//FCFS/FIFO CPU SCHEDULING ALGORITHM

#include<iostream>

using namespace std;

int main(){

cout<<"Enter Jobs:"<<endl;

string a[5];

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

cin>>a[i];

cout<<"Enter the burst time:"<<endl;

int b[5];

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

cin>>b[i];

int wt[5];

int tat[5];

wt[0]=0;

tat[0]=b[0];

float avgwt=0.00;

float avgtat=0.00;

for(int i=1;i<5;i++){

tat[i]=b[i]+tat[i-1];

wt[i]=wt[i-1]+b[i-1];

}

cout<<"Process "<<"Burst time "<<"Waiting Time "<<"Turn around Time"<<endl;

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

avgtat+=tat[i];

avgwt+=wt[i];

cout<<a[i]<<" "<<b[i]<<" "<<wt[i]<<" "<<tat[i]<<endl;

}

cout<<"Average Waiting Time = "<<(avgwt/5)<<endl;

cout<<"Average Turn Ariund time = "<<(avgtat/5);

}

//SJF CPU SCHEDULING ALGORITHM

#include<iostream>

#include<algorithm>

using namespace std;

int main(){

cout<<"Enter Jobs:"<<endl;

string p[5],psjf[5];

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

cin>>p[i];

cout<<"Enter the burst time:"<<endl;

int bt[5],btsjf[5];

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

cin>>bt[i];

btsjf[i]=bt[i];

}

sort(btsjf,btsjf+5);

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

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

if(btsjf[i]==bt[j]&&bt[j]!=-1){

psjf[i]=p[j];

bt[j]=-1;

}

}

}

int wt[5];

int tat[5];

wt[0]=0;

tat[0]=btsjf[0];

float avgwt=0.00;

float avgtat=0.00;

for(int i=1;i<5;i++){

tat[i]=btsjf[i]+tat[i-1];

wt[i]=wt[i-1]+btsjf[i-1];

}

cout<<"Process "<<"Burst time "<<"Waiting Time "<<"Turn around Time"<<endl;

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

avgtat+=tat[i];

avgwt+=wt[i];

cout<<psjf[i]<<" "<<btsjf[i]<<" "<<wt[i]<<" "<<tat[i]<<endl;

}

cout<<"Average Waiting Time = "<<(avgwt/5)<<endl;

cout<<"Average Turn Ariund time = "<<(avgtat/5);

}

1.UNIX SYSTEM CALLS

(I)PROCESS MGMT FORK,WAIT,EXEC

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

int pid;

pid=fork();

if(pid<0)

printf("Error");

else if(pid==0)

{

printf("Child Process\n");

printf("Child ID: %d\n",getpid());

printf("Print ID:%d\n",getppid());

exit(0);

}

else

{

printf("Parent Process\n");

wait(NULL);

printf("Parent ID:%d\n",getpid());

}

return 0;

}

(II)DIRECTORY MGMT OPENDIR,READDIR,CLOSEDIR

#include<stdio.h>

#include<dirent.h>

#include<stdlib.h>

int main()

{

DIR \*dirp;

struct dirent \*dp;

if((dirp=opendir("C:\\Users\\Deepa\\Desktop"))==NULL)

{

printf("Can't open the file");

exit(0);

}

for(dp=readdir(dirp);dp!=NULL;dp=readdir(dirp))

{

if(dp)

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

}

closedir(dirp);

return 0;

}

(III)I/O SYSTEM CALLS

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

int main()

{

int n, fd;

char buff[25];

printf("Enter text to write in the file :\n");

n=read(0, buff, 50);

fd=open("C:\\Users\\Deepa\\Desktop\\A.txt",O\_CREAT | O\_RDWR,0777);

write(fd,buff,n);

write(1,buff,n);

int close(int fd);

return 0;

}

--------------------------------------------------------

2) C LOOK DISK SCHEDULE

#include<stdio.h>

#include<stdlib.h>

void sort(int req[], int n) {

int i, j, temp;

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

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

if(req[j] > req[j+1]) {

temp = req[j];

req[j] = req[j+1];

req[j+1] = temp;

}

}

}

}

void clook(int req[], int n, int head) {

int seek\_count = 0;

int distance, cur\_track;

int j = 0;

sort(req, n);

while(j < n && req[j] < head) {

j++;

}

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

cur\_track = req[i];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

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

cur\_track = req[i];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

printf("Total seek count = %d\n", seek\_count);

}

int main() {

int n, head;

printf("Enter the number of requests: ");

scanf("%d", &n);

int req[n];

printf("Enter the requests: ");

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

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

}

printf("Enter the initial position of the head: ");

scanf("%d", &head);

clook(req, n, head);

return 0;

}

------------------------------------------------------

3.FCFS

(I)#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[], int tat[]) {

wt[0] = 0;

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

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

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

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

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

}

void findAvgTime(int processes[], int n, int bt[]) {

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt, tat);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes Burst Time Waiting Time Turnaround Time\n");

int total\_wt = 0, total\_tat = 0;

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

total\_wt += wt[i];

total\_tat += tat[i];

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

}

float avg\_wt = (float)total\_wt / (float)n;

float avg\_tat = (float)total\_tat / (float)n;

printf("Average Waiting Time: %.2f\n", avg\_wt);

printf("Average Turnaround Time: %.2f\n", avg\_tat);

}

int main() {

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

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

int burst\_time[] = {10, 5, 8};

findAvgTime(processes, n, burst\_time);

return 0;

}

-------------------------------------------------------------

4)LOOK

#include<stdio.h>

#include<stdlib.h>

void sort(int req[], int n) {

int i, j, temp;

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

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

if(req[j] > req[j+1]) {

temp = req[j];

req[j] = req[j+1];

req[j+1] = temp;

}

}

}

}

void look(int req[], int n, int head) {

int seek\_count = 0;

int distance, cur\_track;

int i, j;

sort(req, n);

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

if(req[i] >= head)

break;

}

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

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

for(j = i - 1; j >= 0; j--) {

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

printf("Total seek count = %d\n", seek\_count);

}

int main() {

int n, head;

printf("Enter the number of requests: ");

scanf("%d", &n);

int req[n];

printf("Enter the requests: ");

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

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

}

printf("Enter the initial position of the head: ");

scanf("%d", &head);

look(req, n, head);

return 0;

}

---------------------------------------------------

5. Shortest job first

(I)#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[]) {

int rt[n];

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

rt[i] = bt[i];

int complete = 0, t = 0, minm = 10000, shortest = 0, finish\_time;

int flag = 0;

while (complete != n) {

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

if ((rt[j] <= t) && (rt[j] < minm) && (rt[j] > 0)) {

minm = rt[j];

shortest = j;

flag = 1;

}

}

if (flag == 0) {

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)

minm = 10000;

if (rt[shortest] == 0) {

complete++;

flag = 0;

finish\_time = t + 1;

wt[shortest] = finish\_time - bt[shortest];

if (wt[shortest] < 0)

wt[shortest] = 0;

}

t++;

}

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

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

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

}

void findAvgTime(int processes[], int n, int bt[]) {

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes Burst Time Waiting Time Turnaround Time\n");

int total\_wt = 0, total\_tat = 0;

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

total\_wt += wt[i];

total\_tat += tat[i];

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

}

float avg\_wt = (float)total\_wt / (float)n;

float avg\_tat = (float)total\_tat / (float)n;

printf("Average Waiting Time: %.2f\n", avg\_wt);

printf("Average Turnaround Time: %.2f\n", avg\_tat);

}

int main() {

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

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

int burst\_time[] = {6, 8, 7};

findAvgTime(processes, n, burst\_time);

return 0;

}

-------------------------------------------------

6)C SCAN DISK

#include<stdio.h>

#include<stdlib.h>

void sort(int req[], int n) {

int i, j, temp;

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

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

if(req[j] > req[j+1]) {

temp = req[j];

req[j] = req[j+1];

req[j+1] = temp;

}

}

}

}

void cscan(int req[], int n, int head) {

int seek\_count = 0;

int distance, cur\_track;

int i, j;

sort(req, n);

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

if(req[i] >= head)

break;

}

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

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

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

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

printf("Total seek count = %d\n", seek\_count);

}

int main() {

int n, head;

printf("Enter the number of requests: ");

scanf("%d", &n);

int req[n];

printf("Enter the requests: ");

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

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

}

printf("Enter the initial position of the head: ");

scanf("%d", &head);

cscan(req, n, head);

return 0;

}

--------------------------------------------------

7.ROUND ROBIN

(I)#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum) {

int rem\_bt[n];

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

rem\_bt[i] = bt[i];

int t = 0;

while (1) {

int done = 1;

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

if (rem\_bt[i] > 0) {

done = 0;

if (rem\_bt[i] > quantum) {

t += quantum;

rem\_bt[i] -= quantum;

} else {

t += rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done == 1)

break;

}

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

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

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

}

void findAvgTime(int processes[], int n, int bt[], int quantum) {

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt, quantum);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes Burst Time Waiting Time Turnaround Time\n");

int total\_wt = 0, total\_tat = 0;

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

total\_wt += wt[i];

total\_tat += tat[i];

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

}

printf("Average Waiting Time: %.2f\n", (float)total\_wt / (float)n);

printf("Average Turnaround Time: %.2f\n", (float)total\_tat / (float)n);

}

int main() {

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

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

int burst\_time[] = {10, 5, 8};

int quantum = 2;

findAvgTime(processes, n, burst\_time, quantum);

return 0;

}

------------------------------------------------

8)SCAN

#include <stdio.h>

#include <stdlib.h>

void sort(int req[], int n) {

int temp;

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

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

if (req[j] > req[j+1]) {

temp = req[j];

req[j] = req[j+1];

req[j+1] = temp;

}

}

}

}

void scan(int req[], int n, int head, int direction, int disk\_size) {

int seek\_count = 0;

int distance, cur\_track;

sort(req, n);

int i;

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

if (req[i] >= head)

break;

}

if (direction == 1) {

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

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

if (head != disk\_size - 1) {

distance = abs(disk\_size - 1 - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, disk\_size - 1, distance);

head = disk\_size - 1;

}

for (int j = i - 1; j >= 0; j--) {

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

} else {

for (int j = i - 1; j >= 0; j--) {

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

if (head != 0) {

distance = abs(head - 0);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, 0, distance);

head = 0;

}

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

cur\_track = req[j];

distance = abs(cur\_track - head);

seek\_count += distance;

printf("Disk head moves from %d to %d with seek %d\n", head, cur\_track, distance);

head = cur\_track;

}

}

printf("Total seek count = %d\n", seek\_count);

}

int main() {

int n, head, direction, disk\_size;

printf("Enter the number of requests: ");

scanf("%d", &n);

int req[n];

printf("Enter the requests: ");

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

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

}

printf("Enter the initial position of the head: ");

scanf("%d", &head);

printf("Enter the direction (1 for high, 0 for low): ");

scanf("%d", &direction);

printf("Enter the disk size: ");

scanf("%d", &disk\_size);

scan(req, n, head, direction, disk\_size);

return 0;

}

---------------------------------------------

9)DINING PHILOSOPHER

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t chopstick[5];

void \* philos(void \*);

void eat(int);

int main()

{

int i,n[5];

pthread\_t T[5];

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

sem\_init(&chopstick[i],0,1);

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

n[i]=i;

pthread\_create(&T[i],NULL,philos,(void \*)&n[i]);

}

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

pthread\_join(T[i],NULL);

}

void \* philos(void \* n)

{

int ph=\*(int \*)n;

printf("Philosopher %d wants to eat\n",ph);

printf("Philosopher %d tries to pick left chopstick\n",ph);

sem\_wait(&chopstick[ph]);

printf("Philosopher %d picks the left chopstick\n",ph);

printf("Philosopher %d tries to pick the right chopstick\n",ph);

sem\_wait(&chopstick[(ph+1)%5]);

printf("Philosopher %d picks the right chopstick\n",ph);

eat(ph);

sleep(2);

printf("Philosopher %d has finished eating\n",ph);

sem\_post(&chopstick[(ph+1)%5]);

printf("Philosopher %d leaves the right chopstick\n",ph);

sem\_post(&chopstick[ph]);

printf("Philosopher %d leaves the left chopstick\n",ph);

}

void eat(int ph)

{

printf("Philosopher %d begins to eat\n",ph);

}

--------------------------------------------------------------

10)SSTF

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

void sort(int arr[], int n) {

int temp;

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

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

if (arr[j] > arr[j+1]) {

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

void sstf(int req[], int n, int head) {

int seek\_count = 0;

int distance, cur\_track;

int visited[n];

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

visited[i] = 0;

}

printf("Seek Sequence is:\n");

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

int min = 1000, min\_index;

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

if (!visited[j] && abs(req[j] - head) < min) {

min = abs(req[j] - head);

min\_index = j;

}

}

visited[min\_index] = 1;

cur\_track = req[min\_index];

printf("%d ", cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

printf("\nTotal seek count = %d\n", seek\_count);

}

int main() {

int n, head;

printf("Enter the number of requests: ");

scanf("%d", &n);

int req[n];

printf("Enter the requests: ");

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

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

}

printf("Enter the initial position of the head: ");

scanf("%d", &head);

sstf(req, n, head);

return 0;

}

------------------------------------------------------

11).BANKER ALGO

#include <stdio.h>

#include <stdbool.h>

int n, m;

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

int need[10][10], work[10];

bool finish[10];

void calculateNeed() {

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

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

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

}

}

}

bool isSafe() {

for (int i = 0; i < n; i++) finish[i] = false;

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

int safeSequence[n], index = 0;

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

bool found = false;

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

if (!finish[p]) {

int j;

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

if (need[p][j] > work[j])

break;

}

if (j == m) {

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

work[k] += alloc[p][k];

}

safeSequence[index++] = p;

finish[p] = true;

found = true;

}

}

}

if (!found) {

printf("The system is not in a safe state\n");

return false;

}

}

printf("The system is in a safe state\nSafe sequence is: ");

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

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

}

printf("\n");

return true;

}

int main() {

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter the number of resources: ");

scanf("%d", &m);

printf("Enter the allocation matrix:\n");

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

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

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

}

}

printf("Enter the maximum matrix:\n");

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

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

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

}

}

printf("Enter the available resources:\n");

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

scanf("%d", &avail[j]);

}

calculateNeed();

if (isSafe()) {

printf("System is in a safe state.\n");

} else {

printf("System is not in a safe state.\n");

}

return 0;

}

------------------------------------------------------------

12)FCFS

#include <stdio.h>

#include <stdlib.h>

void fcfs(int req[], int n, int head) {

int seek\_count = 0;

int distance, cur\_track;

printf("Seek Sequence is:\n");

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

cur\_track = req[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

printf("%d ", cur\_track);

}

printf("\nTotal seek count = %d\n", seek\_count);

}

int main() {

int n, head;

printf("Enter the number of requests: ");

scanf("%d", &n);

int req[n];

printf("Enter the requests: ");

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

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

}

printf("Enter the initial position of the head: ");

scanf("%d", &head);

fcfs(req, n, head);

return 0;

}

-------------------------------------------------

13)First fit

#include <stdio.h>

void firstFit(int blockSize[], int m, int processSize[], 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 (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

printf(" %d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main() {

int m, n;

printf("Enter the number of blocks: ");

scanf("%d", &m);

int blockSize[m];

printf("Enter the size of each block: ");

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

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

}

printf("Enter the number of processes: ");

scanf("%d", &n);

int processSize[n];

printf("Enter the size of each process: ");

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

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

}

firstFit(blockSize, m, processSize, n);

return 0;

}

-----------------------------------------------------------

14)FILE ORGANIZATION

#include <stdio.h>

#include <string.h>

#define MAX\_FILES 100

typedef struct {

char name[30];

} File;

File directory[MAX\_FILES];

int fileCount = 0;

void createFile(char name[]) {

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

if (strcmp(directory[i].name, name) == 0) {

printf("File %s already exists.\n", name);

return;

}

}

strcpy(directory[fileCount].name, name);

fileCount++;

printf("File %s created successfully.\n", name);

}

void deleteFile(char name[]) {

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

if (strcmp(directory[i].name, name) == 0) {

for (int j = i; j < fileCount - 1; j++) {

strcpy(directory[j].name, directory[j + 1].name);

}

fileCount--;

printf("File %s deleted successfully.\n", name);

return;

}

}

printf("File %s not found.\n", name);

}

void listFiles() {

if (fileCount == 0) {

printf("No files in the directory.\n");

return;

}

printf("Files in the directory:\n");

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

printf("%s\n", directory[i].name);

}

}

int main() {

int choice;

char name[30];

while (1) {

printf("\n1. Create File\n2. Delete File\n3. List Files\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the name of the file: ");

scanf("%s", name);

createFile(name);

break;

case 2:

printf("Enter the name of the file: ");

scanf("%s", name);

deleteFile(name);

break;

case 3:

listFiles();

break;

case 4:

return 0;

default:

printf("Invalid choice!\n");

}

}

}

-------------------------------------------------

15)BEST FIT

#include <stdio.h>

void bestFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

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

allocation[i] = -1;

}

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

int bestIdx = -1;

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

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j]) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

printf(" %d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main() {

int m, n;

printf("Enter the number of blocks: ");

scanf("%d", &m);

int blockSize[m];

printf("Enter the size of each block: ");

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

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

}

printf("Enter the number of processes: ");

scanf("%d", &n);

int processSize[n];

printf("Enter the size of each process: ");

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

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

}

bestFit(blockSize, m, processSize, n);

return 0;

}

-----------------------------------------

16)OPTIMAL REPLACEMENT

#include <stdio.h>

int findOptimal(int pages[], int n, int frames[], int numFrames, int currentIndex) {

int res = -1, farthest = currentIndex;

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

int j;

for (j = currentIndex; j < n; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

res = i;

}

break;

}

}

if (j == n) {

return i;

}

}

return (res == -1) ? 0 : res;

}

void optimalPageReplacement(int pages[], int n, int numFrames) {

int frames[numFrames];

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

frames[i] = -1;

}

int hit = 0, miss = 0;

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

int j;

for (j = 0; j < numFrames; j++) {

if (frames[j] == pages[i]) {

hit++;

break;

}

}

if (j == numFrames) {

miss++;

int pos = findOptimal(pages, n, frames, numFrames, i + 1);

frames[pos] = pages[i];

}

printf("\nFrames after accessing page %d: ", pages[i]);

for (int k = 0; k < numFrames; k++) {

if (frames[k] != -1)

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

else

printf("- ");

}

}

printf("\nTotal Page Hits: %d\n", hit);

printf("Total Page Misses: %d\n", miss);

}

int main() {

int n, numFrames;

printf("Enter the number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the pages: ");

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

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

}

printf("Enter the number of frames: ");

scanf("%d", &numFrames);

optimalPageReplacement(pages, n, numFrames);

return 0;

}

--------------------------------------------------------

17)WORST FIT

#include <stdio.h>

void worstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

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

allocation[i] = -1;

}

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

int worstIdx = -1;

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

if (blockSize[j] >= processSize[i]) {

if (worstIdx == -1 || blockSize[worstIdx] < blockSize[j]) {

worstIdx = j;

}

}

}

if (worstIdx != -1) {

allocation[i] = worstIdx;

blockSize[worstIdx] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

printf(" %d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main() {

int m, n;

printf("Enter the number of blocks: ");

scanf("%d", &m);

int blockSize[m];

printf("Enter the size of each block: ");

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

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

}

printf("Enter the number of processes: ");

scanf("%d", &n);

int processSize[n];

printf("Enter the size of each process: ");

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

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

}

worstFit(blockSize, m, processSize, n);

return 0;

}

----------------------------------------------------

18)LRU

#include <stdio.h>

int findLRU(int time[], int n) {

int minimum = time[0], pos = 0;

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

if (time[i] < minimum) {

minimum = time[i];

pos = i;

}

}

return pos;

}

void LRU(int pages[], int n, int frames[], int numFrames) {

int counter = 0, time[numFrames], flag1, flag2, pos, faults = 0;

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

frames[i] = -1;

}

printf("\nPage\tFrames\n");

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

flag1 = flag2 = 0;

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

if (frames[j] == pages[i]) {

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if (flag1 == 0) {

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

if (frames[j] == -1) {

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if (flag2 == 0) {

pos = findLRU(time, numFrames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("%d\t", pages[i]);

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

if (frames[j] != -1) {

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

} else {

printf("- ");

}

}

printf("\n");

}

printf("\nTotal Page Faults = %d\n", faults);

}

int main() {

int n, numFrames;

printf("Enter the number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the pages: ");

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

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

}

printf("Enter the number of frames: ");

scanf("%d", &numFrames);

int frames[numFrames];

LRU(pages, n, frames, numFrames);

return 0;

}

-----------------------------------------------------------------

19)UNIX SYSTEM CALLS

(I)PROCESS MGMT FORK,WAIT,EXEC

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

int pid;

pid=fork();

if(pid<0)

printf("Error");

else if(pid==0)

{

printf("Child Process\n");

printf("Child ID: %d\n",getpid());

printf("Print ID:%d\n",getppid());

exit(0);

}

else

{

printf("Parent Process\n");

wait(NULL);

printf("Parent ID:%d\n",getpid());

}

return 0;

}

(II)DIRECTORY MGMT OPENDIR,READDIR,CLOSEDIR

#include<stdio.h>

#include<dirent.h>

#include<stdlib.h>

int main()

{

DIR \*dirp;

struct dirent \*dp;

if((dirp=opendir("C:\\Users\\Deepa\\Desktop"))==NULL)

{

printf("Can't open the file");

exit(0);

}

for(dp=readdir(dirp);dp!=NULL;dp=readdir(dirp))

{

if(dp)

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

}

closedir(dirp);

return 0;

}

(III)I/O SYSTEM CALLS

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

int main()

{

int n, fd;

char buff[25];

printf("Enter text to write in the file :\n");

n=read(0, buff, 50);

fd=open("C:\\Users\\Deepa\\Desktop\\A.txt",O\_CREAT | O\_RDWR,0777);

write(fd,buff,n);

write(1,buff,n);

int close(int fd);

return 0;

}

------------------------------------------

20)FIFO

#include <stdio.h>

void FIFO(int pages[], int n, int numFrames) {

int frames[numFrames];

int front = 0, faults = 0;

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

frames[i] = -1;

}

printf("\nPage\tFrames\n");

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

int flag = 0;

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

if (frames[j] == pages[i]) {

flag = 1;

break;

}

}

if (flag == 0) {

frames[front] = pages[i];

front = (front + 1) % numFrames;

faults++;

}

printf("%d\t", pages[i]);

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

if (frames[j] != -1) {

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

} else {

printf("- ");

}

}

printf("\n");

}

printf("\nTotal Page Faults = %d\n", faults);

}

int main() {

int n, numFrames;

printf("Enter the number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the pages: ");

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

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

}

printf("Enter the number of frames: ");

scanf("%d", &numFrames);

FIFO(pages, n, numFrames);

return 0;

}

--------------------------------------------------------------------------//PAGE FAULT

#include<algorithm>

#include<iostream>

using namespace std;

int main(){

int n;

cin>>n;

int p[n];

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

cin>>p[i];

}

int a=-1,b=-1,c=-1,pf=0;

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

if(p[i]==a){

continue;

}

else if(p[i]==b){

continue;

}

else if(p[i]==c){

continue;

}

else{

pf++;

if(pf%3==1){

a=p[i];

}

else if(pf%3==2){

b=p[i];

}

else{

c=p[i];

}

}

cout<<a<<" "<<b<<" "<<c<<endl;

}

cout<<"No of Page fault: "<<pf;

}

//SYSTEM CALL

#include <stdio.h>

#include <sys/stat.h>

#include <stdlib.h>

int main()

{

struct stat s;

int a; int b;

if(stat("suba",&s)==(-1))

{

perror("Error: cannot stat file");

exit(0);

}

a=s.st\_blksize;

b=s.st\_size;

printf("ALLOCATED SIZE OF THE FILE :%d\nActual size of the file :%d\n",a,b);

return 0;

}

PROCESS SYSTEM CALLS

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

void main()

{

int pid;

pid=fork();

if (pid < 0)

{

printf("The fork cannot be created");

exit(0);

}

else

if (pid==0)

{

execlp("/home/ls","ls",NULL);

printf("\n The process id of the child: %d", getpid());

printf("\n The process id of the parent: %d", getppid());

}

else{

printf("\n The process id of the parent: %d", getpid());

printf("\n The process id of the grandparent: %d", getppid());

}

}