SISTEMAS OPERATIVOS

3004610 - 1

German Sánchez Torres, I.S., M.Sc., Ph.D.

Profesor, Facultad de Ingeniería - Programa de Sistemas

Universidad del Magdalena, Santa Marta.

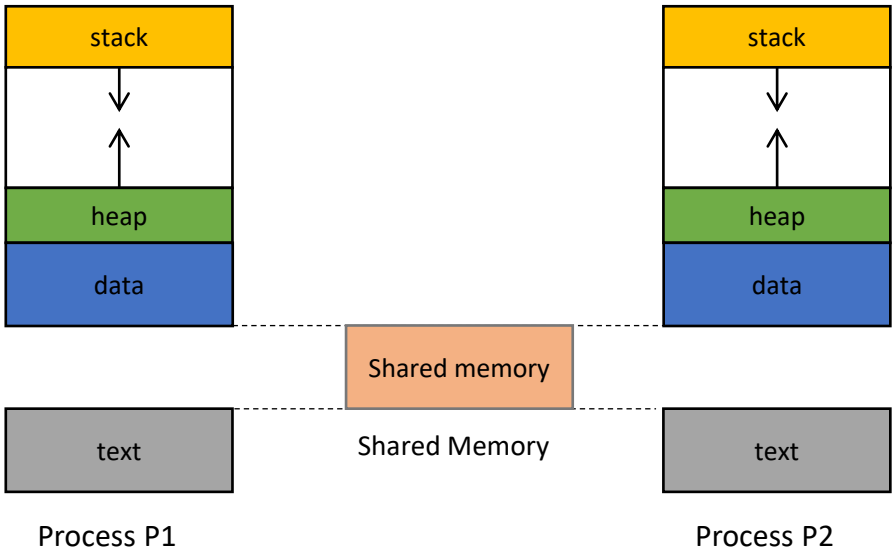
Phone: +57 (5) 4214079 Ext 1138 - 301-683 6593

Edificio Docente, Cub 3D401.

Email: sanchez.gt@gmail.com –gsanchez@unimagdalena.edu.co

Ejemplos Shared Memory (Unix system V) Shmget, shmat, shmdt, shmctl

Shared Memory System Calls



Shared Memory System Calls

1. Creating a Shared Memory Segment
   - Create - Allocated in byte amounts

Only one process

2. Shared Memory Operations and Use
   - Attach
   - Detach

All of the process

3. Shared Memory Control
   - Remove

Only one process

Shared Memory System Calls

1. Creating a Shared Memory Segment
   - Create - Allocated in byte amounts (shmget)
2. Shared Memory Operations and Use
   - Attach (shmat)
   - Detach (shmdt)
3. Shared Memory Control
   - Remove (shmctl – IPC_RMID)

shmget System Call

Function

- The shmget system call is used to create or access a shared memory segment.

Include: <sys/types.h> <sys/ipc.h> <sys/shm.h>

Command: ***int shmget (key_t key, int size, int shmflg);***

Returns: Success: unique shared memory identifier.

Failure: -1 ; Sets errno: Yes.

Arguments:

- ***key_t key***: key for creating or accessing shared memory
- ***int size***: size in bytes of shared memory segment to create. Use 0 for accessing an existing segment.
- ***int shmflg***: segment creation condition and access permission.

shmget(): Argument Values

key

- Use getuid() to make it unique or ftok (System V)

size

- number of bytes to allocate

shmflag

- Creating: IPC_CREAT and permissions
- 0 for accessing only

Defaults Values:

Maximum segment size	1,048,576 bytes (see */proc/sys/kernel/shmmax*)
Minimum segment size	1 byte

shmat System Call

Used to attach (map) the referenced shared memory segment into the calling process's data segment.

The pointer returned is to the first byte of the segment

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

Returns:

- Success:Reference to the data segment address of shared memory;
- Failure:-1; Sets errno:Yes.

Arguments:

- ***int shmid:*** a valid shared memory identifier.
- ***void *shmaddr***: allows the calling process some flexibility in assigning the location of the shared memory.
- ***int shmflg***: access permissions and attachment conditions.

shmdt System Call

The ***shmdt*** is used to detach the calling process's data segment from the shared memory segment.

int shmdt (void * shmaddr);

Returns: Success: 0; Failure: -1; Sets errno: Yes.

Argument:

- ***void *shmaddr***: a reference to an attached memory segment (the shared memory pointer).

shmctl call - Shared Memory Control

shmctl permits the user to perform a number of generalized control operations on an existing shared memory segment and on the system shared memory data structure.

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

Return: Success: 0; Failure: -1; Sets errno: Yes.

Arguments

- ***int shmid***: a valid shared memory segment identifier.
- ***int cmd***: the operation shmctl is t perform.
- ***struct shmid_ds * buf***: a reference to the shmid_ds structure

Operations of ***shmctl()***

- IPC_STAT: Return the current value of the shmid_ds structure for the shared memory segment indicated by the ***shmid*** value.
- IPC_SET: Modify a limited number of members in the permission structure found within the ***shmid_ds*** structure.
- IPC_RMID: Remove the system data structure for the referenced shared memory identifier (***shmid***). Once all references to the shared memory segment are eliminated, the system will remove the actual shared memory segment.
- SHM_LOCK: Lock, in memory, the shared memory segment referenced by the ***shmid*** argument. Superuser access required
- SHM_UNLOCK: Unlock the shared memory segment referenced by the ***shmid*** argument. Superuser access required

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main(){

int *ptr;
int shm_id, c=0;
int shm_size = sizeof(int);

shm_id=shmget(IPC_PRIVATE, shm_size, IPC_CREAT | S_IRUSR | S_IWUSR);


if(!fork()){
 printf("[%d]child process started\n", getpid() );
 ptr = (int *) shmat(shm_id, 0, 0);
 do{}while(*ptr==0);
 printf("[%d]shm_value->%d\n", getpid(),*ptr);shmdt(ptr);
 }
else{
 printf("[%d]parent process started\n", getpid() );
 ptr = (int *) shmat(shm_id, 0, 0);
 sleep(2);
 *ptr = 500;
 printf("[%d]shm_value->%d\n", getpid(),*ptr);
 wait(NULL);
 shmdt(ptr);
 shmctl(shm_id, IPC_RMID, 0);
 }

return EXIT_SUCCESS;
}
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main(){
char *ptr;
int shm_id, c=0;
int shm_size = 256;

shm_id=shmget(IPC_PRIVATE, shm_size, IPC_CREAT | S_IRUSR | S_IWUSR);

if(!fork()){
  printf("[%d]child process started\n", getpid() );
  ptr = (char *) shmat(shm_id, 0, 0);
  do{}while(strcmp(ptr,"quit") != 0 );
  printf("[%d]shm_value->%s\n", getpid(), ptr);
  shmdt(ptr);
}
else{
  printf("[%d]parent process started\n", getpid() );
  ptr = (char *) shmat(shm_id, 0, 0);
  char msg[256];
  do{
    fgets(msg, 256, stdin);
    msg[strlen(msg)-1] = '\0';
    strcpy(ptr, msg);
    }while(strcmp(ptr,"quit") != 0 );
  wait(NULL);
  shmdt(ptr);
  shmctl(shm_id, IPC_RMID, 0);
}
```

```
  return EXIT_SUCCESS;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
#include <sys/shm.h>
#include <sys/stat.h>

void sig_handler(int s){};

int main(){
pid_t child;
int *ptr;
int shm_id, c=0;
int shm_size = sizeof(int)*10;
shm_id=shmget(IPC_PRIVATE, shm_size, IPC_CREAT | S_IRUSR | S_IWUSR);
signal(SIGUSR1, sig_handler);
child = fork();
if(!child){
  printf("[%d]child process started\n", getpid() );
  ptr = (int *) shmat(shm_id, 0, 0);
  pause();
  printf("[%d]reading shm_values:\n", getpid());
  for(int i=0; i<10; i++)
    printf("%d ", ptr[i]);
  printf("\n");
  shmdt(ptr);
  }
else{
  printf("[%d]parent process started\n", getpid() );
  int vect[15]={10,20,5,0,2,14,1,8,9,0};
  ptr = (int *) shmat(shm_id, 0, 0);
  usleep(3000);
  for(int i=0; i<10; i++)
     ptr[i] = vect[i];
  printf("[%d]written shm_values.\n", getpid());
  kill(child, SIGUSR1);
  wait(NULL);
  shmdt(ptr);
  shmctl(shm_id, IPC_RMID, 0);
  }

return EXIT_SUCCESS;
}
```