MEERUT INSTITUTE OF ENGINEERING & TECHNOLOGY, MEERUT

Department of Computer Science & Technology

B. Tech. (Session 2014– 15)

Lab Manual

Operating System Lab(NCS-451)

Faculty Name:

Mr. Davesh Singh Som

Ms. Nidhi Chaudhary

INDEX

INTRODUCTION TO OPERATING SYSTEM

An operating system is a layer of software which takes care of technical aspects of a computer's operation. It shields the user of the machine from the low-level details of the machine's operation and provides frequently needed facilities. There is no universal definition of what an operating system consists of. You can think of it as being the software which is already installed on a machine, before you add anything of your own. Normally the operating system has a number of key elements: (i) a *technical layer of software* for driving the hardware of the computer, like disk drives, the keyboard and the screen; (ii) a *filesystem* which provides a way of organizing files logically, and (iii) a simple *command language* which enables users to run their own programs and to manipulate their files in a simple way. Some operating systems also provide text editors, compilers, debuggers and a variety of other tools. Since the operating system (OS) is in charge of a computer, all requests to use its resources and devices need to go through the OS. An OS therefore provides legal *entry points* into its code for performing basic operations like writing to devices.

Operating systems may be classified by both how many tasks they can perform `simultaneously' and by how many users can be using the system `simultaneously'. That is: single-user *or* multi-user *and* single-task *or* multi-tasking. A multi-user system must clearly be multi-tasking. The table below shows some examples.

OS	Users	Tasks	Processors
MS/PC DOS	S	S	1
Windows 3x	S	QM	1
Macintosh System 7.*	S	QM	1
Windows 9x	S	M*	1
AmigaDOS	S	M	1

hline MTS	M	M	1
UNIX	M	M	n
VMS	M	M	1
NT	S/M	M	n
Windows 2000	M	M	n
BeOS (Hamlet?)	S	M	n

The first of these (MS/PC DOS/Windows 3x) are single user, single-task systems which build on a ROM based library of basic functions called the BIOS. These are system calls which write to the screen or to disk etc. Although all the operating systems can service *interrupts*, and therefore simulate the appearance of multitasking in some situations, the older PC environments cannot be thought of as a multi-tasking system in any sense. Only a single user application could be open at any time. Windows 95 replaced the old coroutine approach of quasi-multitasking with a true context switching approach, but only a single user system, without proper memory protection. Windows NT added a proper kernel with memory protection, based on the VMS system, originally written for the DEC/Vax. Later versions of Windows NT and Windows 2000 (a security and kernel enhanced version of NT) allow multiple logins also through a terminal server. Windows 2000 thus has comparable functionality to Unix in this respect.

The Macintosh system 7 can be classified as single-user quasi-multitasking. That means that it is possible to use several user applications simultaneously. A window manager can simulate the appearance of several programs running simultaneously, but this relies on each program obeying specific rules in order to achieve the illusion. The Macintosh not a true multitasking system in the sense that, if one program crashes, the whole system crashes. Windows $9x$ is purported to be preemptive multitasking but most program crashes also crash the entire system. This might be due to the lack of proper memory protection. The claim is somewhat confusing.

AmigaDOS is an operating system for the Commodore Amiga computer. It is based on the UNIX model and is a fully multi-tasking, single-user system. Several programs may be actively running at any time. The operating system includes a window environment which means that each independent program has a `screen' of its own and does not therefore have to compete for the screen with other programs. This has been a major limitation on multi-tasking operating systems in the past.

MTS (Michigan timesharing system) was the first time-sharing multi-user system. It supports only simple single-screen terminal based input/output and has no hierarchical file system

MAJOR FUNCTIONS OF OPERATING SYSTEM

The major functions of an OS are:

-resource management,

-data management,

-job (task) management, and

-standard means of communication between user and computer.

-The resource management function of an OS allocates computer resources such as CPU time, main memory, secondary storage, and input and output devices for use.

-The data management functions of an OS govern the input and output of the data and their location, storage, and retrieval.

-The job management function of an OS prepares, schedules, controls, and monitors jobs submitted for execution to ensure the most efficient processing. A job is a collection of one or more related programs and their data.

-A job is a collection of one or more related programs and their data.

The OS establishes a standard means of communication between users and their computer -systems. It does this by providing a user interface and a standard set of commands that control the hardware.

Typical Day-to-Day Uses of an Operating System

-Executing application programs.

-Formatting floppy diskettes.

-Setting up directories to organize your files.

-Displaying a list of files stored on a particular disk.

-Verifying that there is enough room on a disk to save a file.

-Protecting and backing up your files by copying them to other disks for safekeeping

Recommended Systems/Software Requirements:

- Intel based desktop PC with minimum of 1.66 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.

- Linux Operating System.

MEERUT INSTITUTE OF ENGINEERING & TECHNOLOGY, MEERUT
N.H. 58, Delhi-Roorkee Highway, Baghpat Road Bypass Crossing, Meerut-250005

Computer Science & Engineering Department
B. Tech. 2nd Year/ 4th Semester
(Session: 2014 – 15)

Operating System Lab (NCS-451)

L	T	P
0	0	2

1. Write Basic Commands for VI-Editor used in Linux.
2. Write a basic program on VI Editor in Linux.
3. Write a program to show the process ID and parent process ID of a process.
4. Write a program to create a child process using fork () system call.
5. Write a program to create child process and verify the child' s parent ID and parent process ID.
6. Write a program to show the orphan process concept.
7. Write a program to show the status of processes [Zombie(Z), sleeping(S), running(R)].
8. Write a program to show the concurrent execution of child and parent process using fork() system call.
9. Write a program to simulate FCFS scheduling algorithm without arrival time.
10. Write a program to simulate FCFS scheduling algorithm with arrival time.
11. Write a program to simulate Shortest Job First (SJF) scheduling algorithm without arrival time.
12. Write a Program to simulate Non- Preemptive Priority Scheduling Algorithm.
13. Write a program to simulate Priority Scheduling Algorithm with arrival time.
14. Write a Program to simulate Round Robin Scheduling Algorithm.

Programs

1. Write Basic Commands for VI-Editor used in Linux.

Commands	Description
i	Instruction mode
a	Append to right mode
/word	Move to the occurrence of "word"
n	Locate next occurrence
w	Advance to next mode
e	Advance to the next end of the word
b	Move to the previous word
3b	Move backwards 3 words
YY	Delete line
3dd	Delete 3 lines
D	Deletes remainder of a line
dw	Deletes a word
X	Delete character
o (small)	Open space for new line below the cursor line
O (big)	Opens a line above the cursor
Ctrl-w	Move back a word in append mode
u (small)	Undo last
U	Undo all changes to current line
Esc	Command mode
:w new file name	Save the file to the new file name from the command mode
:wq	Save and quit
q!	Quit without saving
r	Replaces then type a character to be replaced with r then return to break up a line
J	Joins 2 lines
S	Substitute (sentence) type text over a character, Esc when done
cw	Change word
c	Change part of line from the cursor to the end of line
cc	Substitute new text for a line, Esc when done
h	Move the cursor back one space
H	Move the cursor to the highest line on the space
L	Move the cursor to the lowest line on the screen
M	Position the cursor at the midpoint of the screen
G	Last line in the file
0 (ZERO)	Move the cursor to the beginning of the line it is on
View filename	Open a file for view only
set number	Turn on line number
set no number	Turn on line number
:n	Access the next file for editing
:wq filename.c	save command

gcc space – o space filename space filename.c	compilation command
./filename	run command
Vi	return terminal(new)
Vi space filename.c	(return program)

2. Write a basic program on VI Editor in Linux.

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

main ()
{
   int n;

   printf("Enter an integer\n");

  scanf("%d",&n);

   if ( n%2 == 0 )
      printf("Even\n");
   else
      printf("Odd\n");

   return 0;
}
```

3. Write a program to show the process ID and parent process ID of a process.

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

main ()
{
   int pid;
   pid=getpid();
   ppid=getppid();
   printf("id of the process is=%d\n", getpid());

   printf("id of the process is=%d\n", getppid());
 }
```

4. Write a program to create a child process using fork () system call.

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

main ()
{
   int pid;
   pid=fork();
   printf("id of the process is=%d\n", getpid());
   printf(□hello□);


 }
```

5. Write a program to create child process and verify the child' s parent ID and parent process ID.

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

```
main ()
{
   int pid;
   pid=fork();
   if(pid==0)
   {
   printf("id of the child  process is=%d\n", getpid());

   printf("id of the parent  process is=%d\n", getppid());
   }
 else
 {
   printf("id of the parent  process is=%d\n", getpid());

   printf("id of the parent of parent  process is=%d\n", getppid());
 }
}
```

6. Write a program to show the orphan process concept.

```
#include <stdio.h>
#include <sys/types.h>
main ()
{
   int pid;
   pid=fork();
   if(pid==0)
   {
   printf("id of the child  process is=%d\n", getpid());

   printf("id of the parent  process is=%d\n", getppid());
    sleep(30);
   }
 else
 {
   printf("id of the parent  process is=%d\n", getpid());

   printf("id of the parent of parent  process is=%d\n", getppid());
 }
}
```

7. Write a program to show the status of processes [Zombie(Z), sleeping(S), running(R)].

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

main ()
{
   int pid;
   pid=fork();
if(pid>0)
{
   printf("id of the process is=%d\n", getpid());
```

```
    sleep(20);
  }
```

for(; ;)

{

}

8. Write a program to show the concurrent execution of child and parent process using fork() system call.

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

#define   MAX_COUNT  200

void  ChildProcess(void);                /* child process prototype  */
void  ParentProcess(void);               /* parent process prototype */

void  main(void)
{
pid_t  pid;

pid = fork();
if (pid == 0)
ChildProcess();
else
ParentProcess();
 }

void  ChildProcess(void)
{
     int   i;

      for (i = 1; i <= MAX_COUNT; i++)
      printf("   This line is from child, value = %d\n", i);
       printf("   *** Child process is done ***\n");
 }

void  ParentProcess(void)
{
      int   i;

      for (i = 1; i <= MAX_COUNT; i++)
     printf("This line is from parent, value = %d\n", i);
      printf("*** Parent is done ***\n");
}
```

9. Write a program to simulate FCFS scheduling algorithm without arrival time

```
//first program of OS : FCFS
#include<stdio.h>
#include<string.h>
void main()
{
   char pro[20][6];
   int i=0,j=0,k=0,prot[20],no,wt=0,ta=0,ta1=0;
   printf("Enter the number of process (less than 20) you want to enter\n");
   scanf("%d",&no);
   printf("Enter the process name (max 5 characters) and burst time\n");
   for(i=0;i<no;i++)
   {
         printf("name : ");
         scanf("%s",&pro[i]);
         printf("time : ");
         scanf("%d",&prot[i]);
   }
   for(i=0;i<no;i++)
   printf("--------");
   printf("\n");
   for(i=0,j=0;i<no;i++,j++)
   {
         printf("|");
         printf(" %s\t",pro[i]);
   }
   printf("|\n");
   for(i=0;i<no;i++)
   printf("--------");
   printf("\n");
```

```
   for(i=0,j=0;i<=no;i++)
   {
         printf("%d\t",j);
         if(i<no)
         wt=wt+j;
         j=j+prot[i];
   }
   for(i=0,j=0;i<no;i++)
   {
         ta=ta+prot[i];
         printf("\nWaiting time for process %s is : %d, ",pro[i],j);
         printf("Turn Around time for process %s is : %d",pro[i],ta);
         ta1=ta1+ta;
         if(i<no)
         j=j+prot[i];
   }
   printf("\nAverage waiting time is : %d/%d = %f",wt,no,(float)wt/no);
   printf("\nAverage Turn Around time is : %d/%d = %f\n",ta1,no,(float)ta1/no);
}
```

10.Write a program to simulate FCFS scheduling algorithm with arrival time

```
  //second program of OS : FCFS with arrival time
  #include<stdio.h>
  #include<string.h>
  void main()
 {
   int i=0,j=0,k=0,maxat,no,rt=0,wt=0,ta=0,ta1=0;
   struct pro
   {
         char pn[6];
         int at;
         int bt;
    } proc[20],temp;
```

```
       printf("Enter the number of process (less than 20) you want to enter\n");
       scanf("%d",&no);
       printf("Enter the process name (max 5 characters), arrival time and burst time\n");
      for(i=0;i<no;i++)
      {
            printf("name : ");
            scanf("%s",&proc[i].pn);
            printf("arrival time : ");
            scanf("%d",&proc[i].at);
            printf("burst time : ");
            scanf("%d",&proc[i].bt);
      }
      maxat=0;
      for(i=0;i<no-1;i++)
      {
            for(j=0;j<no-i-1;j++)
            {
                  if(proc[j].at>proc[j+1].at)
                  {
                        temp=proc[j];
                        proc[j]=proc[j+1];
                        proc[j+1]=temp;
                  }
            }
      }
      for(i=0;i<no;i++)
      {
```

```
      if(proc[i].at>maxat)
      maxat=proc[i].at;
}
for(i=0;i<no;i++)
printf("--------");
printf("\n");
for(i=0;i<no;i++,j++)
{
      printf("|");
      printf(" %s\t",proc[i].pn);
}
printf("|\n");
for(i=0;i<no;i++)
printf("--------");
printf("\n");
for(i=0;i<=no;i++)
{
      printf("%d\t",rt);
      if(i<no)
      {
            wt=wt+rt;
            if(i>0)
            wt=wt-proc[i].at;
      }
      rt=rt+proc[i].bt;
}
rt=0;
```

```
      for(i=0,j=0;i<no;i++)

      {

            ta=rt+proc[i].bt-proc[i].at;

            printf("\nWaiting time for process %s is : %d, ",proc[i].pn,rt-
proc[i].at);

            printf("Turn Around time for process %s is : %d",proc[i].pn,ta);

            ta1=ta1+ta;

            rt=rt+proc[i].bt;

      }

      printf("\nAverage waiting time is : %d/%d = %f",wt,no,(float)wt/no);

      printf("\nAverage Turn Around time is : %d/%d = %f\n",ta1,no,(float)ta1/no);

}
```

11. Write a program to simulate Shortest Job First (SJF) scheduling algorithm without arrival time

```
  #include<stdio.h>
  #include<string.h>
  void main()
 {
int i=0,j=0,k=0,maxat,prod[15],no,rt=0,wt=0,ta=0,ta1=0;
struct pro
{
      char pn[6];
      int bt;
} proc[20],temp;
printf("Enter the number of process (less than 20) you want to enter\n");
scanf("%d",&no);
printf("Enter the process name (max 5 characters) and burst time\n");
for(i=0;i<no;i++)
{
      printf("name : ");
      scanf("%s",&proc[i].pn);
      printf("burst time : ");
      scanf("%d",&proc[i].bt);
}
maxat=0;
for(i=0;i<no-1;i++)
{
      for(j=0;j<no-i-1;j++)
      {
```

```
            if(proc[j].bt>proc[j+1].bt)
            {
                  temp=proc[j];
                  proc[j]=proc[j+1];
                  proc[j+1]=temp;
            }
      }
}
for(i=0;i<no;i++)
{
      if(proc[i].bt>maxat)
      maxat=proc[i].bt;
}
for(i=0;i<no;i++)
printf("--------");
printf("\n");
for(i=0;i<no;i++,j++)
{
      printf("|");
      printf(" %s\t",proc[i].pn);
}
printf("|\n");
for(i=0;i<no;i++)
printf("--------");
printf("\n");
for(i=0;i<=no;i++)
{
      printf("%d\t",rt);
      if(i<no)
      {
            wt=wt+rt;
      }
      rt=rt+proc[i].bt;
}
rt=0;
for(i=0,j=0;i<no;i++)
{
      ta=rt+proc[i].bt;
      printf("\nWaiting time for process %s is : %d, ",proc[i].pn,rt);
      printf("Turn Around time for process %s is : %d",proc[i].pn,ta);
      ta1=ta1+ta;
      rt=rt+proc[i].bt;
}
printf("\nAverage waiting time is : %d/%d = %f",wt,no,(float)wt/no);
printf("\nAverage Turn Around time is : %d/%d = %f\n",ta1,no,(float)ta1/no);
}
```

12. **Write a program to simulate Priority Scheduling Algorithm without arrival time.**

```
#include<stdio.h>
#include<string.h>
struct pro
{
char pn[6];
int bt;
 int pr;
```

```
        } proc[20],temp;
          int procno=-1;
        void main()
         {
            int i=0,j=0,k=0,prod[15],no,rt=0,minpr=0,p=0,totalt=0,wt=0,ta=0,ta1=0;
             printf("Enter the number of process (less than 20) you want to
enter\n");
              scanf("%d",&no);
           printf("Enter the process name (max 5 characters), priority and burst
time\n");
      for(i=0;i<no;i++)
      {
            printf("name : ");
            scanf("%s",&proc[i].pn);
            printf("priority : ");
            scanf("%d",&proc[i].pr);
            printf("burst time : ");
            scanf("%d",&proc[i].bt);
            totalt=totalt+proc[i].bt;
      }
      for(i=0;i<no-1;i++)
      {
            for(j=0;j<no-i-1;j++)
            {
                  if(proc[j].pr>proc[j+1].pr)
                  {
                        temp=proc[j];
                        proc[j]=proc[j+1];
                        proc[j+1]=temp;
                  }
            }
      }
      for(i=0;i<no;i++)
      printf("--------");
      printf("\n");
      for(i=0;i<no;i++,j++)
      {
            printf("|");
            printf(" %s\t",proc[i].pn);

      }
      printf("|\n");
      for(i=0;i<no;i++)
      printf("--------");
      printf("\n");
      for(i=0;i<=no;i++)
      {
            printf("%d\t",rt);
            if(i<no)
            {
                  wt=wt+rt;
            }
            rt=rt+proc[i].bt;
      }
      rt=0;
      for(i=0,j=0;i<no;i++)
      {
            ta=rt+proc[i].bt;
            printf("\nWaiting time for process %s is : %d, ",proc[i].pn,rt);
```

```
            printf("Turn Around time for process %s is : %d",proc[i].pn,ta);
            ta1=ta1+ta;
            rt=rt+proc[i].bt;
      }
      printf("\nAverage waiting time is : %d/%d = %f",wt,no,(float)wt/no);
      printf("\nAverage Turn Around time is : %d/%d = %f\n",ta1,no,(float)ta1/no);
}
```

13.Write a program to simulate Priority Scheduling Algorithm with arrival time.

```
#include<stdio.h>
#include<string.h>
struct pro
{
      char pn[6];
      int at;
      int bt;
      int pr;
} proc[20],procf[20],temp;
int procfno=-1;
int check(struct pro s)
{
      int i;
      if(procfno==(-1))
      return 0;
      for(i=0;i<=procfno;i++)
      {
            if(strcmp(procf[i].pn,s.pn)==0)
            {
                  return 1;
            }
      }
      return 0;
}
void main()
{
      int i=0,j=0,k=0,prod[15],no,rt=0,minpr=0,p=0,totalt=0,wt=0,ta=0,ta1=0;
      printf("Enter the number of process (less than 20) you want to enter\n");
      scanf("%d",&no);
      printf("Enter the process name (max 5 characters), arrival time, priority and
burst time\n");
      for(i=0;i<no;i++)
      {
            printf("name : ");
            scanf("%s",&proc[i].pn);
            printf("arrival time : ");
            scanf("%d",&proc[i].at);
            printf("priority : ");
            scanf("%d",&proc[i].pr);
            printf("burst time : ");
            scanf("%d",&proc[i].bt);
            totalt=totalt+proc[i].bt;
      }
      for(i=0;i<no-1;i++)
      {
            for(j=0;j<no-i-1;j++)
```

```
	{
		if(proc[j].at>proc[j+1].at)
		{
			temp=proc[j];
			proc[j]=proc[j+1];
			proc[j+1]=temp;
		}
	}
}
for(i=0;i<totalt;)
{
	for(j=0;j<no;j++)
	{
		if(proc[j].at>i)
		break;
		else if(!(check(proc[j])))
		{
			if(p==0)
			{
				minpr=proc[j].pr;
				k=j;
				p=1;
			}
			if(proc[j].pr<minpr)
			{
				minpr=proc[j].pr;
				k=j;
			}
		}
	}
	if(!check(proc[k]))
	{
			procfno++;
			procf[procfno]=proc[k];
			i=i+procf[procfno].bt;
	}
	p=0;
}
for(i=0;i<no;i++)
printf("--------");
printf("\n");
for(i=0;i<no;i++,j++)
{
	printf("|");
	printf(" %s\t",procf[i].pn);

}
printf("|\n");
for(i=0;i<no;i++)
printf("--------");
printf("\n");
for(i=0;i<=no;i++)
{
	printf("%d\t",rt);
	if(i<no)
	{
		wt=wt+rt;
		if(i>0)
		wt=wt-procf[i].at;
```

```
            }
            rt=rt+procf[i].bt;
      }
      rt=0;
      for(i=0,j=0;i<no;i++)
      {
            ta=rt+procf[i].bt-procf[i].at;
            printf("\nWaiting time for process %s is : %d, ",procf[i].pn,rt-
procf[i].at);
            printf("Turn Around time for process %s is : %d",procf[i].pn,ta);
            ta1=ta1+ta;
            rt=rt+procf[i].bt;
      }
      printf("\nAverage waiting time is : %d/%d = %f",wt,no,(float)wt/no);
      printf("\nAverage Turn Around time is : %d/%d = %f\n",ta1,no,(float)ta1/no);
}
```

14.Write a Program to simulate Round Robin Scheduling Algorithm.

```
#include<stdio.h>
#include<string.h>
struct pro
{
      char pn[6];
      int bt,rt,wt,tat;
} proc[20],procf[40],temp;
int procfno=-1;
int check(struct pro s)
{
      int i;
      if(procfno==(-1))
      return 0;
      if(strcmp(procf[procfno].pn,s.pn)==0)
      return 1;
      return 0;
}
void main()
{
      int i=0,j=0,k=0,l=0,no,rt=0,p=0,totalt=0,ts=0,wt=0,m=0,ta1=0;
      printf("Enter the number of process (less than 20) you want to enter\n");
      scanf("%d",&no);
      printf("Enter the process name (max 5 characters) and burst time\n");
      for(i=0;i<no;i++)
      {
            printf("name : ");
            scanf("%s",&proc[i].pn);
            printf("burst time : ");
```

```
	scanf("%d",&proc[i].bt);
	totalt=totalt+proc[i].bt;
	proc[i].rt=proc[i].bt;
	proc[i].wt=0;
	proc[i].tat=1;
}
printf("Enter the time slice : ");
scanf("%d",&ts);
for(i=0,j=0;i<totalt;i++)
{
	if(proc[j].rt>0)
	{
		k=j;
		p=1;
	}
	else
	{
		j=(++j)%no;
		i--;
		continue;
	}
	if(check(proc[k])&&p!=0)
	{
			procf[procfno].rt++;
			proc[k].rt--;
			l++;
	}
	else if(p!=0)
	{
			procfno++;
			strcpy(procf[procfno].pn,proc[k].pn);
			procf[procfno].bt=proc[k].bt;
			procf[procfno].rt=1;
			proc[k].rt--;
			l=1;
	}
	for(m=0;m<no&&p!=0;m++)
	{
		if(proc[m].rt>0)
		proc[m].tat++;
		if(m==k)
		continue;
		if(proc[m].rt<=0)
		continue;
		proc[m].wt++;
	}
	p=0;
	if(l%ts==0)
	{
		j=(++j)%no;
	}
}
for(i=0;i<=procfno;i++)
printf("--------");
printf("\n");
for(i=0;i<=procfno;i++,j++)
{
	printf("|");
	printf(" %s\t",procf[i].pn);
```

```
	}
	printf("|\n");
	for(i=0;i<no;i++)
	{
		wt=wt+proc[i].wt;
		ta1=ta1+proc[i].tat;
	}
	for(i=0;i<=procfno;i++)
	printf("--------");
	printf("\n");
	for(i=0;i<=procfno+1;i++)
	{
		printf("%d\t",rt);
		rt=rt+procf[i].rt;
	}
	for(i=0,j=0;i<no;i++)
	{
		printf("\nWaiting time for process %s is : %d, ",proc[i].pn,proc[i].wt);
		printf("Turn Around time for process %s is : %d",proc[i].pn,proc[i].tat);
	}
	printf("\nAverage waiting time is : %d/%d = %f",wt,no,(float)wt/no);
	printf("\nAverage Turn Around time is : %d/%d = %f\n",ta1,no,(float)ta1/no);
}
```