

**INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES) DHANBAD**

**Department of Computer Science and Engineering Operating System Lab**

**(CSC15202)**

**Monsoon Semester (2020-21)**

# Name:

\**Adm.No.:**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program No.** | **Name of The Program** | **Date** | **Signature** | **Remarks** |
| **1** | Basics Of UNIX Commands. |  |  |  |
| **2** | Shell Programming |  |  |  |
| **3** | Implementation Of CPU Scheduling.  A) FCFS B) SJF |  |  |  |
| **4** | Implementation Of CPU Scheduling (Continue...)  C)Priority D)Round Robin |  |  |  |
| **5** | Priority Scheduling |  |  |  |
| **6** | Round Robin  Scheduling |  |  |  |
| **7** | Producer-Consumer Problem Using  Semopheres |  |  |  |
| **8** | First Fit Memory  Management |  |  |  |
| **9** | File Manipulation-I |  |  |  |
| **10** | File Manipulation-Ii |  |  |  |
| **11** | Simulate Page  Replacement Algorithms: Fifo |  |  |  |
| **12** | Simulate Page  Replacement Algorithms: Lru |  |  |  |
| **13** | Simulate Page Replacement  Algorithms: Optimal |  |  |  |
| **14** | Simulate Algorithm For Deadlock  Prevention |  |  |  |
| **15** | Understanding Features Of Modern File Systems Understanding Features Of Modern Filesystems |  |  |  |

**EX.NO:1 Basics of UNIX commands**

**DATE:**

**AIM:** To write and learn the basic linux command

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Linux Command*** | | ***DOS Command*** | ***Description*** | | |
| **pwd** | | cd | “Print Working Directory”. | Shows | the current |
| location in the directory tree. |  |  |
| **cd** | | cd, chdir | “Change Directory”. When | typed all | by itself, it |
| returns you to your home directory. | |  |
| **cd** | **directory** | cd directory | Change into the specified directory name. | | |
|  |  | Example: cd /usr/src/linux | |  |
| **cd** | **~** |  | “~” is an alias for your home directory. It can be | | |
|  |  | used as a shortcut to your “home”, or other | | |
|  |  | directories relative to your home. | |  |
| **cd ..** | | cd.. | Move up one directory. For example, if you are in | | |
| /home/vic and you type “cd ..”, you will end | | |
| up in /home. |  |  |
| **cd** | **-** |  | Return to previous directory. An easy way to get | | |
|  |  | back to your previous location! | |  |
| **ls** | | dir /w | List all files in the current directory, in column | | |
| format. |  |  |
| **ls** | **directory** | dir directory | List the files in the specified directory. | |  |
|  |  | Example: ls /var/log |  |  |
| **ls** | **-l** | dir | List files in “long” format, one file per line. This | | |
|  |  | also shows you additional info about the file, such | | |
|  |  | as ownership, permissions, date, and size. | | |
| **ls** | **-a** | dir /a | List all files, including “hidden” files. Hidden files | | |
|  |  | are those files that begin | with a “.”, e.g. The | |
|  |  | .bash\_history file in your home directory. | | |
| **ls -ld** | |  | A “long” list of “directory”, but instead of showing | | |
| **directory** | | the directory contents, show the directory's detailed | | |
|  | | information. For example, compare the output of | | |
|  | | the following two commands: |  |  |
|  | | ls -l /usr/bin |  |  |
|  | | ls -ld /usr/bin |  |  |
| **ls** | **/usr/bin/d\*** | dir d\*.\* | List all files whose names begin with the letter “d” | | |
|  |  | in the /usr/bin directory. |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Linux Command*** | | ***DOS Command*** | ***Description*** | | |
| **pwd** | | cd | “Print Working Directory”. | Shows | the current |
| location in the directory tree. |  |  |
| **cd** | | cd, chdir | “Change Directory”. When | typed all | by itself, it |
| returns you to your home directory. | |  |
| **cd** | **directory** | cd directory | Change into the specified directory name. | | |
|  |  | Example: cd /usr/src/linux | |  |
| **cd** | **~** |  | “~” is an alias for your home directory. It can be | | |
|  |  | used as a shortcut to your “home”, or other | | |
|  |  | directories relative to your home. | |  |
| **cd ..** | | cd.. | Move up one directory. For example, if you are in | | |
| /home/vic and you type “cd ..”, you will end | | |
| up in /home. |  |  |
| **cd** | **-** |  | Return to previous directory. An easy way to get | | |
|  |  | back to your previous location! | |  |
| **ls** | | dir /w | List all files in the current directory, in column | | |
| format. |  |  |
| **ls** | **directory** | dir directory | List the files in the specified directory. | |  |
|  |  | Example: ls /var/log |  |  |
| **ls** | **-l** | dir | List files in “long” format, one file per line. This | | |
|  |  | also shows you additional info about the file, such | | |
|  |  | as ownership, permissions, date, and size. | | |
| **ls** | **-a** | dir /a | List all files, including “hidden” files. Hidden files | | |
|  |  | are those files that begin | with a “.”, e.g. The | |
|  |  | .bash\_history file in your home directory. | | |
| **ls -ld** | |  | A “long” list of “directory”, but instead of showing | | |
| **directory** | | the directory contents, show the directory's detailed | | |
|  | | information. For example, compare the output of | | |
|  | | the following two commands: |  |  |
|  | | ls -l /usr/bin |  |  |
|  | | ls -ld /usr/bin |  |  |
| **ls** | **/usr/bin/d\*** | dir d\*.\* | List all files whose names begin with the letter “d” | | |
|  |  | in the /usr/bin directory. |  |  |

**EX.NO:2 Shell programming**

**DATE:**

**AIM:** To write the program to implement SHELL PROGRAMMING

Q-1 write a shell program to add two numbers.

Program - echo enter the first number

read a

echo enter the second number read b

c=$(($a+$b)) echo the sum is $c

Q- 2 write a shell program to find a number is even or odd Program - echo –n enter the number

read x

b=$(( $x %2)) If[ $b –eq 0 ] then

echo given no is even else

echo given no is odd fi

Q -3write a shell program for fibbonacci series.

Program –

Count =1 F1=1 F2=1 K=0

Echo $f1 Echo $f2

While [ $count –le 10 ] Do

K=$(($f1=$f2)) echo $k

f1=$ f2 f2=$k

count=$(($count+1)) done

**EX.NO:3 Implementation of CPU Scheduling**.

**DATE:**

**AIM:** To write the program to implement CPU & scheduling algorithm for first come first serve scheduling.

**ALGORITHM:**

|  |  |
| --- | --- |
| 1. | Start the program. |
| 2. | Get the number of processes and their burst time. |

1. Initialize the waiting time for process 1 and 0.

4. Process for(i=2;i<=n;i++),wt.p[i]=p[i-1]+bt.p[i-1].

|  |  |
| --- | --- |
| 5. | The waiting time of all the processes is summed then average value time is calculated. |
| 6. | The waiting time of each process and average times are displayed |
| 7. | Stop the program |

**PROGRAM :( FIRST COME FIRST SERVE SCHEDULING)**

#include<stdio.h> struct process

{

int pid; int bt; int wt,tt;

}p[10];

int main()

{

int i,n,totwt,tottt,avg1,avg2; clrscr();

printf("enter the no of process \n"); scanf("%d",&n); for(i=1;i<=n;i++)

{

p[i].pid=i;

printf("enter the burst time n"); scanf("%d",&p[i].bt);

} p[1].wt=0;

p[1].tt=p[1].bt+p[1].wt; i=2;

while(i<=n)

{

p[i].wt=p[i-1].bt+p[i-1].wt; p[i].tt=p[i].bt+p[i].wt; i ++;

}

i=1;

totwt=tottt=0;

printf("\n processid \t bt\t wt\t tt\n"); while(i<=n){

printf("\n\t%d \t%d \t%d \t%d",p[i].pid,p[i].bt,p[i].wt,p[i].tt); totwt=p[i].wt+totwt;

tottt=p[i].tt+tottt; i++;}

avg1=totwt/n; avg2=tottt/n; printf("\navg1=%d \t avg2=%d

\t",avg1,avg2); getch(); return 0;

}

|  |  |  |  |
| --- | --- | --- | --- |
| **OUTPUT:** |  |  |  |
| enter the no of process | |  | 3 |
| enter the burst time | |  | 2 |
| enter the burst time | |  | 4 |
| enter the burst time | |  | 6 |
| Process sid | bt | wt | tt |
| 1 | 2 | 0 | 2 |
| 2 | 4 | 2 | 6 |
| 3 | 6 | 6 | 12 |
| avg1=2 |  | avg2=6 | |

**RESULT:**

Thus the FIFO process scheduling program was executed and verified successfully.

**EX.NO:4 Implementation of CPU Scheduling (Continue...)**

**DATE**:

**AIM:** To write a program to implement CPU scheduling algorithm for shortest job first scheduling.

**ALGORITHM:**

|  |  |
| --- | --- |
| 1. | Start the program. Get the number of processes and their burst time. |
| 2. | Initialize the waiting time for process 1 as 0. |
| 3. | The processes are stored according to their burst time. |

1. The waiting time for the processes are calculated as follows: for(i=2;i<=n;i++).wt.p[i]=p[i=1]+bt.p[i-1].

|  |  |
| --- | --- |
| 5. | The waiting time of all the processes summed and then the average time is calculate |
| 6. | The waiting time of each processes and average time are displayed. |
| 7. | Stop the program. |

**PROGRAM: (SHORTEST JOB FIRST SCHEDULING)**

#include<stdio.h> #include<conio.h> struct process

{

int pid; int bt;

int wt; int tt;

}p[10],temp; int main()

{

int i,j,n,totwt,tottt; float avg1,avg2; clrscr();

printf("\nEnter the number of process:\t"); scanf("%d",&n);

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

{

p[i].pid=i;

printf("\nEnter the burst time:\t"); scanf("%d",&p[i].bt);

}

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

{

if(p[i].bt>p[j].bt)

{

temp.pid=p[i].pid;

p[i].pid=p[j].pid; p[j].pid=temp.pid;

temp.bt=p[i].bt;p[i].bt=p[j].bt; p[j].bt=temp.bt;

}}}

p[1].wt=0; p[1].tt=p[1].bt+p[1].wt; i=2;

while(i<=n){

p[i].wt=p[i-1].bt+p[i-1].wt;

p[i].tt=p[i].bt+p[i].wt; i++;

}

i=1;

totwt=tottt=0;

printf("\nProcess id \tbt \twt \ttt"); while(i<=n){

printf("\n\t%d \t%d \t%d t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt); totwt=p[i].wt+totwt;

tottt=p[i].tt+tottt; i++;

**} avg1=totwt/n; avg2=tottt/n;**

printf("\nAVG1=%f\t AVG2=%f",avg1,avg2); getch();

return 0; }

|  |  |  |  |
| --- | --- | --- | --- |
| **OUTPUT:** |  |  |  |
| enter the number of | | process | 3 |
| enter the burst time: 2 | | | |
| enter the burst time: 4 | | | |
| enter the burst time: 6 | | | |
| processid | bt | wt | tt |
| 1 | 2 | 0 | 2 |
| 2 | 4 | 2 | 6 |
| 3 | 6 | 6 | 12 |
| AVG1=2.000000 | | AVG2=6.000000 | |

### RESULT:

Thus the SJF program was executed and verified successfully

**EX.NO:5 PRIORITY SCHEDULING**

**DATE:**

**AIM:** To write a ‘C’ program to perform priority scheduling.

**ALGORITHM:**

|  |  |
| --- | --- |
| 1. | Start the program. |
| 2. | Read burst time, waiting time, turn the around time and priority. |

1. Initialize the waiting time for process 1 and 0.
2. Based up on the priority process are arranged
3. The waiting time of all the processes is summed and then the average waiting time
4. The waiting time of each process and average waiting time are displayed based on the priority.
5. Stop the program.

**PROGRAM: (PRIORITY SCHEDULING)**

#include<stdio.h> #include<conio.h> struct process

{

int pid; int bt; int wt; int tt; int prior;

}

p[10],temp; int main()

{

int i,j,n,totwt,tottt,arg1,arg2; clrscr();

printf("enter the number of process"); scanf("%d",&n);

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

{

p[i].pid=i;

printf("enter the burst time"); scanf("%d",&p[i].bt);

printf("\n enter the priority"); scanf("%d",&p[i].prior);

}

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

{

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

{

if(p[i].prior>p[j].prior)

{

temp.pid=p[i].pid; p[i].pid=p[j].pid;

p[j].pid=temp.pid; temp.bt=p[i].bt;

p[i].bt=p[j].bt;

p[j].bt=temp.bt; temp.prior=p[i].prior; p[i].prior=p[j].prior; p[j].prior=temp.prior;

}

}

}

p[i].wt=0; p[1].tt=p[1].bt+p[1].wt; i=2;

while(i<=n)

{

p[i].wt=p[i-1].bt+p[i-1].wt;

p[i].tt=p[i].bt+p[i].wt; i++;

}

i=1;

totwt=tottt=0;

printf("\n process to \t bt \t wt \t tt"); while(i<=n)

{

printf("\n%d\t %d\t %d\t %d\t",p[i].pid,p[i].bt,p[i].wt,p[i].tt); totwt=p[i].wt+totwt;

tottt=p[i].tt+tottt;

i++;

}

arg1=totwt/n; arg2=tottt/n;

printf("\n arg1=%d \t arg2=%d\t",arg1,arg2); getch();

return 0;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OUTPUT:** |  |  |  |  |
| enter the no of process:3 | | | | |
| enter the burst time:2 | | | | |
| enter the priority:3 | | | | |
| enter the burst time:4 | | | | |
| enter the priority:1 | | | | |
| enter the burst time:6 | | | | |
| enter the priority:2 | | | | |
| process to | bt | wt | tt |  |
| 1 | 4 | 0 | 4 | 4 |
| 2 | 6 | 4 | 10 | 14 |
| 3 | 2 | 10 | 12 | 22 |
| avg1=4 |  |  | avg2=8 |  |

### RESULT:

Thus the priority scheduling program was executed and verified successfully

**EX.NO:6 ROUND ROBIN SCHEDULING**

**DATE:**

**AIM:** To write a program to implement CPU scheduling for Round Robin Scheduling.

### ALGORITHM:

|  |  |
| --- | --- |
| 1. | Get the number of process and their burst time. |
| 2. | Initialize the array for Round Robin circular queue as ‘0’. |

* 1. The burst time of each process is divided and the quotients are stored on the round Robin array.
  2. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling.

|  |  |
| --- | --- |
| 5. | The waiting time for each process and average times are displayed. |
| 6. | Stop the program. |

**PROGRAM :( ROUND ROBIN SCHEDULING)**

#include<stdio.h> #include<conio.h> struct process

{

int pid,bt,tt,wt;

};

int main()

{

struct process x[10],p[30]; int i,j,k,tot=0,m,n;

float wttime=0.0,tottime=0.0,a1,a2; clrscr();

printf("\nEnter the number of process:\t"); scanf("%d",&n);

for(i=1;i<=n;i++){ x[i].pid=i;

printf("\nEnter the Burst Time:\t"); scanf("%d",&x[i].bt); tot=tot+x[i].bt;

}

printf("\nTotal Burst Time:\t%d",tot); p[0].tt=0;

k=1;

printf("\nEnter the Time Slice:\t"); scanf("%d",&m);

for(j=1;j<=tot;j++)

{

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

{

if(x[i].bt !=0)

{

p[k].pid=i; if(x[i].bt-m<0)

{

p[k].wt=p[k-1].tt;

p[k].bt=x[i].bt; p[k].tt=p[k].wt+x[i].bt;

x[i].bt=0; k++;

}

else

{

p[k].wt=p[k-1].tt;

p[k].tt=p[k].wt+m;

x[i].bt=x[i].bt-m; k++;

}

}

}

}

printf("\nProcess id \twt \ttt"); for(i=1;i<k;i++){

printf("\n\t%d \t%d \t%d",p[i].pid,p[i].wt,p[i].tt); wttime=wttime+p[i].wt;

tottime=tottime+p[i].tt; a1=wttime/n; a2=tottime/n;

}

printf("\n\nAverage Waiting Time:\t%f",a1); printf("\n\nAverage TurnAround Time:\t%f",a2); getch();

return 0;

}

### OUTPUT:

enter the no of process3

|  |  |  |
| --- | --- | --- |
| enter the burst time3 | | |
| enter the burst time5 | | |
| enter the burst time7 | | |
| total burst time : 15 | | |
| enter the time slice: 2 | | |
| process id | wt | tt |
| 1 | 0 | 2 |
| 2 | 2 | 4 |
| 3 | 4 | 6 |
| 1 | 6 | 7 |
| process id | wt | tt |
| 2 | 7 | 9 |
| 3 | 9 | 11 |
| 2 | 11 | 12 |
| 3 | 12 | 14 |
| 3 | 14 | 15 |
| avg waiting time: 21.666666 | | |
| avg turnaround time: | | 26.666666 |

**RESULT:**

Thus the Round Robin scheduling program was executed and verified successfully.

**EX.NO:7 PRODUCER-CONSUMER PROBLEM USING SEMOPHERES**

**DATE**:

**AIM:** To implement producer/consumer problem using semaphore.

**ALGORITHM:**

|  |  |
| --- | --- |
| 1. | Declare variable for producer & consumer as pthread-t-tid produce tid consume. |
| 2. | Declare a structure to add items, semaphore variable set as struct. |
| 3. | Read number the items to be produced and consumed. |
| 4. | Declare and define semaphore function for creation and destroy. |

* 1. Define producer function.
  2. Define consumer function.
  3. Call producer and consumer.
  4. Stop the execution.

**PROGRAM: (PRODUCER-CONSUMER PROBLEM)**

#include<stdio.h> void main()

{

int buffer[10], bufsize, in, out, produce, consume, choice=0; in = 0;

out = 0;

bufsize = 10;

while(choice !=3)

{

printf("\n1. Produce \t 2. Consume \t3. Exit"); printf("\nEnter your choice: =");

scanf("%d", &choice); switch(choice)

{

printf("\nBuffer is Full");

else

{

printf("\nEnter the value: "); scanf("%d", &produce); buffer[in] = produce;

in = (in+1)%bufsize;

}

break;

case 2: if(in == out) printf("\nBuffer is Empty");

else

{

consume = buffer[out];

printf("\nThe consumed value is %d", consume); out = (out+1)%bufsize;

}

break;

}

}

}

|  |  |  |
| --- | --- | --- |
| **OUTPUT:** |  |  |
| 1. Produce | 2. Consume | 3. Exit |
| Enter your choice: 2 | | |
| Buffer is Empty | | |
| 1. Produce | 2. Consume | 3. Exit |
| Enter your choice: 1 | | |
| Enter the value: 100 | | |
| 1. Produce | 2. Consume | 3. Exit |
| Enter your choice: 2 | | |
| The consumed value is 100 | | |
| 1. Produce | 2. Consume | 3. Exit |

Enter your choice: 3

**RESULT:**

Thus the producer consumer program was executed and verified successfully

**DATE:**

**EX.NO:8 FIRST FIT MEMORY MANAGEMENT**

**AIM:** To implement first fit, best fit algorithm for memory management

|  |  |
| --- | --- |
| **ALGORITHM:** | |
| 1. | Start the program. |
| 2. | Get the segment size, number of process to be allocated and their corresponding size. |
| 3. | Get the options. If the option is ‘2’ call first fit function. |
| 4. | If the option is ‘1’ call best fit function. Otherwise exit. |

1. For first fit, allocate the process to first possible segment which is free and set the personnel slap as ‘1’. So that none of process to be allocated to segment which is already allocated and vice versa.
2. For best fit, do the following steps,
3. Sorts the segments according to their sizes.
4. Allocate the process to the segment which is equal to or slightly greater than the process Size and set the flag as the ‘1’ .So that none of the process to be allocated to the segment which is already allocated and vice versa. Stop the program.
5. Stop the program

**PROGRAM: (FIRST FIT MEMORY MANAGEMENT)**

#include<stdio.h> #include<conio.h> #define max 25 void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0; static int bf[max],ff[max]; clrscr(); printf("\n\tMemory Management Scheme - First Fit"); printf("\nEnter the number of blocks:"); scanf("%d",&nb);

printf("Enter the number of files:"); scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n"); for(i=1;i<=nb;i++)

{

printf("Block %d:",i);

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

}

printf("Enter the size of the files :-\n"); for(i=1;i<=nf;i++)

{

printf("File %d:",i);

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

}

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

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1) //if bf[j] is not allocated

{

temp=b[j]-f[i]; if(temp>=0)

if(highest<temp)

{

ff[i]=j;

highest=temp;

}

}

}

frag[i]=highest; bf[ff[i]]=1; highest=0;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement"); for(i=1;i<=nf;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i,f[i],ff[i],b[ff[i]],frag[i]); getch();

}

### OUTPUT:

Enter the number of blocks: 3 Enter the number of files: 2

Enter the size of the blocks:- Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:- File 1: 1

File 2: 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OUTPUT |  |  |  |  |  |
| File No | File Size | Block No |  | Block Size | Fragment |
| 1 | 1 | 3 | 7 | 6 |  |
| 2 | 4 | 1 | 5 | 1 |  |

**RESULT:**

Thus the First Bit and Best Fit program was executed and verified successfully.

**EX.NO:9 FILE MANIPULATION-I**

**DATE:**

**AIM:** To write a program for file manipulation for displays the file and directory in memory

**ALGORITHM:**

|  |  |
| --- | --- |
| 1. | Start the program. |
| 2. | Use the pre-defined function list out the files in directory. |
| 3. | Main function is used to check the file present in the directory or not. |

4. Using the file pointer in the file to that the argument is less than a times means print

|  |  |
| --- | --- |
| 5. | By using if loop check in file, open two means print error |
| 6. | Stop the program. |

**PROGRAM: (FILE MANIPULATION-I)**

#include <dirent.h> #include <stdio.h> int main(void)

{

DIR \*d;

struct dirent \*dir; d = opendir("."); if (d)

{

while ((dir = readdir(d)) != NULL)

{

printf("%s\n", dir->d\_name);

}

closedir(d);

}

return(0);

}



### OUTPUT:

**RESULT:**

Thus the file management program was executed and verified successfully.

**EX.NO:10 FILE MANIPULATION-II**

**DATE:**

**AIM:** To write a program performs file manipulation.

**ALGORITHM:**

1. Start the program.
2. Declare the arguments for file open and file create.
3. Print the file in directory otherwise display the error message error in creation
4. if check the files in directory
5. close the files and directory
6. Stop the program.

**PROGRAM :( FILE MANIPULATION-II)**

#include<stdio.h> #include<sys/stat.h> #include<time.h> main(int ag,char\*arg[])

{

char buf[100]; struct stat s; int fd1,fd2,n;

fd1=open(arg[1],0); fd2=creat(arg[2],0777); stat(arg[2],&s); if(fd2==-1)

printf("ERROR IN CREATION");

while((n=read(fd1,buf,sizeof(buf)))>0)

{

if(write(fd2,buf,n)!=n)

{

close(fd1); close(fd2);

}

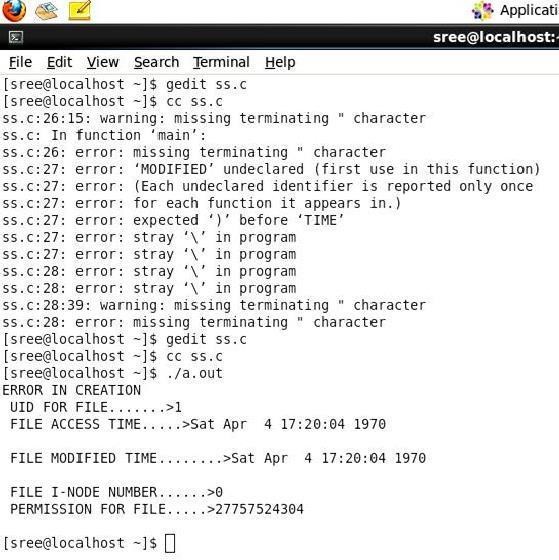
}

printf("\t\n UID FOR FILE.......>%d \n FILE ACCESS TIME.....>%s \n FILE MODIFIED TIME........>%s \n FILE I-NODE NUMBER......>%d \n PERMISSION FOR

FILE. >%o\n\n",s.st\_uid,ctime(&s.st\_atime),ctime(&s.st\_mt

ime),s.st\_mode); close(fd1); close(fd2);

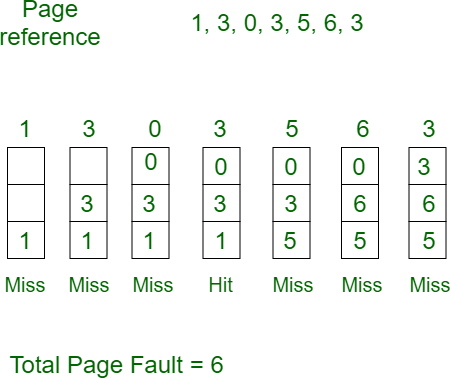
}



### OUTPUT:

**RESULT:**

Thus the File Manipulation II program was executed and verified successfully.



**EX NO: 11 SIMULATE PAGE REPLACEMENT ALGORITHMS FIFO**

**DATE:**

**AIM:** To Simulate FIFO page replacement algorithms.

**First In First Out (FIFO) –**

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

**Example-1**Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames. Find number of page faults.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**

When 3 comes, it is already in memory so —> **0 Page Faults.**

Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**

6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**

Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

[**Belady’s anomaly**](https://www.geeksforgeeks.org/operating-system-beladys-anomaly/) **–** Belady’s anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2,

1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

### ALGORITHM:

1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames in First in first out order.
7. Display the number of page faults.
8. Stop the program

**PROGRAM: (SIMULATE PAGE REPLACEMENT ALGORITHMS FIFO)**

#include<stdio.h> #include<conio.h>

int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1; void main()

{

clrscr();

printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");

printf("\n Enter no.of frames...."); scanf("%d",&nof);

printf("Enter number of Pages.\n"); scanf("%d",&nor);

printf("\n Enter the Page No..."); for(i=0;i<nor;i++) scanf("%d",&ref[i]); printf("\nThe given Pages are:"); for(i=0;i<nor;i++) printf("%4d",ref[i]); for(i=1;i<=nof;i++)

frm[i]=-1;

printf("\n"); for(i=0;i<nor;i++)

{

flag=0;

printf("\n\t page no %d->\t",ref[i]); for(j=0;j<nof;j++)

{

if(frm[j]==ref[i])

{

flag=1; break;

}}

if(flag==0)

{

pf++; victim++;

victim=victim%nof; frm[victim]=ref[i]; for(j=0;j<nof;j++) printf("%4d",frm[j]);

} }

printf("\n\n\t\t No.of pages faults...%d",pf); getch();

}

### OUTPUT:

FIFO PAGE REPLACEMENT ALGORITHM

Enter no.of frames ... 4

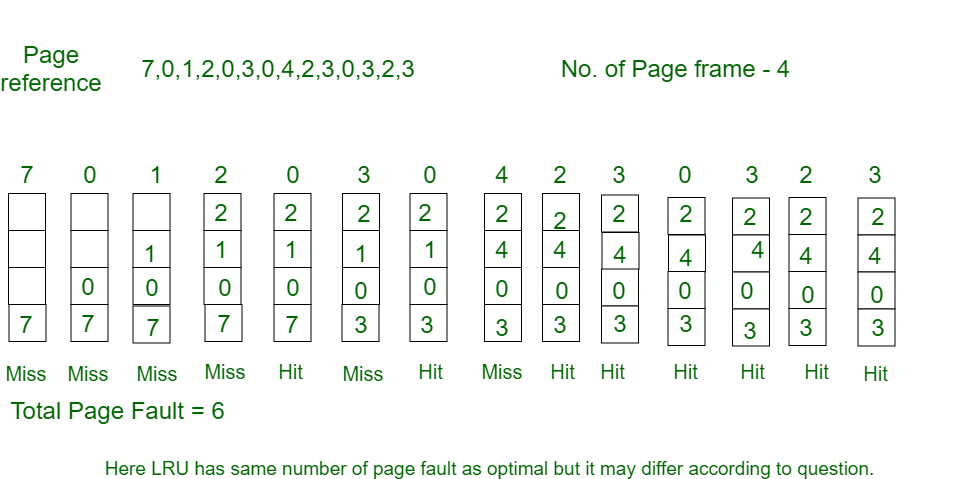
Enter number of reference string.. 6

Enter the reference string.. 564123

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| The given reference string: | | | | |
| ...................................... | 5641 |  | 2 | 3 |
| Reference np5-> | 5 | -1 | -1 | -1 |
| Reference np6-> | 5 | 6 | -1 | -1 |
| Reference np4-> | 5 | 6 | 4 | -1 |
| Reference np1-> | 5 | 6 | 4 | 1 |
| Reference np2-> | 2 | 6 | 4 | 1 |
| Reference np3-> | 2 | 3 | 4 | 1 |
| No.of pages faults... | 6 |  |  |  |

**RESULT:**

Thus the program FIFO page replacement was successfully executed.



**EX NO: 12 SIMULATE PAGE REPLACEMENT ALGORITHMS: LRU**

**DATE:**

**AIM:** To Simulate LRU page replacement algorithms

**Least Recently Used –**

In this algorithm page will be replaced which is least recently used.

**Example-3**Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**

0 is already there so —> **0 Page fault.**

## When 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**

0 is already in memory so —> **0 Page fault**. 4 will takes place of 1 —> **1 Page Fault**

## Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

### ALGORITHM:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
7. Display the number of page faults.
8. stop

**PROGRAM: (SIMULATE PAGE REPLACEMENT ALGORITHMS: LRU)**

#include<stdio.h> #include<conio.h>

int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=- 1; int recent[10],lrucal[50],count=0; int lruvictim();

void main()

{

clrscr();

printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM"); printf("\n Enter no.of Frames. ");

scanf("%d",&nof);

printf(" Enter no.of reference string.."); scanf("%d",&nor);

printf("\n Enter reference string.."); for(i=0;i<nor;i++) scanf("%d",&ref[i]);

printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM "); printf("\n\t The given reference string:");

printf("\n… ");

for(i=0;i<nor;i++) printf("%4d",ref[i]); for(i=1;i<=nof;i++)

{

frm[i]=-1; lrucal[i]=0;

}

for(i=0;i<10;i++)

recent[i]=0; printf("\n"); for(i=0;i<nor;i++)

{

flag=0;

printf("\n\t Reference NO %d->\t",ref[i]); for(j=0;j<nof;j++)

{

if(frm[j]==ref[i])

{

flag=1; break;

}

}

if(flag==0)

{

count++; if(count<=nof) victim++;

else victim=lruvictim(); pf++; frm[victim]=ref[i]; for(j=0;j<nof;j++) printf("%4d",frm[j]);

}

recent[ref[i]]=i;

}

printf("\n\n\t No.of page faults...%d",pf); getch();

}

int lruvictim()

{

int i,j,temp1,temp2; for(i=0;i<nof;i++)

{

temp1=frm[i]; lrucal[i]=recent[temp1];

}

temp2=lrucal[0]; for(j=1;j<nof;j++)

{

if(temp2>lrucal[j]) temp2=lrucal[j];

}

for(i=0;i<nof;i++) if(ref[temp2]==frm[i]) return i;

return 0;

}

### OUTPUT:

LRU PAGE REPLACEMENT ALGORITHM

Enter no.of Frames 3

Enter no.of reference string 6

Enter reference string.. 654231

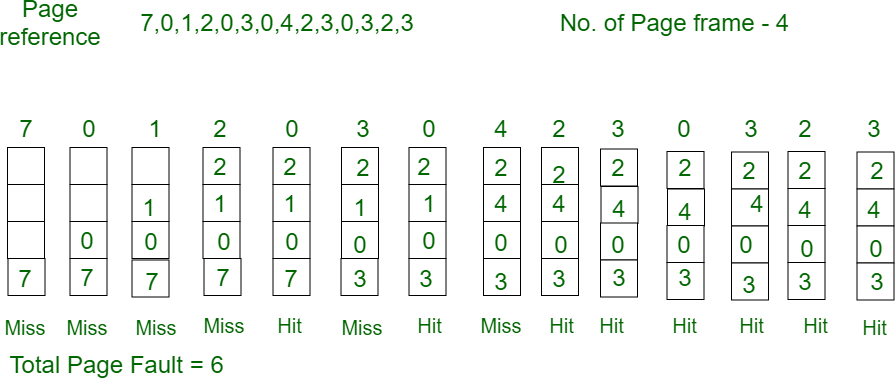
LRU PAGE REPLACEMENT ALGORITHM

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| The given reference string: | | | | | |
| …………………. 6 | 5 | 4 | 2 | 3 | 1 |
| Reference NO 6-> |  |  | 6 | -1 -1 | |
| Reference NO 5-> |  |  | 6 | 5 | -1 |
| Reference NO 4-> |  |  | 6 | 5 | 4 |
| Reference NO 2-> |  |  | 2 | 5 | 4 |
| Reference NO 3-> |  |  | 2 | 3 | 4 |
| Reference NO 1-> |  |  | 2 | 3 | 1 |

No.of page faults. 6

**RESULT:**

Thus the process LRU page replacement was executed and verified successfully.



**EX.NO:13 SIMULATE PAGE REPLACEMENT ALGORITHMS: OPTIMAL**

**DATE**:

**AIM:** To create program for optimal page replacement algorithms.

**Optimal Page replacement –**

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

**Example:** Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**

0 is already there so —> **0 Page fault.**

## When 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**

0 is already there so —> **0 Page fault.**

4 will takes place of 1 —> **1 Page Fault.**

## Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

### ALGORITHM:

1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. Stop the program

**PROGRAM: (SIMULATE PAGE REPLACEMENT ALGORITHMS: OPTIMAL)**

#include<stdio.h> #include<conio.h>

int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=- 1; int recent[10],optcal[50],count=0; int optvictim();

void main()

{

clrscr();

printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN"); printf("\n. ");

printf("\nEnter the no.of frames"); scanf("%d",&nof);

printf("Enter the no.of reference string"); scanf("%d",&nor);

printf("Enter the reference string"); for(i=0;i<nor;i++) scanf("%d",&ref[i]);

clrscr();

printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM"); printf("\n. ");

printf("\nThe given string");

printf("\n. \n");

for(i=0;i<nor;i++) printf("%4d",ref[i]); for(i=0;i<nof;i++)

{

frm[i]=-1;

optcal[i]=0;

}

for(i=0;i<10;i++) recent[i]=0;

printf("\n"); for(i=0;i<nor;i++)

{

flag=0;

printf("\n\tref no %d ->\t",ref[i]); for(j=0;j<nof;j++)

{

if(frm[j]==ref[i])

{

flag=1; break;

}

}

if(flag==0)

{

count++; if(count<=nof)

victim++;

else

victim=optvictim(i); pf++; frm[victim]=ref[i]; for(j=0;j<nof;j++)

printf("%4d",frm[j]);

}

}

printf("\n Number of page faults: %d",pf); getch();

}

int optvictim(int index)

{

int i,j,temp,notfound; for(i=0;i<nof;i++)

{

notfound=1; for(j=index;j<nor;j++)

if(frm[i]==ref[j])

{

notfound=0; optcal[i]=j; break;

}

if(notfound==1)

return i;

}

temp=optcal[0]; for(i=1;i<nof;i++)

if(temp<optcal[i])

temp=optcal[i]; for(i=0;i<nof;i++)

if(frm[temp]==frm[i])

return i;

return 0;

}

### OUTPUT:

**OPTIMAL PAGE REPLACEMENT ALGORITHM**

Enter no. of Frames ... 3

Enter no. of reference string..6

Enter reference string.. 654231

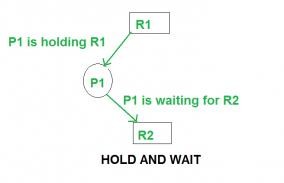
OPTIMAL PAGE REPLACEMENT ALGORITHM

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| The given reference string: | | | | | |
| …………………. 6 | 5 | 4 | 2 | 3 | 1 |
| Reference NO 6-> |  |  | 6 | -1 | -1 |
| Reference NO 5-> |  |  | 6 | 5 | -1 |
| Reference NO 4-> |  |  | 6 | 5 | 4 |
| Reference NO 2-> |  |  | 2 | 5 | 4 |
| Reference NO 3-> |  |  | 2 | 3 | 4 |
| Reference NO 1-> |  |  | 2 | 3 | 1 |

No.of page faults...6

**RESULT:**

Thus the process optimal page replacement was executed and verified successfully.



# EX NO: 14 SIMULATE ALGORITHM FOR DEADLOCK PREVENTION

### DATE:

**AIM:** To Simulate Algorithm for Deadlock prevention

**Deadlock Prevention**

We can prevent Deadlock by eliminating any of the above four conditions.

**Eliminate Mutual Exclusion**

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

**Eliminate Hold and wait**

1. Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. For example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.
2. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.

**Eliminate No Preemption**

Preempt resources from the process when resources required by other high priority processes.

**Eliminate Circular Wait**

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing order of numbering.

For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

**ALGORITHM:**

* 1. Start the program
  2. Attacking mutex condition: never grant exclusive access. But this may not be possible for several resources.
  3. Attacking preemption: not something you want to do.

|  |  |
| --- | --- |
| 4. | Attacking hold and wait condition: make a process hold at the most 1 resource |
| 5. | At a time. Make all the requests at the beginning. Nothing policy. If you feel, retry. |

1. Attacking circular wait: Order all the resources. Make sure that the requests are issued in the
2. Correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n.
3. Resources can be requested only in increasing

|  |  |
| --- | --- |
| 9. | Order. i.e. you cannot request a resource whose no is less than any you may be holding. |
| 10. | Stop the program |

**PROGRAM: (SIMULATE ALGORITHM FOR DEADLOCK PREVENTION)**

#include<stdio.h> #include<conio.h>

int max[10][10], alloc[10][10], need[10][10];

int avail[10],i,j,p,r,finish[10]={0},flag=0; int main()

{

clrscr( );

printf("\n\nSIMULATION OF DEADLOCK PREVENTION");

printf("Enter no. of processes, resources"); scanf("%d%d",&p,&r);printf("Enter allocation matrix"); for(i=0;i<p;i++)

for(j=0;j<r;j++) scanf("%d",&alloc[i][j]); printf("enter max matrix");

for(i=0;i<p;i++) /\*reading the maximum matrix and availale matrix\*/ for(j=0;j<r;j++)

scanf("%d",&max[i][j]); printf("enter available matrix"); for(i=0;i<r;i++) scanf("%d",&avail[i]); for(i=0;i<p;i++) for(j=0;j<r;j++)

need[i][j]=max[i][j]-alloc[i][j]; fun(); /\*calling function\*/ if(flag==0)

{

if(finish[i]!=1)

{

printf("\n\n Failing :Mutual exclusion");

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

{ /\*checking for mutual exclusion\*/ if(avail[j]<need[i][j])

avail[j]=need[i][j];

}fun();

printf("\n By allocating required resources to process %d dead lock is prevented ",i);

printf("\n\n lack of preemption"); for(j=0;j<r;j++)

{

if(avail[j]<need[i][j])

avail[j]=need[i][j];

alloc[i][j]=0;

}

fun( );

printf("\n\n daed lock is prevented by allocating needed resources"); printf(" \n \n failing:Hold and Wait condition ");

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

{

if(avail[j]<need[i][j])

avail[j]=need[i][j];

}

fun( );

printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");

}

}

getch( ); return 0;

}

fun()

{

while(1)

{

for(flag=0,i=0;i<p;i++)

{

if(finish[i]==0)

{

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

{

if(need[i][j]<=avail[j]) continue;

else break;

}

if(j==r)

{

for(j=0;j<r;j++) avail[j]+=alloc[i][j]; flag=1;

finish[i]=1;

}

}

}

if(flag==0) break;

}return 0;

}

### OUTPUT:

SIMULATION OF DEADLOCK PREVENTION

Enter no. of processes, resources 3, 2 Enter allocation matrix 2 4 5

3 4 5

Enter max matrix4 3 4

5 6 1

Enter available matrix2 Failing: Mutual Exclusion

By allocating required resources to process dead is prevented Lack of no preemption deadlock is prevented by allocating needed resources failing: Hold and Wait condition

**RESULT:**

Thus the program deadlock was executed successfully.