System V Shared Memory Support for HP C Run-Time Library for OpenVMS Systems

This document provides reference information about the HP C Run-Time Library (RTL) functions. Also, it supports the following Open Group System V Shared Memory routines:

shmget()
shmctl()
shmat()
shmdt()

System V Shared Memory Limitations

Following are the currently supported limits in System V Shared Memory:

- Maximum number (SHMMNI) of System V Shared Memory segments allowed in a system is 1024.
- Maximum size (SHMMAX) of System V Shared Memory segment allowed is 512 MB.
- Minimum size (SHMMIN) of System V Shared Memory segment allowed is 1 byte. (Note: The global section mapped for System V Shared Memory segment is a multiple of 8 KB. For example, System V Shared Memory segment of size 1 byte will have an 8 KB global section.)

Prerequisites

1. You must have a C RTL ECO kit installed on OpenVMS system running Version 8.4 and UPDATE 600 kit for Alpha or Integrity servers.
2. System V Shared Memory interfaces uses file-backed global sections. These APIs either create or delete the files based on request. Current implementation creates the files in SYS$SPECIFIC:[DECC$SYSV_SHM] directory. System Manager or Administrator must create the directory using the following command before using the Shared Memory APIs:

   ```
   $ CREATE /DIRECTORY SYS$SPECIFIC:[DECC$SYSV_SHM]  -
           /OWNER_UIC = [SYSTEM]  -
            /PROTECTION = (S:RWE, O:RWE, G:RWE, W:RWE)
   ```

   Note: if the SYS$SPECIFIC:[DECC$SYSV_SHM] directory does not exist, the current implementation uses the SYS$SPECIFIC:[PSX$SEMAPHORES] directory. This directory is created during OpenVMS Version 8.4 installation or upgrade procedure.

Restrictions

1. If an application pass its valid non-zero virtual address to **shmat()** API function, the current implementation expects this address to be a 32 bit virtual address; a 64 bit virtual address is not supported.
2. System V Shared Memory APIs are implemented using the file-backed global sections; hence each shared memory segment created would internally create a file to back the data for the section in system device (SYS$SYSDEVICE), and shared memory control operation with IPC_RMID deletes the file. So, the system administrator must provide sufficient disk space on SYS$SYSDEVICE based on the usage of shared memory segments.
3. Performance of these APIs depends on the size of segments; for example creating a file of size 100 MB takes approximately 10 to 15 seconds, so the application using these APIs may experience delay when they are used for the first time. If the same segments are used frequently without IPC_RMID, then this delay will not be seen because with no IPC_RMID in use, the file will not be deleted, and thus will be a persistent file until such time that it is deleted.
4. Current release guarantees the initialization of shared memory segments with zero values only when the following conditions are true:
   a. When High-Water Marking is enabled on the SYS$SYSDEVICE device.

      AND

   b. If application is using $ERAPAT (erase pattern) system service, the erase pattern generated for disk storage type must be zero.

   In all other conditions, the initial contents of each shared memory segment created may or may not be initialized with zero values.

Note

For the following error cases, you must increase the GBLSECTIONS SYSGEN parameter:

1. If **shmget()** API function returns ENOMEM (value = 12) or ENOSPC (value = 28) .
2. If **shmat()** API function returns ENOMEM (value = 12),

ENOMEM indicates 'Not enough core' and ENOSPC indicates ’no space left on device’.

Shared memory internally uses global sections and hence a system having a large number of shared memory segments may result in exhaustion of GBLSECTIONS.

Shared Memory API Description:

---

shmget *(Integrity servers, Alpha)*

Gets a shared memory segment.

Format

#include <shm.h>

int shmget(key_t *key*, size_t *size*, int *shmflg*);

Argument

key
The key for which the associated shared memory identifier is returned.
size
Shared memory segment size in bytes.
shmflg
Flag used to initialize the low order 9 bits of the `shm_perm.mode` member of the `shmid_ds` data structure associated with the new shared memory segment.
For more information, see the Description.

Description

The **shmget()** API function returns the shared memory identifier associated with the *key*.

A shared memory identifier with its associated `shmid_ds` data structure is created for a *key* if one of the following is TRUE:

- The key argument is equal to IPC_PRIVATE.
- The key argument does not already have a shared memory identifier associated with it and (**shmflg** and IPC_CREAT) is non-zero.

If **shmflg** specifies both IPC_CREAT and IPC_EXCL and a shared memory segment already exists for a *key*, **shmget() API** function fails with *errno* set to EEXIST.

When it is created, the `shmid_ds` data structure associated with the new shared memory identifier is initialized as follows:

- The values of *shm_perm.cuid, shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low order 9 bits of *shm_perm.mode* are set equal to the low order 9 bits of the *shmflg* argument.
- The variable *shm_segsz* is set equal to the value of the *size* argument.
- The variables *shm_lpid, shm_nattch, shm_atime,* and *shm_dtime* are set equal to zero, the variable *shm_cpid* is set equal to the Process ID of the segment creator, and the variable *shm_ctime* is set equal to the current time.

Return Values

n	Successful completion. The function returns a non-negative integer shared memory identifier.
-1	Indicates an error. The function sets *errno* to one of the following values:

- EACCES – A shared memory identifier exists for *key*, but operation permission as specified by the low order 9 bits of *shmflg* are not granted.

- EEXIST – A shared memory identifier exists for a key but ((*shmflg* & IPC_CREAT) && (*shmflg* & IPC_EXCL)) is non-zero.

- EINVAL – The value of *size* is either less than SHMMIN, greater than the SHMMAX, or a shared memory identifier exists for a *key*, but the *size* is greater than the size of that segment. With current implementation SHMMIN is defined to 1 byte and SHMMAX is defined to 512 MB.

- ENOENT – A shared memory identifier does not exist for a *key* and (*shmflg* & IPC_CREAT) is equal to zero.

- ENOSPC – A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory segments system-wide are exceeded.

- EVMSERR – OpenVMS specific non-translatable error code.

shmctl *(Integrity servers, Alpha)*

Shared memory control operations.

Format

#include <shm.h>

int shmctl(int *shmid*, int *cmd*, struct shmid_ds **buf*);

Argument

shmid
A shared memory identifier, a positive integer. It is created by the **shmget()** API function and used to identify the shared memory segment on which to perform the control operation.

cmd
The control operation (IPC_STAT, IPC_SET, or IPC_RMID) to perform on the shared memory segment. For more information, see the Description.

buf
Pointer to a *shmid_ds* structure, defined in <shm.h> as follows:

```
struct shmid_ds {
        struct   ipc_perm shm_perm;     /* permissions structure defined in <ipc.h> */
        size_t   shm_segsz;             /* size of memory segment in bytes */
        pid_t    shm_lpid;              /* Process ID of last memory operation */
        pid_t    shm_cpid;              /* Process ID of segment creator */
        shmatt_t  shm_nattch;           /* Number of current attaches */
        time_t  shm_atime;              /* time of last shmat  */
        time_t  shm_dtime;              /* time of last shmdt  */
        time_t  shm_ctime;              /* time of last change  */
        };
```

Description

The **shmctl()** API function provides a variety of shared memory control operations as specified by the *cmd* argument. *cmd* can have the following values:

IPC_STAT
Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*. The contents of this structure are defined in <shm.h>.

IPC_SET
Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

shm_perm.uid
shm_perm.gid
shm_perm.mode */* only low 9 bits */*

This *cmd* can be executed by a process that has an effective user ID equal to either that of a user having appropriate privileges or to the value of either *shm_perm.uid* or *shm_perm.cuid* in the data structure associated with *shmid*.

IPC_RMID
Remove the shared memory identifier specified by *shmid* from the system and delete the shared memory segment and data structure associated with it. If the segment is attached to one or more processes, the segment *key* is changed to IPC_PRIVATE and the segment is marked as removed. The segment disappears when the last attached process detaches it. This *cmd* can be executed by a process that has an effective user ID equal to either that of a user with appropriate privileges or to the value of either *shm_perm.uid* or *shm_perm.cuid* in the data structure associated with *shmid*.

Return Values

0	Successful completion.
-1	Indicates an error. The function sets *errno* to one of the following values:

- EACCES – The argument *cmd* is equal to IPC_STAT and the calling process does not have read permission.

- EINVAL – The value of *shmid* is not a valid shared memory identifier, or the value of *cmd* is not a valid command.

- EFAULT - The argument *cmd* has value IPC_SET or IPC_STAT but the address pointed to by *buf* is not accessible.

- EPERM - The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in the data structure associated with *shmid*.

- EVMSERR – OpenVMS specific non-translatable error code.

shmat *(Integrity servers, Alpha)*

shared memory attach operation.

Format

#include <shm.h>

void *shmat(int *shmid*, const void **shmaddr*, int *shmflg*);

Argument

shmid
A shared memory identifier, a positive integer. It is created by the **shmget()** API function and used to identify the shared memory segment on which to perform the shared memory attach operation.

shmaddr
Address within the calling process to which the shared memory segment has to be attached. For more information, see the Description.

shmflg
Flag to indicate type of attach operation. For more information, see the Description.

Description

The shmat function attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the address space of the calling process.

The segment is attached at the address specified by one of the following criteria:

- If *shmaddr* is a null pointer, the segment is attached at the first available address as selected by the system. This is the preferred method of using shmat.

- If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is non-zero, the segment is attached at the address given by (*shmaddr* -((__int64)*shmaddr % SHMLBA)).
  The character '%' is the C language remainder operator.

• If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is zero, the segment is attached at the address given by *shmaddr*.

• The segment is attached for reading if (*shmflg* & SHM_RDONLY) is non-zero and the calling process has read permission; otherwise, if it is zero and the calling process has both read and write permission, the segment is attached for reading and writing.

A successful **shmat()** API function updates the members of the *shmid_ds* structure associated with the shared memory segment as follows:

- *shm_atime* is set to the current time.
- *shm_lpid* is set to the process ID of the calling process.
- *shm_nattch* is incremented by one.

It is possible to attach a shared memory segment even if it is already marked for deletion, if it has a valid *shmid*.

If *shmaddr* is not a null pointer and SHM_RND flag is specified in *shmflg*, the attaching address is calculated as (*shmaddr* -((__int64)*shmaddr* % SHMLBA)).

Note:

1. For the current implementation the SHMLBA value is 8 KB.
2. If the application passes its own valid non-zero address, the current implementation requires this address to be 32 bit; a 64 bit virtual address is not supported.

Return Values

x	Successful completion. The function returns the start address of the attached shared memory segment.
-1	Indicates an error. The function sets *errno* to one of the following values: • EACCES - Operation permission is denied to the calling process. • EINVAL - The value of *shmid* is not a valid shared memory identifier; or the *shmaddr* is not a null pointer, and the value of (*shmaddr* – ((__int64)*shmaddr* % SHMLBA)) is an illegal address for attaching shared memory; or the *shmaddr* is not a null pointer, (*shmflg* & SHM_RND) is zero, and the value of *shmaddr* is an illegal address for attaching shared memory.

- EMFILE - The number of shared memory segments attached to the calling process exceeds the system imposed limit.

- ENOMEM - The available data space is not large enough to accommodate the shared memory segment.

- EVMSERR – OpenVMS specific non-translatable error code.

shmdt *(Integrity servers, Alpha)*

shared memory detach operation.

Format

#include <shm.h>

int shmdt(const void **shmaddr*);

Argument

shmaddr
The address returned by a previous call to **shmat**.

Description

The **shmdt()** API function detaches the shared memory segment located at the address specified by *shmaddr* from the address space of the calling process. The to-be-detached segment must be currently attached with *shmaddr* equal to the value returned by the attaching **shmat()** API function.

On a successful **shmdt()** API function the system updates the members of the *shmid_ds* structure associated with the shared memory segment as follows:

- *shm_dtime* is set to the current time.
- *shm_lpid* is set to the process-ID of the calling process.
- *shm_nattch* is decremented by one. If it becomes zero and the segment is marked for deletion, the segment is deleted. For more information see the **shmctl** function

Upon exit, all the attached shared memory segments are detached from the process.

Return Values

0	Successful completion.
-1	Indicates an error. The function sets errno to one of the following values: • EINVAL - The value of *shmaddr* is not the data segment start address of a shared memory segment. • EVMSERR – OpenVMS specific non-translatable error code.

Example

```
/*
  Abstract: This test program creates a Shared Memory Segment and write
        some data and reads the same data after attaching again.
        And also it ensures that an EINVAL error is generated when
        we try to attach a memory that has already been deleted.
*/
#include <errno.h>
#include <types.h>
#include <ipc.h>
#include <shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SHMSZ  128
#define SHMKEY 9229

main()
{

   int shmid;
   char *shm_write;
   char *shm_read;

   printf("Creating Shared Memory Segment\n");
   if ((shmid = shmget(SHMKEY, SHMSZ, IPC_CREAT | 0666)) < 0)
   {
```

```
      perror("shmget");
      printf("\nshmget(): FAILED\n");
   }
   if ((shm_write = shmat(shmid, NULL, 0)) == (char *) -1)
   {
      perror("shmat");
      printf("\nshmat(): FAILED\n");
      return;
   }
   printf("Writing Data to the Created Shared Memory Segment\n\n");
   memcpy(shm_write, "Test Shared Memory Segment", SHMSZ);
   printf("Detaching Shared Memory Segment\n");
   if( shmdt(shm_write)<0)
      perror("shmdt");

   printf("Attach again to read the data from Shared Memory Segment\n\n");
   if ((shm_read = shmat(shmid, NULL, 0)) == (char *) -1)
   {
      perror("shmat");
      printf("\nshmat(): FAILED\n");
   }
   printf("Reding Data from Shared Memory Segment\n");
   printf("Data in Segment is: %s\n\n",shm_read);

   printf("Detaching Shared Memory Segment\n");
   if( shmdt(shm_read)<0)
      perror("shmdt");
   printf("Deleting Shared Memory Segment using IPC_RMID\n\n");
   if( shmctl(shmid, IPC_RMID, NULL)<0)
            perror("shmctl");

   printf("Attaching to the deleted Shared Memory Segment - error EINVAL should be generated\n\n");
   if ((shm_write = shmat(shmid, NULL, 0)) == (char *) -1)
   {
      perror("shmat");
   }
}
```

This example produces the following output:

```
Creating Shared Memory Segment
```

Writing Data to the Created Shared Memory Segment

Detaching Shared Memory Segment
Attach again to read the data from Shared Memory Segment

Reding Data from Shared Memory Segment
Data in Segment is: Test Shared Memory Segment

Detaching Shared Memory Segment
Deleting Shared Memory Segment using IPC_RMID

Attaching to the deleted Shared Memory Segment - error EINVAL should be generated

shmat: invalid argument