OS lab manual

Exercise 1

Basic Linux commands

An image is forwarded write full

File manipulation commands

FILE MANIPULATION COMMANDS

a)cat–this create, view and concatenate files.

Creation:

Syn:$cat>filename

Viewing:

Syn:$cat filename

Add text to an existing file:

Syn:$cat>>filename

Concatenate:

Syn:$catfile1file2>file3

$catfile1file2>>file3 (no over writing of file3)

b)grep–used to search a particular word or pattern related to that word from the file.

Syn:$grep search word filename

Eg:$grep anu student

c)rm–deletes a file from the file system

Syn:$rm filename

d)touch–used to create a blank file.

Syn:$touch file names

e)cp–copies the files or directories

Syn:$cpsource file destination file

Eg:$cp student stud

f)mv–to rename the file or directory

syn:$mv old file new file

Eg:$mv–i student student list(-i prompt when overwrite)

g)cut–it cuts or pickup a given number of character or fields of the file.

Syn:$cut<option><filename>

Eg: $cut –c filename

$cut–c1-10emp

$cut–f 3,6emp

$ cut –f 3-6 emp

-c cutting columns

-f cutting fields

h)head–displays10 lines from the head(top)of a given file

Syn:$head filename

Eg:$head student

Syn:$head-2student

i)tail–displays last 10 lines of the file

Syn:$tail filename

Eg:$tail student

To display the bottom two lines;

Syn:$ tail -2 student

j)chmod–used to change the permissions of a file or directory.

Syn:$ch mod category operation permission file

Where, Category–is the user type

Operation–is used to assign or remove permission

Permission–is the type of permission

File–are used to assign or remove permission all.

Exercise 2

To write C Programs using the following system calls of UNIX operating system fork, exec,

getpid, exit, wait, close, stat, opendir, readdir.

1. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (OPENDIR, READDIR, CLOSEDIR)

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create struct direct.

STEP 3: declare the variable buff and pointer dptr.

STEP 4: Get the directory name.

STEP 5: Open the directory.

STEP 6: Read the contents in directory and print it.

STEP 7: Close the directory.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc, char *argv[])
{
char buff[100];
DIR *dirp;
printf("\n\n ENTER DIRECTORY NAME");
scanf("%s", buff);
if((dirp=opendir(buff))==NULL)
{
printf("The given directory does not exist");
exit(1);
}
while(dptr=readdir(dirp))
{
printf("%s\n",dptr->d_name);
}
closedir(dirp);
}
```

2. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEM

(fork, getpid, exit)

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables pid,pid1,pid2.

STEP 3: Call fork() system call to create process.

STEP 4: If pid==-1, exit.

STEP 5: Ifpid!=-1 , get the process id using getpid().

STEP 6: Print the process id.

STEP 7:Stop the program

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
main()
{
int pid,pid1,pid2;
pid=fork();
if(pid==-1)
{
printf("ERROR IN PROCESS CREATION \n");
exit(1);
}
if(pid!=0)
{
pid1=getpid();
printf("\n the parent process ID is %d\n", pid1);
}
else
{
pid2=getpid();
```

```
printf(“\n the child process ID is %d\n”, pid2);
}
}
```

Chapter 2 shell scripts

1.Write a Shell program to check the given number is even or odd

ALGORITHM:

SEPT 1: Start the program.

STEP 2: Read the value of n.

STEP 3: Calculate „r=expr $n%2“.

STEP 4: If the value of r equals 0 then print the number is even

STEP 5: If the value of r not equal to 0 then print the number is odd.

PROGRAM:

```
echo "Enter the Number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
echo "$n is Even number"
else
echo "$n is Odd number"
fi
```

2.Write a Shell program to check the given year is leap year or not

ALGORITHM:

SEPT 1: Start the program.

STEP 2: Read the value of year.

STEP 3: Calculate „b=expr $y%4“.

STEP 4: If the value of b equals 0 then print the year is a leap year

STEP 5: If the value of r not equal to 0 then print the year is not a leap year.

PROGRAM:

```
echo "Enter the year"
read y
b=`expr $y % 4`
if [ $b -eq 0 ]
then
echo "$y is a leap year"
else
echo "$y is not a leap year"
fi
```

3.Write a Shell program to find the factorial of a number

ALGORITHM:

SEPT 1: Start the program.

STEP 2: Read the value of n.

STEP 3: Calculate „i=expr $n-1“.

STEP 4: If the value of i is greater than 1 then calculate „n=expr $n * $i“ and „i=expr $i – 1“

STEP 5: Print the factorial of the given number.

PROGRAM:

```
echo "Enter a Number"
read n
i=`expr $n - 1`
p=1
while [ $i -ge 1 ]
do
n=`expr $n \* $i`
i=`expr $i - 1`
done
echo "The Factorial of the given Number is $n"
```

1. **To write a C program for implementation of Priority scheduling algorithms.**

ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign p and allocate the memory.

Step 4: Inside the for loop get the value of burst time and priority.

Step 5: Assign wtime as zero .

Step 6: Check p[i].pri is greater than p[j].pri .

Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.

Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
int pno;
int pri;
int pri;
int btime;
int wtime;
}sp;
int main()
{
int i,j,n;
int tbm=0,totwtime=0,totttime=0;
sp *p,t;
printf("\n PRIORITY SCHEDULING.\n");
printf("\n enter the no of process....\n");
scanf("%d",&n);
p=(sp*)malloc(sizeof(sp));
```

```
printf("enter the burst time and priority:\n");
for(i=0;i<n;i++)
{
printf("process%d:”,i+1);
scanf("%d%d",&p[i].btime,&p[i].pri);
p[i].pno=i+1;
p[i].wtime=0;
}
for(i=0;i<n-1;i++)
for(j=i+1;j<n;j++)
{
if(p[i].pri>p[j].pri)
{
t=p[i];
p[i]=p[j];
p[j]=t;
}
}
printf("\n process\tbursttime\twaiting time\tturnaround time\n");
for(i=0;i<n;i++)
{
totwtime+=p[i].wtime=tbm;
tbm+=p[i].btime;
printf("\n%d\t\t%d",p[i].pno,p[i].btime);
printf("\t\t%d\t\t%d",p[i].wtime,p[i].wtime+p[i].btime);
}
totttime=tbm+totwtime;
printf("\n total waiting time:%d",totwtime);
printf("\n average waiting time:%f",(float)totwtime/n);
printf("\n total turnaround time:%d",totttime);
```

```
printf("\n avg turnaround time:%f",(float)totttime/n);
}
```

2. To write a C-program to implement the producer – consumer problem using semaphores.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Initialize the buffer size and get maximum item you want to produce.

Step 4: Get the option, which you want to do either producer, consumer or exit from the operation.

Step 5: If you select the producer, check the buffer size if it is full the producer should not produce the item or otherwise produce the item and increase the value buffer size.

Step 6: If you select the consumer, check the buffer size if it is empty the consumer should not

consume the item or otherwise consume the item and decrease the value of buffer size.

Step 7: If you select exit come out of the program.

Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1) {
printf("\nENTER YOUR CHOICE\n");
scanf("%d",&n);
```

```
switch(n)
{ case 1:
if((mutex==1)&&(empty!=0))
producer();
else
printf("BUFFER IS FULL");
break;
case 2:
if((mutex==1)&&(full!=0))
consumer();
else
printf("BUFFER IS EMPTY");
break;
case 3:
exit(0);
break;
}
}
}
int wait(int s) {
return(--s); }
int signal(int s) {
return(++s); }
void producer() {
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\nproducer produces the item%d",x);
mutex=signal(mutex); }
```

```
void consumer() {
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes item%d",x);
x--;
mutex=signal(mutex);
}
```

3.To write a c program to implement IPC using shared memory.

ALGORITHM:

Step 1: Start the process

Step 2: Declare the segment size

Step 3: Create the shared memory

Step 4: Read the data from the shared memory

Step 5: Write the data to the shared memory

Step 6: Edit the data

Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#define SEGSIZE 100
int main(int argc, char *argv[ ])
{
```

```
int shmid,cntr;
key_t key;
char *segptr;
char buff[]="poooda.....";
key=ftok(".",'s');
if((shmid=shmget(key, SEGSIZE, IPC_CREAT | IPC_EXCL | 0666))== -1)
{
if((shmid=shmget(key,SEGSIZE,0))==-1)
{
perror("shmget");
exit(1);
}
}
else
{
printf("Creating a new shared memory seg \n");
printf("SHMID:%d",shmid);
}
system("ipcs –m");
if((segptr=(char*)shmat(shmid,0,0))==(char*)-1)
{
perror("shmat");
exit(1);
}
printf("Writing data to shared memory...\n");
strcpy(segptr,buff);
printf("DONE\n");
printf("Reading data from shared memory...\n");
printf("DATA:-%s\n",segptr);
printf("DONE\n");
```

```
printf("Removing shared memory Segment...\n");
if(shmctl(shmid,IPC_RMID,0)== -1)
printf("Can‟t Remove Shared memory Segment...\n");
else
printf("Removed Successfully");
}
```

4. To write a C program to implement bankers algorithm for deadlock avoidance.

ALGORITHM:

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and available matrix.

Step-4: Compare each and every process using the banker‟s algorithm.

Step-5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process

Step-6: produce the result of state of process

Step-7: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
```

```
printf("********** Baner's Algo **********\n");
input();
show();
cal();
getch();
return 0;
}
void input()
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resources instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++) {
for(j=0;j<r;j++) {
scanf("%d",&max[i][j]);
}}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++) {
for(j=0;j<r;j++) {
scanf("%d",&alloc[i][j]);
}}
printf("Enter the available Resources\n");
for(j=0;j<r;j++) {
scanf("%d",&avail[j]);
}}
void show() {
int i,j
```

```
;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++) {
printf("\nP%d\t ",i+1);
for(j=0;j<r;j++) {
printf("%d ",alloc[i][j]); }
printf("\t");
for(j=0;j<r;j++) {
printf("%d ",max[i][j]); }
printf("\t");
if(i==0) {
for(j=0;j<r;j++)
printf("%d ",avail[j]);
}}}
void cal()
{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
int safe[100];
int i,j;
for(i=0;i<n;i++) {
finish[i]=0; }
//find need matrix
for(i=0;i<n;i++) {
for(j=0;j<r;j++) {
need[i][j]=max[i][j]-alloc[i][j];
}}
printf("
\n");
while(flag) {
flag=0;
```

```
for(i=0;i<n;i++) {
int c=0;
for(j=0;j<r;j++) {
if((finish[i]==0)&&(need[i][j]<=avail[j])) {
c++;
if(c==r) {
for(k=0;k<r;k++) {
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1; }
printf("P%d->",i);
if(finish[i]==1) {
i=n;
}}}}}}
for(i=0;i<n;i++) {
if(finish[i]==1) {
c1++;
}
else
{printf("P%d->",i);
}}
if(c1==n)
{printf("\n The system is in safe state");
}
Else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}}
```