.4, page 11.

2.1.10.8 Using an Alternate Configuration File

By default, the `/etc/config/acct_config` configuration file is used when any of the CSA commands are executed. You can specify a different file by setting the shell variable `ACCTCONFIG` to another configuration file, and then executing the CSA commands.

For example, you would execute the following commands in order to use the configuration file `/tmp/myconfig` while executing `csarun`(8):

```
ACCTCONFIG=/tmp/myconfig /usr/lib/acct/csarun 2> /usr/adm/acct/nite/fd2log
```

2.1.10.9 Disk Usage Reporting (`diskusg`)

The `diskusg`(8) command can be configured at your site. The `site.c` module of `diskusg` contains an example to help you in customizing a report for your site. You can delete your choice of comment-protection characters in the example, compile the routine, relink `diskusg`, then print a sample report of disk usage for your site. You can execute your modified `diskusg` command in the `USEREXIT` state in `csarun` or `runacct` scripts.

2.1.11 Per-process Accounting Data

This section describes some of the fields found in the `pacct` file. `/usr/include/sys/acct.h` defines the structure of this file.

2.1.11.1 Base Accounting Record

One base accounting record per process is written; each record contains the following fields:

Table 2. Base accounting record fields by function

Type	Field	Description
Header	`ac_header`	Accounting header record word (see `/usr/include/sys/accthdr.h`)
	`ac_flag`	Accounting flags.
Identifiers	`ac_acid`	Account ID.
	`ac_gid`	Real group ID.
	`ac_jobid`	Job ID.
	`ac_pid`	Process ID.
	`ac_ppid`	Parent process ID.
	`ac_uid`	Real user ID.
Process Information	`ac_btime`	Start time of the process.
	`ac_comm`	Command name (first 8 characters).
	`ac_etime`	Elapsed time while process executed (in clocks). This number is not repeatable.
	`ac_himem`	Memory-use high water mark in words.
	`ac_nice`	Nice value, measured at the end of 1 second of system and user time or the most expensive value used thereafter. This allows a process to set the value at which most of its work should be done; only charges for increased cost are levied.
	`ac_stat`	Low-order 8 bits from process's exit value. See the `wait`(2) man page for more information.

Type	Field	Description
	ac_tty	Controlling tty device.
Counters	ac_ctime	Process raw connect time in clocks. For multitasking jobs, ac_ctime is a sum of the connect time across all CPUs used by the job.
	ac_io	Number of characters transferred by the process. If any tape accouting information existed for this process, the number of tape bytes read and written is included in the ac_io field. Thus, the amount of tape I/O is reported twice: once in the ac_io field and again in the tape accounting record. The ac_io field generally is larger, because it includes additional I/O performed by the process. This number is repeatable. Device accounting I/O information is also reported twice: by ac_io and in the device accounting record field acd_ioch. Charges for doing I/O to tape or to a particular device can be adjusted by setting the SBU weight factors for tape and device I/O. These weights are defined in /etc/config/acct_config. The tape SBUs are labeled TAPE_SBU *x*, and the device SBUs are BLOCK_DEVICE *x* and CHAR_DEVICE *x*. Set the weight factors relative to P_BYTEIO. The ac_io value is multiplied by P_BYTEIO. The tape or device I/O value is multiplied by the appropriate tape or device weight factor. For example, if a surcharge is to be applied to tape I/O, the read and write values for the TAPE_SBU *x* variables must reflect the amount over P_BYTEIO that should be charged.
	ac_iobtim	I/O wait time in clocks measured while the process is not locked in memory (unlike ac_iowtime). System buffer I/O accumulates here. This number may vary due to system load.
	ac_iosw	Swap count. This number may vary due to system load.

Type	Field	Description
	ac_iowtime	I/O wait time (in clocks) measured while the process is locked in memory. This means that system buffered I/O does not appear here. Also, this is a measure of the time elapsed from when a process is removed from the run queue until the process is reconnected to a CPU; therefore, it may vary due to system load.
	ac_lio	Logical I/O request count; this is a count of the read, write, reada, writea, and listio (list entries) system calls made. This number is repeatable.
	ac_rw	Number of physical I/O requests initiated by the process. This number varies due to conditions in the system buffer cache. Therefore, if repeatable billing is desired, this number cannot be used.
	ac_sctime	System call time in clocks.
	ac_stime	System CPU time used (in clocks). This number is not repeatable, because it varies with system load.
	ac_utime	User CPU time used (in clocks). For nonmultitasked processes, this number does not include semaphore wait time and is repeatable (within the limitations caused by memory conflicts).
Integrals	ac_iowmem	I/O-wait-time memory integral measured while the process is locked in memory (in click-ticks). This number may vary due to system load.
	ac_mem	Memory integral selected when MEMINT = 1 (in clicks-ticks). (MEMINT is located in /etc/config/acct_config.) This is the only constant memory integral available (within the limitations caused by memory conflicts); therefore, if repeatable billing is required, this number must be used.

Type	Field	Description
	ac_mem2	Memory integral selected when MEMINT = 2 (in clicks-ticks). (MEMINT is located in /etc/config/acct_config.) This integral is not constant and varies with machine load.
	ac_mem3	Memory integral selected when MEMINT = 3 (in clicks-ticks). (MEMINT is located in /etc/config/acct_config.) This integral is not constant and varies with machine load.

2.1.11.2 End-of-job Accounting Record

There is one end-of-job record per job. The record is written when the last process of a job is terminated. The record contains the following fields:

Table 3. End-of-job accounting record fields by function

Type	Field	Description
Header	ac_header	Accounting header record word (see /usr/include/sys/accthdr.h)
	ac_flag	Accounting flags.
Identifiers	ace_jobid	Job ID of the job to which this record belongs.
	ace_uid	User ID from the job table.
Job Information	ace_etime	End time of the job (in seconds).
	ace_fsblkused	Sum of the file system storage used. This value may be negative if more space was freed up than was consumed.
	ace_himem	High-water memory use of job; sum of all processes in a job at any given time (in clicks). this can vary because of scheduling differences.

Type	Field	Description
	ace_nice	Last nice value of the job.
	ace_sdshiwat	Secondary data segment high-water use; sum of all processes in a job at any given time (in SDS units). This can vary because of scheduling differences.

2.1.11.3 Multitasking Accounting Record

If a process is multitasked, a multitasking accounting record is written when the last member of the multitasked group is terminated. The record contains the following fields:

Field	Description
ac_header	Accounting header record word (see /usr/include/sys/accthdr.h)
ac_flag	Accounting flags.
ac_smwtime	(Not on CRAY C90 systems.) Semaphore wait time (in clocks).
ac_mutime[MUSIZE]	Time a process was connected to exactly (*i* + 1) CPUs (in 1/100ths of a second format). The CPU time used when the process was connected to (*i* + 1) CPUs is ac_mutime[*i*] * (*i* + 1) 1/100ths of a second. For example, ac_mutime[1] is the time a process was connected to two CPUs, and ac_mutime[1] * 2) is the CPU time used while connected to two CPUs. ac_mutime[] includes wait semaphore time. Prior to UNICOS release 8.3, the multitasking CPU times were stored as 21-bit pseudo-floating point numbers. Beginning with release 8.3, these values are in 1/100ths of a second and are compressed as 16-bit pseudo-floating point numbers. The compression and unit changes were made so that multitasking information for a

maximum of 32 CPUs can be stored in the `pacct` file without changing the size of the records.

2.1.11.4 SDS Accounting Record

(Not on CRAY EL systems.) If a process utilizes SDS, an SDS accounting record is written. The record contains the following fields:

Field	Description
`ac_header`	Accounting header record word (see `/usr/include/sys/accthdr.h`)
`ac_flag`	Accounting flags.
`acs_mem`	Memory integral based on residency time, not execution time (in click-ticks).
`acs_lio`	Logical I/O request count; this count is the number of `ssread` and `sswrite` system calls made.
`acs_ioch`	Number of characters transferred to and from the SDS, stored in bytes.

2.1.11.5 MPP Accounting Record

If a process uses a Cray MPP system, an MPP accounting record is written that contains the following fields:

Field	Description
`ac_header`	Accounting header record word (see `/usr/include/sys/accthdr.h`)
`ac_flag`	Accounting flags.
`ac_mpppe`	Number of MPP processor elements used.
`ac_mppbe`	Number of MPP barrier bits used.
`ac_mpptime`	Number of clocks that the MPP has been used.

2.1.11.6 Performance Accounting Record

When the optional performance accounting feature is enabled (by using the `devacct`(8) command with the `-b` option), a performance accounting record is written at the end of each process. Each record contains the following fields:

Field	Description
ac_header	Accounting header record word (see /usr/include/sys/accthdr.h)
ac_flag	Accounting flags.
acp_rtime	The process start time offset (in clocks) from the previous second (reported in the ac_btime field of the base accounting record). This field allows you to trace start times of processes that are spawned in the same second.
acp_tiowtime	The terminal I/O wait time (in clocks); in other words, the period of time starting when a process performing I/O to a tty or pseudo-tty is removed from the run queue and ending when the process is reconnected to a CPU. This number may vary due to system load.
acp_srunwtime	This field is currently disabled.
acp_swapclocks	The time (in clocks) that a process spends on the swap device.
acp_rwblks	The number of physical blocks transferred by the process using the system buffer I/O interface. This number varies due to conditions in the system buffer cache.
acp_phrwblks	The number of physical blocks transferred by the process using the raw I/O interface.

2.1.12 Multitasking Incentives

Some sites may want to provide accounting incentives for the use of multitasking. Several of these are available through the appropriate setting of installation parameters.

2.1.12.1 Memory Integrals

Three different memory integrals are available to the accounting software. The differences among them are important to those sites that want to give incentives for use of multitasking.

Memory integral #1 - At each change in memory size, memory integral #1 is incremented by the total CPU time used since the last memory change, times the memory size, as follows:

*MI #1: memory size * (total CPU time since last size change)*

Thus, a program that is connected to two CPUs for some period will pay twice the memory cost for that period. When using memory integral #1, a multitasking program incurs the same charges, no matter how many CPUs it gets. This is helpful if consistent billing is important to your site, but not as helpful if incentives for multitasking are important.

Memory integral #2 - The calculations for memory integral #2 are similar to those for #1, except that the increment is the sum of times when at least one CPU was connected, times the memory size, as follows:

*MI #2: memory size * (total time when program was connected*
to at least one CPU since last size change)

A multitasking program pays (in memory charges) only for the first CPU it receives; additional CPUs do not increase the memory charge. Using memory integral #2, a multitasking program can potentially decrease its memory charge by a factor equal to the number of CPUs in the machine. This is an incentive for using multitasking. However, because the amount of time a program is connected to a number of CPUs varies from run to run, memory integral #2 is not consistent. The maximum value for #2 is equal to #1 (if no connect time overlap occurs). Note that this also means that #1 is equal to #2 for single-tasked programs.

Memory integral #3 - Some sites with multi-CPU machines may wish to allow an individual program to use a proportionally large amount of memory only if it is also able to use more than one CPU. For instance, in a four-CPU machine, allowing one program to use 90% of memory may idle some CPUs if the program uses only one CPU.

Memory integral #3 allows the site to control this aspect of CPU use by adding an extra factor into the calculation for memory integral #2. The total memory available to user programs is divided by the number of CPUs to derive the value of “one CPU worth of memory.” The memory size of the program is then divided by the “CPU worth” factor to get the extra factor in memory integral #3, as follows (this extra factor cannot be less than 1):

*MI #3: memory size * (total time when program was connected*
*to at least one CPU since last size change) **
(memory size / "one CPU worth of memory")

Memory integral #3 provides an incentive for single-tasked programs to limit themselves to one CPU worth of memory. Multitasked programs will also pay more in memory charges for lots of memory, but they can reduce their memory charges by using multiple CPUs. However, memory integral #3 is as inconsistent as #2, and it can also affect the memory charges for single-tasked programs.

Note that the changes from #1 to #2 and #2 to #3 are, in a sense, opposite for multitasking programs. The changes from #1 to #2 reward multitasking programs by a factor of up to the number of CPUs. The changes from #2 to #3 penalize large-memory programs by up to the number of CPUs. Thus, if a multitasking program has used all memory (on a four-CPU machine), memory integrals #1 and #3 would be nearly equal, and the value of #2 would be approximately one-quarter the value of #1 or #3.

The accounting software is released with memory integral #2 as the default. The `MEMINT` variable in `/etc/config/acct_config` can be changed to match the site's needs.

2.1.12.2 Reducing Charges

Another incentive you can provide for the use of multitasking is to reduce the charges for CPU time for multitasking programs. This can be accomplished with weighting factors. The operating system kernel maintains counters of the length of time spent by a user program with one processor connected, two processors connected, and so on.

By default, the charges for a multitasking program would be calculated as follows:

```
sum = 0;
for (i=0; i < ncpu; i++)
                sum += ac_mutime[i] * (i+1);
```

This calculation assumes that all CPU time is charged equally. With the weighting factors, the site can specify, for instance, that a second CPU should be only 75% as expensive as the first CPU. Therefore, a program that gets two CPUs as it executes would have its CPU time charges reduced. Note that, because this charge depends on how much overlap a program gets, charges may vary from execution to execution. However, charges are never more than the full price for all CPUs.

The accounting software is released with all CPUs having a cost of 1. The `MUTIME_WEIGHT` *x* variables, defined in `/etc/config/acct_config`, can be changed to meet the site's needs.

Note that the user time reported by the `time`(1) command is adjusted so that there is no charge for wait-semaphore time. (This is in order to provide consistent CPU time charges.) The multitasking overlap times do not adjust for wait-semaphore times and, hence, may actually calculate to a greater CPU time than the sum of the user times. In this case, the CPU charge is limited to the sum of the user times.

2.1.13 Socket Accounting

Socket accounting tracks network usage from the perspective of sockets, wherein one process may use several sockets, and several processes may use the same socket.

The recorded accounting information tracks all of a socket's usage, but it can only be linked to the process that most recently closed the socket. This information can help you resolve network problems and/or monitor system network usage.

You can use the `csasocket`(8) command to summarize and process the socket data; `csaswitch`(8) can be used to check the status of, enable, and disable accounting methods, including socket accounting. See the `csasocket`(8) and `csaswitch`(8) man pages for more information.

2.1.14 Device Accounting

This section describes device accounting. On large computer systems with expensive peripheral devices, it may be useful to associate device usage with the user who initiated the I/O. Cray device accounting allows a system administrator to specify the accounting data that should be collected for device use. This system allows a site to individually label each mounted disk's partitions and so enables the site to charge each type of secondary storage at a different rate. For example, the amount of I/O on a high-speed storage device such as an SSD may be charged at a different rate than I/O on a disk device.

2.1.14.1 Categories of Devices

The following three categories of devices under the UNICOS operating system are important in device accounting:

- Character special devices, which are devices such as terminals, pseudo tty devices, and the HSX channel.
- Block devices, which are devices such as disks, BMR, and the SSD.

- Logical devices, for device accounting, which are the individual file systems. Such devices do not always correspond to a single device, but are treated as such by device accounting.

The device accounting system accounts for device I/O by device type. For a character device, device type is equivalent to its major number. For example, tty devices are major number 1 (in the default system), so they are accounted for as character device 1 (ios-tty). No accounting is performed for block devices, because block devices are used to create file systems; instead, they are treated as logical devices. Logical devices consist of one or more partitions of disk, SSD, and BMR storage. Each logical device is formatted by the `mkfs`(8) command, which provides it with a superblock. The `devacct`(8) program allows you to write an accounting device type into the superblock of each logical device.

2.1.14.2 Structures and Device Names

The `BLOCK_DEVICE` *x* and `CHAR_DEVICE` *x* parameters in `/etc/config/acct_config` contain the SBU values and names for device accounting. Refer to Section 2.1.10.1.9, page 47, for an explanation of configuring these parameters.

Device accounting uses arbitrary ASCII names for the user interface to accounting; internally, these names are mapped by the accounting library routines `typetonam` and `namtotype`. To be useful, these names should be meaningful to even the beginning user, because the `ja`(1) (job accounting) command displays these names when invoked with the `-d` option. The ASCII names are defined in the device name field of the `BLOCK_DEVICE` *x* and `CHAR_DEVICE` *x* parameters.

Logical device accounting names are displayed to the user by `ja` and the accounting programs, and are used by `devacct`(8) to determine the numeric values the kernel uses.

Logical and character device names should not match; in fact no two names should match, because the user cannot distinguish between them.

If names contain spaces (the shell field separator (`SHELL IFS`)), double quotes must be used around the device type names during command invocation.

Device names are used as output by `ja` and the accounting programs; therefore, keeping the names fairly short (less than 40 characters) will make them more readable.

System billing units (SBUs) have the following meanings:

SBU	Description
`Logical I/O Sbu`	The total number of system calls made to this type of device is multiplied by `Logical I/O Sbu` to determine the SBU cost. This value should be nonnegative.
`Characters Xfer Sbu`	The total number of characters transferred to this device type is multiplied by `Characters Xfer Sbu` to determine the SBU cost. This value should be nonnegative.

2.1.14.3 Configuration Changes

The system is released with the character devices configured to match the released configuration; any changes to `/usr/src/uts/c1/cf/devsw.c` should be reflected in the configuration file.

The block device configuration is released with a simple configuration. Several extensions are possible, although some may require altering the values of `MAXBDEVNO` and `MAXCDEVNO`, and rebuilding the system and accounting commands. First, if a site has a special temporary device that is restricted to a set of users, a special type might be placed on that device to allow the billing process to increase the cost of use, offsetting the lower rate of use. Second, SSD or BMR allocated to logical device cache may be reflected in the configuration.

2.1.14.4 System Header Files

The system header files discussed in this section are important in device accounting.

2.1.14.4.1 `param.h` Header File

The values `MAXBDEVNO` and `MAXCDEVNO` are contained in the `/usr/include/sys/param.h` file; they set the maximum size of the accounting structures in the user structure and the maximum size of the accounting data written. It is recommended that they not be increased beyond the current values unless necessary (although making `MAXCDEVNO` smaller and `MAXBDEVNO` larger by the same amounts is acceptable).

MAXBDEVNO is the maximum number of block (logical) device accounting types. This number can be changed from the current value of 10.

MAXCDEVNO is the maximum number of character device accounting types. This number can be changed from the current value of 35.

2.1.14.4.2 acct.h Header File

The /usr/include/sys/acct.h header file contains all the kernel structures for accounting and sets the following values related to device accounting:

Value	Description
NODEVACCT	The number of devio entries per accounting record. This value is the number of device accounting entries that fit into one accounting record.
ACCT_CHSP	A marker combined by an OR operation into the type field (acd_type) to indicate that the devio entry is for a character device.
_MAXDEVIOREC	The maximum number of device accounting records that can be written for any individual process.

2.1.14.5 Using Device Accounting (Devacct(8))

Use the devacct(8) command to label file systems with accounting types while they are mounted. If a file system does not contain a device type label, device accounting ignores it.

In order to enable device accounting, the system may be built to automatically enable specific device types. However, an easier solution is to insert lines into the system startup procedure (/etc/config/daemons) to enable device accounting when bringing the system to multiuser mode. The following example shows a line that can be added to the daemons file (/etc/config/daemons) to enable device accounting (remember the device type name is a single argument and so it may need to be enclosed in double quotation marks if it contains shell separators):

```
SYS1 devacct YES - /usr/lib/acct/devacct -b "device type name"
```

The devacct command with the -l option may be used to label file systems (file systems may be labeled only while mounted). The names of device types are defined in the `BLOCK_DEVICE`*x* and `CHAR_DEVICE`*x* variables located in `/etc/config/acct_config`. Some of the default names include spaces; such names must be enclosed in double quotation marks on the command line.

For example, to label the device `/dev/dsk/root` with the label `"dd49 with ldcache"`, the command would be as follows:

```
/usr/lib/acct/devacct -l "dd49 with ldcache" /dev/dsk/root
```

Device accounting for any device type may be turned on at any time by invoking the `devacct` command with the `-b` option. While device accounting is on, no records are written unless per-process accounting is enabled.

For example, to enable accounting for the devices labeled `"dd49 with ldcache"`, the command is as follows:

```
/usr/lib/acct/devacct -b "dd49 with ldcache"
```

You can turn on performance accounting using the following command:

```
/usr/lib/acct/devacct -b perf01
```

Device accounting for any device type may be turned off at any time by invoking the `devacct` command with the `-t` option. While accounting is disabled, those processes that have already accumulated data will report that data at termination if per-process accounting is enabled. For example, to disable accounting for the devices labeled `"dd49 with ldcache"`, the command is as follows:

```
/usr/lib/acct/devacct -t "dd49 with ldcache"
```

To determine the current label for a device, use the `devacct` command with the `-L` option. For example, to list the current label of `/dev/dsk/root`, you would execute the following command:

```
/usr/lib/acct/devacct -L /dev/dsk/root
```

2.1.14.5.1 Implications of Device Accounting

The system overhead for device accounting is fairly low. However, the amount of accounting data produced in the worst cases is more than double that produced by standard accounting. The more device accounting data kept, the more file system space that is required. If one device is accounted for, processes that use that device generate twice as much accounting data as a process that did not use the device or the same process without device accounting. However, for 1 to NODEVACCT device types, the maximum size of the accounting data does not increase, except that more processes may use one of the devices.

2.1.14.5.2 Tape Device Accounting

To enable or disable tape device accounting, use the *device type name* associated with the CHAR_DEVICE15 parameter in /etc/config/acct_config. By default, this device name is "bmx daemon".

The device name associated with CHAR_DEVICE11 (the default is "bmx tape") controls device accounting only for tape diagnostics.

To enable device accounting for the tapes, execute the following command:

```
/usr/lib/acct/devacct -b "bmx daemon"
```

2.1.15 Switching / and /usr File Systems

Occasionally, sites run on numerous / and /usr file systems and want to maintain the same accounting files throughout. The easiest way to accomplish this is to put /usr/adm or /usr/adm/acct on a separate file system and mount this file system along with each different system.

In addition, two other files, /etc/csainfo and /etc/wtmp, must be copied from the previously booted /. These files must be copied to the new root file system before it is brought up. Failure to correctly copy /etc/csainfo to the new / can cause csarun to abort abnormally. Incorrect connect time data is reported when /etc/wtmp is not copied.

2.1.16 Logging Information

The following sections describe log files found in the UNICOS operating system.

2.1.16.1 Boot Log

The boot log contains the date and time the system was booted. It is located in /etc/boot.log and can be accessed through normal file manipulations such as tail(1), cat(1), pg(1), and more(1). The /etc/rc (see brc(8)) script appends the record to the boot.log. The format is as follows:

date, uname -a
yy/*mm*/*dd* *hh*:*mm* *system* *node* *release* *version* *hardware*

Example:

```
93/05/10 08:07 sn1703c cool 8.0.0tk dev.6 CRAY Y-MP
```

See `date`(1) and `uname`(1) for further information. See also `who`(1), and the sample `wtmp` and `utmp` files in this chapter.

2.1.16.2 `cron` Log

The `cron` log contains the history of all actions taken by the `cron`(8) command. It is located in `/usr/lib/cron/log` and can be accessed by using normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). The format of this file is as follows:

```
CMD: command_executed username  process_id  job_type
     start_time username  process_id    job_type
     end_timerc=error return code
```

job_type can have one of the following values:

a	`at` job
b	Batch job
c	`cron` job

Example:

```
>  CMD: 645827040.a
>  user1 7191 a Tue Jun 19 15:24:00 1990
>  CMD: /usr/lib/sa/sa1 120 1
>  root 7192 c Tue Jun 19 15:24:00 1990
<  root 7192 c Tue Jun 19 15:24:00 1990
<  user1 7191 a Tue Jun 19 15:24:00 1990
>  CMD: 645827059.b
>  user1 7273 b Tue Jun 19 15:24:19 1990
<  user1 7273 b Tue Jun 19 15:24:20 1990 rc=1
```

2.1.16.3 Dump Log

The dump log contains the time and a reason for each dump. The system supplies the time and the user supplies the reason. By default, the dump is located in `/etc/dump.log` and can be accessed using the normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). When the system is changing out of single-user mode, `brc`(8) calls `coredd`(8) to copy a dump file to a file system. The location of the dump can be reconfigured by using the install tool. Note that the user may also change the location of the log file by using the -l option with the cpdmp command.

Example of /etc/dump.log:

```
cpdmp: 035120 blocks on dump device - waiting to be copied
04/26/93 07:27:09   coredd: Copying system dump into /core//04260727.
Unicos-E dump copied to=/core//04260727/dump
        dump taken: 04/26/93 at 07:18:51
        reason: PANIC: master.s: EEX interrupt in UNICOS kernel
```

2.1.16.4 New User Log

The new user log contains information on new users given logins on the system; this data includes who added the users, the times at which they were added, and information about their environment defaults and IDs. This log is located in /usr/adm/nu.log and can be accessed using normal file manipulations such as tail(1), cat(1), pg(1), and more(1). It is created by the nu(8) command. An example of the format follows:

```
user1:co:user login #1
user1:ui:10702:di:/j/user1:sh:/bin/csh:dr:/:pw:qQfHS6B8XYdzg
user1:gi:128,129,130,131,132
user1:ai:0
user1:dl:0:mx:0:mn:0:lk:0:tp:0
user1:dc:default:cm:default:pm:default
         added by adm1 on Wed Jul 20 08:43:20 1988
```

2.1.16.5 su Log

The su log records su(1) attempts for the current day. It is located in the /usr/adm/sulog file and can be accessed using normal file manipulations such as tail(1), cat(1), pg(1), and more(1). It is written by the su(1) command. The format of the log is as follows:

SU *MM/DD hh:mm flag tty olduser-newuser*

flag can have the following values:

+ su was successful.

- su was not successful.

olduser is the login name of the user executing `su`, and *newuser* is the name of the user the executing user is becoming. For example:

```
SU 06/19 15:13 + console operator-root SU 06/19 15:13 + ttyp025 \n
user1-root SU 06/19 15:14 + ttyp021 user2-adm SU 06/19 15:19 - ttyp026 \n
user3-root SU 06/19 15:19 - ttyp022 user4-root
```

2.1.16.6 `OLDsu` Log

The `OLDsu` log is a directory containing all files of daily `su`(1) attempts. It is located in `/usr/adm/OLDsu/*` and can be accessed using normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). The `/etc/rc` script moved the `/usr/adm/sulog` file to the `/usr/adm/OLDsu` directory at system startup. An example of the format follows:

```
$ ls -al OLDsu

-rw-rw-rw-  1 root    2864 Oct 31 19:02 Oct31
-rw-rw-rw-  1 root   20211 Sep 12 09:15 Sep01
-rw-rw-rw-  1 root     938 Sep 12 09:15 Sep02

$ cat Sep01

SU 09/01 16:29 + tty?? root-root
SU 09/01 16:30 + tty?? root-sys
SU 09/01 16:32 + tty?? root-sys
SU 09/01 16:32 + tty?? root-root
SU 09/01 16:34 + tty?? root-sys
SU 09/01 16:35 + tty?? root-root
SU 09/01 16:36 + tty?? root-sys
```

2.1.16.7 System Logs

The system logs are files into which the `syslogd`(8) command has logged messages. They are located in the `/usr/adm/syslog/*` directory. Note that these files are described by the configuration file `/etc/syslog.conf`. They can be accessed using normal file manipulations such as `tail`(1), `cat`(1), `page`(1), and `more`(1). They are written by the `/etc/syslogd` command; the `logger`(1B) command also makes entries in the system logs.

These logs consist of ASCII messages, which may include debug messages, kernel messages, and so on.

The following example is the configuration file for `/etc/syslogd` (these fields are described on the `syslogd`(8) and `syslog`(3) man pages):

```
$ cat /etc/syslog.conf

# USMID @(#)man/2302/02.accounting      92.2    02/05/96 13:26:44
#
#       This is a configuration file for /etc/syslogd
#
#*.debug                                /usr/adm/syslog/debug
#
mail.debug                              /usr/spool/mqueue/syslog
#
kern.debug                              /usr/adm/syslog/kern
#
daemon,auth.debug                       /usr/adm/syslog/auth
#
#*.err;kern.debug;auth.notice           /dev/console
#
*.err;kern.debug;daemon,auth.notice;    /usr/adm/syslog/daylog
#
#*.alert;kern.err;daemon.err            operator
*.alert                                 root
```

Note: The `/etc/syslogd.conf` file does not work if spaces are in it; only tabs can be used to separate items in this file.

The following example shows a listing of the files in the `/usr/adm/syslog` directory:

```
$ ls -l /usr/adm/syslog
total 10
-rw-r--r--   1 root     root         168 Jun 19 15:35 auth
-rw-r--r--   1 root     root        5164 Jun 19 15:45 daylog
-rw-r--r--   1 root     root        4107 Jun 19 15:45 kern
drwxr-xr-x   2 root     root       16864 Jun 19 15:09 oldlogs
```

2.1.16.8 Error Log

The error log is a file containing error records from the operating system. The default error file is `/usr/adm/errfile`. There are two facilities available for generating reports from the data collected by the error-logging mechanism. The first is `errpt`(8), which processes error reports from the data, and the second is `olhpa`, a hardware performance analyzer that reports the hardware errors and statuses recorded in the system error log.

Note: The `olhpa` facility is only available on IOS-E based systems. It is not available on GigaRing based systems.

The `/etc/errdemon` command (see `errdemon`(8)) reads `/dev/error` and places the error records from the operating system into either the specified file, or `errfile`, by default. The `/etc/rc` (see `brc`(8)) script starts `/etc/errdemon`, and `/etc/mverr` is used to start a new `errfile`.

The following example shows sample `errpt` output:

```
Tue Jun 7 12:01:49 1988
       Error reported from IOS 0 for device S49-0-21
       Major:0  Minor:6         Block:140868   status: Recovered
       Iop:0  Channel:21        Unit:0
       Cylinder:1156  Head:5    Sector:0
       Function:Read  Requested:344064 bytes   Received:344064 bytes

       IOS 0 ERROR LOGGING ENABLED
```

See `errpt`(8) for further information. See the *Online Maintenance Tools Guide for Cray PVP Systems*, publication SD–1012, or the `olhpa`(8) man page for information concerning `olhpa`. [2]

2.1.16.9 Multilevel Security (MLS) Log

The multilevel security (MLS) log is a file containing security-relevant event loggings. The security log, `/usr/adm/sl/slogfile`, can be analyzed by using the `reduce` command. `reduce` extracts, formats, and outputs entries from UNICOS security event files. The security event logging daemon, `slogdemon`(8), collects security-relevant records from the operating system by reading the character special pseudo device `/dev/slog`. For more information regarding the format of the security log and on the UNICOS MLS feature, see the `reduce`(8) man page and *General UNICOS System Administration*, publication SG–2301.

[2] CRAY RESEARCH PRIVATE. This document contains information private to Cray Research, Inc. It can be distributed to non-CRI personnel only with approval of the appropriate Cray manager.

2.1.16.10 System Activity Log

The system activity report facility provides commands for generating various system activity reports. Two reporting capabilities exist (one automatic and one user-driven); however, the actual reports are created by the `sar`(8) program in either case. The system activity log is located in `/usr/adm/sa/sa`*DD* and can be accessed with `sar`.

Warning: The log files located in `/usr/adm/sa/sa`*DD* on a Cray ML-Safe configuration of the UNICOS system are considered to be covert channels. You may want to consider restricting access to these files to the `adm` group.

With this command, you can generate system activity reports in real time and save system activities in a file for later use. The `sa1`, `sa2`, and `sadc` commands (see `sar`(8)) generate system activity data on a routine basis, with `sa2` calling `sar` to generate the report.

UNICOS counters are incremented as various system actions occur. These counters provide system-wide measurements. `sadc` accesses `/dev/kmem` to read these system activity counters.

Refer to the `sar`(8) man page for more information on the format of the system activity log.

2.1.16.11 Message Log

The message log contains messages and replies to the operator. It is located in `/usr/spool/msg/msglog.log` and can be accessed using normal file manipulations, such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). All messages and replies to and from the operator console are put into this file by the console. An example of the file format follows:

```
Apr  1 07:11:06  Message daemon stopped
Apr  1 09:36:54  Message daemon started
Apr  1 08:09:49  Message   1: TM122 - mount tape WK1102(sl) on a CART
 device for user1 50,  () or reply cancel / device name
```

Warning: The `msglog.log` file is considered a covert channel on a Cray ML-Safe configuration of the UNICOS system. You may want to consider restricting access to this file to the `adm` group.

2.1.16.12 Accounting Logs

The accounting logs are files containing various accounting information, as follows:

Log	Description
csainfo	A file containing boot times. It can be accessed with the od(1) command (the -d option will give the seconds). Each time the system is booted, the boot time is written to /etc/csainfo by the /etc/csaboots (see csaboots(8)) command. csaboots is invoked by /etc/rc (see brc(8)). See also the description of the boot log in Section 2.1.16.1, page 70.
utmp	A file containing active system and terminal connection information. This log is used by write(1), who(1), wall(8), and mail(1) in getting user information. It is located in /etc/utmp and can be accessed using the who(1) and last(1B) commands. It is written to by init(8), date(1), login(1), and getty(8). For information on the format of utmp, see utmp(5). **Warning:** On a Cray ML-Safe configuration of the UNICOS system, utmp and wtmp are considered to be covert channels. You may want to consider restricting access to these files to the adm group.
wtmp	A file containing a system and terminal connection history record. This log includes usage statistics for each terminal, date change, time stamp, boot records, reboots, shutdowns, and state changes. wtmp must exist; programs that access it do not create it (the /etc/rc script creates /etc/wtmp by default). Records are in the form of utmp(5); acctcon(8) and csaline(8) convert /etc/wtmp into session and charging records. This data is merged into the system accounting reports. wtmp can also be accessed using the who(1) and last(1) commands. wtmp is written by init(8), date(1), login(1), and getty(8). For information on the format of wtmp, see utmp(5).
pacct	Files containing per-process accounting data; these are located in /usr/adm/acct/day/pacct* and can be accessed using the acctcom(1) command. Note that these files are read by system accounting programs, and the information appears in the accounting reports. pacct is written by the kernel, and its format is described in /usr/include/sys/acct.h.

Warning: On systems running a Cray ML-Safe configuration of the UNICOS system, access to `pacct*` files should be restricted to the `adm` group.

The following data files are accessed by system accounting programs, and their information appears in the accounting reports:

Log	Description
`disktacct`	A file containing disk accounting data, located in `/usr/adm/acct/nite/disktacct`. The `/usr/lib/acct/dodisk` (see `dodisk`(8)) command writes this file.
`fee`	A file containing user fees for accounting data, located in `/usr/adm/acct/day/fee`. This file is written by `/usr/lib/acct/chargefee` (see `chargefee`(8)).
`nqacct`	A file containing NQS daemon accounting data, located in `/usr/adm/acct/day/nqacct*`. This file is written by `/usr/lib/nqs/nqsdaemon`. See `/usr/include/acct/dacct.h` for the format.
`soacct`	A file containing socket accounting data, located in `/usr/adm/acct/day/soacct*`. This file is written by the kernel. See `/usr/include/sys/acct.h` for the format.
`tpacct`	A file containing tape daemon accounting data, located in `/usr/adm/acct/day/tpacct*`. This file is written by `/usr/lib/tp/tpdaemon` (see `tpdaemon`(8)). See `/usr/include/acct/dacct.h` for the format.

2.1.16.13 NQS Log

The NQS log contains NQS information. Its default location is the ASCII file `/usr/spool/nqs/log` (you can change the location of the log file with the `qmgr set log_file` command; to see where the current log file resides, use the `qmgr show parameters` command). Access to `/usr/spool/nqs` is restricted; however, if you have the correct permissions, you can access the NQS log file using normal file manipulations, such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). This log is created by the NQS log daemon.

Warning: On systems running a Cray ML-Safe configuration of the UNICOS system, access to the NQS log should be restricted to the `adm` group.

An example of the log file's format is as follows:

```
05/13 08:00:00 I getpkt(): Received packet from local process: <89775>.
05/13 08:00:00 I getpkt(): Client process real UID=<900>.
05/13 08:00:00 I getpkt(): Packet type=<PKT_QUEREQVLPQ(30)>.
05/13 08:00:00 I getpkt(): Packet contents are as follows:
05/13 08:00:00 I getpkt(): Pkt_str[1] = <batnam1>.
05/13 08:00:00 I getpkt(): Pkt_int[1] = <40>.
05/13 08:00:00 I getpkt(): Pkt_int[2] = <119>.
05/13 08:00:00 T nqs_quereq(): Request <40.cool>: Attempting to read request.
05/13 08:00:00 T nqs_quereq(): Request <40.cool>: Request was read.
```

2.2 Standard UNIX Accounting

The standard UNIX accounting feature of the UNICOS system provides methods for collecting resource use data per process, recording connect sessions, monitoring disk usage, and charging fees to specific logins. A set of C language programs and shell procedures is provided to reduce this accounting data into summary files and reports. This section describes the structure, implementation, and management of the accounting system; it also describes the reports generated and the meaning of the columnar data.

The following list is a synopsis of the standard accounting actions:

- At process termination, the UNICOS system kernel writes one record per process in `/usr/adm/acct/day/pacct`.
- The `login`(1) and `init`(8) programs record connect sessions by writing records into `/etc/wtmp`. Date changes, reboots, and shutdowns are also recorded in this file. The `wtmp` file is described in `utmp`(5).
- The programs `acctdusg`(8) and `diskusg`(8) break down disk usage by login.
- Fees can be charged to specific logins with the `chargefee`(8) shell procedure.
- Each day the `cron`(8) shell procedure executes the `runacct`(8) shell procedure, which reduces accounting data and produces summary files and reports.

- The monacct(8) procedure can be executed on a monthly or fiscal period basis. It saves and restarts summary files, generates a report, and cleans up the sum directory. These saved summary files could be used to charge users for UNICOS system usage.

2.2.1 Files and Directories

The /usr/lib/acct directory contains all the C language programs and shell procedures necessary for running the accounting system. The adm login is used by the accounting system and has the login directory structure shown in Figure 3.

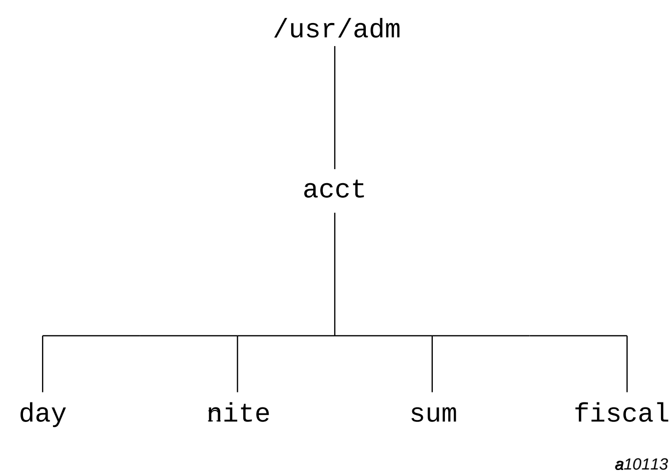

Figure 3. Directory structure of the adm login

The /usr/adm/acct/day directory contains the active data collection files. The nite directory contains files that are reused daily by the runacct(8) procedure. The sum directory contains the cumulative summary files updated by runacct(8). The fiscal directory contains periodic summary files created by monacct(8).

In addition, configurable parameters are located in /etc/config/acct_config. You should modify these parameters to meet your site's needs.

2.2.2 Daily Operation

When the UNICOS system is switched into multiuser mode, /usr/lib/acct/startup is executed, as follows:

1. The acctwtmp(8) program adds a boot record to /etc/wtmp. This record is signified by use of the system name as the login name in the wtmp record.

2. Process accounting is started with turnacct(8). The turnacct command specified with the on argument, as follows, executes the accton(8) program with the /usr/adm/acct/day/pacct argument:

   ```
   /usr/lib/acct/turnacct on
   ```

3. The remove shell procedure is executed to clean up the saved pacct and wtmp files left in the sum directory by runacct(8).

The ckpacct(8) procedure is run with cron(8) every hour to check if there is enough space on the current file system (the default is /usr). If there are fewer than MIN_BLKS free blocks (by default 500), accounting is stopped, and the system administrator is notified about the action. MIN_BLKS is defined in the configuration file /etc/config/acct_config. The ACCT_FS variable in /etc/config/acct_config must be set to the file system containing /usr/adm/acct. If the free space increases to 500 free blocks at a later time, accounting is restarted, again with notification to the system administrator.

You can use the chargefee(8) program to bill users. It adds to /usr/adm/acct/day/fee records that are picked up and processed by the next execution of runacct and merged into the total accounting records.

The runacct command is executed with cron each night. It processes the following active accounting files:

```
/usr/adm/acct/day/pacct
/etc/wtmp
/usr/adm/acct/day/fee
/usr/adm/acct/nite/disktacct
```

It produces command summaries and usage summaries by login. When the system is shut down with shutdown(8), the shutacct(8) shell script is executed. It writes a shutdown reason record into /etc/wtmp (see utmp(5)) and turns process accounting off.

2.2.3 Setting up the Accounting System

This section explains how to automate the operation of the accounting system. It also contains information on converting UNICOS 8.0, 8.3, 9.0, 9.1, 9.2, and 9.3 standard UNIX accounting files to UNICOS 10.0 CSA format.

To automate the operation of the accounting system, complete the following steps:

1. Modify any necessary parameters in the file `/etc/config/acct_config`, which contains configurable parameters for the accounting system. Ensure that the parameters, such as `MEMINT`, reflect the needs of your site. You can specify an alternate configuration file when running any of the accounting commands. See Section 2.1.10.8, page 55, for more information.

2. If you maintain startup options with the Installation Configuration Menu System (ICMS), configure `RC_ACCT` to have a value of `YES`. Otherwise, edit the `/etc/config/rcoptions` file to set `RC_ACCT` to a `YES` value.

3. Add an entry similar to the following to `/usr/spool/cron/crontabs/root` so that `cron` automatically runs `dodisk`:

   ```
   0 2 * * 4 /usr/lib/acct/dodisk
   ```

   `dodisk` must be executed by `root`, because no other user has the correct permissions to read `/dev/dsk/*`.

4. For most installations, you should make entries similar to the following in `/usr/spool/cron/crontabs/adm` so that `cron` will run the daily accounting automatically:

   ```
   0 4 * * 1-6 /usr/lib/acct/runacct 2>/usr/adm/acct/nite/fd2log
   50 * * * * /usr/lib/acct/ckpacct
   ```

   The `runacct`(8) command should be run at a time when the `dodisk`(8) routine has had sufficient time to complete. If `dodisk` has not completed before `runacct` executes, disk information may be missing.

5. To facilitate monthly merging of accounting data, make an entry similar to the following in `/usr/spool/cron/crontabs/adm`:

   ```
   15 5 1 * * /usr/lib/acct/monacct
   ```

   This entry allows the `monacct`(8) procedure to clean up all daily reports and daily total accounting files and to deposit one monthly total report and one monthly total accounting file in the `fiscal` directory. It takes advantage of the default action of monacct, which uses the current month's date as the suffix for the file names. The entry is executed when the runacct(8) procedure has sufficient time to complete. This results in the creation of monthly accounting files on the first day of each month containing the entire previous month's data.

6. Set the PATH shell variable in /usr/adm/.profile to the following:

   ```
   PATH=/usr/lib/acct:/bin:/usr/bin
   ```

2.2.3.1 Setting up a User Exit

Daily accounting provides one user exit, /usr/lib/acct/run.user, that you can call from the runacct command. This user exit allows you to tailor the runacct procedure to your site's needs by creating a shell script to perform any additional processing during the daily run of accounting. You do not have to modify the runacct script.

While executing, runacct checks in the USEREXIT state for a shell script named /usr/lib/acct/run.user. If the script exists, it is executed via the shell . (dot) command. If the script does not exist, the user exit is ignored. The . (dot) command will not execute a compiled program, but the user exit script can. runacct variables are available, without being exported, to the user exit script. runacct checks the return status from the user exit and, if it is nonzero, the execution of csarun is terminated.

2.2.3.2 Converting Standard UNIX Accounting to CSA Accounting

If your site decides to run CSA instead of standard UNIX accounting, you should wait until the start of an accounting period before implementing CSA. (An accounting period usually begins on the first day of a month.) Before switching to CSA, use the standard UNIX accounting package to process the previous month's accounting data.

Follow these steps to convert from standard UNIX accounting to CSA:

1. Run the current version of UNICOS standard UNIX accounting programs until the first day of the next month. Use the runacct(8) command to process the daily accounting data.

2. On the first day of the month, use the monacct(8) command to generate an accounting report for the previous month.

3. On the first day of the month, switch from running the standard UNIX accounting package to CSA.

4. (Optional step) The daily `tacct` files must be converted to `cacct` format if you later want to summarize this data by using `csaperiod`(8). The conversion should be done by using the `csaconvert`(8) command. Refer to the `csaconvert`(8) man page and the UNICOS Installation Guide, publication SG-2112, for more information on conversion.

For details on how to set up CSA, see Section 2.1.4, page 11.

2.2.4 the `runacct` Command

The `runacct`(8) command is the main daily accounting shell procedure. It processes connect, fee, disk, and process accounting files and prepares daily and cumulative summary files for use by `prdaily`(8) or for billing purposes. `runacct` also contains one user exit point that allows you to tailor the daily accounting run to your site's needs. It is normally initiated with the `cron`(8) command during nonprime hours.

The following files in `/usr/adm/acct`, which are produced by `runacct`, are of particular interest:

File	Description
`nite/daytacct`	The total accounting file for the previous day in `tacct.h` format.
`nite/lineuse`	Produced by `acctcon`(8). It reads the `wtmp` file and produces usage statistics for each terminal line on the system. This report is not especially useful, but is a carryover from traditional UNIX systems.
`sum/cms`	The accumulation of each day's command summaries. It is restarted by the execution of `monacct`(8). The ASCII version of this file is `nite/cms`.
`sum/daycms`	Produced by the `acctcms`(8) program. It contains the daily command summary. The ASCII version of this file is `nite/daycms`.
`sum/loginlog`	Produced by the `lastlogin`(8) shell procedure. This file contains a record of the last time each login was used.
`sum/rprt`*MMDD*	Each execution of `runacct`(8) saves a copy of the daily report as produced by `prdaily`(8).

sum/tacct	The accumulation of each day's nite/daytacct. It can be used for billing purposes and is restarted each month or fiscal period by the monacct(8) procedure.

The runacct command does not damage files in the event of errors. It contains a series of protection mechanisms that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that runacct can be restarted with minimal intervention.

The runacct command records its progress by writing descriptive messages into the file active. (Files used by runacct are assumed to be in the /usr/adm/acct/nite directory unless otherwise noted.) All diagnostic output during the execution of runacct is written into fd2log. runacct terminates execution if the lock and lock1 files exist when it is invoked. The lastdate file contains the month and day runacct was last invoked and is used to prevent more than one execution per day. If runacct detects an error, it writes a message to /dev/console, sends mail to root and adm, removes locks, saves diagnostic files, and terminates execution.

Processing is broken down into separate reentrant states so that runacct can be restarted. The last state completed is recorded in a file. As each state completes, statefile is updated to reflect the next state. When runacct reaches the CLEANUP state, it removes the locks and terminates. States are executed as follows:

State	Description
SETUP	The turnacct(8) command switch is executed. The process accounting files, /usr/adm/acct/day/pacct*, are moved to /usr/adm/acct/day/Spacct*.*MMDD*. The /etc/wtmp file is moved to /usr/adm/acct/nite/wtmp.*MMDD*, with the current date added at the end.
WTMPFIX	The wtmpfix (see fwtmp(8)) program checks the wtmp file in the nite directory for accuracy. Some date changes cause acctcon1 (see acctcon(8)) to fail, so wtmpfix attempts to adjust the time stamps in the wtmp file if a date change record appears.

If `wtmpfix` is unable to fix the `wtmp` file, the `wtmp` file must be manually repaired. Refer to Section 2.2.5.1, page 89.

`CONNECT1`
: Connect session records are written to `ctmp` in the form of `ctmp.h`. The `lineuse` file and the `reboots` file are created, showing all of the boot records found in the `wtmp` file.

`CONNECT2`
: The `ctmp` file is converted to `ctacct.`*MMDD*, which is comprised of connect accounting records. (Accounting records are in `tacct.h` format.)

`PROCESS`
: The `acctprc1` and `acctprc2` programs (see `acctprc`(8)) are used to convert the process accounting files, `/usr/adm/acct/day/Spacct*.`*MMDD*, into total accounting records in `ptacct*.`*MMDD*. The `Spacct` and `ptacct` files are correlated by number so that, if `runacct` fails, the `Spacct` files are not reprocessed. One precaution should be noted: when restarting `runacct` in this state, remove the last `ptacct` file, because it will not be complete.

`MERGE`
: The process accounting records are merged with the connect accounting records, the output going to `daytacct`.

`FEES`
: Any ASCII `tacct` records from the file `fee` are merged into `daytacct`.

`DISK`
: On the day after the `dodisk`(8) procedure runs, `disktacct` is merged with `daytacct`.

`MERGETACCT`
: The `daytacct` file is merged with `sum/tacct`, the cumulative total accounting file. Each day, `daytacct` is saved in `sum/tacct.`*MMDD* so that `sum/tacct` can be recreated if it becomes corrupted or lost.

`CMS`
: Today's command summary is merged with the cumulative command summary file `sum/cms`. ASCII and internal format command summary files are produced.

`USEREXIT`	User exit point. If a script named `/usr/lib/acct/run.user` exists, it will be executed via the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. `runacct` variables are available, without being exported, to the user exit script. You might use this user exit to run local accounting programs.
`CLEANUP`	Clean up temporary files, run `prdaily`(8) and save its output in `sum/rprt`*MMDD*, remove the locks, and then exit.

2.2.4.1 Failure Recovery for `runacct`

The `runacct`(8) program can fail for a variety of reasons; the most common reasons are a system crash, a lack of space in the file system containing `/usr/adm/acct`, and a corrupted `wtmp` file. If the `active` *MMDD* file exists, check it first for error messages. If the `active` file and lock files exist, check `fd2log` for messages.

The following are error messages produced by `runacct` and the recommended recovery actions:

`ERROR: locks found, run aborted`

The `lock` and `lock1` files were found. These files must be removed before `runacct` can restart.

`ERROR: acctg already run for date:  check /usr/adm/acct/nite/lastdate`

The date in `lastdate` and today's date are the same. Remove `lastdate`.

`ERROR: turnacct switch returned rc=?`

Check the integrity of `turnacct`(8) and `accton`(8). The `accton` program must be owned by `root`, and the `setuid` bit must be set.

`ERROR: Spacct?.MMDD already exists`

File setups have probably already been run. Check status of files, then run setups manually.

`ERROR: /usr/adm/acct/nite/wtmp.MMDD already exists, run setup manually.`

This message is self-explanatory.

`ERROR: wtmpfix errors see /usr/adm/acct/nite/wtmperror`

The `wtmpfix`(8) program detected a corrupted `wtmp` file. Use `fwtmp`(8) to correct the corrupted file.

`ERROR: Connect acctg failed:  check /usr/adm/acct/nite/log`

The `acctcon1`(8) program encountered a bad `wtmp` file. Use `fwtmp` to correct the bad file.

`ERROR: Invalid state, check /usr/adm/acct/nite/active`

The `statefile` file is probably corrupted. Check `statefile` and read the `active` file before restarting.

2.2.4.2 Restarting `runacct`

If you invoke `runacct`(8) without arguments, the invocation is assumed to be the first one of the day. The *MMDD* argument is necessary if `runacct` is being restarted. It specifies the month and day for which `runacct` is to rerun the accounting. The entry point for processing is based on the contents of `statefile`. To override `statefile`, include the desired state on the command line. For each case, see the appropriate example, as follows:

To start `runacct`:

```
nohup runacct 2> /usr/adm/acct/nite/fd2log&
```

To restart `runacct` using the state specified in `statefile`:

```
nohup runacct 0601 2> /usr/adm/acct/nite/fd2log&
```

To restart `runacct` at a specific state, overriding `statefile`:

```
nohup runacct 0601 WTMPFIX 2> /usr/adm/acct/nite/fd2log&
```

2.2.5 Fixing Corrupted Files

When file corruption occurs, some files can be ignored or restored from the file save backup. Certain files, however, must be fixed in order to maintain the integrity of the accounting system.

2.2.5.1 Fixing wtmp Errors

The wtmp files generally cause the highest number of errors in the day-to-day operation of the accounting system. When the date is changed, and the UNICOS system is in multiuser mode, a set of date change records is written into the /etc/wtmp file. The wtmpfix program (see fwtmp(8)) is designed to adjust the time stamps in the wtmp records when a date change is encountered.

Some combinations of date changes and reboots, however, slip through wtmpfix and cause acctcon1 (see acctcon(8)) to fail.

The following example shows how to repair a wtmp file:

```
$ cd /usr/adm/acct/nite
$ /usr/lib/acct/fwtmp < wtmp.MMDD > xwtmp
$ ed xwtmp
  (Delete corrupted records)
$ /usr/lib/acct/fwtmp -ic < xwtmp > wtmp.MMDD
  (Restart runacct at the WTMPFIX state)
```

If the wtmp file is beyond repair, create a null wtmp file, which prevents any charging of connect time. The acctprc1 program (see acctprc(8)) cannot determine which login owned a particular process, but the process is charged to the first login in the /etc/udb file for that user ID.

2.2.5.2 Fixing tacct Errors

If your installation is using the accounting system to charge users for system resources, the integrity of sum/tacct is quite important. Occasionally, tacct records appear with negative numbers, duplicate user IDs, or a user ID of 65535. First, check the sum/tacctprev file with prtacct(8). If it looks correct, the latest sum/tacct. *MMDD* should be corrected; sum/tacct must then be recreated. A correctional procedure is as follows:

```
$ cd /usr/adm/acct/sum
$ /usr/lib/acct/acctmerg -v tacct.MMDD xtacct
$ ed xtacct
  (Remove the bad records, write duplicate user ID records to another file)
$ /usr/lib/acct/acctmerg -i xtacct tacct.MMDD
$ /usr/lib/acct/acctmerg tacctprev tacct.MMDD tacct
```

The monacct(8) procedure removes all tacct. *MMDD* files; therefore, you can recreate sum/tacct by merging these files.

2.2.6 Updating Holidays

The /usr/lib/acct/holidays file contains the prime/nonprime time table for the accounting system. You should edit the table to reflect your site's holiday schedule for the year. By default, the holidays file is located in the /usr/lib/acct directory. You can change the location of this file by modifying the HOLIDAY_FILE variable in /etc/config/acct_config. If necessary, you should modify the NUM_HOLIDAYS variable (also located in acct_config), which sets the upper limit on the number of holidays that can be defined in HOLIDAY_FILE.

The format is composed of three types of entries:

1. Comment lines: These lines may appear anywhere in the file as long as the first character in the line is an asterisk.

2. Year and time designation line: This line should be the first data line (noncomment line) in the file and must appear only once. The line consists of three fields of 4 digits each (leading white space is ignored). For example, to specify the year as 1982, prime time at 9:00 A.M., and nonprime time at 4:30 P.M., the following entry would be appropriate:

   ```
   1982 0900 1630
   ```

   As a special condition for the time field, the time 2400 is automatically converted to 0000.

3. Company holidays lines: These entries follow the year designation line and have the following general format:

   day-of-year Month Day Description of Holiday

   The day-of-year field is a number in the range of 1 through 366, indicating the day for a given holiday (leading white space is ignored). The other three fields are commentary and are not currently used by other programs.

2.2.7 Reports

The runacct(8) program generates five basic reports upon each invocation. These reports cover the areas of connect accounting, usage by user on a daily basis, command usage reported by daily totals, command usage reported by monthly totals, and last login time by user. The diskusg command can be configured at your site; see Section 2.1.10.9, page 55, for a description of how to customize a report for your site.

The following sections describe the reports and interpretation of their tabulated data.

2.2.7.1 Daily Report

In the first part of the report, the from/to banner alerts you to the time period being reported. The specified times are the time the last accounting report was generated until the time the current accounting report was generated. This banner is followed by a log of system reboots, shutdowns, power failure recoveries, and any other record dumped into the `/etc/wtmp` file by the `acctwtmp`(8) program.

The second part of the report is a breakdown of line usage. The `TOTAL DURATION` value is the difference between the time stamps of the first and the last record found in the `wtmp` file. The columns are as follows:

Column	Description
`LINE`	The terminal line or access port
`MINUTES`	The total number of minutes the line was in use during the accounting period
`PERCENT`	The total number of `MINUTES` the line was in use, divided into the `TOTAL DURATION`
`#SESS`	The number of times this port was accessed for a `login`(1) session

2.2.7.2 Daily Usage Report

The daily usage report gives a breakdown of system resource usage by user. Its data consists of the following:

Heading	Description
`ACCOUNT NAME`	If the UNICOS user-information database is enabled, this field contains the account name; otherwise, it contains `default`.
`UID`	User ID.
`LOGIN NAME`	Login name of the user; there can be more than one login name for a single user ID (although this is not recommended); this identifies the user.
`CPU SECS`	The amount of time in seconds the user's process used the CPU.

KCORE-MINS	A cumulative measure of the amount of memory a process used while running. The amount shown reflects kiloword segments multiplied by minutes used.
CONNECT (MINS)	The real time used. Real time is the amount of time that a user was logged in to the system. If this time is rather high, and column # OF PROCS is low, this person probably logs in first thing in the morning and rarely uses the terminal the rest of the day. This type of user can be a system resource problem. If this user is logged in and is not using the system at all, he or she may be using a line to the system that someone else needs.
DISK BLOCKS	Output from the disk accounting programs after that output has been merged into the total accounting record (tacct.h). Disk accounting is accomplished by the acctdusg(8) program.
# OF PROCS	The number of processes invoked by the user. Large numbers indicate an uncontrolled user shell procedure.
# OF JOBS	Number of times the user logged in to the system (interactive or batch).
# DISK SAMPLES	Number of times disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.
FEE	The total accumulation of billing units charged against the user by the chargefee(8) shell procedure. The chargefee procedure is used to levy charges against a user for special services (such as file restores) performed. This field is often unused.
SBU	A site-specific system billing unit (SBU); default is 0. You can modify the SBU calculation for your

site by editing the source and recompiling the accounting software (see Section 2.1.10.1, page 40).

2.2.7.3 Daily Command and Monthly Total Command Summaries

The daily command and monthly total command summaries are virtually the same, except that the daily command summary reports only on the current accounting period, while the monthly total command summary reports on the time from the start of the fiscal period to the current date. That is, the monthly report reflects the data accumulated since the last invocation of the `monacct`(8) procedure.

The data included in these reports tells you which commands are used most often. Based on this information, you can identify areas of the system using a majority of system resources.

These two reports are sorted by `TOTAL CPU-MIN`. The following categories are used:

Heading	Description
`COMMAND NAME`	The name of the command. All shell procedures are under the name `sh`, because only object modules are reported by the process accounting system. The `acctcom`(1) program is a good tool to use for identifying a user who executed a suspiciously named command and also for determining whether super-user privileges were used.
`NUMBER CMDS`	The total number of invocations of this particular command.
`TOTAL KCOREMIN`	The total cumulative measurement of the number of kiloword segments of memory used by a process per run-time minute.
`TOTAL CPU-MIN`	The total processing time this program has accumulated.
`TOTAL REAL-MIN`	The total real-time (wall-clock) minutes this program has accumulated.
`MEAN SIZE-K`	The mean of the `TOTAL KCOREMIN` over the number of invocations reflected by `NUMBER CMDS`.

MEAN CPU-MIN	The mean derived between the NUMBER CMDS and TOTAL CPU-MIN.
HOG FACTOR	A relative measurement of the ratio of system availability to system usage. It is computed by the following formula: (total CPU time) / (elapsed time) This gives a relative measure of the total available CPU time consumed by the process during its execution.
K-CHARS TRNSFD	The total number of characters moved by the read(2) and write(2) system calls.
I/O BUFS RD/WR	The total number of physical reads and writes that a process performed.

2.2.7.4 Last Login Report

The last login report provides the date on which a particular login was last used. You can use this report as a source of likely candidates to be moved to the archives, or, of unused logins and login directories to be deleted.

2.2.8 Accounting Files

This section lists files relevant to the accounting system in the /usr/adm/acct/day, /usr/adm/acct/nite, /usr/adm/acct/sum, and /usr/adm/acct/fiscal directories.

Files in the /usr/adm/acct/day directory are as follows:

File	Description
dtmp	Output from the acctdusg(8) program.
fee	Output from the chargefee(8) program (ASCII tacct records).
pacct	Active process-accounting file.
pacct*	Process-accounting files switched using turnacct(8).

`Spacct*.`*MMDD*	Process-accounting files for *MMDD* during execution of `runacct`(8).

Files in the `/usr/adm/acct/nite` directory are as follows:

File	Description
`active`	Used by `runacct` to record progress and print warning and error messages. The `active`*MMDD* file is the same as `active` after `runacct` detects an error.
`cms`	ASCII total command summary used by `prdaily`(8).
`ctacct.`*MMDD*	Connect accounting records in `tacct.h` format.
`ctmp`	Output of `acctcon1` program (see `acctcon`(8)); connect session records in `ctmp.h` format.
`daycms`	ASCII daily command summary used by `prdaily`.
`daytacct`	Total accounting records for one day in `tacct.h` format.
`disktacct`	Disk accounting records in `tacct.h` format; created by `dodisk`(8) procedure.
`fd2log`	Diagnostic output during execution of `runacct`.
`lastdate`	Last day `runacct` executed in *date* `+%` *m%* `d` format.
`lineuse`	The tty line usage report used by `prdaily`.
`lock lock1`	Used to control serial use of `runacct`.
`log`	Diagnostic output from `acctcon1`.
`log` *MMDD*	Same as log after `runacct` detects an error.
`reboots`	The beginning and ending dates from `wtmp`, and a listing of reboots.
`statefile`	A record of the current state during execution of `runacct`.
`tmpwtmp`	The `wtmp` file corrected by `wtmpfix` (see `fwtmp`(8)).
`wtmperror`	Place for `wtmpfix` error messages.

`wtmperror`*MMDD*	Same as `wtmperror` after `runacct` detects an error.
`wtmp.`*MMDD*	Previous day's `wtmp` file.

Files in the `/usr/adm/acct/sum` directory are as follows:

File	Description
`cms`	Total command summary file for current fiscal year in internal summary format.
`cmsprev`	Command summary file without latest update.
`daycms`	Command summary file for yesterday in internal summary format.
`loginlog`	Login record file created by `lastlogin`(8).
`pacct.`*MMDD*	Concatenated version of all `pacct` files for *MMDD*; removed after reboot by `remove`(8) procedure.
`rprt`*MMDD*	Saved output of `prdaily`(8) program.
`tacct`	Cumulative total accounting file for current fiscal period.
`tacctprev`	Same as `tacct` without latest update.
`tacct`*MMDD*	Total accounting file for *MMDD*.
`wtmp.`*MMDD*	Saved copy of `wtmp` file for *MMDD*, removed after reboot by `remove`(8) procedure.

Files in the `/usr/adm/acct/fiscal` directory are as follows:

File	Description
`cms`	Total command summary file for the fiscal period in internal summary format.
`fiscrpt`	Report similar to `prdaily`(8) for fiscal period.

`tacct*`	Total accounting file for fiscal period.

2.3 Front-end Formatting

Front-end formatting facilities let you customize accounting reports and generate output files that can be processed on a front-end computer system. The front-end formatting process consists of two main parts:

- Consolidating the accounting data you have collected to select useful information and to reduce it to a manageable amount of data for the front-end system.
- Formatting the consolidated data into meaningful reports and files for further processing on the front-end system.

Accounting data is consolidated using identifier keys. These keys may include user ID (`uid`), account ID (`acid`), job ID (`jid`), group ID (`gid`), and job class (`jclass`). The front-end formatters then can send the consolidated data output to either an ASCII report or to a binary file.

Note: Disk usage information is not available on a job basis in the UNICOS operating system; thus, it cannot be consolidated by job ID or job class. However, output from the `dodisk`(8) utility can be used for billing disk usage on a user ID or account ID basis.

2.3.1 Why Use Front-end Formatting

Sites may want to use a front-end formatter to customize Cray Research accounting data in the following situations:

- All billing is done on a single system. When accounting data from several systems are processed on a single system, the units of measure may need to be standardized. For example, all CPU time should be expressed in milliseconds.
- The front-end system is an IBM machine that requires character fields to be in EBCDIC format.
- Only a few fields are important to the billing system; these usually include CPU time, memory use, disk use, and swap use.

Cray Research accounting products let you choose from two types of front-end formatting:

- Cray Research system accounting (CSA) front-end formatters are templates of C programs that show you how to consolidate session file records and delivers output in VM, MVS, or ASCII format.
- The generic front-end formatter, `csagfef`(8), accepts as input a generic consolidated data file or multiple `pacct` (per-process accounting data) files. It delivers output as either an ASCII report or a Cray Research binary file. `csagfef` cannot convert output to VM or MVS format.

You should consider several factors when deciding which front-end formatter to use:

- The CSA front-end formatters require a source license, while the generic formatter does not.
- The generic front-end formatter delivers either ACSII or Cray binary data output, where binary numbers are always written as a 64-bit word. CSA formatters can be modified to write 32-bit numeric values or EBCDIC output.
- Both types of formatters process session record files, which are created by `csabuild`(8). However, the generic formatter is also capable of processing multiple `pacct` files.

2.3.2 Preparing to Use a Formatter

Before you attempt either to modify a CSA formatter or to execute the generic formatter, you must make several decisions based on what you want the final report or data file to contain. The issues you must decide upon include the following:

- Identifying the data that needs to be reported.

  A multitude of data can be extracted from a session or a `pacct` file. For efficiency and the conservation of disk space, only the necessary data should be consolidated by the CSA formatters or by `csagcon`(8).

- Selecting the consolidation keys.

  You can use various keys to consolidate the data. Both types of formatters support data consolidation by account ID, group ID, job ID, and user ID or some combination thereof. `csagcon` also supports data consolidation by job class, which is either interactive or batch through the Network Queuing System (NQS).

- Determining which sessions should be consolidated when the input is a session file.

You can consolidate data for only terminated sessions, only active sessions, or both terminated and active sessions.

- Selecting the format of the ASCII report or binary data file.

  Among the things to be decided are the units of the various fields, the precision, the order of the data, the character set, the length of character strings, and the size and format of binary integer and floating point numbers.

After making these decisions, you should modify or set up the front-end formatter to generate reports or data files based on these specifications. Normally, front-end formatters are executed by `csarun` in either the `FEF` or the `USEREXIT` state. See Section 2.1.10.3, page 51, for more information on these user exits.

2.3.3 CSA Front-end Formatting

All CSA front-end formatters contain code both to consolidate session record data and to send consolidated data to a report or file. You must modify one of these templates in order to consolidate and send the data output specifically needed by your site.

> **Note:** `csafef`(8), `csafef2`(8), and `csaibm`(8) are templates; if you execute them as released, they produce a message stating that they are templates. If your site wants to use one of these programs, you must have a source license and you must make modifications to the code. Any local changes made to these templates are not supported by Cray Research.

2.3.4 Generic Front-end Formatting

The generic accounting data consolidator `csagcon`(8) and the generic front-end formatter `csagfef`(8) are more flexible versions of the `csacon`(8) and `csacrep`(8) utilities. They let you do the following tasks:

- Consolidate a session file
- Consolidate one or more `pacct` accounting files
- Generate an ASCII report or a binary file based on a file created by `csagcon`

The `csagcon` and `csagfef` utilities let you specify the fields to be consolidated and the format of the report. In contrast, `csacon` and `csacrep` have hardcoded data specifications and formats that cannot be changed without source code and local modifications.

Administrators who execute `csagcon` may need privilege to access the the `/dev/kmem` file. If this privilege is needed and you do not possess it, `csagcon` will terminate with an error.

The `csagcon` and `csagfef` utilities can be executed from the `csarun` user exit scripts. Both commands can be invoked from either the `FEF` or `USEREXIT` state of `csarun`. See Section 2.1.10.3, page 51, for more information on user exits.

To invoke `csagcon` and `csagfef` from the `FEF` state, put these or similar commands in the file `/usr/lib/acct/csa.fef`:

```
csagcon -S ${SESSION_FILE} -s username -o ${SESSION_DIR}/gacct
csagfef -f ${SESSION_DIR}/gacct source_file > ${CRPT_DIR}/site.rpt
```

Alternately, the same two commands can be placed into the `/usr/lib/acct/csa.user` file; then, `csagcon` and `csagfef` will execute from the `csarun` `USEREXIT` state.

2.3.4.1 Data Consolidation

The `csagcon` command consolidates data either from a session file, which is created by `csabuild`(8), or from `pacct` files. You can choose the data that is to be consolidated by using the `csagcon -R` option. If a data list is not specified, a set of default variables is selected. In addition, some variables are always selected.

The variable names listed throughout this section are used by both `csagcon` and `csagfef`.

2.3.4.2 Required Data Variables

The following table lists the required variables that are always included in the consolidated data file. You must not include any of these variables in a `csagcon` request file (`-R` option). If you do, `csagcon` will terminate with an error.

Table 4. Required data variables

Variable	Type or Value	Description	
`acid` *	Integer	Account ID.	
`con_key`	Integer	`csagcon` consolidation option(s) you specify. If you specify multiple options, the values are added together.	
		Value	`csagcon` consolidation option
		0001	`-a` (consolidate by the account ID (`acid`) variable)
		0002	`-c` (consolidate by the job class (`jclass`) variable; job class is either interactive or NQS)
		0004	`-g` (consolidate by the group ID (`gid`) variable)
		0010	`-j` (consolidate by the job ID (`jid`) variable)
		0020	`-u` (consolidate by the user ID (`uid`) variable)
		0040	`-N` (consolidate NQS requests strictly by job ID)
		0100	`-A` (consolidate active and terminated sessions)
		0200	`-C` (consolidate only active sessions)
`creatime`	Integer	Creation time of the file in seconds since 00:00:00 GMT, 1 January 1970.	
`file_end`	Integer	If the input was a `pacct` file, this is the latest process end time found in the file. If the input was a session file, this is the end time of the last uptime period. Measured in seconds.	
`file_start`	Integer	If the input was a `pacct` file, this is the earliest process end time found in the file. If the input was a session file, this is the start time of the first uptime period. Measured in seconds.	
`gid` *	Integer	Group ID.	
`ios`	Integer	I/O subsystem type.	
		Value	I/O subsystem type
		1	Model E
`jclass` *	Integer	Job class.	
		Value	Job class
		1	Interactive job

Variable	Type or Value	Description
		2 — NQS job
jid *	Integer	Job ID.
ncpus	Integer	Number of CPUs started.
njobs	Integer	Number of jobs. Calculated as the number of pacct end-of-job records found.
nproc	Integer	Number of processes.
nsess	Integer	Number of sessions. This is meaningful only when the input was a session file.
num_datarec	Integer	Number of data records in the file.
sort_opt	Integer	csagcon sort option used. Value — csagcon sort option 0 — None (unsorted) 1 — -s acid (sorted by account ID then user ID) 2 — -s acname (sorted by account name then user name) 3 — -s jclass (sorted by job class then job ID) 4 — -s uid (sorted by user ID then account ID) 5 — -s uname (sorted by user name then account name)
tp_devgrp	String	An array that is indexed by 0 through (tp_ndevgrp -1) and contains the names of the tape device groups. The names are prefixed with tp_. If there are fewer than tp_ndevgrp tape device groups, the unused entries have values of tp_null0, tp_null1, and so on. This field is reported when the input was a session file and tape information was requested.
tp_ndevgrp	Integer	Number of tape device groups. This field is reported when the input was a session file and tape information was requested.
uid *	Integer	User ID.
us_nttype	Integer	Number of UNICOS station call processor (USCP) transfer types. This field is reported when the input was a session file and USCP information was requested.

Variable	Type or Value	Description
us_tname	String	An array that is indexed by 0 through (us_nttype -1) and contains the names of the USCP transfer types. The names are prefixed with us_. This field is reported when the input was a session file and USCP information was requested.
BYTE_CLICK	Integer	Number of bytes per click.
BYTE_WORD	Integer	Number of bytes per word.
CLK_TCK	Integer	Number of clocks per second.
FPTYPE	String	Floating point type: Cray or IEEE.
HARDWARE	String	Machine identification. Includes serial number and mainframe type.
MACHINE	String	Machine name.
MAXBDEVNO	Integer	Maximum number of block devices.
MAXCDEVNO	Integer	Maximum number of character devices.
MAXCPUS	Integer	Maximum number of CPUs for this mainframe type and subtype.
MEMORY	String	Memory configuration.
MEMORY_NWORD	Integer	Total system and user memory in words.
NODENAME	String	Network node name.
OS_HZ	Integer	Clock rate (the frequency per second with which the clock routine is called); usually 60 or 100.
RELEASE	String	Release of the operating system.
SDS_WGHT	Integer	Number of clicks per SDS allocation unit.
SOFTWARE	String	Software release information.
SYSNAME	String	Operating system name.
VERSION	String	Release version of the operating system.
WORD_CLICK	Integer	Number of words per click.

* If this variable is not selected as a consolidation key, its value is -2. For example, the following command consolidates session record file by job ID:

```
csagcon -jN -S Session-Record.0928 -o gacct.0928
```

In the file `gacct.0928` the values for the `acid`, `gid`, `jclass`, and `uid` variables will be -2 for records. This is because these variables were not selected as consolidation keys on the command line.

2.3.4.3 Default and Optional Data Variables

The following sections describe the data that you can specify in a `csagcon` request file (`-R` option). The request file contains a list of variables that will be consolidated by `csagcon`. By default, `csagcon` consolidates the same data as `csacon`(8).

The `csagcon` utility gets the default and optional data variables from the file `/usr/lib/acct/table_init`. Specifying a different file using the `-T` option is not recommended because `csagcon` expects the data variable names given in this file. Use caution in specifying the `-T` option; normally it is used only for debugging source code.

The column headings are defined as follows:

Heading	Meaning
Variable	The name that `csagcon` and `csagfef` use for the data item. This name, except where noted, should appear in the request file when you use the `csagcon -R` option.
Type	The data type of the variable. Valid types are `integer`, `float`, and `string`.
Unit	The unit, if any, of the data item. The item can be converted to another unit by `csagfef` (see Section 2.3.4.5.7, page 126).
Default	Specifies whether a data item is consolidated when the `csagcon -R` option is omitted. Yes — The data item is consolidated by default. No — The data item is not consolidated by default.
Job	Specifies whether the `csagcon -j` option must be used when the data item is consolidated. Yes — The `csagcon -j` option must be used. For this data item to be consolidated when `-j` is specified, either the `-R` option is not used and this is a default item, or the `-R` option is used and this item is listed in the request file.

No	The csagcon -j option does not have to be used.

2.3.4.3.1 Pacct Record Variables

This section describes the variables that contain pacct process information. These variables are available when the csagcon input is either a session file or one or more pacct files.

Note: The values in Table 5 are only available when using the csagcon -I option.

Table 5. pacct base record variables — per-process values

Variable	Type	Unit	Default	Job	Description
pp_p_cmd	String	-	No	No	Command name (first 8 characters).
pp_p_flag	Integer	-	No	No	Record flags (See ac_flag in /user/include/sys/acct.h).
pp_p_nice	Integer	-	No	No	Nice value.
pp_p_pid	Integer	-	No	No	Process ID.
pp_p_ppid	Integer	-	No	No	Parent process ID.
pp_p_stat	Integer	-	No	No	Exit status.
ps_p_tty	String	-	No	No	Controlling tty device (maximum of 8 characters).

Table 6. pacct base record variables - total values

Variable	Type	Unit	Default	Job	Description
pb_t_btime	Integer	Seconds	No	Yes	Process start time.
pb_t_ctime	Integer	Clocks	No	No	Process connect time.
pb_t_etime	Integer	Clocks	No	No	Elapsed time.

Variable	Type	Unit	Default	Job	Description
pb_t_io	Integer	Bytes	No	No	Number of characters transferred.
pb_t_iobtime	Integer	Clocks	No	No	I/O wait time.
pb_t_iosw	Integer		No	No	I/O swap count.
pb_t_iowmem	Integer	Click-ticks	No	No	I/O wait time memory integral while locked in memory.
pb_t_iowtime	Integer	Clocks	No	No	I/O wait while locked in memory.
pb_t_kcore	Float	Kiloword-minute	No	No	Kcore-minutes.
pb_t_lio	Integer		No	No	Number of logical I/O requests.
pb_t_mem	Integer	Click-ticks	No	No	Memory integral.
pb_t_phimem_max	Integer	Words	No	No	Maximum process highwater memory mark.
pb_t_phimem_min	Integer	Words	No	No	Minimum process highwater memory mark.
pb_t_rw	Integer		No	No	Number of physical I/O requests.
pb_t_sctime	Integer	Clocks	No	No	System call time.
pb_t_stime	Integer	Clocks	No	No	System CPU time.
pb_t_utime	Integer	Clocks	No	No	User CPU time.

Table 7. pacct base record variables - prime time values

Variable	Type	Unit	Default	Job	Description
pb_p_ctime	Float	Clocks	No	No	Process connect time.
pb_p_etime	Float	Clocks	No	No	Elapsed time.
pb_p_io	Float	Bytes	Yes	No	Number of characters transferred.
pb_p_iobtime	Float	Clocks	Yes	No	I/O wait time.
pb_p_iosw	Float		No	No	I/O swap count.
pb_p_iowmem	Float	Click-ticks	Yes	No	I/O wait time memory integral while locked in memory.
pb_p_iowtime	Float	Clocks	Yes	No	I/O wait while locked in memory.
pb_p_kcore	Float	Kiloword-minute	Yes	No	Kcore-minutes.
pb_p_lio	Float		Yes	No	Number of logical I/O requests.
pb_p_mem	Float	Click-ticks	No	No	Memory integral.
pb_p_rw	Float		Yes	No	Number of physical I/O requests.
pb_p_sctime	Float	Clocks	Yes	No	System call time.
pb_p_stime	Float	Clocks	Yes	No	System CPU time.
pb_p_utime	Float	Clocks	Yes	No	User CPU time.

Table 8. pacct base record variables - nonprime time values

Variable	Type	Unit	Default	Job	Description
pb_n_ctime	Foat	Clocks	No	No	Process connect time.
pb_n_etime	Float	Clocks	No	No	Elapsed time.

Variable	Type	Unit	Default	Job	Description
pb_n_io	Float	Bytes	Yes	No	Number of characters transferred.
pb_n_iobtime	Float	Clocks	Yes	No	I/O wait time.
pb_n_iosw	Float		No	No	I/O swap count.
pb_n_iowmem	Float	Click-ticks	Yes	No	I/O wait time memory integral while locked in memory.
pb_n_iowtime	Float	Clocks	Yes	No	I/O wait while locked in memory.
pb_n_kcore	Float	Kiloword-minute	Yes	No	Kcore-minutes.
pb_n_lio	Float		Yes	No	Number of logical I/O requests.
pb_n_mem	Float	Click-ticks	No	No	Memory integral.
pb_n_rw	Float		Yes	No	Number of physical I/O requests.
pb_n_sctime	Float	Clocks	Yes	No	System call time.
pb_n_stime	Float	Clocks	Yes	No	System CPU time.
pb_n_utime	Float	Clocks	Yes	No	User CPU time.

Table 9. pacct secondary data storage (SDS) record variables - total values

Variable	Type	Unit	Default	Job	Description
ps_t_memsw	Integer	Click-ticks	No	No	SDS execution memory integral.
ps_t_sdioch	Integer	Bytes	No	No	Number of bytes transferred to or from SDS.

Variable	Type	Unit	Default	Job	Description
`ps_t_sdlio`	Integer		No	No	Number of logical SDS I/O requests.
`ps_t_sdsmem`	Integer	Click-ticks	No	No	SDS residency memory integral.

Table 10. `pacct` SDS record variables - prime time values

Variable	Type	Unit	Default	Job	Description
`ps_p_memsw`	Float	Click-ticks	No	No	SDS execution memory integral.
`ps_p_sdioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from SDS.
`ps_p_sdlio`	Float		Yes	No	Number of logical SDS I/O requests.
`ps_p_sdsmem`	Float	Click-ticks	No	No	SDS residency memory integral.

Table 11. `pacct` SDS record variables - nonprime time values

Variable	Type	Unit	Default	Job	Description
`ps_n_memsw`	Float	Click-ticks	No	No	SDS execution memory integral.
`ps_n_sdioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from SDS.
`ps_n_sdlio`	Float		Yes	No	Number of logical SDS I/O requests.
`ps_n_sdsmem`	Float	Click-ticks	No	No	SDS residency memory integral.

Note: All of the variables in Table 12 are available when -E is specified. Job-specific variables (the Job value is Yes) are also accessible when the csagcon -j option is used.

Table 12. pacct end-of-job record variables

Variable	Type	Unit	Default	Job	Description
pe_t_fsblkused	Integer		No	Yes	Number of file system blocks used.
pe_t_jetime	Integer	Seconds	No	Yes	Time the job ended.
pe_t_jhimem	Integer	Clicks	No	Yes	Job highwater memory mark.
pe_t_jnice	Integer	-	No	No	Nice value at job termination.
pe_t_sdshiwat	Integer	SDS allocation units	No	Yes	Job SDS highwater mark.

In Table 13, the pd_t_b*xxxx* variables are arrays that are indexed by 0 through (MAXBDEVNO - 1). The pd_t_c*xxxx* variables are arrays that are indexed by 0 through (MAXCDEVNO - 1).

Table 13. pacct device I/O record variables - total values

Variable	Type	Unit	Default	Job	Description
pd_t_bioch	Integer	Bytes	No	No	Number of bytes transferred to or from the block device.
pd_t_blio	Integer		No	No	Number of logical I/O requests for the block device.

Variable	Type	Unit	Default	Job	Description
`pd_t_btype`	Integer		No	No	Major device number for block devices. A device number of -1 indicates that there is no accounting information for the array index.
`pd_t_cioch`	Integer	Bytes	No	No	Number of bytes transferred to or from the character device.
`pd_t_clio`	Integer		No	No	Number of logical I/O requests for the character device.
`pd_t_ctype`	Integer		No	No	Major device number for character devices. A device number of -1 indicates that there is no accounting information for this array index.

In Table 14, the `pd_p_b`*xxxx* variables are arrays that are indexed by 0 through (`MAXBDEVNO` - 1). The `pd_p_c`*xxxx* variables are arrays that are indexed by 0 through (`MAXCDEVNO` - 1).

Table 14. `pacct` device I/O record variables - prime time values

Variable	Type	Unit	Default	Job	Description
`pd_p_bioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from the block device.
`pd_p_blio`	Float		Yes	No	Number of logical I/O requests for the block device.

Variable	Type	Unit	Default	Job	Description
`pd_p_btype`	Integer		Yes	no	Major device number for block devices. A device number of -1 indicates that there is no accounting information for the array index.
`pd_p_cioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from the character device.
`pd_p_clio`	Float		Yes	No	Number of logical I/O requests for the character device.
`pd_p_ctype`	Integer		Yes	No	Major device number for character devices. A device number of -1 indicates that there is no accounting information for this array index.

In Table 15, the `pd_n_b`*xxxx* variables are arrays that are indexed by 0 through (`MAXBDEVNO` - 1). The `pd_n_c`*xxxx* variables are arrays that are indexed by 0 through (`MAXCDEVNO` - 1).

Table 15. `pacct` device I/O record variables - non-prime time values

Variable	Type	Unit	Default	Job	Description
`pd_n_bioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from the block device.
`pd_n_blio`	Float		Yes	No	Number of logical I/O requests for the block device.

Variable	Type	Unit	Default	Job	Description
pd_n_cioch	Float	Bytes	Yes	No	Number of bytes transferred to or from the character device.
pd_n_clio	Float		Yes	No	Number of logical I/O requests for the character device.

Table 16. pacct massively parallel processing (MPP) record variables - total values

Variable	Type	Unit	Default	Job	Description
pm_t_pe	Integer		No	No	Number of MPP processing elements.
pm_t_pe_max	Integer		No	No	Largest number of MPP processing elements used by a single process.
pm_t_pe_time	Integer	Clocks	No	No	Sum of (number of PEs used multiplied by time used).
pm_t_time	Integer	Clocks	No	No	MPP time used.
pm_t_time_max	Integer	Clocks	No	No	Greatest amount of MPP time used by a single process.

Table 17. `pacct` MPP record variables - prime time values

Variable	Type	Unit	Default	Job	Description
`pm_p_pe`	Float		Yes	No	Number of MPP processing elements.
`pm_p_pe_time`	Float	Clocks	Yes	No	Sum of (number of PEs used multiplied by time used).
`pm_p_time`	Float	Clocks	Yes	No	MPP time used.

Table 18. `pacct` MPP record variables - nonprime time values

Variable	Type	Unit	Default	Job	Description
`pm_n_pe`	Float		Yes	No	Number of MPP processing elements.
`pm_n_pe_time`	Float	Clocks	Yes	No	Sum of (number of PEs multiplied by time used).
`pm_n_time`	Float	Clocks	Yes	No	MPP time used.

Note: Each item in the following multitasking tables is an array that is indexed by 0 through (`MAXCPUS` - 1).

Table 19. `pacct` multitasking record variables - total values

Variable	Type	Unit	Default	Job	Description
`pu_t_mutime`	Integer	Clocks	No	No	Time connected to [i+1] CPUs.
`pu_t_smwtime`	Integer	Clocks	No	No	Semaphore wait time.

Table 20. pacct multitasking record variables - prime time values

Variable	Type	Unit	Default	Job	Description
pu_p_mutime	Float	Clocks	Yes	No	Time connected to [i+1] CPUs.
pu_p_smwtime	Float	Clocks	No	No	Semaphore wait time.

Table 21. pacct multitasking record variables - nonprime time values

Variable	Type	Unit	Default	Job	Description
pu_n_mutime	Float	Clocks	Yes	No	Time connected to [i+1] CPUs.
pu_n_smwtime	Float	Clocks	No	No	Semaphore wait time.

Table 22. pacct performance record variables - total values

Variable	Type	Unit	Default	Job	Description
pp_t_phrwblks	Integer		No	No	Number of raw physical blocks moved.
pp_t_rwblks	Integer		No	No	Number of buffered physical blocks moved.
pp_t_rtime	Integer	Clocks	No	No	Process start time past pb_t_btime.
pp_t_srunwtime	Integer	Seconds	No	No	SRUN wait time.
pp_t_swapclocks	Integer	Clocks	No	No	Swapped time.
pp_t_tiowtime	Integer	Clocks	No	No	Terminal I/O wait time.

Table 23. pacct performance record variables - prime time values

Variable	Type	Unit	Default	Job	Description
pp_p_phrwblks	Float		No	No	Number of raw physical blocks moved.
pp_p_rwblks	Float		No	No	Number of buffered physical blocks moved.
pp_p_rtime	Float	Clocks	No	No	Process start time past pb_t_btime.
pp_p_srunwtime	Float	Seconds	No	No	SRUN wait time.
pp_p_swapclocks	Float	Clocks	No	No	Swapped time.
pp_p_tiowtime	Float	Clocks	No	No	Terminal I/O wait time.

Table 24. pacct performance record variables - nonprime time values

Variable	Type	Unit	Default	Job	Description
pp_n_phrwblks	Float		No	No	Number of raw physical blocks moved.
pp_n_rwblks	Float		No	No	Number of buffered physical blocks moved.
pp_n_rtime	Float	Clocks	No	No	Process start time past pb_t_btime.
pp_n_srunwtime	Float	Seconds	No	No	SRUN wait time.
pp_n_swapclocks	Float	Clocks	No	No	Swapped time.
pp_n_tiowtime	Float	Clocks	No	No	Terminal I/O wait time.

2.3.4.3.2 Daemon Accounting Variables

The accounting variables that contain daemon usage information are available only when the csagcon input file is a session file.

In Table 25 each item is an array that is indexed by the tape device group names prefixed by tp_ (see the tp_devgrp array in Table 14, page 111) or by 0 through (tp_ndevgrp - 1).

Table 25. Tape accounting variables

Variable	Type	Unit	Default	Job	Description
tp_nmount	Integer		Yes	No	Number of volumes mounted.
tp_nread	Integer	Bytes	Yes	No	Number of bytes read.
tp_nwrite	Integer	Bytes	Yes	No	Number of bytes written.
tp_rtime	Integer	Seconds	Yes	No	Reservation time.
tp_stime	Integer	Clocks	Yes	No	System CPU time.
tp_utime	Integer	Clocks	Yes	No	User CPU time.

In Table 26, the values for nq_init, nq_disp, and nq_term are found in /usr/include/acct/dacct.h.

Table 26. NQS accounting variables

Variable	Type	Unit	Default	Job	Description
nq_btime *	Integer	Seconds	No	Yes	Start time of the request.
nq_disp *	Integer		No	Yes	Dispose subtype (NQ_DISP).
nq_elapse **	Integer	Seconds	No	Yes	Wall-clock time used while the request was running.
nq_init *	Integer		No	Yes	Initiation subtype (NQ_INIT).

Variable	Type	Unit	Default	Job	Description
nq_machname	String		No	Yes	Originating machine name (16 characters).
nq_mid *	Integer		No	Yes	Originating machine ID.
nq_nreq	Integer		Yes	No	Number of NQS requests.
nq_quename	String		No	Yes	Name of the last queue in which the request was located (16 characters).
nq_qwtime	Integer	Seconds	No	No	Queue wait time.
nq_reqname	String		No	Yes	Request name (16 characters).
nq_seqno *	Integer		No	Yes	Sequence number.
nq_stime	Integer	Clocks	Yes	No	Shepherd system CPU time.
nq_term *	Integer		No	Yes	Termination subtype (NQ_TERM).
nq_utime	Integer	Clocks	Yes	No	Shepherd user CPU time.
nq_wallclock ***	Integer	Seconds	No	Yes	Total wall-clock time for the request to complete.

* If the value for this field is unknown, or if this is an interactive session, this field is set automatically to -9.

** nq_elapse is the amount of wall-clock time which elapsed while the request was running on a CPU. This does not include queue wait time, system down time, or the period when the request was suspended, checkpointed, or held.

*** nq_wallclock is the total amount of wall-clock time it took the request to complete. This includes queue wait time and system down time. This value is reported only once for a request. It is possible that the amount of CPU time the request uses is greater than the wall-clock time, because the request could have created additional processes, been multitasked, or done work in the background.

Table 27. Connect time accounting variables

Variable	Type	Unit	Default	Job	Description
ct_con_n	Integer	Seconds	Yes	No	Nonprime time connect time.
ct_con_p	Integer	Seconds	Yes	No	Prime time connect time.
ct_nlogin	Integer		Yes	No	Number of interactive logins.

2.3.4.3.3 System Billing Units (SBU) Variables

The following table describes the variables that contain information about the system billing units (SBUs). If the input to csagcon is a session file, all the SBUs are multiplied by the appropriate NQS weighting factor. The NQS weighting factors are defined in the accounting configuration file /etc/config/acct_config.

Table 28. System billing units (SBU) variables

Variable	Type	Unit	Default	Job	Description
sb_pacct	Float	Billing units	No	No	pacct SBUs.
sb_tape	Float	Billing units	No	No	Tape SBUs.
sb_uscp	Float	Billing units	No	No	USCP SBUs.
sb_ctime	Float	Billing units	No	No	Connect time SBUs.
sb_total	Float	Billing units	Yes	No	Total SBU value.

2.3.4.4 Data File Format

The `csagcon` consolidated data file consists of header and data records. The header records describe both the machine on which the data was collected and the data records.

The `csagfef -h` option displays some of the information found in the header records.

The file is organized as follows:

Record Type	Description
Header word	File identifier that is defined in `/usr/include/sys/accthdr.h`.
`gc-defs`	Definitions record.
`gc-imeta`	Meta record for integer data.
`gc-fmeta`	Meta record for floating point data.
`gc-cmeta`	Meta record for character string data.
`gc-data`	Indicator for the start of data record 1.
`gc-int`	Data record 1 containing integer data.
`gc-float`	Data record 1 containing floating point data.
`gc-char`	Data record 1 containing character string data.
`gc-data`	Indicator for the start of data record 2.
`gc-int`	Data record 2 containing integer data.
`gc-float`	Data record 2 containing floating point data.
`gc-char`	Data record 2 containing character string data.

(Additional `gc-data`, `gc-int`, `gc-float`, and `gc-char` records for each data record.)

2.3.4.4.1 Header Records

Header records appear only once, at the beginning of the consolidated data file. There are three types of header records:

Header Record Type	Description
Header word	Identifies the file according to the format specified in the file `/usr/include/sys/accthdr.h`. This word allows other accounting programs to check for a valid input file type before attempting to process the file.
Definitions record	Contains constants and character strings that describe the machine on which the data was consolidated and array element names. These variables can be accessed by `csagfef`(8) and are listed in Section 2.3.4.2, page 100.
Meta record	Describes the data in the data records. A meta record lists the name, type, and size of each item or array in the data records and the order of the data found in the data records. There is a separate meta record for integer data, floating point data, and character string data.

2.3.4.4.2 Data Records

Data records follow the header records in a file. The `gc-data` record denotes the start of the data for a unique consolidation identifier.

2.3.4.5 `csagfef` Source Scripts

The `csagfef`(8) command is a translator that formats `csagcon`(8) output into an ASCII report or a binary file according to the directives found in a source script.

The `csagfef` scripts are based on four sections including the body, any of which may be empty or missing. Scripts can contain any of the following sections in any order:

`BEGIN` *{ statements }*
`END` *{ statements }*
`function` *name (arglist) { statements }*
statements

The `csagfef` command can process multiple source scripts, and one script can contain multiple `BEGIN`, `END`, or body sections. In these cases, `csagfef` executes the statements for all like sections in the order that they appear in the scripts or script.

For example, all statements in the various `BEGIN` sections will be combined into one `BEGIN` section. The statements will be in the same order as they appear in the scripts or script.

2.3.4.5.1 `BEGIN` Section

The statements associated with `BEGIN` comprise the preamble. The preamble is executed once after the definition and meta-data records are read. The preamble can be used to print report headings and to initialize variables used in the body

2.3.4.5.2 `END` Section

The statements associated with `END` comprise the postamble. The postamble is executed once after all the records in the data file have been read. You can instruct `csagfef` in this section to process and print summary data.

2.3.4.5.3 `function` Section

The statements in the `function` section of `csagfef` define functions as specified by you. Functions always begin with the word `function` followed by the function name and the argument list. The `arglist` consists of names separated by commas. These argument names are the formal parameters of the function and the variables that are local to the function. Function calls may be nested and recursive. The return statement can be used to return a value.

2.3.4.5.4 Body

Statements that are not in any of the above sections form the body of the `csagfef` source script. Typically, these statements print out information from the data records. This section is executed once for each data record encountered.

2.3.4.5.5 Example Source Scripts

Examples of `csagfef` source scripts can be found in the `/usr/src/cmd/acct/src/csa/csagfef/examples` directory.

2.3.4.5.6 `csagfef` Language Description

The `csagfef` language is the action language of `nawk` without the string processing operations. If you are familiar with `nawk`, you will have little difficulty writing and understanding `csagfef` scripts. The pattern part of `nawk` is unnecessary in `csagfef`, because the data format is defined in the data file. You merely select the data items to process by name.

`csagfef` implements a version of the `awk` language (new `awk`, or `nawk`) described in *The AWK Programming Language*, by Alfred Aho, Brian Kernighan, and Peter Weinberger (1988).

A `csagfef` script can include any of the following statements:

```
if ( expression ) statement [ else statement ]
while ( expression ) statement
do statement while ( expression )
for ( expression; expression; expression ) statement
break
continue
{ [ statements ] }
expression
print expression-list [ >expression ]
printf format[, expression-list ] [ >expression ]
next
exit [ expression ]
return [ expression ]
```

The following describes further the contents of statements in a `csagfef` script:

- Statement terminators. Statements are terminated by semicolons, right braces, or newlines.

- Statement continuation. Statements can be continued on successive lines by using \ as the last character of the line. Statements can also be continued after the following symbols:

```
,       (comma)
{       (left brace)
&&      (logical AND)
||      (logical OR)

do
else
)       (right parenthesis in an "if" or "for" statement)
```

- Comments. Nonexecutable comments begin with # and end with a newline. They can appear anywhere in the source script.
- Expressions. Expressions include constants, variables, and operators. Parentheses can be used to control the grouping of the operations in an expression.
- Logical expressions. Logical expressions have a value of 1 (true) and 0 (false). As in the C language, any nonzero value is taken to be true.
- Numbers. Numbers can be integers or floating points. The format is the same as that recognized by `strtod`(3C) and `strtol`(3C): digits, decimal point, digits, `e` or `E`, signed exponent. At least one digit or a decimal point must be present; the other components are optional. Octal integers begin with 0. Hexadecimal integers begin with 0 *x*.
- Variable names. Variable names consist of a letter followed by a string of letters, numbers, or the character _. Variables are used to name the data items found in the data records of the consolidated file.

  Some variables in the consolidated data file are arrays. The elements of these arrays can be referenced by indexing. For example, the variable, `pu_t_mutime`, is an array that contains the time a process was connected to (i+1) CPUs; see Table 19, page 114 (table: `pacct` multitasking record variables). The time a process was connected to one CPU is referenced by `pu_t_mutime` [0].

  You can also define additional variables within the `csagfef` source script; however, user-defined arrays are not supported.

A `csagfef` script can include prefix, infix, and suffix operators as follows:

Prefix operators

The `csagfef` command applies a prefix operator immediately preceding a term and any suffix operators. It then applies any prefix operators to the left of that operator, grouping them from right to left.

Operator	Action
++X	Preincrement
–X	Predecrement
+X	Plus
-X	Minus
!X	Logical NOT

Infix operators

The `csagfef` command applies infix operators, in descending order of precedence, as follows:

Operator	Action
X^Y	Exponentiation
X*Y	Multiplication
X/Y	Division
X%Y	Remainder
X+Y	Addition
X-Y	Subtraction
X<Y	Less than
X<=Y	Less than or equal
X>Y	Greater than
X>=Y	Greater than or equal
X==Y	Equals
X!=Y	Not equals
X&&Y	Logical AND
X||Y	Logical OR
Z?X:Y	Conditional
X=Y	Assignment
X*=Y	Multiply assign

X/=Y	Divide assign
X%=Y	Remainder assign
X+=Y	Add assign
X-=Y	Subtract assign
X,Y	Comma

Suffix operators

The `csagfef` command applies a suffix operator immediately following a term before it applies any other operator. It then applies any suffix operators to the right of that operator, grouping them from left to right. The following list shows the suffix operators:

Operator	Action
X++	Postincrement
X–	Postdecrement
X[Y]	Subscript
X(Y)	Function call

2.3.4.5.7 Built-in Functions

The `csagfef` command has the following built-in functions, with the function parameters (given in parentheses) defined at the end of the list:

Function name	Description
`abs(`*exp*)	Returns the absolute value of *exp*.
`acid2nam`(*num*)	Returns the character string associated with the account ID (*num*). If there is no associated string, return `Unknown`.
`bytes_to`(*num*[, *unit*])	Converts bytes to some other unit. If [, *unit*] is not specified, kilobytes are returned.
`clicks_to`(*num*[, *unit*])	Converts clicks to some other unit. If [, *unit*] is not

	specified, kilobytes are returned.
`clocks_to`(*num*[, *tunit*])	Converts clocks to some other unit. If [, *tunit*] is not specified, seconds are returned.
`close(`*str*)	Closes the file stream specified by *str*.
`frac(`*exp*)	Returns the fractional part of *exp*.
`gid2nam`(*num*)	Returns the character string associated with the group ID (*num*). If there is no associated string, return `Unknown`.
`imax(`*arr*)	Returns the index of the maximum element of array *arr*.
`imin(`*arr*)	Returns the index of the minimum element of array *arr*.
`int(`*exp*)	Returns the integer part of *exp*.
`isdefined(`*sym*)	Returns 1 if *sym* is defined. Otherwise, returns 0.
`nam2acid`(*str*)	Returns the numeric account ID associated with the account name (*str*). If there is no account ID, return -1.
`nam2gid`(*str*)	Returns the numeric group ID associated with the group name (*str*). If there is no group ID, returns -1.
`nam2uid`(*str*)	Returns the numeric ID associated with the user name (*str*). If there is no user ID, returns -1.

`strcmp`(*str1*, *str2*)	Compares two strings. Returns a value that is greater than, equal to, or less than 0 according to whether *str1* is greater than, equal to, or less than *str2*.
`strftime`(*fmt*)	Formats the time into a string according to (*fmt*).
`strlen`(*str*)	Returns the number of characters in string (*str*).
`sum(`*arr*)	Returns the sum of the elements in array *arr*.
`system(`*str*)	Passes *str* to the shell for execution.
`ticks_to`(*num*[, *tunit*])	Converts ticks to some other unit. If [, *tunit*] is not specified, seconds are returned.
`uid2nam`(*num*)	Returns the character string associated with the user ID (*num*). If there is no associated string, returns `Unknown`.
`words_to`(*num*[, *unit*])	Converts words to some other unit. If [, *tunit*] is not specified, kilowords are returned.

The definitions of the function parameters are as follows:

Parameter	Definition
arr	An array name. For example: `imax` (`pd_t_cioch`)
exp	A variable name or a function invocation. For example:

`abs` (`pb_t_rw`)

Variable name

fmt

NULL or a valid `strftime`(3C) format that is enclosed in double quotes. For example:

`strftime` ()

NULL format

`strftime` (" `%X` ")

`strftime` format

num

Either an integer value or the name of a variable that contains an integer value. For example:

`bytes_to` (`pb_t_io`)

Variable name

`uid2nam` (*uid*)

Variable name

`words_to` (`5125`)

Integer value

str, *str1*, *str2*

Either character strings enclosed in double quotation marks or the names of a variables whose values are character strings. For example:

`close` (" `cpu_data` ")

Character string

command = `"date; uname -a"`

`system` (*command*)

Variable that contains a character string

sym

A variable name. Names of array elements are not valid symbols. *sym* can be defined by the `csagfef -D` option. For example:

	`if` (`isdefined` (`ios_e`))
	Variable
	`if` (`isdefined` (`us_stime` [`us_Dispose`]))
	Not valid
	`csagfef -DCPU`
	`if` (`isdefined` (`CPU`))
	Symbol defined by the `csagfef -D` option
unit	May be one of the following:

`B`	Converts to bytes
`KB`	Converts to kilobytes (2^{10} bytes)
`MB`	Converts to megabytes (2^{20} bytes)
`GB`	Converts to gigabytes (2^{30} bytes)
`W`	Converts to words
`KW`	Converts to kilowords (2^{10} words)
`MW`	Converts to megawords (2^{20} words)
`GW`	Converts to gigawords (2^{30} words)
`number`	Uses `number` as the divisor and divides the value by number

`variable_name`	Uses `variable_name` as the divisor

Examples of using the `unit` function parameter follow:

`bytes_to` (`tp_nread, MB`)

Converts bytes to megabytes

`words_to` (`pb_t_phimem_max`, `1000`)

Uses 1000 as the divisor and returns (`pb_t_phimem_max / 1000`)

tunit

May be one of the following:

`SEC`	Converts to seconds
`MIN`	Converts to minutes
`HOUR`	Converts to hours
`DAY`	Converts to days

Example:

`clocks_to` (`pb_t_iowtime`, `MIN`)

Convert clocks to minutes

2.3.4.5.8 Built-in Variables

The `csagfef` command has the following built-in variables, as shown in Table 29:

Table 29. Built-in variables

Variable	Default	Description
`FILENAME`	None	Name of the current input file
`NR`	None	Number of data records read so far
`OFMT`	`%.6g`	Output format for printing numbers

Variable	Default	Description
OFS	" "	Output field separator
ORS	\n	Output record separator
RSIZE	None	Size of the data records in bytes

2.3.4.5.9 Generic Front-end Formatting Example

The extended example presented here illustrates how you can consolidate and format data for NQS requests using `csagcon` and `csagfef`. It assumes input from a session file. The example follows the steps listed in Section 2.3, page 97.

1. Identify the data that needs to be reported.

   Determine the information that is useful to your site. In this case, for each NQS request the example will report the following fields:

   - User name
   - Account name
   - Request name
   - Request ID
   - Queue name
   - CPU time
   - Memory high-water value
   - Queue wait time
   - Locked I/O wait time
   - Unlocked I/O wait time

   Because some of these items are not default `csagcon` consolidation items, you must specify a request file when executing `csagcon`. The following variable names, which are described in Table 14, page 111, through Table 29, page 131, must be in the request file (`nqs.req`). You can find a copy of this file in the `/usr/src/cmd/acct/src/csa/csagfef/examples` directory.

   ```
   nq_reqname
   nq_seqno
   nq_quename
   ```

```
pb_t_stime
pb_t_utime
pb_t_phimem_max
nq_qwtime
pb_t_iowtime
pb_t_iobtime
```

Pass the request file name (`nqs.req`) to `csagcon` by using the `-R` option (`-R nqs.req`).

2. Select the `csagcon` consolidation keys.

   To extract information for each NQS request, you must select consolidation keys: appropriate job ID (`-j` option) and job class (`-c` option). However, you must be certain that all portions of an NQS request are processed as though they have the same job ID, which is the default. (For this example, do not specify the `-N` option, which consolidates each portion of an NQS request according to its job ID).

   To report the username and account name that is associated with each request, you also must specify the `-u` and `-a` options. If these two keys are not specified, the username and account name will not be known.

   Note: All consolidation keys (`acid`, `gid`, `jclass`, `jid`, and `uid`) that are not selected on the `csagcon` command line by the `-a`, `-g`, `-c`, `-j`, and `-u` options, will have a value of -9.

   For example, if you do not specify the `-u` option, the `uid` variable will always have a value of -9.

   If you want to sort the output, use the `-s` option. In this example, the output is sorted alphabetically by `username` (`-s username` option).

   To summarize, the consolidation and sort options used in this example are the following: `-j -c -u -a -s username`.

3. Determine which sessions should be consolidated.

   This example will consolidate only terminated sessions (default option). You can use the `-A` or `-C` option to consolidate all sessions or only active sessions.

   The data to be consolidated now is identified and you are ready to execute `csagcon`. If you assume that the input comes from a session file named `Super-Record.1130` and the output is written to the file `gacct.1130`, you would execute the following command:

```
csagcon -S Super-Record.1130 -o gacct.1130 -R nqs.req -jcua -s username
```

4. Format the consolidated data into a report.

   You must decide the units and length of the various fields. In this example, memory highwater is reported in megawords and CPU time, queue wait time, locked I/O wait time, and unlocked I/O wait time is reported in seconds. Data that is not already in the correct units is converted by `csagfef`. Tables Table 14, page 111 through Table 29, page 131 list the default units of the various fields.

   After deciding on the format, you must write a `csagfef` source script that tells `csagfef` how to generate the report. The following script can be used as input to `csagfef` and is found in the following file:

   `/usr/src/cmd/acct/src/csa/csagfef/examples/nqs.ss`

   The script contains variables that control the writing of the header and summary lines. When `-D HEADER` is specified on the command line, `csagfef` outputs the header. When `-D SUMMARY` is specified, summary information is written.

   If you assume that the consolidated data file is named `gacct.1130`, and the source script is named `nqs.ss`, the following command will generate a report without the header and summary lines:

   ```
   csagfef -f gacct.1130 nqs.ss
   ```

   If you want both, the header and summary information, you should execute the following command:

   ```
   csagfef -f gacct.1130 -D HEADER -D SUMMARY nqs.ss
   ```

   The `nqs.ss` source script listing follows.

```
BEGIN {
#
#
#       Figure out which sessions were consolidated.
#
if (con_key & 0100) {
        CONSOL = "ACTIVE AND COMPLETED SESSIONS"
} else if (con_key & 0200) {
        CONSOL = "ONLY ACTIVE SESSIONS"
} else {
        CONSOL = "ONLY COMPLETED SESSIONS"
}
```

```
#
#       Initialize counters.
ntot_sess = 0                   # Total number of sessions
nnqs = 0                        # Number of NQS sessions
#
#       Print the header if "-D HEADER" was specified on the command line.
#
if ( isdefined(HEADER) ) {
        printf("%s   DAILY REPORT FOR %s (Rel %s, %s)\n\n",
           strftime("%c", creatime), SYSNAME, RELEASE, VERSION)

        printf("INCLUDES DATA FOR %s BETWEEN\n", CONSOL)
        printf("    %s AND %s\n\n",
           strftime("%c", file_start), strftime("%c", file_end))

        printf("                                               REQUEST  ")
        printf("                 CPU TIME    MEM HIWAT QWAIT   LCK IO  ")
        printf("UNLCK  \n")

        printf("USER NAME    ACCOUNT NAME     REQUEST NAME     ID       ")
        printf("QUEUE NAME       [SECS]      [MW]      [SECS]  WAIT    ")
        printf("IO WAIT\n")

        printf("============ ================ ================ ======== ")
        printf("================ =========== ========= ======= ======= ")
        printf("=======\n")
}
}

ntot_sess++             # count the total number of sessions

if ( jclass == 2 ) {    # output information only about NQS requests
        nnqs++          # count the number of NQS requests

        username = uid2nam(uid)                   # user name
        acname = acid2nam(acid)                   # account name

        cputime = clocks_to(pb_t_stime, SEC) + \ clocks_to(pb_t_utime, SEC)
                                                  # CPU time in seconds
        memhiwat = words_to(pb_t_phimem_max, MW)  # memory high water in megwords

        lockio = clocks_to(pb_t_iowtime, SEC)     # locked I/O wait in seconds
```

```
        ulockio = clocks_to(pb_t_iobtime, SEC)    # unlocked I/O wait in seconds

        printf("%-12.12s %-16.16s %-16.16s %-8d %-16.16s ",
           username, acname, nq_reqname, nq_seqno, nq_quename)
        printf("%11.3f %8.0f %7d %7.1f %7.1f\n",
           cputime, memhiwat, nq_qwtime, lockio, ulockio)

}

#
#       Print summary information about the input file if "-D SUMMARY"
#       was specified on the command line.
#
END {
if ( isdefined(SUMMARY) ) {
        printf("\n\nInput file: %s\nTotal number of sessions: %d\n",
           FILENAME, ntot_sess)
        printf("Number of NQS requests: %d\n", nnqs)
        printf("Number of non-NQS requests: %d\n", ntot_sess - nnqs)
}
```

The script above produces the following output. Both the header and summary information are included.

```
Wed Nov 30 10:04:50 1994   DAILY REPORT FOR sn1703c (Rel 9.0.0ao, d90.50)

INCLUDES DATA FOR ONLY COMPLETED SESSIONS BETWEEN
    Wed Nov 30 07:58:09 1994 AND Wed Nov 30 09:51:45 1994

                                    REQ             CPU TIME MEM HIWAT QWAIT  LCK IO  UNLCK
USER NAME ACCOUNT NAME REQUEST NAME ID  QUEUE NAME [SECS]   [MW]      [SECS] WAIT    IO WAIT
========= ============ ============ ==  ========== ======== ========= ====== ======= =======
fe        Xydev        STDIN        3   b_30_1     0.411        1       4       0.1     0.4
fe        Xydev        STDIN        4   b_30_1     0.414        1       3       0.1     0.3
pds       SysAdm       STDIN        6   b_30_1     0.570        0       4       0.0     0.4
root      SysAdm       STDIN        6   b_30_1     0.544        0       0       0.0     0.5
root      SysAdm       SLSCRUB      7   b_1200_1   0.958        0       0       0.0     1.6
root      SysAdm       STDIN        5   b_30_1     0.531        0       0       0.0     0.1
root      Xydev        STDIN        4   b_30_1     0.558        0       0       0.0     0.3
root      Xydev        STDIN        3   b_30_1     0.552        0       0       0.1     0.4
user1     SysAdm       SLSCRUB      7   b_1200_1   2.079        0       9       0.1    14.8


Input file: gacct
Total number of sessions: 175
Number of NQS requests: 10
Number of non-NQS requests: 165
```