Operating System Lab

**Experiment No-1:**

**Aim: Practicing of Basic UNIX Commands.**

1 . Study of Unix/Linux general purpose utility command list: man, who, cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod, chown, finger, pwd, cal, logout, shutdown.

1. man : the man command is used to display the manual or documentation
2. who: The who command displays a list of currently logged-in users on the system, along with information about their login sessions.
3. cat: The cat command is used to concatenate and display the content of one or multiple files.
4. cd: The cd command is used to change the current working directory .
5. cp: The cp command is used to copy files and directories from one location to another.
6. ps: is used to display information about the currently running processes on the system.
7. ls: The ls command lists the contents of a directory, showing files and subdirectories in the specified location.
8. mv: The mv command is used to move or rename files and directories.
9. rm: The rm command is used to remove (delete) files and directories.
10. mkdir: The mkdir command is used to create new directories (folders) in the file system.
11. rmdir: The rmdir command is used to remove empty directories (folders) from the file system.
12. echo: The echo command is used to display a message or text. It is also frequently used in shell scripting.
13. more: The more command is used to display the content of a file one screen at a time, allowing you to scroll through it.
14. date: The date command displays the current date and time.
15. time: The time command is used to measure the execution time of other commands.
16. kill: The kill command is used to terminate or send signals to running processes. It is commonly used to end processes gracefully or forcefully.
17. history: The history command displays a list of previously executed commands in the current session.
18. chmod: The chmod command is used to change the permissions (read, write, execute) of files and directories.
19. chown: The chown command is used to change the ownership of files and directories, assigning them to different users or groups.
20. finger: The finger command is used to display information about user accounts on the system.
21. pwd: The pwd command stands for print working directory and it displays the full path of the current working directory.
22. cal: The cal command displays the calendar for the specified month or year.
23. logout: The logout command is used to log out from the current user session in the terminal.
24. shutdown: The shutdown command is used to shut down or restart the system.

**Experiment No-2:**

**Aim: Write programs using the following UNIX operating system calls**

* 1. **open, read, write, seek and close Source code :**

#include <stdio.h>

#include <conio.h> // for getch() int main() {

FILE \*file;

char buffer[100];

file = fopen("testfile.txt", "w+"); // "w+" creates a file for reading and writing if (file == NULL) {

printf("Error opening/creating file.\n"); return 1;

}

fprintf(file, "Hello, Students"); fseek(file, 0, SEEK\_SET);

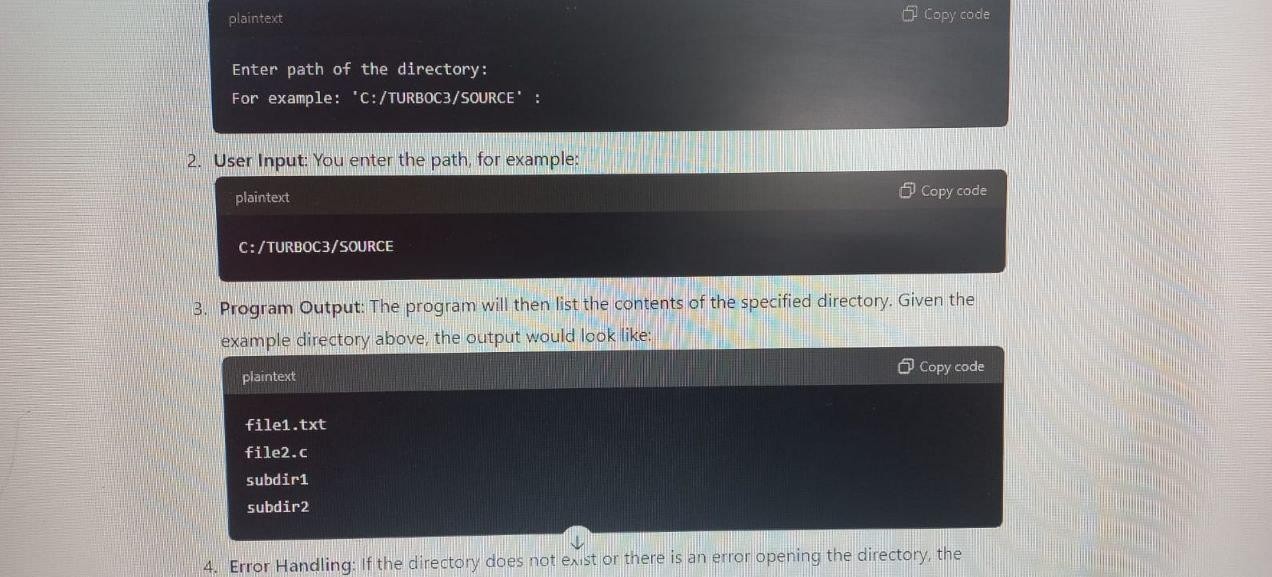
fgets(buffer, 100, file);

printf("Content read from file: %s\n", buffer);

fclose(file);

getch(); // Wait for key press before exiting (specific to Turbo C++) return 0;

}

Output:

* 1. **opendir, closedir Source code :**

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

DIR \*d;

struct dirent \*dir;

char \*directory\_path = " "; // "." refers to the current directory clrscr();

printf("\nEnter path of the directory...!\n"); printf("\For example: 'C:/TURBOC3/SOURCE' :\n"); scanf("%s",directory\_path);

d = opendir(directory\_path); if (d) {

while ((dir = readdir(d)) != NULL) { printf("%s\n", dir->d\_name);

}

closedir(d);

} else {

perror("Unable to open directory"); return EXIT\_FAILURE;

}

getch();

return EXIT\_SUCCESS;

}

**Output:** C:/TURBOC3/Sourcedir1 file1.txt

file2.c subdir1 program.exe

C:/TURBOC3/Sourcedir2

Unable to open directory: No such file or directory

**Experiment No-3:**

**AIM:**

**To write C programs to simulate UNIX commands like cp, ls, grep.**

1. **Program for simulation of cp unix commands**

**Source code :**

#include <stdio.h> #include <stdlib.h>

int main(int argc, char \*argv[]) { FILE \*source, \*destination; char ch;

if (argc != 3) {

printf("Usage: ./cp\_simulation<source\_file><destination\_file>\n"); return 1;

}

source = fopen(argv[1], "r"); if (source == NULL) {

printf("Unable to open source file: %s\n", argv[1]); return 1;

}

destination = fopen(argv[2], "w"); if (destination == NULL) {

printf("Unable to open or create destination file: %s\n", argv[2]);

fclose(source);

return 1;

}

while ((ch = fgetc(source)) != EOF) { fputc(ch, destination);

}

fclose(source); fclose(destination);

printf("File copied successfully.\n");

return 0;

}

**Output**: ./cp\_simulation source.txt destination.txt File copied successfully.

1. Program for simulation of ls unix commands

**Source code :**

#include<stdio.h>

#include<dirent.h> main(intargc,char\*\*argv)

{

DIR\*dp;

struct dirent \*link; dp=opendir(argv[1]);

printf(“\ncontentsofthedirectory%sare\n”,argv[1]); while((link=readdir(dp))!=0)

printf(“%s”,link->d\_name); closedir(dp);

}

**Output:** ./list\_directoryexample\_dir contents of the directory example\_dir are

.

..

file1.txt file2.txt subdir1 subdir2

1. **Program for simulation of grep unix commands Source code :**

#include<stdio.h> #include<string.h> #define max 1024 void usage()

{

printf(“usage:\t. /a.out filename word \n “);

}

int main(int argc, char \*argv[])

{

FILE \*fp;

char fline[max]; char \*newline; int count=0; int occurrences=0; if(argc!=3)

{

usage();

exit(1);

}

if(!(fp=fopen(argv[1],”r”)))

{

printf(“grep: couldnot open file : %s \n”,argv[1]); exit(1);

}

while(fgets(fline,max,fp)!=NULL)

{

count++; if(newline=strchr(fline, „\n‟))

\*newline=‟\0‟; if(strstr(fline,argv[2])!=NULL)

{

printf(“%s: %d %s \n”, argv[1],count, fline); occurrences++;

}

}

}

Input and Output:

Suppose you have a text file named example.txt with the following content: Hello world!

This is a test file.

The word "test" appears twice in this test file. This is the last line.

Input: ./search\_word example.txt test Output: example.txt: 2 This is a test file.

example.txt: 3 The word "test" appears twice in this test file.

**Experiment 4:**

**Aim: Simulate the following CPU scheduling algorithms:**

**(a) Round Robin (b) SJF (c) FCFS (d) Priority**

1. **Round Robin CPU SCHEDULING ALGORITHM PROGRAM:**

**Source code :**

#include<stdio.h>

int main()

{

int st[10],bt[10],wt[10],tat[10],n,tq; int i,count=0,swt=0,stat=0,temp,sq=0; float awt=0.0,atat=0.0;

printf("Enter number of processes:"); scanf("%d",&n);

printf("Enter burst time for sequences:"); for(i=0;i<n;i++)

{

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

}

printf("Enter time quantum:"); scanf("%d",&tq);

while(1)

{

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

{

temp=tq; if(st[i]==0)

{

count++; continue;

}

if(st[i]>tq) st[i]=st[i]-tq; else if(st[i]>=0)

{

temp=st[i]; st[i]=0;

}

sq=sq+temp; tat[i]=sq;

}

if(n==count)

break;

}

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

{

wt[i]=tat[i]-bt[i]; swt=swt+wt[i]; stat=stat+tat[i];

}

awt=(float)swt/n; atat=(float)stat/n;

printf("Process\_no Burst time Wait time Turn around time"); for(i=0;i<n;i++)

printf("\n%d\t %d\t %d\t %d",i+1,bt[i],wt[i],tat[i]);

printf("\nAvg wait time is %f Avgturn around time is %f",awt,atat); return 0;

} INPUT:

Enter number of processes:3 Enter burst time for sequences:12 8

20

Enter time quantum:5

EXPECTED OUT PUT:

Process\_no Burst time Wait time Turn around time

1 12 18 30

2 8 15 23

3 20 20 40

Avg wait time is 17.666666 Avgturn around time is 31.000000

1. **SJF CPU SCHEDULING ALGORITHM PROGRAM:**

**Source code :**

#include<stdio.h>

int main()

{

int i,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0; float aw,at;

printf("enter no. of processes:\n"); scanf("%d",&n);

printf("enter the burst time of processes:"); for(i=0;i<n;i++)

scanf("%d",&bt[i]); for(i=0;i<n;i++)

{

for(j=i;j<n;j++) if(bt[i]>bt[j])

{

t=bt[i]; bt[i]=bt[j]; bt[j]=t;

}

}

for(i=0;i<n;i++) printf("%d",bt[i]); for(i=0;i<n;i++)

{ wt[0]=0;

tt[0]=bt[0];

wt[i+1]=bt[i]+wt[i];

tt[i+1]=tt[i]+bt[i+1]; w1=w1+wt[i]; t1=t1+tt[i];

}

aw=w1/n; at=t1/n;

printf("\nbt\t wt\t tt\n"); for(i=0;i<n;i++)

printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]); printf("aw=%f\n,at=%f\n",aw,at);

return 0;

} INPUT:

enter no. of processes:

3

enter the burst time of processes:12 8

20

EXPECTED OUT PUT 8 12 20

btwttt 8 0 8

12 8 20

20 20 40 aw=9.000000

,at=22.000000

1. **FCFS CPU SCHEDULING ALGORITHM PROGRAM:**

**Source code:**

#include<stdio.h> int main()

{

int i,j,bt[10],n,wt[10],tt[10],w1=0,t1=0; float aw,at;

printf("enter no. of processes:\n"); scanf("%d",&n);

printf("enter the burst time of processes:"); for(i=0;i<n;i++)

scanf("%d",&bt[i]); for(i=0;i<n;i++)

{ wt[0]=0;

tt[0]=bt[0];

wt[i+1]=bt[i]+wt[i];

tt[i+1]=tt[i]+bt[i+1]; w1=w1+wt[i]; t1=t1+tt[i];

}

aw=w1/n; at=t1/n;

printf("\nbt\t wt\t tt\n"); for(i=0;i<n;i++)

printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]); printf("aw=%f\n,at=%f\n",aw,at);

return 0;

} INPUT:

enter no. of processes:

3

enter the burst time of processes:12 8

20

EXPECTED OUTPUT:

btwttt 12 0 12

8 12 20

20 20 40 aw=10.000000

,at=24.000000

1. **Priority based CPU SCHEDULING ALGORITHM PROGRAM:**

**Source code :**

#include<stdio.h> int main()

{

int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s; float aw,at;

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

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

{

printf("The process %d:\n",i+1); printf("Enter the burst time of processes:"); scanf("%d",&bt[i]);

printf("Enter the priority of processes %d:",i+1); scanf("%d",&prior[i]);

pno[i]=i+1;

}

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

{

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

{

if(prior[i]<prior[j])

{

s=prior[i]; prior[i]=prior[j]; prior[j]=s; s=bt[i]; bt[i]=bt[j]; bt[j]=s;

s=pno[i]; pno[i]=pno[j]; pno[j]=s;

}

}

}

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

{ wt[0]=0;

tt[0]=bt[0];

wt[i+1]=bt[i]+wt[i];

tt[i+1]=tt[i]+bt[i+1]; w1=w1+wt[i]; t1=t1+tt[i]; aw=w1/n;

at=t1/n;

}

printf(" \n job \t bt \t wt \t tat \t prior\n"); for(i=0;i<n;i++)

printf("%d \t %d \t %d\t %d\t %d\n",pno[i],bt[i],wt[i],tt[i],prior[i]); printf("aw=%f \t at=%f \n",aw,at);

return 0;

} INPUT:

enter the number of processes:3 The process 1:

Enter the burst time of processes:12 Enter the priority of processes 1:3 The process 2:

Enter the burst time of processes:8 Enter the priority of processes 2:2 The process 3:

Enter the burst time of processes:20 Enter the priority of processes 3:1 EXPECTED OUTPUT:

job btwt tat prior 3 20 0 20 1

2 8 20 28 2

1 12 28 40 3

aw=16.000000 at=29.000000

**Experiment 5:**

**Aim: Control the number of ports opened by the operating system with**

* 1. **Semaphore**
  2. **Monitor**
     1. Semaphore

#include <stdio.h>

#include <stdlib.h> #include <pthread.h>

#include <semaphore.h> #include <unistd.h>

#define MAX\_PORTS 3 sem\_tsemaphore;

void\* open\_port(void\* arg) {

int port\_number = \*((int\*)arg);

printf("Thread %d waiting to open a port...\n", port\_number);

sem\_wait(&semaphore);

printf("Thread %d opened a port.\n", port\_number); sleep(2);

printf("Thread %d closed the port.\n", port\_number);

sem\_post(&semaphore);

free(arg); return NULL;

}

int main() {

sem\_init(&semaphore, 0, MAX\_PORTS); pthread\_tthreads[5];

for (int i = 0; i< 5; i++) {

int\* port\_number = malloc(sizeof(int));

\*port\_number = i + 1;

pthread\_create(&threads[i], NULL, open\_port, port\_number);

}

for (int i = 0; i< 5; i++) { pthread\_join(threads[i], NULL);

}

sem\_destroy(&semaphore);

return 0;

}

**Output:**

Thread 1 waiting to open a port... Thread 1 opened a port.

Thread 2 waiting to open a port... Thread 2 opened a port.

Thread 3 waiting to open a port... Thread 3 opened a port.

Thread 4 waiting to open a port... Thread 5 waiting to open a port... Thread 1 closed the port.

Thread 4 opened a port. Thread 2 closed the port. Thread 5 opened a port. Thread 3 closed the port. Thread 4 closed the port. Thread 5 closed the port.

* + 1. **Monitors** #include <stdio.h> #include <stdlib.h>

#include <pthread.h> #include <unistd.h>

#define MAX\_PORTS 3

int open\_ports = 0; pthread\_mutex\_tmutex; pthread\_cond\_tcond;

void\* open\_port(void\* arg) {

int port\_number = \*((int\*)arg);

printf("Thread %d waiting to open a port...\n", port\_number); pthread\_mutex\_lock(&mutex);

while (open\_ports>= MAX\_PORTS) {

printf("Thread %d is waiting since all ports are busy.\n", port\_number); pthread\_cond\_wait(&cond, &mutex); // Block this thread until the condition is met

}

open\_ports++;

printf("Thread %d opened a port. Currently opened ports: %d\n", port\_number, open\_ports); pthread\_mutex\_unlock(&mutex);

sleep(2); pthread\_mutex\_lock(&mutex);

*open\_ports--;*

printf("Thread %d closed the port. Currently opened ports: %d\n", port\_number, open\_ports); pthread\_cond\_signal(&cond);

pthread\_mutex\_unlock(&mutex);

free(arg); return NULL;

}

int main() { pthread\_tthreads[5];

pthread\_mutex\_init(&mutex, NULL); pthread\_cond\_init(&cond, NULL);

for (int i = 0; i< 5; i++) {

int\* port\_number = malloc(sizeof(int));

\*port\_number = i + 1;

pthread\_create(&threads[i], NULL, open\_port, port\_number);

}

for (int i = 0; i< 5; i++) { pthread\_join(threads[i], NULL);

}

pthread\_mutex\_destroy(&mutex); pthread\_cond\_destroy(&cond);

return 0;

}

# Output:

Thread 1 waiting to open a port...

Thread 1 opened a port. Currently opened ports: 1 Thread 2 waiting to open a port...

Thread 2 opened a port. Currently opened ports: 2 Thread 3 waiting to open a port...

Thread 3 opened a port. Currently opened ports: 3 Thread 4 waiting to open a port...

Thread 4 is waiting since all ports are busy. Thread 5 waiting to open a port...

Thread 5 is waiting since all ports are busy.

Thread 1 closed the port. Currently opened ports: 2 Thread 4 opened a port. Currently opened ports: 3 Thread 2 closed the port. Currently opened ports: 2 Thread 5 opened a port. Currently opened ports: 3 Thread 3 closed the port. Currently opened ports: 2 Thread 4 closed the port. Currently opened ports: 1 Thread 5 closed the port. Currently opened ports: 0

**Experiment No-6:**

**AIM: Write a program to illustrate concurrent execution of threads using pthreads library.**

**SOURCE CODE :**

#include <stdio.h> #include <stdlib.h> #include <pthread.h> #include <unistd.h>

#define NUM\_THREADS 5

void\* print\_thread\_info(void\* threadid) { long tid;

tid = (long)threadid;

printf("Thread %ld: Starting...\n", tid); sleep(1);

printf("Thread %ld: Exiting...\n", tid); pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS]; int rc;

long t;

for (t = 0; t < NUM\_THREADS; t++) {

printf("Main: Creating thread %ld\n", t);

rc = pthread\_create(&threads[t], NULL, print\_thread\_info, (void\*)t); if (rc) {

printf("Error: Unable to create thread %ld, %d\n", t, rc); exit(-1);

}

}

for (t = 0; t < NUM\_THREADS; t++) {

pthread\_join(threads[t], NULL);

}

printf("Main: All threads completed.\n"); pthread\_exit(NULL);

}

# OUT PUT:

Main: creating thread 0 Main: creating thread 1 Main: creating thread 2 Main: creating thread 3 Main: creating thread 4 Thread 0 is starting... Thread 1 is starting... Thread 2 is starting... Thread 3 is starting... Thread 4 is starting... Thread 0 is finishing... Thread 1 is finishing... Thread 2 is finishing... Thread 3 is finishing... Thread 4 is finishing...

All threads have completed.

**Experiment No-7:**

**AIM: Write a program to solve producer-consumer problem using Semaphores. SOURCE CODE :**

#include <stdio.h> #include <stdlib.h> #include <pthread.h> #include <semaphore.h> #include <unistd.h> #define BUFFER\_SIZE 5 int buffer[BUFFER\_SIZE]; int count = 0;

sem\_t empty; sem\_t full;

pthread\_mutex\_t mutex; void\* producer(void\* arg) {

int item; while (1) {

item = rand() % 100; sem\_wait(&empty); pthread\_mutex\_lock(&mutex); buffer[count] = item;

count++;

printf("Produced: %d. Buffer count: %d\n", item, count); pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

sleep(rand() % 2); // Sleep for a random time

}

}

void\* consumer(void\* arg) { int item;

while (1) { sem\_wait(&full);

pthread\_mutex\_lock(&mutex); count--;

item = buffer[count];

printf("Consumed: %d. Buffer count: %d\n", item, count); pthread\_mutex\_unlock(&mutex);

sem\_post(&empty); sleep(rand() % 2);

}

}

int main() {

pthread\_t prod[3], cons[3];

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0); pthread\_mutex\_init(&mutex, NULL); for (int i = 0; i < 3; i++) {

pthread\_create(&prod[i], NULL, producer, NULL);

}

for (int i = 0; i < 3; i++) {

pthread\_create(&cons[i], NULL, consumer, NULL);

}

for (int i = 0; i < 3; i++) { pthread\_join(prod[i], NULL); pthread\_join(cons[i], NULL);

}

sem\_destroy(&empty);

sem\_destroy(&full); pthread\_mutex\_destroy(&mutex); return 0;

}

# OUT PUT :

Producer 1 produced: 12

Producer 1 produced: 34

Producer 2 produced: 45

Consumer 1 consumed: 12

Consumer 2 consumed: 34

Producer 1 produced: 23

Consumer 1 consumed: 45

Producer 2 produced: 67

Consumer 2 consumed: 23

Consumer 1 consumed: 67

Producer 1 produced: 89

Producer 2 produced: 10

Consumer 2 consumed: 89

Consumer 1 consumed: 10

Producer 1 produced: 5

Producer 2 produced: 39

Consumer 2 consumed: 5

Consumer 1 consumed: 39

**Experiment No-8:**

**AIM : Implement the following memory allocation methods for fixed partition a) First fit b) Worst fit c) Best fit**

**First Fit Memory Allocation Source Code: Source code :**

#include <stdio.h>

void firstFit(int blocks[]

, int m,

int processes[], int n) {

int allocation[n];

for (int i = 0; i< n; i++) { allocation[i] = -1;

}

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

for (int j = 0; j < m; j++) {

if (blocks[j] >= processes[i]) { allocation[i] = j;

blocks[j] -= processes[i]; break;

}

}

}

printf("First Fit Allocation:\n");

for (int i = 0; i< n; i++) { if (allocation[i] != -1)

printf("Process %d allocated to Block %d\n", i + 1, allocation[i] + 1); else

printf("Process %d not allocated\n", i + 1);

}

}

int main() {

int blocks[] = {100, 500, 200, 300, 600};

int processes[] = {212, 417, 112, 426}; int m = sizeof(blocks) / sizeof(blocks[0]);

int n = sizeof(processes) / sizeof(processes[0]); firstFit(blocks, m, processes, n);

return 0;

}

# OUTPUT:

First Fit Allocation:

Process 1 allocated to Block 2

Process 2 allocated to Block 5

Process 3 allocated to Block 2 Process 4 not allocated

**B .Best Fit Memory Allocation Source code: Source code:**

#include <stdio.h>

void bestFit(int blocks[], int m, int processes[], int n) { int allocation[n];

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

allocation[i] = -1; // Initialize allocation array

}

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

int bestIdx = -1;

for (int j = 0; j < m; j++) {

if (blocks[j] >= processes[i]) {

if (bestIdx == -1 || blocks[j] < blocks[bestIdx]) { bestIdx = j; // Select the smallest block that fits

}

}

}

if (bestIdx != -1) { allocation[i] = bestIdx;

blocks[bestIdx] -= processes[i];

}

}

printf("Best Fit Allocation:\n"); for (int i = 0; i< n; i++) {

if (allocation[i] != -1)

printf("Process %d allocated to Block %d\n", i + 1, allocation[i] + 1); else

printf("Process %d not allocated\n", i + 1);

}

}

int main() {

int blocks[] = {100, 500, 200, 300, 600}; // Memory blocks

int processes[] = {212, 417, 112, 426}; // Processes requiring memory int m = sizeof(blocks) / sizeof(blocks[0]);

int n = sizeof(processes) / sizeof(processes[0]);

bestFit(blocks, m, processes, n);

return 0;

}

# Output:

Best Fit Allocation:

Process 1 allocated to Block 4

Process 2 allocated to Block 2

Process 3 allocated to Block 3

Process 4 allocated to Block 5

* + 1. **Worst Fit Memory Allocation Source Code: Source code:**

#include <stdio.h>

void worstFit(int blocks[], int m, int processes[], int n) { int allocation[n];

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

allocation[i] = -1; // Initialize allocation array

}

for (int i = 0; i< n; i++) { int worstIdx = -1;

for (int j = 0; j < m; j++) {

if (blocks[j] >= processes[i]) {

if (worstIdx == -1 || blocks[j] > blocks[worstIdx]) { worstIdx = j; // Select the largest block that fits

}

}

}

if (worstIdx != -1) { allocation[i] = worstIdx;

blocks[worstIdx] -= processes[i];

}

}

printf("Worst Fit Allocation:\n"); for (int i = 0; i< n; i++) {

if (allocation[i] != -1)

printf("Process %d allocated to Block %d\n", i + 1, allocation[i] + 1); else

printf("Process %d not allocated\n", i + 1);

}

}

int main() {

int blocks[] = {100, 500, 200, 300, 600}; // Memory blocks

int processes[] = {212, 417, 112, 426}; // Processes requiring memory int m = sizeof(blocks) / sizeof(blocks[0]);

int n = sizeof(processes) / sizeof(processes[0]); worstFit(blocks, m, processes, n);

return 0;

}

# Output:

Worst Fit Allocation:

Process 1 allocated to Block 5

Process 2 allocated to Block 2

Process 3 allocated to Block 5 Process 4 not allocated

**Experiment No-9:**

**AIM Simulate the following page replacement algorithms a) FIFO b) LRU c) LFU**

1. **FIFO Source Code: Source code:**

#include <stdio.h>

void fifoPageReplacement(int pages[], int n, int capacity) { int frames[capacity];

int front = 0, page\_faults = 0; for (int i = 0; i< capacity; i++) frames[i] = -1;

printf("FIFO Page Replacement:\n"); for (int i = 0; i< n; i++) {

int page = pages[i]; int flag = 0;

for (int j = 0; j < capacity; j++) { if (frames[j] == page) {

flag = 1; // Page found, no fault

break;

}

}

if (flag == 0) {

frames[front] = page;

front = (front + 1) % capacity; page\_faults++;

printf("Page %d caused a page fault.\n", page);

}

printf("Frames: [");

for (int j = 0; j < capacity; j++) { if (frames[j] != -1) {

printf("%d ", frames[j]);

} else { printf("- ");

}

}

printf("]\n");

}

printf("Total Page Faults: %d\n\n", page\_faults);

}

int main() {

int pages[] = {1, 3, 0, 3, 5, 6, 3, 1, 6, 3};

int n = sizeof(pages) / sizeof(pages[0]); int capacity = 3;

fifoPageReplacement(pages, n, capacity); return 0;

}

# Output :

Page 1 caused a page fault. Frames: [1 - - ]

Page 3 caused a page fault. Frames: [1 3 - ]

Page 0 caused a page fault. Frames: [1 3 0 ]

Frames: [1 3 0 ]

Page 5 caused a page fault. Frames: [5 3 0 ]

Page 6 caused a page fault. Frames: [5 6 0 ]

Frames: [5 6 0 ]

Frames: [5 6 0 ]

Frames: [5 6 0 ]

Total Page Faults: 5

# LRU Source Code: Source code :

#include <stdio.h>

void lruPageReplacement(int pages[], int n,

int capacity)

{

int frames[capacity], counter[capacity], time = 0; int page\_faults = 0;

for (int i = 0; i< capacity; i++) { frames[i] = -1;

counter[i] = 0;

}

printf("LRU Page Replacement:\n"); for (int i = 0; i< n; i++) {

int page = pages[i]; int flag = 0, least = 0;

for (int j = 0; j < capacity; j++) { if (frames[j] == page) {

flag = 1; // Page found, no fault

counter[j] = ++time; // Update time for LRU

break;

}

}

if (flag == 0) { // Page fault occurred for (int j = 1; j < capacity; j++) {

if (counter[j] < counter[least]) { least = j;

}

}

frames[least] = page; // Replace the least recently used page counter[least] = ++time; // Update time

page\_faults++;

printf("Page %d caused a page fault.\n", page);

}

printf("Frames: [");

for (int j = 0; j < capacity; j++) { if (frames[j] != -1) {

printf("%d ", frames[j]);

} else { printf("- ");

}

}

printf("]\n");

}

printf("Total Page Faults: %d\n\n", page\_faults);

}

int main() {

int pages[] = {1, 3, 0, 3, 5, 6, 3, 1, 6, 3};

int n = sizeof(pages) / sizeof(pages[0]); int capacity = 3;

lruPageReplacement(pages, n, capacity); return 0;

}

# OUTPUT:

Page 1 caused a page fault. Frames: [1 - - ]

Page 3 caused a page fault. Frames: [1 3 - ]

Page 0 caused a page fault. Frames: [1 3 0 ]

Frames: [1 3 0 ]

Page 5 caused a page fault. Frames: [5 3 0 ]

Page 6 caused a page fault. Frames: [5 6 0 ]

Frames: [5 6 0 ]

Frames: [5 6 0 ]

Frames: [5 6 0 ]

Total Page Faults: 5

# LFU Source Code:

**Source code :**

#include <stdio.h>

void lfuPageReplacement(int pages[], int n,

int capacity)

{

int frames[capacity], frequency[capacity], page\_faults = 0; for (int i = 0; i< capacity; i++) {

frames[i] = -1;

frequency[i] = 0;

}

printf("LFU Page Replacement:\n"); for (int i = 0; i< n; i++) {

int page = pages[i];

int flag = 0, least = 0;

for (int j = 0; j < capacity; j++) { if (frames[j] == page) {

flag = 1; // Page found, no fault

frequency[j]++; // Increase frequency of the page

break;

}

}

if (flag == 0) { // Page fault occurred

for (int j = 1; j < capacity; j++) { if (frequency[j] < frequency[least]) {

least = j;

}

}

frames[least] = page; // Replace the least frequently used page frequency[least] = 1; // Reset the frequency for the new page

page\_faults++;

printf("Page %d caused a page fault.\n", page);

}

printf("Frames: [");

for (int j = 0; j < capacity; j++) { if (frames[j] != -1) {

printf("%d ", frames[j]);

} else { printf("- ");

}

}

printf("]\n");

}

printf("Total Page Faults: %d\n\n", page\_faults);

}

int main() {

int pages[] = {1, 3, 0, 3, 5, 6, 3, 1, 6, 3};

int n = sizeof(pages) / sizeof(pages[0]); int capacity = 3;

lfuPageReplacement(pages, n, capacity); return 0;

}

# Output:

Page 1 caused a page fault. Frames: [1 - - ]

Page 3 caused a page fault. Frames: [1 3 - ]

Page 0 caused a page fault. Frames: [1 3 0 ]

Frames: [1 3 0 ]

Page 5 caused a page fault. Frames: [5 3 0 ]

Page 6 caused a page fault. Frames: [5 6 0 ]

Frames: [5 6 0 ]

Frames: [5 6 0 ]

Frames: [5 6 0 ] Total Page Faults: 5

**Experiment No: 10**

**Aim: Simulate Paging Technique of memory management. Source Code:**

#include <stdio.h> #include <stdlib.h> #define PAGE\_SIZE 4

#define MEMORY\_SIZE 16

#define NUM\_PAGES MEMORY\_SIZE / PAGE\_SIZE

int main() {

int logical\_address, page\_number, offset, physical\_address; int page\_table[NUM\_PAGES];

for (int i = 0; i < NUM\_PAGES; i++) { page\_table[i] = i;

}

printf("PAGE TABLE:\n");

printf("Page Number -> Frame Number\n"); for (int i = 0; i < NUM\_PAGES; i++) {

printf(" %d -> %d\n", i, page\_table[i]);

}

printf("\nEnter a logical address (0 to %d): ", MEMORY\_SIZE - 1); scanf("%d", &logical\_address);

if (logical\_address < 0 || logical\_address >= MEMORY\_SIZE) {

printf("Invalid logical address! Please enter an address between 0 and %d.\n", MEMORY\_SIZE

- 1);

return 1;

}

page\_number = logical\_address / PAGE\_SIZE; offset = logical\_address % PAGE\_SIZE;

physical\_address = (page\_table[page\_number] \* PAGE\_SIZE) + offset; printf("\nLogical Address: %d\n", logical\_address);

printf("Page Number: %d, Offset: %d\n", page\_number, offset);

printf("Physical Address: %d (Frame Number: %d, Offset: %d)\n", physical\_address, page\_table[page\_number], offset);

return 0;

}

# Output:

PAGE TABLE:

Page Number -> Frame Number

0 -> 0

1 -> 1

2 -> 2

3 -> 3

Enter a logical address (0 to 15): 6

Logical Address: 6

Page Number: 1, Offset: 2

Physical Address: 6 (Frame Number: 1, Offset: 2)

**Experiment No: 11**

**Aim: Implement Bankers Algorithm for Dead Lock avoidance and prevention Source Code:**

#include <stdio.h> #include <stdbool.h> #define P 5

#define R 3

bool isSafe(int processes[], int avail[], int max[][R], int alloc[][R], int need[][R]) { int work[R], finish[P] = {0};

for (int i = 0; i < R; i++) { work[i] = avail[i];

}

int safeSeq[P];

int count = 0; while (count < P) {

bool found = false;

for (int p = 0; p < P; p++) {

if (!finish[p]) { // If process is not finished bool possible = true;

for (int j = 0; j < R; j++) {

if (need[p][j] > work[j]) { possible = false;

break;

}

}

if (possible) {

for (int k = 0; k < R; k++) { work[k] += alloc[p][k];

}

safeSeq[count++] = p; finish[p] = 1;

found = true;

}

}

}

if (!found) { return false;

}

}

printf("System is in a safe state.\nSafe sequence is: "); for (int i = 0; i < P; i++) {

printf("%d ", safeSeq[i]);

}

printf("\n"); return true;

}

int main() {

int processes[P] = {0, 1, 2, 3, 4};

int avail[R] = {3, 3, 2};

int max[P][R] = {{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}};

int alloc[P][R] = {{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}};

int need[P][R];

for (int i = 0; i < P; i++) { for (int j = 0; j < R; j++) {

need[i][j] = max[i][j] - alloc[i][j];

}

}

if (!isSafe(processes, avail, max, alloc, need)) { printf("System is not in a safe state.\n");

}

return 0;

}

# Output:

System is in a safe state. Safe sequence is: 1 3 4 0 2

**Experiment No: 12**

**Aim: Simulate the following file allocation strategies a) Sequential b) Linked c) Indexed**

1. Sequential Allocation Source Code:

#include <stdio.h>

#define MAX\_BLOCKS 100 int blocks[MAX\_BLOCKS];

void sequentialAllocation(int startBlock, int fileSize) { int i;

for (i = startBlock; i< (startBlock + fileSize); i++) { if (blocks[i] == 1) {

printf("Block %d is already allocated. Sequential allocation failed.\n", i); return;

}

}

for (i = startBlock; i< (startBlock + fileSize); i++) { blocks[i] = 1;

}

printf("File allocated using Sequential Allocation from block %d to block %d.\n", startBlock, startBlock + fileSize - 1);

}

int main() {

int startBlock, fileSize;

for (int i = 0; i< MAX\_BLOCKS; i++) { blocks[i] = 0;

}

printf("Enter the starting block and file size: "); scanf("%d%d", &startBlock, &fileSize); sequentialAllocation(startBlock, fileSize);

return 0;

}

# Output:

Enter the starting block and file size: 5 3

File allocated using Sequential Allocation from block 5 to block 7.

# Linked allocation Source Code:

#include <stdio.h>

#define MAX\_BLOCKS 100 struct Block {

int nextBlock; int isAllocated;

};

struct Block blocks[MAX\_BLOCKS];

void linkedAllocation(int startBlock, int fileSize) { int count = 0, currentBlock = startBlock;

while (count <fileSize) {

if (blocks[currentBlock].isAllocated == 1) {

printf("Block %d is already allocated. Linked allocation failed.\n", currentBlock); return;

}

blocks[currentBlock].isAllocated = 1; if (count <fileSize - 1) {

printf("Enter the next block number after %d: ", currentBlock); scanf("%d", &blocks[currentBlock].nextBlock);

currentBlock = blocks[currentBlock].nextBlock;

} else {

blocks[currentBlock].nextBlock = -1; // End of file

}

count++;

}

printf("File allocated using Linked Allocation starting from block %d.\n", startBlock);

}

int main() {

int startBlock, fileSize;

for (int i = 0; i< MAX\_BLOCKS; i++) { blocks[i].isAllocated = 0;

blocks[i].nextBlock = -1;

}

printf("Enter the starting block and file size: "); scanf("%d%d", &startBlock, &fileSize); linkedAllocation(startBlock, fileSize);

return 0;

}

**Output:**

Enter the starting block and file size: 3 3 Enter the next block number after 3: 5 Enter the next block number after 5: 7

File allocated using Linked Allocation starting from block 3.

1. **Indexed Allocation Source Code:**

#include <stdio.h>

#define MAX\_BLOCKS 100

#define MAX\_INDEX\_SIZE 10 int blocks[MAX\_BLOCKS];

int indexBlock[MAX\_INDEX\_SIZE];

void indexedAllocation(int indexBlockNo, int fileSize) { int i, dataBlock;

for (i = 0; i< MAX\_INDEX\_SIZE; i++) {

indexBlock[i] = -1;

}

printf("Enter the block numbers to store the file data:\n")

for (i = 0; i<fileSize; i++) { scanf("%d", &dataBlock);

if (blocks[dataBlock] == 1) {

printf("Block %d is already allocated. Indexed allocation failed.\n", dataBlock); return;

}

blocks[dataBlock] = 1; indexBlock[i] = dataBlock;

}

printf("File allocated using Indexed Allocation. Index block is at block %d.\n", indexBlockNo); printf("Index block contents: ");

for (i = 0; i<fileSize; i++) { printf("%d ", indexBlock[i]);

}

printf("\n");

}

int main() {

int indexBlockNo, fileSize;

for (int i = 0; i< MAX\_BLOCKS; i++) { blocks[i] = 0;

}

printf("Enter the index block number and file size: "); scanf("%d%d", &indexBlockNo, &fileSize); indexedAllocation(indexBlockNo, fileSize);

return 0;

}

# Output:

Enter the index block number and file size: 2 3 Enter the block numbers to store the file data:

5 7 9

File allocated using Indexed Allocation. Index block is at block 2. Index block contents: 5 7 9

**Experiment No: 13**

**Aim: Download and install nachos operating system and experiment with it**

NACHOS (Not Another Completely Heuristic Operating System) is a pedagogical operating system designed for educational purposes. It allows students to implement and experiment with various operating system functionalities, including process management, memory management, file systems, and synchronization.

# Install Prerequisites

Open the terminal and run the following command to install the required packages: sudo apt update

sudo apt install build-essential gcc g++ git

# Download NACHOS Source Code

download NACHOS from the [Stanford NACHOS page](http://web.stanford.edu/class/cs140/projects/nachos/nachos.html) Change into the downloaded directory:

cd nachos

# Compile NACHOS

Navigate to the code directory and compile the source code: cd code

make clean make

# Run NACHOS

Run NACHOS to format the filesystem:

./nachos -f

Execute a sample program:

./nachos -x ../test/halt