BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

IS F462 – Network Programming

Lab#4

Topic: System V IPC and POSIX IPC

Note: please use programs under *code* directory supplied with this sheet. Do not copy from this sheet

System V Semaphores:

1. Creating and initializing a semaphore

Semaphores are used to synchronize the access to a shared data among multiple processes. The following program shows us on how to create and initialize a semaphore. The complete program is in semget.c

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

#define key (2000) //change this to your id, only digits: 2009100

main ()
{
  int id;
  id = semget (key, 1, IPC_CREAT | 0666);
  if (id < 0)
    printf ("Semaphore is not created\n");
  printf("Semaphore successfully created with id %d\n", id);
return;
}
```

Q?

1. Compile and run the above program. The list of semaphores that can be viewed by your account can be listed by running the command `$ipcs –s`.
2. Run the command $ipcs –s –i <semid> and check what is the value of semaphore.
3. the default value of the semaphore is not useful for practical purposes. It has to be set to the available number of resources. Suppose if we have 6

resources with us, how do we initialize? Use the following line to initialize the semaphore. Modify the above program with the following line. Complete program is available in seminit.c

```
semctl (semid, 0, SETVAL, 6)
```

after running the program, check the modified value using the above command.

4. find out what is the maximum value a semaphore can take.
5. how do we initialize for binary semaphores and counting semaphores?
6. As discussed in the class, the fact that creating and initializing operations not being done atomically, it may lead to deadlock situations. Write a piece of code that ensures in any process that is accessing the semaphore only after initialization.
7. use semctl() call to remove the semaphore in your program.

2. operating with semaphores:

Considering that we have created the semaphore and initialized it, let us see how to utilize it. the following program synchronizes the parent and child in printing the lines. View the file semopPC.c. This is an example of binary semaphores.

Q?

1. Compile and run the above program. What did you observe? Is the child and parent printing lines one after another? Understand how it happens.
2. Change the initial value of the semaphore 5 and check the out put. Justify the output. Remove the sleep() and check.
3. Create one more child running the same code as the first child and check the output. Justify.

3. Producer-consumer problem using semaphores:

Consider a producer-consumer problem. In this problem, producers supplies goods only when availability of goods is zero and at one time he supplies 10 goods. Consumer will consume goods only when goods are available. The Semaphores are used to synchronize the producer and consumer demands. The programs are available as producer.c and consumer.c. This is an example of counting semaphores. Change the key value to something of your idno. Otherwise you may get weird results.

Q?

1. Run both the programs in separate terminals. Observe how the semaphore is used to control the access to objects. if you stop producer.c program from running, how does consumer behave? Also use ipcs to see the semaphores created in the system.
2. Run consumer program in more than one terminal and see the sem value. This is the case when we have multiple consumers with single producer.
3. Change the producer behaviour. Producer will not wait until it becomes zero rather he will keep adding 1 object at a time. Suppose that the storage capacity for storing the goods is only N. that means producer can't produce and store more than N. How do you ensure this using semaphores? [Hint: think of using one more semaphore]
4. Try to run program in 3 in multiple terminals. That is multiple producers and multiple consumers. Justify the output.
5. Suppose that the items are not use and throw. They are re-usable. That is when a consumer takes an item he returns it after sometime say 2 secs. Modify the program to work for this.
6. Suppose while running as in 5, kill some consumers using kill command and observe the sem values. Did the consumers return the acquired values? Use SEM_UNDO flag and check.

Shared Memory:

1. Creating and attaching a shared memory segment

Shared memory is used to share the user memory among multiple processes with the help of kernel. The following program shows how to create and attach shared memory segment. The program is in shmget.c

```
main ()
{
  int id;
  key_t key;
  int shmid;
  char *data;

  key = ftok ("shmget.c", 'R');

  if ((shmid = shmget (key, 1024, 0644 | IPC_CREAT)) == -1)
    {
```

```
      perror ("shmget: shmget failed");
      exit (1);
    }
  data = shmat (shmid, (void *) 0, 0);

  if (data == (char *) (-1))
    perror ("shmat");

  return;
}
```

Q?

1. Compile and run the above program. The list of shared memory segments that can be viewed by your account can be listed by running the command `$ipcs –m`.
2. Run the command $ ipcs -m -i <shmid> and check what is the size of shared memory.
3. Write a line that adds some data to the shared memory. Something like, strcpy(data,"hello world");
4. Copy this program and modify to read and print the contents of shared memory. Verify whether you are getting the same contents are not.
5. Modify the above program as follows. The complete program is available in shmgetC.c

```
if((pid==fork())==0)
{

for(i=0;i<50;i++)
{      printf("%s", data);
sprintf(data, "I am child and my pid is %d and my count is %d\n", getpid(), i);
}
exit(0);
}
else if(pid>0)
{

for(i=0;i<50;i++)
{      printf("%s", data);
sprintf(data, "I am parent and my pid is %d and my count is %d\n", getpid(),
i);
}
}
```

Run the above program and see the output. Did it match you expectations. Justify? Find the solution.

6. Write a line that detaches the shared memory segment
7. Write a line that removes the shared memory segment
8. Design a program that verifies that removing a shared memory segment doesn't erase it from the memory unless all the processes have detached it.

2. Client and server using shared memory segments controlled by semaphores:

The following programs illustrate the use of shared memory for clients and servers running on the same system. This is a very popular use of shared memory. This example is taken from

http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Fapis%2Fapiexusmem.htm

i. Server program (shmserver.c)

ii. Client program (shmclient.c)

Q?

Q1. Observe how the shared memory is protected using the semaphores

Q2. Make changes in the above code as per the following question: The program shmclient.c will writes an array of N numbers where N is occupied in first 1 byte and thereafter every two bytes stores each number. The server sums up all the numbers and writes the output next to the last number in 4 bytes. Client read the result written by the server.

Memory Mapping:

We have seen that fifos are basically named pipes. Similarly, mapped memory is like shared memory with a name. With this little bit of information, we can conclude that there is some sort of relationship between files in the file system and memory mapping techniques - and, in fact, there is. Simply put, the memory mapping mechanism deals with the mapping of a file in the file system to a portion of memory through mmap() function.

```
#include <sys/mman.h>
Void *mmap(void *addr, size_t len, int prot, int flag, int filedes,
off_t off);
```

Consider the following program in which two processes will communicate through memory mapping. "Write.c" will write the data into some file and "read.c" will access the data using mmap() function.

POSIX IPC

POSIX Message Queues

Creating and Using Posix Message Queue

//pmsg_create.c

```
#define QUEUE_NAME  "/test_queue" //Name must begin with /
#define MAX_SIZE    1024

int main(int argc,char *argv[]){
    mqd_t mqd;
    struct mq_attr attr;
    int must_stop = 0;

    /* initialize the queue attributes */
    attr.mq_flags = 0;
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = MAX_SIZE;
    attr.mq_curmsgs = 0;

    /* create the message queue */
    mqd = mq_open(QUEUE_NAME, O_CREAT | O_RDONLY, 0644, &attr);
    if(mqd < 0)
      perror("Error While Creating Message Queue\n");

return 0;
}
```

Q?

Q2. Compile and run the above program.

a. gcc pmsg_create.c -lrt

Q3. Run the programs named pmsg_send and pmsg_receive and understand the code.

a. First run pmsg_send and then simultaneously run pmsg_receive

Q4. Modify the above programs to create a chat application between two processes.

Q5. Read about **mq_timesend** and **mq_timereceive** and try to implement them on code in Q 2.

Displaying The Message Queue

#mount -t mqueue source target

```
$su
Password
#mkdir <mountpoint>
 #mount -t mqueue none <mountpoint>

mkdir mqueue
~/code$ sudo mount -t mqueue none /home/anand/code/mqueue

$cat /proc/mounts | grep mqueue
none /home/anand/code/mqueue mqueue rw,relatime 0 0

~/code/mqueue$ ls
test_queue

$ls -ld <mountpoint>
-rw-r--r-- 1 anand anand 80 Feb  9 11:14 test_queue
```

Deleting Message Queue

```
#include <mqueue.h>

int mq_unlink(const char *name);
```

The mq_unlink() function removes the message queue identified by name, and marks the queue to be destroyed once all processes cease using it. (This may mean immediately, if all processes that had the queue open have already closed it)

A Feature that distinguishes POSIX message queues form their System V counterparts is the ability to receive asynchronous notification of the availability of a message on a previously empty queue (ie when queue makes transitions from being empty to nonempty) . A process can choose to be notified either via a signal or via invocation of a function in a separate thread.

Note : Only one process (“The registered Process”) can be registered to receive a notification from a particular message queue.

```
#include <mqueue.h>
```

```
int mq_notify(mqd mqdes, const struct sigevent *notification);
```

Q?

Q3. Run the code **mq_notify.c** and try to understand it.
Q4. Try using message queue to synchronize parent and child process execution similar to one in earlier labs.

POSIX Semaphores

Creating Semaphore

```
int main(int argc,char *argv[]){

int flags,opt;
mode_t perms;
unsigned int value;
sem_t *sem;
flags = 0;
flags = flags | O_CREAT | O_RDONLY;
perms = S_IRUSR;
value = 1;
sem = sem_open(sem_name,flags,0777,value);
if(sem < SEM_FAILED){
      perror("Error Creating Semaphore\n");
}else{
      printf("Semaphore Created Successfully\n");
}
int currval = 0;
if(sem_getvalue(sem,&currval) < 0){
      perror("Error While Getting The Value Of Semaphore\n");
}else{
      printf("Current Value of Semaphore Is :\t%d\n",currval);
}
sem_unlink(sem_name);
      return 0;
}
```

Q?

Q1. Run the code **psem_create.c**
    a. `gcc -o create psem_create.c -lpthread`
Q2. Check if the semaphore is created in the system or not
    a. `ls -l /dev/shm/sem.*`

Note: You may need to add sleep to your code for this.

System V semaphores, when creating a semaphore object, creates an array of semaphores whereas POSIX semaphores create just one. System V uses keys and identifiers to identify the IPC objects, while POSIX uses names and file descriptors to identify the IPC object. One marked difference between the System V and POSIX semaphore implementations is that in System V you can control how much the semaphore count can be increased or decreased; whereas in POSIX, the semaphore count is increased and decreased by 1. POSIX semaphores provide a mechanism for process-wide semaphores rather than system-wide semaphores. So, if a developer forgets to close the semaphore, on process exit the semaphore is cleaned up. In simple terms, POSIX semaphores provide a mechanism for non-persistent semaphores.

Synchronization using semaphore

Sem_Wait

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
```

Sem_Post

```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

Q?

Q1. Execute **sem_wait_post.c** and understand the synchronization
Q2. Write a program for synchronization between three processes ie one parent and two children similar to sem_wait_post.
Q3. Till now we have seen Named semaphore, now let's look at Unnamed semaphore for synchronization between related processes and threads.

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
```

Q4. Try writing the above programs using Unnamed semaphore.

Shared Memory

Creating Shared Memory

```
int main(int argc,char *argv[]){
      int flags,opt,fd;
      mode_t perms;
      size_t size;
      void *addr;

      flags = O_RDWR | O_CREAT;

      size = 50;
      perms = perms | S_IRUSR | S_IWUSR;
      fd = shm_open(shm_name,flags,perms);
      if(fd==-1){
            perror("Error In Opening\n");
      }
      addr = mmap(NULL,size,PROT_READ | PROT_WRITE,MAP_SHARED,fd,0);;
      if(addr == MAP_FAILED){
            perror("MMAP ERROR\n");
      }
      return 0;
}
```

Q?

Q 1 Execute the above program and check if shared memory is created or not.

```
ls -l /dev/shm/
```

POSIX Shared memory returns a file descriptor while System V shared memory does not. One can use stat over shared memory to get the details of the memory segment.

Writing To Shared Memory And Reading From Shared Memory

Q 2 Execute program **pshm_write.c** and **pshm_read.c**
Q 3 Synchronize parent and child process ie printing pid in alternate manner using shared memory.
Q 4 Try creating a chat application using shared memory

Removing Shared Memory

```
int shm_unlink(const char *name);
```