LAB NO:5

Tribhuvan University

Institute of Engineering

Pulchowk Campus

Lab report on

Shared Memory and Semaphores

Submitted by

Arpan Pokharel
075BCT015
Group: A

Submitted to

Department of Electronics and Computer Engineering

Submission Date: Feb 28,2022

Shared Memory and Semophores

Theory

Shared memory or IPC is a mechanism in which two or more processes sharing a common segment of memory that they can both read to and write from to communicate with one another. It's just other part of process's address space just like calling malloc() Because it's just memory, shared memory is the fastest IPC mechanism of them all.

Shared memory is well, a shared resource. Without some way of letting the process that have access to it know if it's safe to read and write to the shared memory area. We're leaving our code open to race condition This problem arises because these systems which use preemptive multitasking, where the os takes care of managing when and for how long each process gets to execute. In such a situation, the contents of shared memory area can end up modified easily.

One simple way of solving this problem is semaphores.

Semaphores

Semaphores are another IPC mechanism available when developing on UNIX. They allow different process to synchronise their access to certain resources.

The most common type is called a binary semaphore because they have two states locked or unlocked.

when a process wants exclusive access to a resource, shared memory being an example

they attempt to lock the semaphore associated with that resource. If the semaphore they are attempting to lock is already locked, the caller is suspended, otherwise they are granted to lock. When we're finished using resource, we unlock the resource and any processes that have attempted to lock the semaphore in the meantime are woken up again to attempt the lock again. This way only one process can have access to the resource at once.

Program 1:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
  const key_t SHMKEY =7890;
  const int PERMS =0666;
  char exFlag =0;
Void main(){
      int shmid;
      char* data;
  shmid=shmget(SHMKEY, 1, PERM | IPC_CREAT);
  if(shmid<0)
     Printf("can't get the shared memory");
  data =(char*)shmat(shmid,0,0);
  *data =0;
  if(!fork()){
     printF("press any key to exit... \n");
     scanF("%c", &data,1);
     Printf("child exiting...\n");
     exit(0); }
  else{
     while(!data)
     printF("Recieved %c from child\n",data);
     Shmdt(data);
     shmctl(shmid, IPC_RMID, 0);
     Printf("parent exiting..");
     exito;
  }
```

OUTPUT

```
press any key to exit...
r
child exiting...
Recieved r from chid
Parent exiting..
```

Discussion

The code above shows use of shared memory to exchange data in between child and parent process. First of all a shared memory segment is allocated using shm get which could be used by all interested process. And then thus created segment is mapped onto address 0. And then data is written through child process which is read by parent process which is read by parent process which lates detaches the segment which is later destroyed

Program 2:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
void main(){
    int semid, key, nsem;
    key=(key_t)0x20;
    nsem=0;
    semid=0;
    semid=semget(key, nsem, ipc_creat|0666);
    printf("created semaphore with id: %d", semid);
}
```

OUTPUT

created semaphore with id: -1

DISCUSSION

Semget() system call gets a system semaphore set identifies. i.e. returns the system semaphore set identifies associated with the argument key. The system call creates a new set of 'nsem' number of semaphores. Thus in

above lab code, no semaphore is created and -1 is returned by semget() which would have returned semaphore set identifier (a non-negative integer)

changing nsem = 1

OUTPUT

created semaphore with id: 0

- - - - semaphore Arrays - - -

key	semid	owner	perms	nsems
0x00000020	0	Arpan	666	1

Discussion

changing value of nsem from 0 to 1, one semaphore is created with identifier 0

Program 3

```
#include<sys/types.h>
#include<sys/ipc.h>
main()
{
  int semid, key, nsem, flag;
  nsem = 1;
  flag = 0666|IPC_CREAT;
  for(i=0;;i++){
  key = (key_t)i;
  semid = semget(key, nsem, flag);
  if(semid > 0)
    printf("created semaphore with Id: %d",
                                    semid)
  else{
  printf("maximum number of semaphore set
          are %d \n", i);
  exit(0);
  }
```

Output

Created semaphore with ID =: 32769
Created Semaphore with ID =: 65538
:
Created Semaphore with ID =: [redacted] 1048608
maximum number of semaphore set are 32

Discussion

- - - - - Semaphore Arrays - - - - -

key	semid	owner	perms	nsems
0x00000020	0	Arpan	666	1
0x0000000[redacted]	32769	Arpan	666	1
:				
0x0000001f	1048608	Arpan	666	1

The maximum number of semaphore that can be created is 32. (ie. from 0x00000000 to 0x0000001F). Because a semaphore with key 0x00000020 is already created in previous program and so it returns some negative value as it cannot create one and loop is terminated.

Program 4

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

const key_t SEMKEY=4567;
const int perms=0666;
char exflag=0
void main() {
    int semid;
semid=semget(SEMKEY,1,PERMS|IPC_CREAT);
```

```
if(semid< 0){
    printf("can't create semaphore"); exit(1)
if(!fork()){
   char data;
   sembuf *sbuf=(sembuf*) malloc(sizeof
                                 (sembuf);
 sbuf→sem_num = 0
 sbuf→sem_op = 1;
 sbuf→sem_flg = 0;
 printf("press any key to exit...\n");
 scanf("%c", &data);
 semop(semid, sbuf, 1);
 printf("child exiting");
 exit(0);
  }
    else{
 sembuf* sbuf = (sembuf*) malloc(sizeof(sem
                                 - buf)
      sbuf→sem_num = 0;
      sbuf→sem_op = -1
      sbuf→sem_flg = 0;
   semop(semid, sbuf, 1);
     printf("child released semaphore \n");
     printf("parent exiting");
     exit(0); }}
```

output

```
press any key to exit..
a
child exiting...
child released semaphore
parent exiting..
```

Discussion

Here semaphose is released by the child, then the parent continues. until then the parent will be waiting at the statement

sem_op(semid, sbud, 1);

Here sem_op ~~is~~ # in the child is 1 which means to increment semvalue by 1 and sem_op value in parent is -1 which means to decrease the value by 1.

If sem_op is less than 0 and sem_flag is not equal to IPC_NOWAIT and if the semaphore value is too low, that particular operation is blocked until another operation release the resource or modifies the value of semaphore such that the requested operation can proceed. In above program, the sem_op() in parent is blocked and is executed after the child increase the value of semaphore.

conclusion

Thus, shared memory (shm) and semaphores (sem) can be used as a means of inter process communication.