Dynamic Memory Management [8]

The loader supports the following two types of dynamic memory management:

- The HEAP, STACK, and TSTACK directives let you use dynamic memory managed by the system heap routines. The ADDBSS directive lets you expand the initial size of your program and reserve space for later memory expansion.
- The DYNAMIC directive lets you specify a common block that can be expanded or contracted at your discretion.

You can use either one or both of these schemes in a single program.

All directives in this section are global directives.

Managing global heap memory
8.1

The HEAP, STACK, and TSTACK directives let you control the size and location of the system-managed heap and stack. Memory space can be acquired from the heap by use of the system heap routines. Under the UNICOS operating system, the heap is always present and resides after the longest segment branch of your program. Heap space is available to all segments of your program.

HEAP *directive*
8.1.1

The HEAP directive allocates memory that the heap manager can manage dynamically. All memory requests are satisfied with space from a common heap. The HEAP directive allows the memory use within a job to increase.

The heap is located in memory following the segment tree branch that occupies the largest amount of memory. HEAP has the same effect on both segmented and nonsegmented programs.

Format:

```
HEAP=[init][+inc]
```

init Initial number of decimal words available to the heap manager; the default is specific to each system.

inc Increment size, in decimal words, of a request to the operating system for additional memory if the heap overflows.

A value of 0 indicates that heap size is fixed. If you specify the DYNAMIC directive, the loader ignores an increment size other than 0. The default is specific to each system.

Command-line equivalent: `-H` option

STACK *directives* 8.1.2

The STACK directive allocate part of heap memory to a stack for use by re-entrant programs. When you use STACK, the HEAP directive is not needed unless you want to change the default heap values.

The STACK directive is intended for use by individual users to set the stack size for their programs. The following paragraphs outline the steps the loader takes in determining a program's stack size:

1. If a STACK directive has been used, the initial value specified with the STACK directive becomes the program's initial stack size.
2. If no STACK directive is present, the loader analyzes the module calling structure of the program. It estimates what the stack requirements of the program will be. Run-time characteristics of the program, such as regression or indirectly invoked procedures, can cause the estimate to be inaccurate. The loader may underestimate the stack requirements needed for execution. The loader rarely overestimates program stack requirements.

3. If a DEFSTACK directive, see page 118, has been encountered, the loader will compare the estimated size with the initial value specified with the DEFSTACK directive. The larger of the two values will be used as the initial stack size of the program.
4. If no DEFSTACK directive is present, the loader will use the estimated value as the initial stack size.

Format:

```
STACK=[init][+inc]
```

init Initial size, in decimal words, of a stack. If *init* is less than or equal to 128 words or is absent, an installation-defined value is used.

inc Size, in decimal words, of additional increments to a stack if the stack overflows. A value of zero (0) implies that stack overflow is prohibited. An installation-defined value defines the default increment value.

Command-line equivalent: -S option

The use of more than one of the STACK, HEAP, and FREEHEAP directives can easily result in an inconsistent specification. If this occurs, the maximum size heap is used.

TSTACK *directive*
8.1.3

Multitasked programs often have more extensive stack requirements than unitasked programs. Slave tasks often require a different amount of stack space than the main program task. The TSTACK directive allows you to specify a stack size to be used whenever slave tasks are initiated. In the absence of a TSTACK directive, the loader estimates the amount of stack space required for the slave tasks by using the same algorithm that is used to estimate main program stack size.

Format:

```
TSTACK=init[+inc]
```

init Initial size, in decimal words, of the stack space assigned to each slave task when the slave task begins execution.

inc Size, in decimal words, of additional increments to a stack if the stack overflows. A value of zero (0) implies that stack overflow is prohibited. An installation-defined value defines the default increment value.

ADDBSS *directive*
8.1.4

The ADDBSS directive tells the loader to expand the initial size of your program. This provides preallocated space for later requests of the program to expand its heap space.

Format:

```
ADDBSS=value
```

value The number of 1024-word blocks of space to add to the uninitialized data area of your program.

Command-line equivalent: `-b` option

DYNAMIC *directive*
8.1.5

The DYNAMIC directive specifies the common block that can expand or contract under your control. You must call the system routines to expand your program size before referencing the portions of the dynamic common block not initially allocated to your program. The common block occupies memory following the largest segment, and all segments have access to it at any time during program execution. The contents of the dynamic common block may not be declared at load or compile time (data loaded).

Format:

```
DYNAMIC=comblk|//
```

comblk Allocates the specified common block to the first word following the longest segment branch. Only one common block can be specified.

`//` Specifies the blank common block as dynamic.

If no HEAP is required, blank common is always dynamic (default); otherwise, there is no default dynamic common block.

If you expand a common block that is not the dynamic common block, you may overwrite a segment in memory, or, when the loader brings in the successor segment, the loader may overwrite the common block. Use the dynamic common block instead.

Example:

CFT program

```
PROGRAM X
COMMON /DYNCOM/ SPACE(1)
.               In this user-supplied code, the
.               user requests 9999 additional
.               words of memory.
DO 100 I=1,10000
   SPACE(I)=0   This code zeroes out 10,000 words,
.               but only 1 word is actually
.               preallocated by the loader.
100 CONTINUE
```

SEGLDR directive

```
DYNAMIC=DYNCOM  Identifies /DYNCOM/ as the dynamic
                common block.
```

Using the heap and dynamic common together

8.2

You can use the heap and dynamic common together in a program if you are careful to adhere to the following guidelines: When both the heap and dynamic common are used, the heap begins immediately after the longest segment branch of your program, and it has a fixed size. No expansion of the heap is allowed. The dynamic common block begins after the heap, and it can expand.

Because the heap cannot expand, the initial size assigned to it must be large enough to accommodate all requests for heap space. This is critically important under the UNICOS operating system because many system library routines request heap space to perform their functions. In general, the initial size of the heap should be at least 5000 words.

The following examples use several system routines for memory management. Additional memory management routines are also available. If you require further information about any of the library routines used in these examples, consult the library manual appropriate to the language and operating system.

Fortran example for acquiring space from the heap

8.2.1

The following is an example of a Fortran program that acquires a 1-Mword block of heap space. You do not need loader directives, but you may use some to set heap values to something other than their defaults.

```
    PROGRAM USEHEAP
    INTEGER SPACE(0:0), ERRCODE, INDEX
    POINTER (SPTR,SPACE)

    CALL HPALLOC (SPTR, 1000000, ERRCODE, 0)
    IF (ERRCODE .EQ. 0) THEN
        DO 1 INDEX = 0, 999999
            SPACE(INDEX) = INDEX

1       CONTINUE
    ENDIF
    END
```

Fortran example for using dynamic common 8.2.2

The following is an example of a Fortran program that runs under the UNICOS operating system and sets up a dynamic common block of 1 million words. The example requires the use of SBREAK, a Fortran interface to the system library routine `sbreak`, documented in the *UNICOS System Calls Reference Manual*, publication SR–2012. SBREAK expands the field length of the program for the additional space. For this example, you also need the two loader directives: `DYNAMIC=DYNCOM`, to identify the dynamic common block, and `HEAP=10000+0`, to set up a heap size large enough and to indicate that it cannot expand. Both of these directives are described in this manual.

```
      PROGRAM USEDYN
      COMMON /DYNCOM/ SPACE(1)
      INTEGER SBREAK, ERRCODE
      . Only one word of space is preallocated to the program.
      . The user must call the system library routine SBREAK to
      . expand the program's field length and to acquire the
      . additional space.
      ERRCODE=SBREAK(1000000)
      IF (ERRCODE .GE. 0) THEN
            DO 100 I=1,1000000
                  SPACE(I) = 0.0

100         CONTINUE
      ENDIF
      END
```