IPC:Semaphores

Assignment No 6.a

Subject:- Unix Operating System

System Lab Class :- TYIT

Name PRN

Aditi Sudhir Ghate 2020BTEIT00044

Title-Write a program to illustrate the semaphore concept. Use fork so that 2 process running simultaneously and communicate via semaphore.

Objective:

1. To learn about IPC through semaphore.
2. Use of system call and IPC mechanism to write effective application programs.

Theory:

A semaphore controls access to a shared resource through the use of a counter. If the counter is greater than zero, then access is allowed. If it is zero, then access is denied. What the counter is counting are permits that allow access to the shared resource. Thus, to access the resource, a thread must be granted a permit from the semaphore.

Working of semaphore :

In general, to use a semaphore, the thread that wants access to the shared resource tries to acquire a permit. If the semaphore's count is greater than zero, then the thread acquires a permit, which causes the semaphore's count to be decremented. Otherwise, the thread will be blocked until a permit can be acquired. When the thread no longer needs an access to the shared resource, it releases the permit, which causes the semaphore's count to be incremented. If there is another thread waiting for a permit, then that thread will acquire a permit at that time.

The function semget() initializes or gains access to a semaphore.

It is prototyped by: int semget(key_t key, int nsems, int semflg);

When the call succeeds, it returns the semaphore ID (semid). The key argument is a access value associated with the semaphore ID. The nsems argument specifies the number of elements in a semaphore array. The call fails when nsems is greater than the number of elements in an existing array; when the correct count is not known, supplying 0 for this argument ensures that it will succeed. POSIX Semaphores:

- sem_open() -- Connects to, and optionally creates, a named semaphore
- sem_init() -- Initializes a semaphore structure (internal to the calling program, so not a named semaphore).
- sem_close() -- Ends the connection to an open semaphore.

- sem_unlink() -- Ends the connection to an open semaphore and causes the semaphore to be removed when the last process closes it.
- sem_destroy() -- Initializes a semaphore structure (internal to the calling program, so not a named semaphore).
- sem_getvalue() -- Copies the value of the semaphore into the specified integer.
- sem_wait(), sem_trywait() -- Blocks while the semaphore is held by other processes or returns an error if the semaphore is held by another process.
- sem_post() -- Increments the count of the semaphore.

Data Dictionary:

Number	Variable/function	DataType	Use
1.	pid	int	Get Process ID
2.	semflg	int	Flag to pass to semget
3.	semid	int	Id of semaphore
4.	key	Key_t	Key to pass to semget
5.	nops	int	Number of Operations

Program-

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include<stdlib.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));
/* ptr to operations to perform */
```

```
/* set up semaphore */
(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 */
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;
}
}
}
```

Output-

```
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments/6A$ ./a.out

semget: Setting up seamaphore: semget(0x4d2, %#o)
semget: semget succeeded: semid =0

semop:Parent Calling semop(0, &sops, 2) with:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
semop: semop returned 0
Parent Process Taking Control of Track: 1/3 times

semop:Child Calling semop(0, &sops, 2) with:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
Parent Process Giving up Control of Track: 1/3 times
        semop: semop returned 0


Child Process Taking Control of Track: 1/3 times

semop:Parent Calling semop(0, &sops, 2) with:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
Child Process Giving up Control of Track: 1/3 times
semop: semop returned 0
Parent Process Taking Control of Track: 2/3 times

semop:Child Calling semop(0, &sops, 2) with:Parent Process Giving up Control of Track: 2/3 times

        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
        semop: semop returned 0
```

```
Child Process Taking Control of Track: 2/3 times

semop:Parent Calling semop(0, &sops, 2) with:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
Child Process Giving up Control of Track: 2/3 times
semop: semop returned 0
Parent Process Taking Control of Track: 3/3 times

semop:Child Calling semop(0, &sops, 2) with:Parent Process Giving up Control of Track: 3/3 times

        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
        semop: semop returned 0


Child Process Taking Control of Track: 3/3 times
Child Process Giving up Control of Track: 3/3 times
```

Conclusion-

Use of semaphore for IPC where one process is child of other and in same program using various system calls like semget,semctl is studied

Reference-

Dave's Programming in C Tutorials