**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)**

**IS F462 – Network Programming**

**Lab#5**

**Topic: POSIX IPC**

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

#### In today’s lab we will do some example programs on POSIX message queues and POSIX semaphores and POSIX Shared Memory.

### **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?

1. Compile and run the above program.
   1. gcc pmsg\_create.c -lrt
2. Run the programs named pmsg\_send and pmsg\_receive and understand the code.
   1. First run pmsg\_send and then simultaneously run pmsg\_receive
3. Modify the above programs to create a chat application between two processes.
4. 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?

1. Run the code **mq\_notify.c** and try to understand it.
2. 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?

1. Run the code **psem\_create.c**
   1. gcc -o create psem\_create.c -lpthread
2. Check if the semaphore is created in the system or not
   1. 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?

1. Execute **sem\_wait\_post.c** and understand the synchronization
2. Write a program for synchronization between three processes ie one parent and two children similar to sem\_wait\_post.
3. 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);

1. 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);