**Exp. 3 Process Scheduling**

#include <stdio.h>

int main()

{

int pid[15];

int bt[15];

int n;

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

scanf("%d",&n);

printf("Enter process id of all the processes: ");

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

{

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

}

printf("Enter burst time of all the processes: ");

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

{

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

}

int i, wt[n];

wt[0]=0;

//for calculating waiting time of each process

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

{

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

}

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

float twt=0.0;

float tat= 0.0;

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

{

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

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

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

//calculating and printing turnaround time of each process

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

printf("\n");

//for calculating total waiting time

twt += wt[i];

//for calculating total turnaround time

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

}

float att,awt;

//for calculating average waiting time

awt = twt/n;

//for calculating average turnaround time

att = tat/n;

printf("Avg. waiting time= %f\n",awt);

printf("Avg. turnaround time= %f",att);

}

**Exp. 4 Priority scheduling algorithm. Code:** #include<stdio.h>

struct Proccess

{

int pid; int arival; int burst; int priority; int completion; int turnaround;

int waiting;

};

void swap(struct Proccess \*a,struct Proccess \*b)

{

struct Proccess temp = \*a;

\*a =\* b;

\*b=temp;

}

void sortByPriority(struct Proccess proc[],int n)

{

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

{

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

{

if(proc[j].priority < proc[i].priority ||(proc[j].priority == proc[i].priority && proc[j].arival<proc[i].arival) )

{

swap(&proc[i],&proc[j]);

} }

}

}

void calculateTimes (struct Proccess proc[],int n)

{

int currentTime=0;

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

{

if(currentTime<proc[i].arival)

{

currentTime=proc[i].arival;

}

proc[i].completion=currentTime+proc[i].burst; proc[i].turnaround=proc[i]. completion-proc[i].arival; proc[i].waiting=proc[i].turnaround-proc[i].burst;

currentTime=proc[i].completion;

}

}

void printGanttChart(struct Proccess proc[],int n)

{

printf("Gannt Chart: \n");

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

{

printf("|P%d",proc[i].pid);

}

printf("|\n"); printf("\nPID\tPriority\tArrival\tBurst\tCompletion\tTurnAround

\tWaiting\n");

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

{

printf("%d\t%d\t\t%d\t%d\t%d\t\t%d\t\t%d\n",proc[i].pid, proc[i].priority, proc[i].arival,proc[i].burst, proc[i].completion, proc[i].

turnaround, proc[i].waiting);

}

}

int main()

{

int n;

printf("Enter the number of proccesses:"); scanf("%d",&n); struct Proccess proc[n];

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

{

printf("Enter ProccessID,Arrival Time, Burst Time, Priority for Proccess%d", i+1); scanf("%d %d %d %d",&proc[i].pid, &proc[i]. arival, &proc[i].

burst, &proc[i].priority);

}

sortByPriority(proc,n); calculateTimes(proc,n); printGanttChart(proc,n);

return 0;

}

**Output:**

Enter the number of proccesses:3

Enter ProccessID, Arrival Time, Burst Time, Priority for

Proccess11 2 5 2

Enter ProccessID, Arrival Time, Burst Time, Priority for

Proccess21 10 14 1

Enter ProccessID, Arrival Time, Burst Time, Priority for Proccess33 4 8 3 Gannt Chart:

| P1 | P1 | P3 |

PID Priority Arrival Burst Completion TurnAround Waiting

1 1 10 14 24 14 0

1 2 2 5 29 27 22

3 3 4 8 37 33 25

**Exp. 5 Producer Consumer**

// C program for the above approach

#include <stdio.h>

#include <stdlib.h>

// Initialize a mutex to 1

int mutex = 1;

// Number of full slots as 0

int full = 0;

// Number of empty slots as size

// of buffer

int empty = 10, x = 0;

// Function to produce an item and

// add it to the buffer

void producer()

{

// Decrease mutex value by 1

--mutex;

// Increase the number of full

// slots by 1

++full;

// Decrease the number of empty

// slots by 1

--empty;

// Item produced

x++;

printf("\nProducer produces"

"item %d",

x);

// Increase mutex value by 1

++mutex;

}

// Function to consume an item and

// remove it from buffer

void consumer()

{

// Decrease mutex value by 1

--mutex;

// Decrease the number of full

// slots by 1

--full;

// Increase the number of empty

// slots by 1

++empty;

printf("\nConsumer consumes "

"item %d",

x);

x--;

// Increase mutex value by 1

++mutex;

}

// Driver Code

int main()

{

int n, i;

printf("\n1. Press 1 for Producer"

"\n2. Press 2 for Consumer"

"\n3. Press 3 for Exit");

// Using '#pragma omp parallel for'

// can give wrong value due to

// synchronization issues.

// 'critical' specifies that code is

// executed by only one thread at a

// time i.e., only one thread enters

// the critical section at a given time

#pragma omp critical

for (i = 1; i > 0; i++) {

printf("\nEnter your choice:");

scanf("%d", &n);

// Switch Cases

switch (n) {

case 1:

// If mutex is 1 and empty

// is non-zero, then it is

// possible to produce

if ((mutex == 1)

&& (empty != 0)) {

producer();

}

// Otherwise, print buffer

// is full

else {

printf("Buffer is full!");

}

break;

case 2:

// If mutex is 1 and full

// is non-zero, then it is

// possible to consume

if ((mutex == 1)

&& (full != 0)) {

consumer();

}

// Otherwise, print Buffer

// is empty

else {

printf("Buffer is empty!");

}

break;

// Exit Condition

case 3:

exit(0);

break;

}

}

}

**Exp. 6 Dining Philosophers problem. Code:**

#include <stdio.h> #include<stdlib.h> int sticks[5]={1,1,1,1,1};

int mutex=1; int phil=5; int eating[5]={0,0,0,0,0}; void pick\_chopsticks(int phil){

--mutex; if(eating[phil]==1){

printf("philo%d is already eating.They cannot pick up chopsticks again.\n",phil+1);

}else if(sticks[phil]==1&&sticks[(phil+1)%phil]==1){

sticks[phil]=sticks[(phil+1)%phil]=0; eating[phil]=1;

printf("philo%d is now eating\n",phil+1);

}else{

printf("philo%d is waiting for chopsticks\n",phil+1);

}

++mutex;

}

void put\_chopsticks(int phil){

--mutex; if(eating[phil]==1){

sticks[phil]=sticks[(phil+1)%phil]=1; eating[phil]=0;

printf("philo %d puts down chopsticks %d and

%d\n",phil+1,phil+1,(phil+1)%phil+1);

printf("philo%d is done eating\n",phil+1);

}else{

printf("philo%d cannot put down chopsticks because they are not holding any\n",phil+1);

}

++mutex;

}

int main(){

int choice,philosophere;

printf("press 1 for pick\n2.press2 for to put\n3.press3 to exit"); while(1){ printf("enter"); scanf("%d",&choice); switch(choice){ case 1:

printf("enter num(1-5):"); scanf("%d",&phil); if(phil<1||phil>5){ printf("invalid philo num!\n");

}else

{pick\_chopsticks(phil-1);} break; case 2:

printf("enter num(1-5):"); scanf("%d",&phil); if(phil<1||phil>5){

printf("invalid philo num!\n");

}else

{put\_chopsticks(phil-1);} break; case 3: exit(0); default:

printf("incorrect");

}} return 0;

}

**Output:**

press 1 for pick 2.press2 for to put 3.press3 to exitenter1 enter num(1-5):2 philo2 is now eating enter1 enter num(1-5):3 philo3 is waiting for chopsticks enter1 enter num(1-5):4 philo4 is waiting for chopsticks enter1 enter num(1-5):5 philo5 is waiting for chopsticks enter3

**Exp. 7 (Deadlock Avoidance) Banker's Algorithm. Code:**

#include <stdio.h>

int main()

{

int n = 5, m = 3; int alloc[5][3] =

{

{0, 1, 0},

{2, 0, 0}, {3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

int max[5][3] =

{

{7, 5, 3},

{3, 2, 2}, {9, 0, 2}, {2, 2, 2},

{4, 3, 3}

};

int avail[3] = {3, 3, 2}; int f[n], ans[n], ind = 0;

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

{ f[k] = 0;

}

int need[n][m];

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

{

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

{

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

}

}

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

{

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

{

if (f[i] == 0)

{

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

{

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

{

flag = 1;

break;

}

}

if (flag == 0)

{

ans[ind++] = i; for (int y = 0; y < m; y++)

{

avail[y] += alloc[i][y];

} f[i] = 1;

}

}

}

}

int flag = 1;

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

{

if (f[i] == 0)

{

flag = 0;

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

}

}

if (flag == 1)

{

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

printf("Following is the safe sequence:\n"); for (int i = 0; i < n - 1; i++) {

printf("P%d -> ", ans[i]);

}

printf("P%d\n", ans[n - 1]);

}

return 0;

}

**Exp. 9 Page replacement algorithms.**

**a)First in First Out (FIFO) Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_PAGES 12 #define FRAME\_SIZE 3

void displayFrames(int frames[], int size)

{

printf("Frames: ");

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

{

if (frames[i] != -1)

{

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

}

else

{

printf("- ");

}

}

printf("\n");

}

int main()

{

int pages[MAX\_PAGES] = {0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4};

int frames[FRAME\_SIZE]; int pageFaults = 0;

int time = 0;

for (int i = 0; i < FRAME\_SIZE; i++)

{

frames[i] = -1;

}

printf("Page reference string: ");

for (int i = 0; i < MAX\_PAGES; i++)

{

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

}

printf("\n");

for (int i = 0; i < MAX\_PAGES; i++)

{

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

for (int j = 0; j < FRAME\_SIZE; j++)

{

if (frames[j] == page)

{

found = 1;

break;

}

}

if (!found)

{

frames[time % FRAME\_SIZE] = page; time++; pageFaults++;

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

displayFrames(frames, FRAME\_SIZE);

} else {

printf("Page %d already in frame. No page fault.\n", page);

}

}

printf("Total page faults: %d\n", pageFaults); return 0;

}

**Output:**

Page reference string: 0 1 2 3 0 1 4 0 1 2 3 4 Page 0 caused a page fault.

Frames: 0 - -

Page 1 caused a page fault.

Frames: 0 1 -

Page 2 caused a page fault.

Frames: 0 1 2

Page 3 caused a page fault.

Frames: 3 1 2

Page 0 caused a page fault.

Frames: 3 0 2

Page 1 caused a page fault.

Frames: 3 0 1

Page 4 caused a page fault.

Frames: 4 0 1

Page 0 already in frame. No page fault.

Page 1 already in frame. No page fault.

Page 2 caused a page fault.

Frames: 4 2 1

Page 3 caused a page fault.

Frames: 4 2 3

Page 4 already in frame. No page fault.

Total page faults: 9

**b) Least Recently Used (LRU) Code:**

#include<stdio.h> int findLRU(int time[],int n)

{

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

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

{

if(time[i]<minimum)

{

minimum=time[i];

pos=i;

}

}

return pos;

}

int main()

{

int

no\_of\_frames,no\_of\_pages,frames[3],pages[13],counter=0,time[3],flag1,fl ,i,j,pos,faults=0;

printf("Enter no of frames:"); scanf("%d",&no\_of\_frames); printf("Enter no of pages:"); scanf("%d",&no\_of\_pages); printf("Enter reference string:");

for(i=0;i<no\_of\_pages;++i)

{

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

}

for(i=0;i<no\_of\_frames;++i)

{

frames[i]=-1;

}

for(i=0;i<no\_of\_pages;++i)

{

flag1=flag2=0;

for(j=0;j<no\_of\_frames;++j)

{

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

{

counter++; time[j]=counter; flag1=flag2=1; break;}}

if(flag1==0)

{

for(j=0;j<no\_of\_frames;++j)

{

if(frames[j]==-1)

{

counter++; faults++; frames[j]=pages[i]; time[j]=counter; flag2=1;

break;

}

}

}

if(flag2==0)

{

pos=findLRU(time,no\_of\_frames); counter++; faults++; frames[pos]=pages[i];

time[pos]=counter;

}

printf("\nFrames:");

for(j=0;j<no\_of\_frames;++j)

{

if(frames[j]==-1) printf("-"); else printf("%d",frames[j]);

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

return 0;

}

**Output:**

Enter no of frames:3

Enter no of pages:13

Enter reference string:7 0 1 2 0 3 0 4 2 3 0 3 2

Frames:7-- Frames:70- Frames:701

Frames:201

Frames:201

Frames:203

Frames:203

Frames:403

Frames:402

Frames:432

Frames:032

Frames:032

Frames:032

Total Page Faults=9

**c) Optimal. Code:**

#include <stdio.h>

#define MAX\_FRAMES 10

#define MAX\_PAGES 50

int findOptimalPage(int pages[], int page\_count, int frames[], int frame\_cou int

current\_index)

{

int i, j, farthest = current\_index, replace\_index = -1; for (i = 0; i < frame\_count; i++)

{

int found = 0;

for (j = current\_index; j < page\_count; j++)

{

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

{

found = 1;

if (j > farthest)

{

farthest = j;

replace\_index = i;

}

break;

}

}

if (!found)

{

return i;

}

}

replace\_index;

}

void optimalPageReplacement(int pages[], int page\_count, int frame\_count)

{

int frames[MAX\_FRAMES], i, j, page\_faults = 0; for (i=0; i<frame\_count; i++)

{

frames[i] = -1;

}

printf("Reference String: ");

for (i = 0; i < page\_count; i++)

{

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

}

printf("\n");

for (i = 0; i < page\_count; i++)

{

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

for (j = 0; j <frame\_count; j++)

{

if (frames[j] == page)

{

found = 1;

break;

}

}

if (! found)

{

int replace\_index = findOptimalPage(pages, page\_count, frames, frame\_count, i); frames[replace\_index] = page; page\_faults++; printf("Page Fault: %d -> ", page);

for (j = 0; j <frame\_count; j++)

{

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

}

printf("\n");

}

}

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

}

int main() { int pages [MAX\_PAGES], page\_count, frame\_count, i; printf("Enter the number of pages: "); scanf("%d", &page\_count);

printf("Enter the reference string (space separated): "); for (i = 0; i < page\_count; i++)

{

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

}

printf("Enter the number of frames: "); scanf("%d", &frame\_count);

optimalPageReplacement(pages, page\_count, frame\_count); return 0;

}

**Output:**

Enter the number of pages: 3

Enter the reference string (space separated): 7 0 1 2 0 3 0 4 2 3 0 3 2

Enter the number of frames: Reference String: 701

Page Fault: 7 -> 7-1

Page Fault: 0 -> 0-1

Page Fault: 1 -> 1-1

Total Page Faults: 3