# OS

*Lab Exam*

**Registration Number:** _____
**Name:** _____

**Time:** 90 minutes
**Total Marks:** 75
**Approximate time required:** 80 minutes

**Note:** The timings with each question are given to guide you about the average time required for the question (if you know it. If you don't, don't waste time on it).

## Q1: Answer the following question precisely and briefly: (marks 10) time 10 min

### Why do we need to study Operating Systems?

Because operating systems are the heart of all modern devices, especially communication devices. Communication devices use protocols and software level features borrowed from the operating systems. Hence it is essential that a cs graduate should know about the operating systems.

### Why study Linux in an Operating Systems course?

Because of the openness of the linux operating system. The availability of source-code and freeness with which you can modify and study it is good for the course.

### Why not study Microsoft Windows?

Because its more of a consumer oriented operating system designed to be more userfreindly. It does not provide as many features as Linux for studying its architecture and working. For the most part it does not allow you to play with deep down system level things.

### What is a Safe State and what is its use in deadlock avoidance??

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state. System is in safe state if there exists a safe sequence of all processes. Deadlock Avoidance: ensure that a system will never enter an unsafe state.

### What facilities does a kernel provide? What are they called?

Kernel manages the devices and processes. It provides a set of functions called system calls.

### Q2: What is the difference between "an argument" and "a switch" in a typical Linux command? What is the importance of each? How to use them? Time 2 min (4 marks)

Arguments are supplied as is where as switches are preceded by a '-'. For example in a sort command, 'sort myfile –r –n' , myfile is the argument and –r –n are the switches telling the "mode" in which to work. So switches define "what" to do with the arugments. Most linux commands follow this convention. Another example is 'pstree –hpc'. This takes no arguments but has 3 switches merged. It will show all processes with their process id, highlight the current process in the tree and expand child processes.

**Q3: What do the following commands and their arguments and switches do?**
**(12 marks) time 5 min**

**Uniq**

The uniq (unique) utility displays a file, skipping adjacent duplicate lines, but does not change the original file.

**cat**

Displays a Text File content

**passwd**

Change your login password .

**cd ..**

change the directory to current working directory (basically do nothing)

**file**

file utility tells about the contents of any file on a Linux system without having to open and examine the file yourself.

**gedit newfile.cpp &**

start a new instance of gedit (notepad) with newfile.cpp (if newfile.cpp exists it will be opened) and put this process in the background giving back the prompt.

**Q4: What is execvp() supposed to do? Give an example of its use**
**(4 marks) time 3 min**

Execvp replaces the current process with another application's code. All open file descriptors are kept intact, the process id is kept intact, just the code is replaced and the instruction pointer is reset to 0. The stack is lost as well. We need to give the application's path, name and its arguments.

**Q5: Shell Scripting**

**(10 marks) time 15 min**

**Write a script which does the following:**

- **Display the total number of arguments passed in the command line.**
- **If the second argument is greater than 10**
- **Display the string we are studying "operating systems " in summer.**
- **If the third argument is not 10**
- **Display the following string using for loop**

**Pid:1 Pid:5 Pid10 Pid:21**

- **Display all the arguments passed.**

```
echo  $#
if test $2 –gt 10
then
echo we are studying `"operating system"` in summer
fi
if test $3 –ne 10
then
for  i in 1 5 10 21
do
echo pid $i
done
```

```
fi
echo $*
```

**Q6: argc, argv**

**(5 marks) time 10 min**

**If arguments passed are more than two, do the following:**

**Add 2 to the 2nd argument and display the result.**

```
int main(inr argc, char * argv[ ])
{
        If(argc>2)
        {
        int a=atoi(argv [ 1]);
        a=a+2;
        cout<<a;
        }
        return 0;
}
```

**Q7: pipes**

**(10 marks) time 10 min**

**Create a pipe which is accessible in both parent and child.**

**The parent should ONLY be allowed to read and the child should ONLY be allowed to write.**

```
int main()
```

```
{
        int pfds[2];
        pipe(pfds);
        if(fork())
        {
                close(pfds[1]) ; //for parent
        }
        else
        {
                close(pfds[0]) ; //for parent
        }
        return 0;
}
```

**Q8:Concurrency problem: Dining Philosophers.**

**(10 marks) time 15 min**

**The goal of this exercise is to implement a solution to the Dining Philosophers problem using monitors instead of semaphores. Create a method Dine(), which waits until a diner has two chopsticks and can eat, then calls Eat(), and then releases the chopsticks before returning. Your solution should allow multiple philosophers to eat at the same time (as long as there are sufficient chopsticks in a pile in the middle of the table).**

```
State variables:
chopsticks Number of chopsticks left Initial value N
Monitor: waitingP
Lock: lock
```

```
Dine() {
lock.acquire();
while (chopsticks < 2) {
waitingP.wait()
}
chopsticks -= 2;
lock.release();
Eat();
lock.acquire();
chopsticks +=2;
waitingP.broadcast(); or signal
loc
```

**Q9:Print Ping Pong ten times by using threads and semaphore .**

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
sem_t s1,s2;
void *ping(void*)
      {
      for(int i=0;i<10;i++)
            {
            sem_wait(&s1);
            printf("ping ");
            sem_post(&s2);
            }
      }
void *pong(void*)
      {
      for(int i=0;i<10;i++)
            {
            sem_wait(&s2);
            printf("pong \n");
            sem_post(&s1);
            }
      }
int main()
      {
      pthread_t t1,t2;
      sem_init(&s1,0,1);
      sem_init(&s2,0,0);
      pthread_create(&t1,NULL,&pong,NULL);
      pthread_create(&t2,NULL,&ping,NULL);
      pthread_join(t1,NULL);
      pthread_join(t2,NULL);
      return 0;
      }
```