Operating System

Sri Lanka Institute of Information Technology

Worksheet 7

Year 03 Semester 01

Purpose

In this lab, You will learn about Inter process communication with Semaphores.

Basic Concepts

Semaphores can best be described as counters used to control access to shared resources by multiple processes. They are most often used as a locking mechanism to prevent processes from accessing a particular resource while another process is performing operations on it. Semaphores are often dubbed the most difficult to grasp of the three types of System V IPC objects. In order to fully understand semaphores, we'll discuss them briefly before engaging any system calls and operational theory. The name semaphore is actually an old railroad term, referring to the crossroad "arms" that prevent cars from crossing the tracks at intersections. The same can be said about a simple semaphore set. If the semaphore is on (the arms are up), then a resource is available (cars may cross the tracks). However, if the semaphore is off (the arms are down), then resources are not available (the cars must wait).

While this simple example may stand to introduce the concept, it is important to realize that semaphores are actually implemented as sets, rather than as single entities.

Internal Data Structures

Let's briefly look at data structures maintained by the kernel for semaphore sets. Kernel semid ds structure As with message queues, the kernel maintains a special internal data structure for each semaphore set which exists within its addressing space. This structure is of type semid ds, and is defined in linux/sem.h as follows:

```
/* One semid data structure for each set of semaphores in the system. */
struct semid_ds {
struct ipc_perm sem_perm;          /* permissions .. see ipc.h */
time_t sem_otime;                  /* last semop time */
time_t sem_ctime;                  /* last change time */
struct sem *sem_base;              /* ptr to first semaphore in array
struct wait_queue *eventn;
struct wait_queue *eventz;
struct sem_undo *undo;              /* undo requests on this array */
ushort sem_nsems;                  /* no. of semaphores in array */
};
```

Kernel sem structure In the semid ds structure, there exists a pointer to the base of the semaphore array itself. Each array member is of the sem structure type. It is also

defined in linux/sem.h:

```
/* One semaphore structure for each semaphore in the system. */
struct sem {
short sempid;              /* pid of last operation */
ushort semval;             /* current value */
ushort semncnt;          /* num procs awaiting increase in semval
ushort semzcnt;          /* num procs awaiting semval = 0 */
};
```

SYSTEM CALL: semget()
In order to create a new semaphore set, or access an existing set, the semget() system call is used.

SYSTEM CALL: semget();
PROTOTYPE: int semget (key_t key, int nsems, int semflg);
RETURNS: semaphore set IPC identifier on success -1 on error:

SYSTEM CALL: semop()
SYSTEM CALL: semop();
PROTOTYPE: int semop (int semid, struct sembuf *sops, unsigned nsops);
RETURNS: 0 on success (all operations performed) -1 on error:

SYSTEM CALL: semctl()
SYSTEM CALL: semctl();
PROTOTYPE: int semctl (int semid, int semnum, int cmd, union semun arg);
RETURNS: positive integer on success -1 on error:

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

 union semun {
                    int val;
                    struct semid_ds *buf;
                    ushort *array;
                };
main()
{          int i,j;
           int pid;
           int semid; /* semid of semaphore set */
           key_t key = 1234; /* key to pass to semget() */
           int semflg = IPC_CREAT | 0666; /* semflg to pass to semget() */
           int nsems = 1; /* nsems to pass to semget() */
```

```
        int nsops; /* number of operations to do */
        struct sembuf *sops = (struct sembuf *) malloc(2*sizeof(struct sembuf));

 (void) fprintf(stderr, "\nsemget: Setting up seamaphore: semget(%#lx, %\
%#o)\n",key, nsems, semflg);

  if ((semid = semget(key, nsems, semflg)) == -1) {
        perror("semget: semget failed");
        exit(1);
    } else
        (void) fprintf(stderr, "semget: semget succeeded: semid =\ %d\n", semid);

 /* get child process */

                  if ((pid = fork()) < 0) {
                      perror("fork");
                      exit(1);
                  }

                if (pid == 0)
                  { /* child */
                    i = 0;

                while (i  < 3) {/* allow for 3 semaphore sets */

                  nsops = 2;

    /* wait for semaphore to reach zero */

    sops[0].sem_num = 0; /* We only use one track */
    sops[0].sem_op = 0; /* wait for semaphore flag to become zero */
    sops[0].sem_flg = SEM_UNDO; /* take off semaphore asynchronous  */


    sops[1].sem_num = 0;
    sops[1].sem_op = 1; /* increment semaphore -- take control of track */
    sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT; /* take off semaphore */

    /* Recap the call to be made. */

    (void) fprintf(stderr,"\nsemop:Child  Calling semop(%d, &sops, %d) with:", semid,
nsops);
    for (j = 0; j < nsops; j++)
        {
         (void) fprintf(stderr, "\n\tsops[%d].sem_num = %d, ", j, sops[j].sem_num);
         (void) fprintf(stderr, "sem_op = %d, ", sops[j].sem_op);
```

```
          (void) fprintf(stderr, "sem_flg = %#o\n", sops[j].sem_flg);
        }

    /* Make the semop() call and report the results. */
        if ((j = semop(semid, sops, nsops)) == -1) {
                perror("semop: semop failed");
                }
          else
         {
                (void) fprintf(stderr, "\tsemop: semop returned %d\n", j);
(void) fprintf(stderr, "\n\nChild Process Taking Control of Track: %d/3  times\n", i+1);
                sleep(5); /* DO Nothing for 5 seconds */

                 nsops = 1;

                /* wait for semaphore to reach zero */
                sops[0].sem_num = 0;
            sops[0].sem_op = -1; /* Give UP COntrol of track */
sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT; /* take off semaphore, asynchronous
*/

                if ((j = semop(semid, sops, nsops)) == -1) {
                        perror("semop: semop failed");
                        }
                  else
(void) fprintf(stderr, "Child Process Giving up Control of Track: %d/3 times\n", i+1);
                sleep(5); /* halt process to allow parent to catch semaphor change first */
             }
      ++i;
    }

   }
 else /* parent */
   {  /* pid hold id of child */

     i = 0;


    while (i  < 3) { /* allow for 3 semaphore sets */

    nsops = 2;

    /* wait for semaphore to reach zero */
    sops[0].sem_num = 0;
    sops[0].sem_op = 0; /* wait for semaphore flag to become zero */
    sops[0].sem_flg = SEM_UNDO; /* take off semaphore asynchronous  */
```

```
      sops[1].sem_num = 0;
      sops[1].sem_op = 1; /* increment semaphore -- take control of track */
      sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT; /* take off semaphore */

      /* Recap the call to be made. */
 (void) fprintf(stderr,"\nsemop:Parent Calling semop(%d, &sops, %d) with:", semid,
nsops);
      for (j = 0; j < nsops; j++)
          {
           (void) fprintf(stderr, "\n\tsops[%d].sem_num = %d, ", j, sops[j].sem_num);
           (void) fprintf(stderr, "sem_op = %d, ", sops[j].sem_op);
           (void) fprintf(stderr, "sem_flg = %#o\n", sops[j].sem_flg);
          }

      /* Make the semop() call and report the results. */
          if ((j = semop(semid, sops, nsops)) == -1) {
                    perror("semop: semop failed");
                    }
            else
           {
                    (void) fprintf(stderr, "semop: semop returned %d\n", j);

(void) fprintf(stderr, "Parent Process Taking Control of Track: %d/3 times\n", i+1);
                    sleep(5); /* Do nothing for 5 seconds */

                     nsops = 1;

                    /* wait for semaphore to reach zero */
                    sops[0].sem_num = 0;
              sops[0].sem_op = -1; /* Give UP COntrol of track */
    sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT; /* take off semaphore,
asynchronous  */
                              if ((j = semop(semid, sops, nsops)) == -1) {
                              perror("semop: semop failed");
                              }
                       else
  (void) fprintf(stderr, "Parent Process Giving up Control of Track: %d/3 times\n", i+1);
                    sleep(5); /* halt process to allow child to catch semaphor change first */
                }
       ++i;
     }
    }
}
```