## AD52: Operating System Laboratory Programs

1. **date**

–used to check the date and time Syn:$date

|  |  |  |  |
| --- | --- | --- | --- |
| Format | Purpose | Example | Result |
| +%m | To display only month | $date+%m | 06 |
| +%h | To display month name | $date+%h | June |
| +%d | To display day of month | $date+%d | O1 |
| +%y | To display last two digits of years | $date+%y | 09 |
| +%H | To display hours | $date+%H | 10 |
| +%M | To display minutes | $date+%M | 45 |
| +%S | To display seconds | $date+%S | 55 |

### cal

–used to display the calendar Syn:$cal 2 2009

### echo

–used to print the message on the screen.

Syn:$echo “text”

### ls

–used to list the files. Your files are kept in a directory.

Syn:$ls ls–s

ls–l provide file statistics

ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l)

ls–s Order by size

ls–r Reverse order

### man

–used to provide manual help on every UNIX commands.

Syn:$man unix command

$man cat

### who & whoami

–it displays data about all users who have logged into the system currently. The next command displays about current user only.

Syn:$who

$whoami

### uptime

–tells you how long the computer has been running since its last reboot or power-off.

Syn:$uptime

### uname

–it displays the system information such as hardware platform, system name and processor, OS type.

Syn:$uname–a

### hostname

–displays and set system host name Syn:$ hostname

FILE MANIPULATIONS COMMANDS

1. **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)

1. **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

1. **rm**–deletes a file from the file system Syn:$rm filename
2. **touch**–used to create a blank file.

Syn:$touch file names

1. **cp**–copies the files or directories Syn:

$cpsource file destination file

Eg:$cp student stud

1. **mv**–to rename the file or directory

syn:$mv old file new file

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

**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

Examples:

$chmod u-wx student

Removes write and execute permission for users

$chmod u+rw,g+rw student

Assigns read and write permission for users and groups

## PROGRAM 1: Write Programs using System Calls of Unix Operating Systems (OPENDIR, READDIR, CLOSEDIR)

# 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);

}

**Program 2: Program For System Calls Of Unix Operating System (fork, getpid, exit)**

# 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();

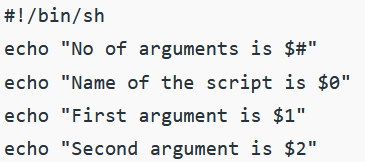
printf(“\n the child process ID is %d\n”, pid2);

}

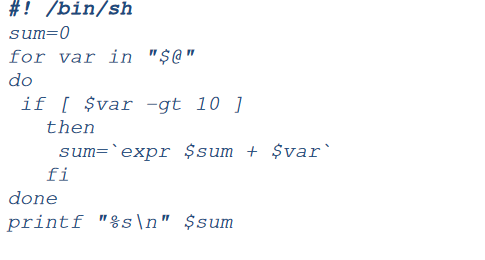
}

**Program 3:Shell Programs**

**3a. Program to demonstrate positional Parameters**



3b. Write a shell script that takes numbers greater then 10 as command line parameters and finds their sum



**Program 4: To write a C program for implementation of Priority scheduling algorithms.**

/\*

 \* C Program to Implement Priority Scheduling

 \*/

#include<stdio.h>

int main()

{

int bt[20],p[20],pri[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

for(i=0;i<n;i++)

{

printf("Enter Burst time p%d:",i+1);

scanf("%d",&bt[i]);

printf("Enter Priority p%d:",i+1);

scanf("%d",&pri[i]);

p[i]=i+1;

}

//sorting of burst times

for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(pri[j]<pri[pos])

pos=j;

}

temp=pri[i];

pri[i]=pri[pos];

pri[pos]=temp;

temp=pri[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

//finding the waiting time of all the processes

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<i;j++)

//individual WT by adding BT of all previous completed processes

wt[i]+=bt[j];

//total waiting time

total+=wt[i];

}

//average waiting time

avg\_wt=(float)total/n;

printf("\nProcess\t Burst Time \tPriority \t Waiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{

//turnaround time of individual processes

tat[i]=bt[i]+wt[i];

//total turnaround time

totalT+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t%d\t\t\t%d",p[i],bt[i],wt[i],pri[i],tat[i]);

}

//average turnaround time

avg\_tat=(float)totalT/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f",avg\_tat);

}

**Program 5: To write a C program for implementation of Round Robin scheduling algorithms.**

/\* Round Robin Scheduling Program in C \*/

#include<stdio.h>

int main()

{

//Input no of processed

int n;

printf("Enter Total Number of Processes:");

scanf("%d", &n);

int wait\_time = 0, ta\_time = 0, arr\_time[n], burst\_time[n], temp\_burst\_time[n];

int x = n;

//Input details of processes

for(int i = 0; i < n; i++)

{

printf("Enter Details of Process %d \n", i + 1);

printf("Arrival Time: ");

scanf("%d", &arr\_time[i]);

printf("Burst Time: ");

scanf("%d", &burst\_time[i]);

temp\_burst\_time[i] = burst\_time[i];

}

//Input time slot

int time\_slot;

printf("Enter Time Slot:");

scanf("%d", &time\_slot);

//Total indicates total time

//counter indicates which process is executed

int total = 0, counter = 0,i;

printf("Process ID Burst Time Turnaround Time Waiting Time\n");

for(total=0, i = 0; x!=0; )

{

// define the conditions

if(temp\_burst\_time[i] <= time\_slot && temp\_burst\_time[i] > 0)

{

total = total + temp\_burst\_time[i];

temp\_burst\_time[i] = 0;

counter=1;

}

else if(temp\_burst\_time[i] > 0)

{

temp\_burst\_time[i] = temp\_burst\_time[i] - time\_slot;

total += time\_slot;

}

if(temp\_burst\_time[i]==0 && counter==1)

{

x--; //decrement the process no.

printf("\nProcess No %d \t\t %d\t\t\t %d\t\t\t %d", i+1, burst\_time[i], total-arr\_time[i], total-arr\_time[i]-burst\_time[i]);

wait\_time = wait\_time+total-arr\_time[i]-burst\_time[i];

ta\_time += total -arr\_time[i];

counter =0;

}

if(i==n-1)

{

i=0;

}

else if(arr\_time[i+1]<=total)

{

i++;

}

else

{

i=0;

}

}

float average\_wait\_time = wait\_time \* 1.0 / n;

float average\_turnaround\_time = ta\_time \* 1.0 / n;

printf("\nAverage Waiting Time:%f", average\_wait\_time);

printf("\nAvg Turnaround Time:%f", average\_turnaround\_time);

return 0;

}

**Program 6: To write a c program to implement IPC using shared memory.**

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");

}

**Program 7: To write a C-program to implement the producer – consumer problem using semaphores.**

## 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);

}