

# **Computer Operating Systems, Practice Session 6**

## **Semaphore and Shared Memory Operations in Unix**

Mustafa Ersen (ersenm@itu.edu.tr)

Istanbul Technical University  
34469 Maslak, İstanbul

25 March 2015

Today

# Computer Operating Systems, PS 6

Shared Memory

Examples

# Shared Memory Data Structure

```
1 struct shmid_ds {  
2     struct ipc_perm shm_perm;    /* Ownership and permissions */  
3     size_t          shm_segsz;   /* Size of segment (bytes) */  
4     time_t          shm_atime;   /* Last attach time */  
5     time_t          shm_dtime;   /* Last detach time */  
6     time_t          shm_ctime;   /* Last change time */  
7     pid_t           shm_cpid;    /* PID of creator */  
8     pid_t           shm_lpid;    /* PID of last shmat/shmdt */  
9     shmatt_t        shm_nattch;  /* No. of current attaches */  
10    ...  
11 };
```

# Shared Memory System Calls

Header files in Unix to be used in shared memory operations:

- ▶ `sys/ipc.h`
- ▶ `sys/shm.h`
- ▶ `sys/types.h`

## Shared Memory Allocation:

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

## Shared Memory Control:

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

## Shared Memory Operations:

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

## Example 1 - Headers and Definitions

```
1  /* to be able to involve an integer variable in strcat */
2  #define _GNU_SOURCE
3  /* for shared memory and semaphores */
4  #include <sys/ipc.h>
5  #include <sys/shm.h>
6  #include <sys/sem.h>
7  #include <sys/types.h>
8  /* for handling signals */
9  #include <signal.h>
10 /* to use fork */
11 #include <unistd.h>
12 /* other necessary headers */
13 #include <stdlib.h>
14 #include <stdio.h>
15 #include <string.h>
16
17 /* The ftok() function returns a key based on path and id that
18    is usable in subsequent calls to semget() and shmget() */
19 #define KEYSEM ftok(strcat(get_current_dir_name(),argv[0]),1)
20 #define KEYSEM2 ftok(strcat(get_current_dir_name(),argv[0]),2)
21 #define KEYSHM ftok(strcat(get_current_dir_name(),argv[0]),3)
```

## Example 1 - Semaphore Operations

```
1  /* semaphore increment operation */
2  void sem_signal(int semid, int val){
3      struct sembuf semaphore;
4      semaphore.sem_num=0;
5      semaphore.sem_op=val;
6      semaphore.sem_flg=1;  /* relative: add sem_op to value */
7      semop(semid, &semaphore, 1);
8  }
9
10 /* semaphore decrement operation */
11 void sem_wait(int semid, int val){
12     struct sembuf semaphore;
13     semaphore.sem_num=0;
14     semaphore.sem_op=(-1*val);
15     semaphore.sem_flg=1;  /* relative: add sem_op to value */
16     semop(semid, &semaphore, 1);
17 }
```

## Example 1 - Signal Handler

```
1 /* signal-handling function */
2 void mysignal(int signum){
3     printf("Received signal with num = %d.\n", signum);
4 }
5
6 void mysigset(int num){
7     struct sigaction mysigaction;
8     mysigaction.sa_handler=(void *)mysignal;
9     /* using signal-catching function identified by sa_handler */
10    mysigaction.sa_flags=0;
11    /* sigaction system call is used to change the action taken by a
12     process on receipt of a specific signal (specified with num) */
13    sigaction(num,&mysigaction ,NULL);
14 }
```

## Example 1 - Creating Child Processes

```
1 int main (int argc, char *argv[]){
2     mysigset(12); /* signal handler with num=12 */
3     int shmid = 0; /* shared memory id */
4     int *globalcp = NULL; /* shared memory area */
5     int localInt; /* a locally defined integer */
6     int termSem = 0, lock = 0; /* semaphore ids */
7     int f; /* return value of fork() */
8     int child[2]; /* child process ids */
9     int i, myOrder = 0; /* order of the running child process */
10
11     /* creating 2 child processes */
12     for (i=0; i<2; i++){
13         f=fork();
14         if (f==-1){
15             printf("FORK error ....\n");
16             exit(1);
17         }
18         if (f==0)
19             break;
20         child[i]=f;
21     }
```



## Example 1 - Parent Process

```
1  if (f != 0){
2      /* creating a semaphore for synchronization(value=0)
3       between parent and its children */
4      termSem = semget(KEYSEM2, 1, 0700|IPC_CREAT);
5      semctl(termSem, 0, SETVAL, 0);
6      /* creating a semaphore for mutual exclusion(value=1)
7       between child processes */
8      lock = semget(KEYSEM, 1, 0700|IPC_CREAT);
9      semctl(lock, 0, SETVAL, 1);
10     /* creating a shared memory area with the size of an int */
11     shmid = shmget(KEYSHM, sizeof(int), 0700|IPC_CREAT);
12     /* attaching the shared memory segment identified by shmid
13      to the address space of the calling process (parent) */
14     globalcp = (int *)shmat(shmid, 0, 0);
15     /* initializing the value in the shared memory as 0 */
16     *globalcp = 0;
17     /* detaching the shared memory segment from the address
18      space of the calling process (parent) */
19     shmdt(globalcp);
20     sleep(2); /* waiting for 2 seconds */
```

## Example 1 - Parent Process (Cont.)

```
1  /* sending the signal 12 to start child processes */
2  printf("Parent has created resources. \n");
3  printf("Now, it will start the child processes. \n");
4  for (i=0; i<2; i++)
5      kill(child[i],12);
6  /* decreasing semaphore value by 2(wait for all children) */
7  sem_wait(termSem,2);
8  printf("All child processes have finished their jobs.\n");
9  /* removing the created semaphores and shared memory */
10 semctl(termSem,0,IPC_RMID,0);
11 semctl(lock,0,IPC_RMID,0);
12 shmctl(shmid,IPC_RMID,0);
13 /* parent process is exiting */
14 exit(0);
15 }
```

## Example 1 - Child Process

```
1  else{
2      /* to show which child process is running */
3      myOrder = i;
4      /* wait until receiving a signal (kill signal) */
5      pause();
6      printf("child %d is starting ....\n",myOrder);
7      /* returning the semaphore ids for KEYSEM and KEYSEM2 */
8      lock = semget(KEYSEM, 1, 0);
9      termSem = semget(KEYSEM2, 1, 0);
10     /* returning the shared memory id associated with KEYSHM */
11     shmid = shmget(KEYSHM, sizeof(int), 0);
12     /* attaching the shared memory segment identified by shmid
13     to the address space of the calling process (child) */
14     globalcp = (int *) shmat(shmid,0,0);
15     for (i=0; i<5; i++){
16         /* waiting for the semaphore with id=lock to enter
17         the critical section */
18         sem_wait(lock, 1);
19         printf(" child %d: Found the value: %d, i:%d\n",
20               myOrder, *globalcp, i);
```

## Example 1 - Child Process (Cont.)

```
1      /* updating the value of the shared memory segment */
2      localInt = *globalcp;
3      sleep(1);    /* waiting for a second */
4      localInt += i;
5      *globalcp = localInt;
6      printf(" child %d: Made the value: %d, i:%d\n",
7             myOrder, *globalcp, i);
8      /* making the critical section available again */
9      sem_signal(lock, 1);
10     sleep(1);    /* waiting for a second */
11 }
12 /* detaching the shared memory segment from the address
13 space of the calling process (child) */
14 shmdt(globalcp);
15 /* increase semaphore by 1(synchronization with parent) */
16 sem_signal(termSem, 1);
17 exit(0);    /* child process is exiting */
18 }
19 return 0;
20 }
```

# Output of Example 1

```
Parent has created resources.  
Now, it will start the child processes.  
Received signal with num = 12.  
child 1 is starting ....  
Received signal with num = 12.  
child 0 is starting ....  
child 0: Found the value: 0, i:0  
child 0: Made the value: 0, i:0  
child 1: Found the value: 0, i:0  
child 1: Made the value: 0, i:0  
child 0: Found the value: 0, i:1  
child 0: Made the value: 1, i:1  
child 1: Found the value: 1, i:1  
child 1: Made the value: 2, i:1  
child 0: Found the value: 2, i:2  
child 0: Made the value: 4, i:2  
child 1: Found the value: 4, i:2  
child 1: Made the value: 6, i:2  
child 0: Found the value: 6, i:3  
child 0: Made the value: 9, i:3  
child 1: Found the value: 9, i:3  
child 1: Made the value: 12, i:3  
child 0: Found the value: 12, i:4  
child 0: Made the value: 16, i:4  
child 1: Found the value: 16, i:4  
child 1: Made the value: 20, i:4  
All child processes have finished their jobs.
```

## Example 2 - Creating a shared memory segment with permission flags

```
1 #include <stdio.h>
2 #include <sys/shm.h> /* shared memory */
3 #include <sys/stat.h> /* S_IRUSR and S_IWUSR */
4
5 int main () {
6     /* Allocate a Shared Memory Location */
7     int segment_id;
8     const int shared_segment_size = 0x6400;
9     segment_id = shmget(IPC_PRIVATE, shared_segment_size,
10                        IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
11     /* IPC_PRIVATE: shared memory cannot be accessed by other processes
12     IPC_CREAT: create the segment if it doesn't already exist
13     IPC_EXCL: fail if segment already exists
14     S_IRUSR: read permission, owner
15     S_IWUSR: write permission, owner */
```

## Example 2 - Attaching to a variable and writing

```
1  /* Attach a Connection */
2  char* shared_memory;
3  shared_memory = (char*) shmat(segment_id, 0, 0);
4  printf("Shared Memory Attached Address %p\n", shared_memory);
5
6  /* Learn the Segment Size */
7  int segment_size;
8  struct shmid_ds shmbuffer;
9  shmctl(segment_id, IPC_STAT, &shmbuffer);
10 segment_size = shmbuffer.shm_segsz;
11 printf("Segment size is: %d\n", segment_size);
12
13 /* Write a String into the Shared Memory Location*/
14 sprintf(shared_memory, "Hello, World.");
15
16 /* Detach Connection */
17 shmdt(shared_memory);
```

## Example 2 - Attaching to another address and reading

```
1  /* Make a Shared Memory Connection to another Address */
2  shared_memory = (char*) shmat(segment_id, (void*) 0x5000000, 0);
3  printf("Shared Memory New Connection Address %p\n", shared_memory);
4
5  /* Read String from the Shared Memory Location */
6  printf("%s\n", shared_memory);
7
8  /* Detach Connection */
9  shmdt(shared_memory);
10
11 /* Remove the Shared Memory Segment */
12 shmctl(segment_id, IPC_RMID, 0);
13
14 return 0;
15 }
```



## Output of Example 2

```
Shared Memory Attached Address 0xb778e000  
Segment size is: 25600  
Shared Memory New Connection Address 0x5000000  
Hello, World.
```