

# SHARED MEMORY ON SOLARIS/LINUX

Y. Alagiannis, S. Evangelatos, V. Nikakis & A. Delis

*Depart. of Informatics & Telecoms*

*University of Athens, Athens, Greece*

## Shared Memory

- Initially, one can create a shared segment using the call `shmget()`
- You may “attach” the shared segment into the address space of a process by invoking `shmat()`.
- The segment is detached through the call `shmdt()`.
- The last process using the segment should *erase* it using the call `shmctl()`.

**Important Note:** shared memory is a *limited* resource; if you do not delete it after you use it, you will eventually deplete this critical resource of the system. Once this happens, programs will not be able to allocate any shared memory.

- To print the allocated shared memory, simply type-in the “`ipcs`” command.
- To remove, an allocated shared memory segment with ID `SharedMemoryID` from the system, type in: “`ipcrm -m SharedMemoryID`”

## Creating a Shared Segment

- **int shmget(key\_t key, int size, int shmflg)**
  - All shared segments have a globally unique key. You can specify this key while trying to use an existing segment. To create a new segment, set the **key** to `IPC_PRIVATE`.
  - **size** indicates the length/size of the segment under creation.
  - **shmflg** offers permissions (very much like file permissions)
    - \* 0642 means: owner can read/write, group can read and everybody else can write.
    - \* execute bit(s) is meaningless.
- **If you are using an existing segment, you do not need to issue an shmget().**
- **Example on creating a new segment:**

```
int id;  
id = shmget(IPC_PRIVATE, 10, 0666);  
if ( id == -1 ) perror( "CREATION" );
```

## Attaching a Shared Segment

- **void \*shmat(int shmid, void \*shmaddr, int shmflg)**
  - **shmid** is the segmentID
  - **shmaddr** is used for setting up the segment at a specific location in the process address space. In general, just use 0.
  - **shmflg** lets you put the segment at a particular location (see man page) or to be read-only (*SHM\_RDONLY*). If **shmflg** is zero then the segment is attached in read-write fashion.

- Usage of shmat:

```
int *mem;
```

```
mem = (int *)shmat(id, (void *)0, 0);
```

```
if ((int)mem == -1 ) perror("Attachment.");
```

- After attachment, the process can use the segment as any other piece of memory!

## Detaching a Shared Segment

- **int shmdt(void \*shmaddr);**
  - **shmaddr** is the address of the shared segment
- Usage:  
*int err;*  
*err = shmdt((void \*)mem);*  
*if (err == -1 ) perror("Detachment");*
- It is recommended to detach the segment after usage. Most systems have a specific value for the maximum number of segments that can be attached simultaneously.

## Removing a Shared Segment

- One of the processes (often the very last) has to remove the shared segment; otherwise the segment will remain in the system for ever.
- Use the versatile `shmctl()` call.
- **`int shmctl(int shmid, int cmd, struct shmid_ds *buf)`**
  - **`shmid`** is the segmentID.
  - **`cmd`** designates what to do; use `IPC_RMID`.
  - Set **`buf`** to 0.
- Usage:  

```
int err;  
err = shmctl(id, IPC_RMID, 0);  
if ( err == -1 ) perror("Removal");
```

## Shell Script for easier removal

- **Linux**

```
- ipcs |  
  awk -v u='whoami' '/Shared/,/^$/{ if($6==0&&$3==u)  
  print "ipcrm shm",$2,";"} /Semaphore/,/^$/{ if($3==u)  
  print "ipcrm sem",$2,";"}'  
  | /bin/sh
```

- **Solaris**

```
- ipcs -a |  
  awk -v u='whoami' '$5==u && (($1=="m" && $9==0) || ($1=="s"))  
  {print "ipcrm -$1,$2,";"}' | /bin/sh
```

- This script is provided by Stefanos Stamatis

## Sample Code for Shared Segment Use

*Output from shareMem1.c*

```
./out1  
Allocated. 38731782  
Attached. Mem contents 0  
Start other process. >
```

```
Start other process. >s  
mem is now 2  
Removed. 0
```

*Output from shareMem2.c*

```
./out2 38731782  
Id is 38731782  
Attached. Mem contents 1  
Changed mem is now 2  
Detachment 0
```

## shareMem1.c source code

```
/* shareMem1.c -- Y. Alagiannis-A.Delis */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char *argv[])
{
    int id=0, err=0, *mem;
    /* Make shared memory segment */
    id = shmget(IPC_PRIVATE,10,0666);
    if (id == -1) perror ("Creation");
    else      printf("Allocated. %d\n", id);
    /* Attach the segment */
    mem = (int *) shmat(id, (void*)0, 0);
    if ((int) mem == -1) perror("Attachment.");
    else printf("Attached. Mem contents %d\n",*mem);

    *mem=1; /* Initialize shared int to 1 */
    printf("Start other process. >"); getchar();
    /* Print out new value */
    printf("mem is now %d\n", *mem);
    /* Remove segment */
    err = shmctl(id, IPC_RMID, 0);
    if (err == -1) perror ("Removal.");
    else printf("Removed. %d\n", err);
    return 0;
}
```

## shareMem2.c source code

```
/* shareMem2.c: Y. Alagiannis-A.Delis */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char *argv[])
{
    int id, err, *mem;
    if (argc <= 1) {printf("Need shmem id. \n"); exit(1); }
    /* Get id from command line. */
    sscanf(argv[1], "%d", &id);
    printf("Id is %d\n", id);

    /* Attach the segment */
    mem = (int *) shmat(id, (void*) 0,0);
    if ((int) mem == -1) perror("Attachment.");
    else printf("Attached. Mem contents %d\n",*mem);

    /* Give it a different value */
    *mem=2;
    printf("Changed mem is now %d\n", *mem);
    /* Detach segment */
    err = shmdt((void *) mem);
    if (err == -1) perror ("Detachment.");
    else printf("Detachment %d\n", err);
    return 0;
}
```

## *POSIX vs. System V Semaphores*

There are two types of semaphores: *POSIX* and *System V*.

- You can use either one (do not mix).
- *POSIX* semaphores are considered easier to use in application programs and follow the definition of the “classic” operations  $P()/V()$ .
- *System V* semaphores are system-wide resources (persistent), have larger footprint, are more versatile (can control the value by which are increased/decreased) but are more clumsy to use.

## (POSIX) Semaphores on Solaris/Linux

- `#include <semaphore.h>`
- `sem_init`, `sem_destroy`, `sem_post`, `sem_wait`, `sem_trywait`
- `int sem_init(sem_t *sem, int pshared, unsigned int value);`
  - The above initializes a semaphore.
  - Compile with `-lrt`
  - `pshared` is:
    - \* *non-zero*: semaphore is shared between processes.
    - \* *zero*: semaphore is shared between threads of the process.

## (POSIX) Semaphore Operations

- **sem\_wait(), sem\_trywait()**
  - **int sem\_wait(sem\_t \*sem);**
  - **int sem\_trywait(sem\_t \*sem);**
  - Perform P(S) operation.
  - **sem\_wait** blocks; **sem\_trywait** will fail rather than block.
- **sem\_post()**
  - **int sem\_post(sem\_t \*sem);**
  - Perform V(S) operation.
- **sem\_destroy()**
  - **int sem\_destroy(sem\_t \*sem);**
  - Destroys a semaphore.

## Example semtest.c

```
/* semtest.c: Semaphore test example. Y. Alagiannis-A.Delis */
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/ipc.h>
extern int errno;

int main(int argc, char *argv[])
{
    sem_t sp;
    int retval;

    /* Initialize the semaphore. */
    retval = sem_init(&sp,1,2);
    if (retval != 0) { perror("Couldn't initialize."); exit(3); }

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    retval = sem_trywait(&sp);
    printf("Did trywait. Returned %d >\n", retval); getchar();

    sem_destroy(&sp);
}
```

## Sample Output

```
kronos:/home/ad/my-semaphores>gcc -o semtest semtest.c -lrt
kronos:/home/ad/my-semaphores>semtest
Did trywait. Returned 0 >

Did trywait. Returned 0 >

Did trywait. Returned -1 >

kronos:/home/ad/my-semaphores>
```

## More Examples

- *semtest3.c* creates a shared memory segment and sets up an semaphore on it.
- *semtest3a.c* gets a shared memory segment ID from the command line and use it (along with the semaphore).

### Running semtest3 alone - Output:

```
kronos:/home/ad/my-semaphores>semtest3
Allocated 401
Did trywait. Returned 0 >

Did trywait. Returned 0 >

Did trywait. Returned -1 >

Removed. 0
kronos:/home/ad/my-semaphores>
```

## Running semtest3 and semtest3a together

```
kronos:/home/ad/my-semaphores>semtest3
```

```
Allocated 601
```

```
Did trywait. Returned 0 >
```

```
Did trywait. Returned -1 >
```

```
Did trywait. Returned -1 >
```

```
Removed. 0
```

```
kronos:/home/ad/my-semaphores>
```

```
kronos:/home/ad/my-semaphores>semtest3a 601
```

```
Allocated 601
```

```
Did trywait. Returned 0 >
```

```
Did trywait. Returned -1 >
```

```
Did trywait. Returned -1 >
```

```
kronos:/home/ad/my-semaphores>
```

```

/* semtest3.c: Semaphore test example using shared memory. Y. Alagiannis- A. Delis */
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SEGMENTSIZe sizeof(sem_t)
#define SEGMENTPERM 0666

int main(int argc, char *argv[])
{
    sem_t *sp;
    int retval;
    int id, err;

    /* Make shared memory segment. */
    id = shmget(IPC_PRIVATE, SEGMENTSIZe, SEGMENTPERM);
    if (id == -1) perror("Creation");
    else printf("Allocated %d\n", id);
    /* Attach the segment. */
    sp = (sem_t *) shmat(id, (void*) 0, 0);
    if ((int) sp == -1) { perror("Attachment."); exit(2);}

    /* Initialize the semaphore. */
    retval = sem_init(sp, 1, 2);
    if (retval != 0) {
        perror("Couldn't initialize.");
        exit(3);
    }

    retval = sem_trywait(sp);

```

```
printf("Did trywait. Returned %d >\n", retval); getchar();

retval = sem_trywait(sp);
printf("Did trywait. Returned %d >\n", retval); getchar();

retval = sem_trywait(sp);
printf("Did trywait. Returned %d >\n", retval); getchar();

sem_destroy(sp);

/* Remove segment. */
err = shmctl(id, IPC_RMID, 0);
if (err == -1) perror("Removal.");
else printf("Removed. %d\n",err);
return 0;
}
```

```

/* semtest3a.c Semaphore test example using shared memory. Y.Alagiannis-A.Delis*/
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SEGMENTSIZe sizeof(sem_t)
#define SEGMENTPERM 0666

extern int errno;
int main(int argc, char *argv[])
{
    sem_t *sp;
    int retval, id, err;

    if (argc <= 1) { printf("Need shm id. \n"); exit(1); }

    /* Get id from command line. */
    sscanf(argv[1], "%d", &id);
    printf("Allocated %d\n", id);

    /* Attach the segment. */
    sp = (sem_t *) shmat(id, (void*) 0, 0);
    if ((int) sp == -1) { perror("Attachment."); exit(2); }
    /* Initialize the semaphore. */
    retval = sem_init(sp, 1, 1);
    if (retval != 0) {
        perror("Couldn't initialize.");
        exit(3);
    }
}

```

```
retval = sem_trywait(sp);
printf("Did trywait. Returned %d >\n", retval); getchar();

retval = sem_trywait(sp);
printf("Did trywait. Returned %d >\n", retval); getchar();

retval = sem_trywait(sp);
printf("Did trywait. Returned %d >\n", retval); getchar();

/* Remove segment. */

err = shmdt((void *) sp);
if (err == -1) perror ("Detachment.");
return 0;
}
```

## Semaphores (System V calls)

- `#include <sys/sem.h>`
- `#include <sys/ipc.h>`
- `#include <sys/types.h>`
- **semget, semop, semctl**
- **int semget(key\_t key, int nsems, int semflg);**
  - The above creates a set of semaphores of nsems items
  - Usually nsems is set to 1 (unique semaphore id for each semaphore)
  - **key** is semaphore set's unique id. Use `IPC_PRIVATE` and system will provide a new, non-existed value.
  - **semflg** offers permissions
  - **Return value:** new semaphore set id on success, -1 on failure.

## Semaphores (System V calls)

- *#include <sys/sem.h>*
- *#include <sys/ipc.h>*
- **int semctl(int semid, int semnum, int cmd, ...);**
  - The semctl() function provides a variety of semaphore control operations.
  - Most common operations are GETVAL, SETVAL, SETALL, IPC\_RMID.
  - The following union has to be declared:
    - union semun {  
int val;  
struct semid\_ds \*buf;  
unsigned short \*array;  
};
  - **semid**: semaphore set id
  - **semnum**: specific semaphore for this operation to be applied
  - **cmd**: operation to be applied
  - The fourth argument is optional and depends upon the operation requested
  - **Return value**: on error returns -1 and errno indicates the error

## Semaphores (System V calls)

- `#include <sys/sem.h>`
- `#include <sys/ipc.h>`
- `#include <sys/types.h>`
- **int semop(int semid, struct sembuf \*sops, size\_t nsops);**
  - The above call is used to perform atomically a user-defined array of semaphore operations on the set of semaphores
  - **semid** is semaphore set's unique id.
  - **sops** is a pointer to a user-defined array of semaphore operation structures
  - Each structure, `sembuf`, contains the following members, that have to be set each time:
    - \* `short sem_num;` (semaphore number)
    - \* `short sem_op;` (semaphore operation, -1 for P(s), +1 for V(s))
    - \* `short sem_flg;` (operation flags, usually set to 0)
  - **nsops** is the number of structures in the array `sops`
  - **Return value:** 0 on success, -1 on error and `errno` will be set to indicate the error.

```
/* Example code using System V calls for semaphores and shared memory */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/ipc.h>

union semun {
    int val;                /* value for SETVAL */
    struct semid_ds *buf;   /* buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* array for GETALL, SETALL */
}

void free_resources(int shm_id, int sem_id)
{
    /* Delete the shared memory segment */
    shmctl(shm_id,IPC_RMID,NULL);

    /* Delete the semaphore */
    semctl(sem_id,0,IPC_RMID,0);
}
```

```

/* Semaphore P - down operation, using semop */
int sem_P(int sem_id)
{
    struct sembuf sem_d;
    sem_d.sem_num = 0;
    sem_d.sem_op = -1;
    sem_d.sem_flg = 0;
    if (semop(sem_id, &sem_d, 1) == -1) {
        perror("# Semaphore down (P) operation ");
        return -1;
    }
    return 0;
}

/* Semaphore V - up operation, using semop */
int sem_V(int sem_id)
{
    struct sembuf sem_d;
    sem_d.sem_num = 0;
    sem_d.sem_op = 1;
    sem_d.sem_flg = 0;
    if (semop(sem_id, &sem_d, 1) == -1) {
        perror("# Semaphore up (V) operation ");
        return -1;
    }
    return 0;
}

```

```

/* Semaphore init - set a semaphore's value to val */
int sem_init(int sem_id, int val)
{
    union semun arg;

    arg.val = val;
    if (semctl(sem_id,0,SETVAL,arg) == -1) {
        perror("# Semaphore setting value ");
        return -1;
    }
    return 0;
}

int main ()
{
    int shm_id, sem_id, t = 0, *sh, pid;

    /* Create a new shared memory segment */
    shm_id = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0660);
    if (shm_id == -1) {
        perror("Shared memory creation");
        exit(EXIT_FAILURE);
    }

    /* Create a new semaphore id */
    sem_id = semget(IPC_PRIVATE,1,IPC_CREAT | 0660);
    if (sem_id == -1) {
        perror("Semaphore creation ");
        shmctl(shm_id,IPC_RMID,(struct shmid_ds *)NULL);
        exit(EXIT_FAILURE);
    }

    /* Set the value of the semaphore to 1 */

```

```

if (sem_Init(&sem_id, 1) == -1) {
    free_resources(shm_id, sem_id);
    exit(EXIT_FAILURE);
}

/* Attach the shared memory segment */
sh = (int *)shmat(shm_id, NULL, 0);
if (sh == NULL) {
    perror("Shared memory attach ");
    free_resources(shm_id, sem_id);
    exit(EXIT_FAILURE);
}
/* Setting shared memory to 0 */
*sh = 0;

/* New process */
if ((pid = fork()) == -1) {
    perror("fork");
    free_resources(shm_id, sem_id);
    exit(EXIT_FAILURE);
}

if (pid == 0) {
    /* Child process */
    printf("# I am the child process with process id: %d\n", getpid());
} else {
    /* Parent process */
    printf("# I am the parent process with process id: %d\n", getpid());
}

printf("(%d): trying to access the critical section\n", getpid());
sem_P(&sem_id);
printf("(%d): accessed the critical section\n", getpid());

```

```
(*sh)++;
printf("(%d): value of shared memory is now: %d\n", getpid(), *sh);

printf("(%d): getting out of the critical section\n", getpid());
sem_V(sem_id);

printf("(%d): got out of the critical section\n", getpid());

/* Child process */
if (!pid)
    exit(EXIT_SUCCESS);

/* Wait for child process */
wait(NULL);

/* Clear resources */
free_resources(shm_id, sem_id);

return EXIT_SUCCESS;
}
```

## Man Pages

- Always read carefully the manual pages for the system calls.