BAHIR DAR UNIVERSITY

FACULTY OF COMPUTING DEPARTMENT OF SOFTWARE ENGINEERING

Individual assignment on "Operating System and System Programing"

Name **ID**

HENOK MEKOYET----------BDU1307773

Submitted to lecturer Wendmu B.

Submission Date June 24/2022

Contents

1. What / Why / How, this system call?

A system call is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS. System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

In this paper we will focus on shmget(2) system call in detail

the shmget() system call is used In order to create a new message queue, or access an existing queue.

SYSTEM CALL: shmget();

PROTOTYPE: int shmget (key_t key, int size, int shmflg);
RETURNS: shared memory segment identifier on success
-1 on error: errno = EINVAL (Invalid segment size specified)
EEXIST (Segment exists, cannot create)
EIDRM (Segment is marked for deletion, or was removed)
ENOENT (Segment does not exist)
EACCES (Permission denied)
ENOMEM (Not enough memory to create segment)

Shmget() stands for **shared memory segment**. It is mainly used for Shared memory communication. This system call is used to access the shared memory and access the messages in order to communicate with the process.

It is the fastest form of inter process communication. The main advantage of shared memory is that the copying of message data is eliminated.

shmget() returns the identifier of the System V shared memory segment associated with the value of the argument *key*. It may be used either to obtain the identifier of a previously created shared memory segment (when *shmflg* is zero and *key* does not have the value **IPC_PRIVATE**), or to create a new set.
A new shared memory segment, with size equal to the value of *size* rounded up to a multiple of **PAGE_SIZE**, is created if *key* has the value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no shared memory segment corresponding to *key* exists, and **IPC_CREAT** is specified n *shmflg*.

If *shmflg* specifies both **IPC_CREAT** and **IPC_EXCL** and a shared memory segment already exists for *key*, then **shmget**() fails with *errno* set to **EEXIST**. (This is analogous to the effect of the combination **O_CREAT | O_EXCL** for open(2).)

2.Briefly describe about the list of parameters and flag?

The **shmget()** function either creates a new shared memory segment or returns the shared memory identifier associated with the *key* parameter for an existing shared memory segment. A new shared memory segment is created if one of the following conditions is met:

- The *key* parameter is equal to IPC_PRIVATE.

- The *key* parameter does not have a shared memory identifier associated with it and the IPC_CREAT flag is specified in the *shmflg* parameter.

Parameters

key

(Input) The key associated with the shared memory segment. A key of IPC_PRIVATE (0x00000000) guarantees that a unique shared memory segment is created. A key can also be specified by the caller or generated by the ftok() function.

size

(Input) The size of the shared memory segment being created. If an existing shared memory segment is being accessed, *size* may be zero.

shmflg

(Input) Operation and permission flags. The value of the *shmflg* parameter is either zero or is obtained by performing an OR operation on one or more of the following constants:

S_IRUSR (0x00000100)
Allow the owner of the shared memory segment to attach to it in read mode.
S_IWUSR (0x00000080)
Allow the owner of the shared memory segment to attach to it in write mode.
S_IRGRP (0x00000020)
Allow the group of the shared memory segment to attach to it in read mode.
S_IWGRP (0x00000010)
Allow the group of the shared memory segment to attach to it in write mode.
S_IROTH (0x00000004)
Allow others to attach to the shared memory segment in read mode.
S_IWOTH (0x00000002)
Allow others to attach to the shared memory segment in write mode.
IPC_CREAT (0x00000200)
Create the shared memory segment if it does not exist.
IPC_EXCL (0x00000400)
Return an error if the IPC_CREAT flag is set and the shared memory segment already exists.
SHM_TS_NP (0x00010000)
If creating a new shared memory segment, make the shared memory segment a teraspace shared memory segment. When a job attaches to the shared memory segment, the shared memory segment will be added to the job's teraspace. Some compilers permit the user to indicate that the teraspace versions of storage functions should be used. For example, if a C module is compiled using CRTCMOD TERASPACE(*YES *TSIFC), this flag will be set automatically.

If accessing an existing shared memory segment, only specify this constant if it was specified when the shared memory segment was created.

SHM_RESIZE_NP (0x00040000)

If creating a new teraspace shared memory segment, allow the size of the shared memory segment to be changed with the shmctl() and shmctl64() functions. The maximum size of this teraspace shared memory segment is 268 435 456 bytes (256 MB). This flag is ignored for nonteraspace shared memory segments. A nonteraspace shared memory segment may always be resized up to 16 773 120 bytes (16 MB minus 4096 bytes).

SHM_MAP_FIXED_NP (0x00100000)

If creating a new teraspace shared memory segment, make all jobs that successfully attach to the shared memory segment attach to the shared memory segment at the same address. The shared memory segment may not be attached in read-only mode. This flag is ignored for nonteraspace shared memory segments.

If accessing an existing shared memory segment, only specify this constant if it was specified when the shared memory segment was created.

The value shmflg is composed of: **IPC_CREAT** Create a new segment. If this flag is not used, then shmget () will find the segment associated with key and check to see if the user has permission to access the segment. IPC_EXCL This flag is used with IPC_CREAT to ensure that this call creates the segment.

If IPC_CREAT is used alone, shmget() either returns the segment identifier for a newly created segment, or returns the identifier for a segment which exists with the same key value.

If IPC_EXCL is used along with IPC_CREAT, then either a new segment is created, or if the segment exists, the call fails with -1. IPC_EXCL is useless by itself, but when combined with IPC_CREAT, it can be used as a facility to guarantee that no existing segment is opened for access.

3.. List the flags, their purpose with code implementation (give Example source code with output)

The value *shmflg* is composed of:

IPC_CREAT

Create a new segment. If this flag is not used, then **shmget**() will find the segment associated with *key* and check to see if the user has permission to access the segment.

IPC_EXCL

This flag is used with **IPC_CREAT** to ensure that this call creates the segment. If the segment already exists, the call fails.

SHM_HUGETLB (since Linux 2.6)

Allocate the segment using "huge" pages.

SHM_HUGE_2MB, **SHM_HUGE_1GB** (since Linux 3.8)

Used in conjunction with **SHM_HUGETLB** to select alternative hugetlb page sizes (respectively, 2 MB and 1 GB) on systems that support multiple hugetlb page sizes.

More generally, the desired huge page size can be configured by encoding the base-2 logarithm of the desired page size in the six bits at the offset **SHM_HUGE_SHIFT**. Thus, the above two constants are defined as:

```
#define SHM_HUGE_2MB    (21 << SHM_HUGE_SHIFT)
#define SHM_HUGE_1GB    (30 << SHM_HUGE_SHIFT)
```

SHM_NORESERVE (since Linux 2.6.15)

This flag serves the same purpose as the mmap(2) MAP_NORESERVE flag. Do not reserve swap space for this segment. When swap space is reserved, one has the guarantee that it is possible to modify the segment. When swap space is not reserved one might get SIGSEGV upon a write if no physical memory is available.

In addition to the above flags, the least significant 9 bits of *shmflg* specify the permissions granted to the owner, group, and others. These bits have the same format, and the same meaning, as the *mode* argument of open(2). Presently, execute permissions are not used by the system.

When a new shared memory segment is created, its contents are initialized to zero values, and its associated data structure, *shmid_ds* (see shmctl(2)), is initialized as follows:

- *shm_perm.cuid* and *shm_perm.uid* are set to the effective user ID of the calling process.
- *shm_perm.cgid* and *shm_perm.gid* are set to the effective group ID of the calling process.
- The least significant 9 bits of *shm_perm.mode* are set to the least significant 9 bit of *shmflg*.
- *shm_segsz* is set to the value of *size*.
- *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set to 0.

- *shm_ctime* is set to the current time.

If the shared memory segment already exists, the permissions are verified, and a check is made to see if it is marked for destruction.

Normally, IPC_CREAT | IPC_EXCL is used if you want to create and initialize a new memory block. E.g.:

```
int shmid = shmget(key, sizeof(struct messageQueue), IPC_CREAT | S_IRWXU |
IPC_EXCL);
if( shmid != -1 )
{
   /* initialization code */
}
/* if it already exists, open it: */
if( shmid == -1 && errno == EEXIST )
   shmid = shmget(key, sizeof(struct messageQueue), S_IRWXU );

if( shmid == -1 ) {
   perror("shmget");
}
```

If you don't need to initialize it, you can skip the IPC_EXCL:

```
int shmid = shmget(key, sizeof(struct messageQueue), IPC_CREAT | S_IRWXU );
```

and if you don't need to create it, you can skip the IPC_CREAT:

```
int shmid = shmget(key, sizeof(struct messageQueue), S_IRWXU );
```

Let's create a wrapper function for locating or creating a shared memory segment :

```
int open_segment( key_t keyval, int segsize )
{
        int     shmid;

        if((shmid = shmget( keyval, segsize, IPC_CREAT | 0660 )) == -1)
        {
                return(-1);
        }

        return(shmid);
}
```

RETURN VALUE

On success, a valid shared memory identifier is returned. On error, -1 is returned, and *errno* is set to indicate the error.

ERRORS

EACCES The user does not have permission to access the shared memory segment, and does not have the **CAP_IPC_OWNER** capability in the user namespace that governs its IPC namespace.

EEXIST IPC_CREAT and **IPC_EXCL** were specified in *shmflg*, but a shared memory segment already exists for *key*.

EINVAL A new segment was to be created and *size* is less than **SHMMIN** or greater than **SHMMAX**.

EINVAL A segment for the given *key* exists, but *size* is greater than the size of that segment.

ENFILE The system-wide limit on the total number of open files has been reached.

ENOENT No segment exists for the given *key*, and **IPC_CREAT** wasnot specified.

ENOMEM No memory could be allocated for segment overhead.

ENOSPC All possible shared memory IDs have been taken (**SHMMNI**), or allocating a segment of the requested *size* would cause the system to exceed the system-wide limit on shared memory (**SHMALL**).

EPERM The **SHM_HUGETLB** flag was specified, but the caller was not privileged (did not have the **CAP_IPC_LOCK** capability) and is not a member of the *sysctl_hugetlb_shm_group* group; see the description of */proc/sys/vm/sysctl_hugetlb_shm_group* in proc(5).

NOTES top

IPC_PRIVATE isn't a flag field but a *key_t* type. If this special value is used for *key*, the system call ignores all but the least significant 9 bits of *shmflg* and creates a new shared memory segment.

Shared memory limits

The following limits on shared memory segment resources affect the **shmget**() call:

SHMALL System-wide limit on the total amount of shared memory, measured in units of the system page size.

On Linux, this limit can be read and modified via */proc/sys/kernel/shmall*. Since Linux 3.16, the default

value for this limit is:

ULONG_MAX - 2^24

The effect of this value (which is suitable for both 32-bit and 64-bit systems) is to impose no limitation on allocations. This value, rather than **ULONG_MAX**, was chosen as the default to prevent some cases where historical applications simply raised the existing limit without first checking its current value. Such applications would cause the value to overflow if the limit was set at **ULONG_MAX**.

From Linux 2.4 up to Linux 3.15, the default value for this limit was:

SHMMAX / PAGE_SIZE * (SHMMNI / 16)

If **SHMMAX** and **SHMMNI** were not modified, then multiplying the result of this formula by the page size (to get a value in bytes) yielded a value of 8 GB as the limit on the total memory used by all shared memory segments.

SHMMAX Maximum size in bytes for a shared memory segment.
On Linux, this limit can be read and modified via */proc/sys/kernel/shmmax*. Since Linux 3.16, the default value for this limit is:
ULONG_MAX - 2^24
The effect of this value (which is suitable for both 32-bit and 64-bit systems) is to impose no limitation on allocations. See the description of **SHMALL** for a discussion of why this default value (rather than **ULONG_MAX**) is used.
From Linux 2.2 up to Linux 3.15, the default value of this limit was 0x2000000 (32 MB).
Because it is not possible to map just part of a shared memory segment, the amount of virtual memory places another limit on the maximum size of a usable segment: for example, on i386 the largest segments that can be mapped

have a size of around 2.8 GB, and on x86-64 the limit is around 127 TB.

SHMMIN Minimum size in bytes for a shared memory segment: implementation dependent (currently 1 byte, though **PAGE_SIZE** is the effective minimum size).

SHMMNI System-wide limit on the number of shared memory segments. In Linux 2.2, the default value for this limit was 128; since Linux 2.4, the default value is 4096. On Linux, this limit can be read and modified via */proc/sys/kernel/shmmni*.

The implementation has no specific limits for the per-process maximum number of shared memory segments (**SHMSEG**).