1) Describe at least 3 differences between Shared Memory and Mapped Memory

- Shmem use process memory to share the data / mmap use the file and map into the process memory and let others access that file
- Shmem use the key to access the memory / mmap use the file permission to determine the access
- Shmem may disappear after system shutdown / mmap will last until the file is removed

2) Describe at least 3 differences between Pipes and Message Queues

- Pipe is unidirectional / msgQ is bidirectional
- Pipe sends the stream of buffer / msgQ send the struct of data
- Pipe can handle the simple data / msgQ needs the first variable of struct to be long (data type) msgType

3.1) Two pipes are created before forking a process, how does the child process access the file descriptors of those pipes in order to communicate with its parent process?

Inheritance: After forking, the child process inherits the file descriptors of the pipes created by the parent process. These file descriptors are part of the child's file descriptor table.

File Descriptor Numbers: The child process knows the file descriptor numbers of the pipes minherited from the parent process. Typically, these file descriptor numbers are stored in variables or passed as arguments to functions.

Using Standard I/O Functions: The child process can use the file descriptor numbers to perform read and write operations on the pipes. Standard I/O functions such as read() and write() are used to read from and write to the pipes using the file descriptor numbers.

3.2) What is the purpose of the statements in line 30 to 32?

3.3) Describe the main purpose of the program

3.4) What could be the results from the given program?

4) Describe the meaning of Deadlock and Starvation.

Deadlock is when two or more processes are stuck because each is waiting for the other to release a resource. A locks, wait for B signal. B locks, wait for A signal.

Starvation is when a process keeps waiting to access a resource, like CPU time or memory, but other processes keep getting priority, so it never gets a chance to execute or complete its task.happens in LIFO(Last In First Out) (which is stack)

When two processes (a, b) are both waiting on a semaphore, and “a” cannot proceed until “b” signals, and “b” cannot continue until “a” signals. They are both asleep, waiting. Neither can signal the other, wake the other up. This is called a *deadlock*.

- P1 locks “a” which succeeds, then waits on “b”
- P2 locks “b” which succeeds, then waits on “a”

Indefinite blocking, or *starvation*, occurs when one process is constantly in a wait state, and is never signaled. This often occurs in LIFO situations.

5)

```
void* file_memory;

……(1)…… (time (NULL));
fd = ……(2)…… (argv[1], O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
……(3)…… (fd, FILE_LENGTH+1, SEEK_SET);
write (fd, ……(4)……, 1);
lseek (fd, 0, SEEK_SET);
file_memory = ……(5)…… (0, FILE_LENGTH, PROT_WRITE, MAP_SHARED, fd, 0);
……(6)…… (fd);
sprintf((char*) file_memory, “%d\n”, ……(7)…… (-100, 100));
……(8)…… (file_memory, FILE_LENGTH);
return 0;
}
```

5.1) srand
5.2) open
5.3) lseek
5.4) “”
5.5) mmap
5.6) close
5.7) (if function is declared : random_range; else rand;)
5.8) munmap

6.1 line 26, signal type
line 34, use for terminate the process by typr Ctrl+C
line 41, signal define user2
line 44, signal define user1

6.2 line 15, send the signal to user 2
line 19, send the signal to user 1
line 42, send the signal to user 1 to go to function fn

6.3 When “O” printed, it will send the signal to user 2. When user 2 woke up, it will print “X” and send the signal to user 1. And when the processes of user 1 is terminated, it will print "O is terminated" and when the process of user 2 is terminated, it will printed "X is terminated".

6.4 OXOXOOXOXXXXO //I think it might be random between O and X. If the user type Ctrl+C, it will print X is termianted and O is terminated and exit the program.

6.1)xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
6.2) What is the purpose of the “kill” function in such line 15, 19 and 42
Send signal to some other processes

6.3) Describe the purpose of the program
6.4) What would be the outputs from the given program?

7) Write a program that creates two new threads, where each thread prints a different pair of characters with a different number of times on a terminal. Each created thread must join the main thread after it has completed its task. The first thread must print “OO” 10 times and the second thread must print “YY” 20 times Both threads need to use the same thread function to print the characters. The program must xxxxxxxxxxxx

Posix named sem

```
struct params{
  sem_t *semID;
  int count;
  char letters[3];
};

void *func(void *args){
  struct params *p = (struct params *) args;

  for(int i = 0; i < p->count; i++){
    sem_wait(p->semID);
    printf("%s", p->letters);
    fflush(stdout);
    sem_post(p->semID);
  }
  printf("\n");

  return NULL;
}

int main(){
  struct params args1;
  struct params args2;

  args1.semID = sem_open("/mysem1", O_CREAT | O_RDWR, 0666, 1);
  args1.count = 10;
```

```
  strcpy(args1.letters, "OO");

  args2.semID = sem_open("/mysem2", O_CREAT | O_RDWR, 0666, 1);
  args2.count = 20;
  strcpy(args2.letters, "YY");

  pthread_t tid1, tid2;

  pthread_create(&tid1, NULL, &func, &args1);
  pthread_create(&tid2, NULL, &func, &args2);

  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);

  sem_unlink("/mysem1");
  sem_unlink("/mysem2");

  return 0;
}
```

This one doesnt use semaphore

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
/*Write a code using fork() to create thread to print OO 10times XX 20times using
semaphore(flag) to make printing alternatingly. Only use 1 thread function. [create struct for
inputs, parse to semaphore in thread function. Also use start and join thread]*/

struct thread_args{
  char* c;
  int flag;
  int count;
};

void *print_function(void *parameters){
  struct thread_args* p = (struct thread_args*) parameters;
  if(p->flag==0){
```

```
    for(int i=0; i<p->count;i++){
      fputs(p->c, stderr);
      sleep(2);
    }
  }else{
    for(int i=0; i<p->count;i++){
      fputs(p->c, stderr);
      sleep(2);
    }
  }
  return 0;
}

int main(){
  pthread_t thread1, thread2;
  struct thread_args args1, args2;
  char* result;
  pid_t pid;
  pid = fork();
  if(pid == 0){ //child
    args1.c= "OO";
    args1.flag=0;
    args1.count=10;
    pthread_create(&thread1, NULL, &print_function,&args1);
    pthread_join(thread1, NULL);
  }else{ //parent
    args2.c="XX";
    args2.flag=1;
    args2.count=20;
    pthread_create(&thread2, NULL, &print_function,&args2);
    pthread_join(thread2, NULL);
    wait(NULL);
  }


  return 0;
}
```