

Accounting [10]

UNICOS provides two types of system accounting, standard UNIX System V accounting or Cray system accounting (CSA). You may use one or the other of these accounting packages at your site. To help you decide which accounting package to use, see Section 10.1, page 195, which describes the unique features of CSA.

For information on using standard UNIX System V accounting, see *UNICOS Resource Administration*, Cray Research publication SG-2302.

This section describes CSA, which is the more complete and frequently used of the two accounting types. It includes the following:

- An overview of CSA, including unique CSA features, descriptions of directories and files, and the `/usr/lib/acct/csarun` primary daily accounting shell script.
- Procedures to follow so that you can set up CSA and execute daily accounting procedures that result in the generation of a variety of reports.

Note: The UNICOS Station Call Processor (USCP) does not apply to the CRAY J90se system.

Your accounting configuration file is located in `/etc/config/acct_config`. A sample file is provided at the end of this section; the sample file may differ slightly from the one included with your system.

10.1 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.

10.1.1 Concepts and terminology

The following concepts and terms are important in CSA:

<u>Term</u>	<u>Description</u>
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 <i>daily accounting</i> .
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 <code>csarun(8)</code> is invoked with the <code>-A</code> option. <code>csarun</code> places recycled data into data files in the <code>/usr/adm/acct/day</code> directory. These data files are suffixed with <code>0</code> ; for example, per-process accounting data for active sessions from previous accounting periods is in the <code>/usr/adm/acct/day/pacct0</code> file.
Session	<p>CSA organizes accounting data by sessions and boot times and then places the data into a session record file.</p> <p>For non-NQS jobs, a <i>session</i> consists of all accounting data for a given job ID during a single boot period.</p> <p>A <i>session</i> 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.</p>

Uptime period or boot period	A period delineated by the system boot times found in <code>/etc/csainfo</code> . The <code>csaboosts(8)</code> command writes to this file during system boot.
------------------------------	---

10.1.2 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 10.1.7, page 214.

10.1.2.1 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 3 illustrates the directory structure for both directories.

Figure 3. `/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:

<u>Abbreviation</u>	<u>Definition</u>
<code>MMDD</code>	Month, day
<code>hhmm</code>	Hour, minute

10.1.2.2 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/csaboosts` (see `csaboosts(8)`), which records

boot times at system startup. Programs used only by CSA begin with the characters `csa`.

10.1.2.3 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.

<u>Accounting file</u>	<u>Description</u>
<code>/usr/adm/acct/day/dtmp</code>	Disk accounting data
<code>/usr/adm/acct/day/nqacct*</code>	NQS daemon accounting data
<code>/usr/adm/acct/day/pacct*</code>	Per-process accounting data
<code>/usr/adm/acct/day/tpacct*</code>	Tape daemon accounting data
<code>/usr/adm/acct/day/soacct*</code>	Socket accounting data
<code>/etc/csainfo</code>	Boot times
<code>/etc/wtmp</code>	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 10.1.5, page 207, describes how to do this. By default, all raw data files are deleted after use and are not archived.

10.1.2.4 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:

<u>File</u>	<u>Description</u>
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
Wdiskacct	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

10.1.2.5 Processed data files

CSA outputs the following data files:

<u>File</u>	<u>Description</u>
/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.

10.1.2.6 Reports

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

<u>File</u>	<u>Description</u>
<code>/usr/adm/acct/fiscal/rpt/MMDD/hhmm/rprt</code>	Periodic accounting report
<code>/usr/adm/acct/sum/rpt/MMDD/hhmm/rprt</code>	Daily accounting report

10.1.3 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/csaboosts` 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 10.1.4, page 203, 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)`).

10.1.4 Setting up CSA

The following is a brief description of setting up CSA. Site-specific modifications are discussed in detail in Section 10.1.10, page 229. 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 10.1.10.7, page 244, 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.

- 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.

- 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

10.1.5 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.

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.

10.1.5.1 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.

10.1.5.2 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`.

10.1.5.3 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.

<u>State</u>	<u>Description</u>
SETUP	The current accounting files are switched via <code>turnacct(8)</code> and <code>turndacct(8)</code> . These files are then moved to the <code>/usr/adm/acct/work/MMDD/hhmm</code> directory. File names are prefaced with <code>w.</code> <code>/etc/wtmp</code> and <code>/etc/csainfo</code> are also moved to this directory.
WTMPFIX	The <code>wtmp</code> file in the <code>work</code> directory is checked for accuracy by <code>wtmpfix</code> (see <code>fwtmp(8)</code>). Some date changes cause <code>csaline(8)</code> to fail, so <code>wtmpfix</code> attempts to adjust the time stamps in the <code>wtmp</code> file if a date change record appears. If <code>wtmpfix</code> is unable to fix the <code>wtmp</code> file, the <code>wtmp</code> file must be manually repaired. This is described in Section 10.1.6.1, page 212.
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 <code>BAD.</code> in the <code>/usr/adm/acct/work/*</code> directory. The corrected files do not have this prefix.
PREPROC	The NQS and connect time (<code>wtmp</code>) accounting files are run through preprocessors. File names of preprocessed files are prefixed with a <code>P</code> in the <code>/usr/adm/acct/work/MMDD/hhmm</code> directory.
ARCHIVE1	First user exit of the <code>csarun</code> script. If a script named <code>/usr/lib/acct/csa.archive1</code> exists, it will be executed through the shell <code>.</code> (<code>dot</code>) command. The <code>.</code> (<code>dot</code>) 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 <code>\${WORK}</code> .
BUILD	The per-process, NQS, tape, socket, and connect accounting data is organized into a session record file.
ARCHIVE2	Second user exit of the <code>csarun</code> script. If a script named <code>/usr/lib/acct/csa.archive2</code> exists, it will be executed through the shell <code>.</code> (<code>dot</code>) command. The <code>.</code> (<code>dot</code>) 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 <code>cacct.h</code> format. The <code>cacct</code> file is put into the <code>/usr/adm/acct/sum/data/MMDD/hhmm</code> directory for use by <code>csaperiod(8)</code> .
REPORT	Generates the daily accounting report and puts it into <code>/usr/adm/acct/sum/rpt/MMDD/hhmm/rprt</code> . A consolidated data file, <code>/usr/adm/acct/sum/data/MMDD/hhmm/cacct</code> , 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, <code>/usr/adm/acct/sum/rpt/MMDD/hhmm/rprt</code> .
FEF	Third user exit of the <code>csarun</code> script. If a script named <code>/usr/lib/acct/csa.fef</code> exists, it will be executed through the shell <code>.</code> (<code>dot</code>) command. The <code>.</code> (<code>dot</code>) command will not execute a compiled program, but the user exit script can. <code>csarun</code> 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 <code>csarun</code> script. If a script named <code>/usr/lib/acct/csa.user</code> exists, it will be executed through the shell <code>.</code> (<code>dot</code>) command. The <code>.</code> (<code>dot</code>) command will not execute a compiled program, but the user exit script can. <code>csarun</code> 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.

10.1.5.4 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.

10.1.6 Verifying and correcting data files

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

10.1.6.1 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.

10.1.6.2 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.

10.1.6.3 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.

10.1.7 Files and directories

This section describes the files and directories used by CSA.

10.1.7.1 /usr/adm/acct directory

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

<u>Directory</u>	<u>Description</u>
<code>day</code>	Current accounting files
<code>fiscal</code>	Periodic accounting data and reports
<code>nite</code>	Processing messages and errors
<code>sum</code>	Daily accounting data and reports
<code>work</code>	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.

<u>File</u>	<u>Description</u>
<code>dtmp</code>	Disk accounting data (ASCII) created by <code>dodisk(8)</code>
<code>nqacct*</code>	NQS daemon accounting data
<code>pacct*</code>	Per-process accounting data
<code>soacct*</code>	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:

<u>File</u>	<u>Description</u>
cms	Periodic command usage data in command summary (cms) record format
pdacct	Consolidated periodic data generated on <i>MMDD</i> at <i>hhmm</i>

The `/usr/adm/acct/fiscal/rpt/MMDD/hhmm` directory contains the periodic accounting report, `rprrt`, 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:

<u>File</u>	<u>Description</u>
active	Progress and status of <code>csarun</code>
activeMMDDhhmm	Progress and status of <code>csarun</code> after an error has been detected
clastdate	Last two times <code>csarun</code> was executed; in <i>MMDD hhmm</i> format
disktacct	Disk accounting records in <code>cacct.h</code> format; created by <code>dodisk(8)</code>
dk2log	Diagnostic output created during execution of <code>dodisk</code>
E*MMDDhhmm	Error/warning messages for an accounting run done on <i>MMDD</i> at <i>hhmm</i>
fd2log	Diagnostic output created during execution of <code>csarun</code>
lineuse	tty line usage report
lock, lock1	Controls simultaneous invocations of <code>csarun</code>
pd2log	Diagnostic output created during execution of <code>csaperiod</code>
pdact	Progress and status of <code>csaperiod</code>
pdactMMDDhhmm	Progress and status of <code>csaperiod</code> 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:

<u>File</u>	<u>Description</u>
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, rpt, 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:

<u>File</u>	<u>Description</u>
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:

<u>Program/script</u>	<u>Description</u>
<code>csaaddc</code>	Merges consolidated (cacct) accounting files
<code>csabuild</code>	Creates a session file
<code>csacon</code>	Creates a consolidated (cacct) accounting file
<code>csaconvert</code>	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
<code>csacrep</code>	Generates consolidated accounting reports
<code>csadrep</code>	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 <i>session</i> and <i>pacct</i> 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 (<i>/etc/wtmp</i>)
csanqs	Preprocesses NQS accounting data
csapacct	Verifies and deletes records from a <i>pacct</i> 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:

<u>Program/script</u>	<u>Description</u>
<i>csa.archive1</i>	Site-generated user exit for <i>csarun</i>
<i>csa.archive2</i>	Site-generated user exit for <i>csarun</i>
<i>csa.fef</i>	Site-generated user exit for <i>csarun</i>

`csa.user` Site-generated user exit for `csarun`

10.1.7.2 /etc directory

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

<u>File</u>	<u>Description</u>
<code>csaboosts</code>	Captures system boot times
<code>csainfo</code>	Output file of <code>csaboosts</code>
<code>wtmp</code>	Current connect time data

10.1.7.3 /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.

10.1.8 CSA data processing

The flow of data among the various CSA programs is explained in this section and is illustrated in Figure 4.

Figure 4. 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`, `ngacct`, `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 `csafeef(8)`, `csafeef2(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 10.1.5, page 207 and Section 10.1.10.3, page 241) 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. `csarep(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 10.1.4, page 203, for more information.

10.1.9 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.

10.1.9.1 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 10.1.9.3, page 224.

10.1.9.2 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 `csaboosts` 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.

10.1.9.3 How to remove recycled data

Before choosing to delete recycled data, you should understand the repercussions, as described in Section 10.1.9.4, page 226. 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.

10.1.9.4 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 10.1.9.5, page 228.
- 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 10.1.9.3, page 224.

Table 16. Possible effects of removing recycled data

Method	Underbilling?	Incorrect billing?	Consolidated data file
<code>csarecy -A</code>	Yes. Users are not billed for the portion of the session that was terminated by <code>csarecy -A</code> .	Possible. Manually terminated recycled sessions may be billed improperly in a future billing period.	Does not include data for sessions terminated by <code>csarecy -A</code> .
<code>csabuild -o</code>	No. Users are billed for the portion of the session that was terminated by <code>csabuild -o</code> .	Possible. Manually terminated recycled sessions may be billed improperly in a future billing period.	Includes data for sessions terminated by <code>csabuild -o</code> .
<code>csarun -A</code>	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.
<code>rm</code>	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.

10.1.9.5 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.

10.1.10 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

10.1.10.1 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`.

10.1.10.1.1 `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`.

<u>Value</u>	<u>Description</u>
<code>P_BASIC</code>	Prime-time weight factor. <code>P_BASIC</code> is multiplied by the sum of prime time SBU values to get the final SBU factor for the <code>pacct</code> base record.
<code>P_TIME</code>	General-time weight factor. <code>P_TIME</code> is multiplied by the time SBUs (made up of <code>P_STIME</code> , <code>P_ITIME</code> , <code>P_SCTIME</code> , and <code>P_INTTIME</code>) to get the time contribution to the <code>pacct</code> base record SBU value.
<code>P_STIME</code>	System CPU-time weight factor. The unit used for this weight is <i>billing units</i> per second. <code>P_STIME</code> is multiplied by the system CPU time to get the system CPU factor.
<code>P_UTIME</code>	User CPU-time weight factor. The unit used for this weight is <i>billing units</i> per second. <code>P_UTIME</code> 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 <code>MUTIME_WEIGHT</code> parameters have been set to values other than 1.0. See the following explanation of these values.
<code>P_ITIME</code>	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 <i>billing units</i> per second.
P_INTTIME	Weight factor for interrupt time. The unit used for this weight is <i>billing units</i> 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 <code>pacct</code> base record SBU value.
P_XMEM	CPU-time-memory-integral weight factor. The unit used for this weight is <i>billing units</i> per Kword-minute. P_XMEM is multiplied by the memory integral (see Section 10.1.12.1, page 252). This value is affected by your site's choice of MEMINT (in the accounting configuration file <code>/etc/config/acct_config</code>).
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 <i>billing units</i> 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 <code>pacct</code> base record SBU value.
P_BYTEIO	Characters-transferred weight factor. The unit used for this weight is <i>billing units</i> 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 10.1.11.1, page 246 (field `ac_io`), for more information.

P_PHYIO	Physical-I/O-request weight factor. The unit used for this weight is <i>billing units</i> 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 <i>billing units</i> 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.

10.1.10.1.2 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 10.1.12, page 251.

10.1.10.1.3 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.

10.1.10.1.4 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:

$$P_MPPPE \text{ billing units} = P_MPPPE * \sum_0^{\# \text{ of sessions}} (\text{no. MPP PEs used} * \text{MPP time used})$$

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

$$NP_MPPPE \text{ billing units} = NP_MPPPE * \sum_0^{\# \text{ of sessions}} (\text{no. MPP PEs used} * \text{MPP time used})$$

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:

$$\text{P_MPPBB billing units} = \text{P_MPPBB} * \sum_0^{\# \text{ of sessions}} (\text{no. MPP barrier bits used} * \text{MPP time used})$$

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

$$\text{NP_MPPBB billing units} = \text{NP_MPPBB} * \sum_0^{\# \text{ of sessions}} (\text{no. MPP barrier bits used} * \text{MPP time used})$$

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:

$$\text{P_MPPTIME billing units} = \text{P_MPPTIME} * \sum_0^{\# \text{ of sessions}} (\text{MPP time used})$$

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

$$\text{NP_MPPTIME billing units} = \text{NP_MPPTIME} * \sum_0^{\# \text{ of sessions}} (\text{MPP time used})$$

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.

10.1.10.1.5 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`.

10.1.10.1.6 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.

10.1.10.1.7 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.

10.1.10.1.8 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_SBUi` 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

10.1.10.1.9 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:

<u>SBU factor</u>	<u>Description</u>
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 10.1.14, page 254).

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/cl/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 10.1.14, page 254.

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.

10.1.10.1.10 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:

<u>Variable</u>	<u>Description</u>
<i>stime</i>	System CPU time in seconds.
<i>utime</i>	User CPU time in seconds. User CPU time is the sum of user times after weighting for multitasking.
<i>iowtime</i>	Time (in seconds) spent waiting in the kernel for I/O while the process is locked in memory.
<i>cpumem</i>	Memory integral (see Section 10.1.12.1, page 252).
<i>iowmem</i>	I/O-wait-time memory integral.
<i>bytes</i>	Number of bytes of data transferred.
<i>phy</i>	Number of physical I/O requests made.
<i>log</i>	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 10.1.11.1, page 246.

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 $\$25 / (60 \text{ minutes} * 2^{10} \text{ Kwords})$ 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 $\$3 / 3600 \text{ seconds}$ 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 $\$25 / (60 \text{ seconds} * 2^{10} \text{ Kwords})$ 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 ($63072000 \text{ seconds} * 5\% = 3153600 \text{ seconds}$). The amount of I/O that can be completed in 2 years is $31745177026560 \text{ bytes}$ ($9.6 \text{ Mbytes/second} * 3153600 \text{ seconds}$). Thus, you would charge $\$125,000 / 31745177026560 \text{ bytes}$ 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 $\$0.01 / 1000 \text{ I/O requests}$ 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):

<u>Weight factor</u>	<u>Charge</u>
P_BASIC	1.0

P_TIME	1.0
P_ETIME	0.0277777777777777
P_STIME	0.0
P_ETIME	0.0008333333333333
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:

<u>Weight factor</u>	<u>Charge</u>
P_BASIC	1.0
P_TIME	1.0
P_ETIME	0.0277777777777777
P_STIME	0.0
P_ETIME	0.0008333333333333
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

10.1.10.2 Daemon accounting

Accounting information is available from NQS, online tapes, and sockets. Data is written to the `ngacct`, `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 10.1.4, page 203, 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 10.1.4, page 203). 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
```

10.1.10.3 Setting up user exits

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

<u>csarun state</u>	<u>User exit</u>
ARCHIVE1	<code>/usr/lib/acct/csa.archive1</code>
ARCHIVE2	<code>/usr/lib/acct/csa.archive2</code>

```
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 10.1.10.7, page 244).

10.1.10.4 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:

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

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

10.1.10.5 Tailoring CSA shell scripts and commands

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

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

10.1.10.6 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 10.1.10.3, page 241, for more information.

10.1.10.7 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 10.1.4, page 203, 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 10.1.4, page 203.

10.1.10.8 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
```

10.1.10.9 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.

10.1.11 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.

10.1.11.1 Base accounting record

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

<u>Field</u>	<u>Description</u>
<code>ac_flag</code>	Accounting flags; may be any of the flags defined in <code>sys/acct.h</code> . <code>FORK</code> means that the process forked but did not <code>exec</code> . <code>ASU</code> means that the process used super-user privileges.
<code>ac_stat</code>	Low-order 8 bits from the process's exit value. See the <code>wait(2)</code> man page for more information.
<code>ac_uid</code>	Real user ID.
<code>ac_gid</code>	Real group ID.
<code>ac_tty</code>	Controlling tty device.
<code>ac_btime</code>	Start time of the process (in seconds).
<code>ac_utime</code>	User CPU time used (in clocks). This number is repeatable for nonmultitasked jobs (within the limitations caused by memory conflicts).
<code>ac_stime</code>	System CPU time used (in clocks). This number is not repeatable, because it varies with system load.
<code>ac_etime</code>	Elapsed time while process executed (in clocks). This number is not repeatable.
<code>ac_mem</code>	Memory integral selected when <code>MEMINT = 1</code> (in clicks-ticks). (<code>MEMINT</code> is located in <code>/etc/config/acct_config</code> .) 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. See Section 10.1.12.1, page 252, for more information.

<code>ac_mem2</code>	Memory integral selected when <code>MEMINT = 2</code> (in clicks-ticks). (<code>MEMINT</code> is located in <code>/etc/config/acct_config</code> .) This integral is not constant and varies with machine load. See Section 10.1.12.1, page 252, for more information.
<code>ac_mem3</code>	Memory integral selected when <code>MEMINT = 3</code> (in clicks-ticks). (<code>MEMINT</code> is located in <code>/etc/config/acct_config</code> .) This integral is not constant and varies with machine load. See Section 10.1.12.1, page 252, for more information.
<code>ac_io</code>	<p>Number of characters transferred by the process. If any tape accounting information existed for this process, the number of tape bytes read and written is included in the <code>ac_io</code> field. Thus, the amount of tape I/O is reported twice: once in the <code>ac_io</code> field and again in the tape accounting record. The <code>ac_io</code> field generally is larger, because it includes additional I/O performed by the process. This number is repeatable.</p> <p>Device accounting I/O information is also reported twice: by <code>ac_io</code> and in the device accounting record field <code>acd_ioch</code>.</p> <p>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 <code>/etc/config/acct_config</code>. The tape SBUs are labeled <code>TAPE_SBU x</code>, and the device SBUs are <code>BLOCK_DEVICE x</code> and <code>CHAR_DEVICE x</code>.</p> <p>Set the weight factors relative to <code>P_BYTEIO</code> (see Section 10.1.10.1.8, page 236, and Section 10.1.10.1.9, page 237). The <code>ac_io</code> value is multiplied by <code>P_BYTEIO</code>. The tape or device I/O value is multiplied by the appropriate tape or device weight factor.</p> <p>For example, if a surcharge is to be applied to tape I/O, the read and write values for the <code>TAPE_SBU x</code> variables must reflect the amount over <code>P_BYTEIO</code> that should be charged.</p>

<code>ac_rw</code>	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.
<code>ac_iowtime</code>	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.
<code>ac_iowmem</code>	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.
<code>ac_iosw</code>	Swap count. This number may vary due to system load.
<code>ac_lio</code>	Logical I/O request count; this is a count of the <code>read</code> , <code>write</code> , <code>reada</code> , <code>writea</code> , and <code>listio</code> (list entries) system calls made. This number is repeatable.
<code>ac_pid</code>	Process ID.
<code>ac_ppid</code>	Parent process ID.
<code>ac_ctime</code>	Process raw connect time in clocks.
<code>ac_acid</code>	Account ID.
<code>ac_jobid</code>	Job ID.
<code>ac_nice</code>	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. The 1 second of system and user time is calculated from the billable time, not from the time actually connected. Therefore, because billable time is calculated each time a process is rescheduled, or one time per second, the nice value becomes fixed some time in the interval from 1 to 2 seconds of connect time. (Connect time includes semaphore

	wait, except on CRAY C90 systems; billable time does not include semaphore wait.)
ac_comm	Command name (first 8 characters).
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_himem	Memory-use high water mark in words.
ac_sctime	System call time in clocks.

10.1.11.2 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:

<u>Field</u>	<u>Description</u>
ace_jobid	Job ID of the job to which this record belongs.
ace_uid	User ID from the job table.
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.
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.
ace_nice	Last nice value of the job.
ace_fsblkused	Sum of the file system storage used. This value may be negative if more space was freed up than was consumed.
ace_etime	End time of the job (in seconds).

10.1.11.3 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:

<u>Field</u>	<u>Description</u>
<code>ac_smwtime</code>	(Not on CRAY C90 systems) Semaphore wait time (in clocks).
<code>ac_mutime[MUSIZE]</code>	Time connected to ($i + 1$) CPUs (in compressed 1/100ths of a second format). 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 <code>pacct</code> file without changing the size of the records.

10.1.11.4 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:

<u>Field</u>	<u>Description</u>
<code>acs_mem</code>	Memory integral based on residency time, not execution time (in click-ticks).
<code>acs_lio</code>	Logical I/O request count; this count is the number of <code>ssread</code> and <code>sswrite</code> system calls made.
<code>acs_ioch</code>	Number of characters transferred to and from the SDS, stored in bytes.

10.1.11.5 MPP accounting record

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

<u>Field</u>	<u>Description</u>
<code>ac_mpppe</code>	Number of MPP processor elements used.
<code>ac_mppbe</code>	Number of MPP barrier bits used.

<code>ac_mpptime</code>	Number of clocks that the MPP has been used.
-------------------------	--

10.1.11.6 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:

<u>Field</u>	<u>Description</u>
<code>acp_rtime</code>	The process start time offset (in clocks) from the previous second (reported in the <code>ac_btime</code> field of the base accounting record). This field allows you to trace start times of processes that are spawned in the same second.
<code>acp_tiovertime</code>	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.
<code>acp_srunwtime</code>	This field is currently disabled.
<code>acp_swapclocks</code>	The time (in clocks) that a process spends on the swap device.
<code>acp_rwblks</code>	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.
<code>acp_phrwblks</code>	The number of physical blocks transferred by the process using the raw I/O interface.

10.1.12 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.

10.1.12.1 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.

10.1.12.2 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_muptime[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.

10.1.13 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.

10.1.14 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.

10.1.14.1 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.

10.1.14.2 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 10.1.10.1.9, page 237, 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:

<u>SBU</u>	<u>Description</u>
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.

10.1.14.3 Configuration changes

The system is released with the character devices configured to match the released configuration; any changes to /usr/src/uts/cl/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.

10.1.14.4 System header files

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

10.1.14.4.1 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.

10.1.14.4.2 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:

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

10.1.14.5 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
```


10.1.14.5.1 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.

10.1.14.5.2 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"
```

10.1.15 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.

10.1.16 Logging information

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

10.1.16.1 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.

10.1.16.2 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_time rc=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
```

10.1.16.3 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
```

10.1.16.4 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
```

10.1.16.5 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 + tty025 \n
user1-root SU 06/19 15:14 + tty021 user2-adm SU 06/19 15:19 - tty026 \n
user3-root SU 06/19 15:19 - tty022 user4-root
```

10.1.16.6 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
```

10.1.16.7 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
```

10.1.16.8 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*, Cray Research publication SD-1012, or the `olhpa(8)` man page for information concerning `olhpa`.²

10.1.16.9 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, `slugdemon(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

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.

the `reduce(8)` man page and *General UNICOS System Administration*, Cray Research publication SG-2301.

10.1.16.10 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/saDD` and can be accessed with `sar`.



Warning: The log files located in `/usr/adm/sa/saDD` 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.

10.1.16.11 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(s1) 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.

10.1.16.12 Accounting logs

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

<u>Log</u>	<u>Description</u>
csainfo	A file containing boot times. It can be accessed with the <code>od(1)</code> command (the <code>-d</code> option will give the seconds). Each time the system is booted, the boot time is written to <code>/etc/csainfo</code> by the <code>/etc/csaboosts</code> (see <code>csaboosts(8)</code>) command. <code>csaboosts</code> is invoked by <code>/etc/rc</code> (see <code>brcc(8)</code>). See also the description of the boot log in Section 10.1.16.1, page 260.
utmp	A file containing active system and terminal connection information. This log is used by <code>write(1)</code> , <code>who(1)</code> , <code>wall(8)</code> , and <code>mail(1)</code> in getting user information. It is located in <code>/etc/utmp</code> and can be accessed using the <code>who(1)</code> and <code>last(1B)</code> commands. It is written to by <code>init(8)</code> , <code>date(1)</code> , <code>login(1)</code> , and <code>getty(8)</code> . For information on the format of <code>utmp</code> , see <code>utmp(5)</code> .
	Warning: On a Cray ML-Safe configuration of the UNICOS system, <code>utmp</code> and <code>wtmp</code> are considered to be covert channels. You may want to consider restricting access to these files to the <code>adm</code> 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. <code>wtmp</code> must exist; programs that access it do not create it (the <code>/etc/rc</code> script creates <code>/etc/wtmp</code> by default). Records are in the form of <code>utmp(5)</code> ; <code>acctcon(8)</code> and <code>csaline(8)</code> convert <code>/etc/wtmp</code> into session and charging records. This data is merged into the system accounting reports. <code>wtmp</code> can also be accessed using the <code>who(1)</code> and <code>last(1)</code> commands. <code>wtmp</code> is written by <code>init(8)</code> , <code>date(1)</code> , <code>login(1)</code> , and <code>getty(8)</code> . For information on the format of <code>wtmp</code> , see <code>utmp(5)</code> .
pacct	Files containing per-process accounting data; these are located in <code>/usr/adm/acct/day/pacct*</code> and can be accessed using the <code>acctcom(1)</code> command. Note that these files are read by system accounting programs, and the information appears in the accounting reports. <code>pacct</code> is written by the kernel, and its format is described in <code>/usr/include/sys/acct.h</code> .



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:

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

10.1.16.13 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.
```