**Program 1:**

**Write a C Program to Create 5 Child Process**

**Code:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

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

{

if(fork()==0)

{

printf("[Son] pid %d from[parent] pid %d\n",getpid(),getppid());

exit(0);

}

}

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

wait(NULL);

return 0;

}

**Output:**

[Son] pid 2674 from[parent] pid 2673

[Son] pid 2678 from[parent] pid 2673

[Son] pid 2676 from[parent] pid 2673

[Son] pid 2675 from[parent] pid 2673

[Son] pid 2677 from[parent] pid 2673

**Code:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

printf("Parent Process\n");

if(fork()==0)

{

printf("Child Process\n");

for(int i=1;i<=10;i++)

{

printf("%d\t",i);

}

}

return 0;

}

**Output:**

Parent Process

Child Process

1 2 3 4 5 6 7 8 9 10

**Code:**

#include<stdio.h>

#include<unistd.h>

int main()

{

pid\_t pid=fork();

if(pid<0)

{

fprintf(stderr,"Fork failed\n");

return 1;

}

else if (pid==0)

{

printf("Child Process:PID=%d\n",getpid());

}

else

{

printf("parent process:PID=%d,Child PID=%d\n",getpid(),pid);

}

return 0;

}

**Output:**

parent process:PID=28564,Child PID=28565

Child Process:PID=28565

**First Come First Serve(without Arrival)**

**Prgrm 1:**

**Code:**

#include<stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int turn\_time;

} p[10];

void main()

{

int n;

float total\_wait = 0;

float total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

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

{

printf("Enter the Burst Time for Process %d: ", i + 1);

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

}

p[0].wait\_time = 0;

p[0].turn\_time = p[0].burst\_time;

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

{

p[i].wait\_time = p[i - 1].wait\_time + p[i - 1].burst\_time;

p[i].turn\_time = p[i].wait\_time + p[i].burst\_time;

}

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

{

total\_wait += p[i].wait\_time;

total\_turn += p[i].turn\_time;

printf("Waiting time: %d\t TAT Time: %d\n",p[i].wait\_time,p[i].turn\_time,p[i]);

}

avg\_wait = total\_wait / n;

avg\_turn = total\_turn / n;

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

}

**Output:**

Enter the number of Processes: 4

Enter the Burst Time for Process 1: 21

Enter the Burst Time for Process 2: 3

Enter the Burst Time for Process 3: 6

Enter the Burst Time for Process 4: 2

Average Waiting Time: 18.750000

Average Turnaround Time: 26.750000

**First Come First Serve(without Arrival)**

**(Easy code)**

**Code:**

#include<stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int turn\_time;

} p[10];

void main()

{

int n;

float total\_wait = 0;

float total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

p[0].wait\_time = 0;

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

{

printf("Enter the Burst Time for Process %d: ", i + 1);

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

p[i].wait\_time = p[i - 1].wait\_time + p[i - 1].burst\_time;

p[i].turn\_time = p[i].wait\_time + p[i].burst\_time;

total\_wait += p[i].wait\_time;

total\_turn += p[i].turn\_time;

}

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

{

printf("Waiting time: %d\t TAT Time: %d\n",p[i].wait\_time,p[i].turn\_time,p[i]);

}

avg\_wait = total\_wait / n;

avg\_turn = total\_turn / n;

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

}

**Output:**

Enter the number of Processes: 3

Enter the Burst Time for Process 1: 24

Enter the Burst Time for Process 2: 3

Enter the Burst Time for Process 3: 3

Waiting time: 0 TAT Time: 24

Waiting time: 24 TAT Time: 27

Waiting time: 27 TAT Time: 30

Average Waiting Time: 17.000000

Average Turnaround Time: 27.000000

**First Come First Serve(with Arrival)**

**(Easy code)**

**(Same as Priority Scheduling (Non pre-emptive) Code just replace priority with arrival\_time)**

**Code:**

#include <stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int turn\_time;

int arrival\_time;

int complete\_time;

int status; // to track completion

} p[10];

int main()

{

int n, time = 0, completed = 0, s, i;

int total\_wait = 0;

int total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

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

{

printf("\nEnter the Burst Time for Process %d: ", i + 1);

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

printf("Enter the Arrival Time for Process %d: ", i + 1);

scanf("%d", &p[i].arrival\_time);

p[i].status = 0;

}

while (completed != n)

{

s = -1;

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

{

if (p[i].arrival\_time <= time && p[i].status == 0)

{

s = i;

break;

}

}

if (s != -1)

{

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

{

if (p[i].arrival\_time <= time && p[i].status == 0 && p[i].arrival\_time < p[s].arrival\_time)

{

s = i;

}

}

p[s].complete\_time = time + p[s].burst\_time;

p[s].turn\_time = p[s].complete\_time - p[s].arrival\_time;

p[s].wait\_time = p[s].turn\_time - p[s].burst\_time;

total\_wait += p[s].wait\_time;

total\_turn += p[s].turn\_time;

p[s].status = 1;

completed++;

time = p[s].complete\_time;

}

else

{

time++;

}

}

printf("\nProcess\tWaiting Time\tTurnaround Time\n");

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

{

printf("%d\t\t%d\t\t%d\n",

i + 1, p[i].wait\_time, p[i].turn\_time);

}

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

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

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

return 0;

}

**Output:**

Enter the number of Processes: 5

Enter the Burst Time for Process 1: 6

Enter the Arrival Time for Process 1: 0

Enter the Burst Time for Process 2: 10

Enter the Arrival Time for Process 2: 5

Enter the Burst Time for Process 3: 13

Enter the Arrival Time for Process 3: 7

Enter the Burst Time for Process 4: 2

Enter the Arrival Time for Process 4: 11

Enter the Burst Time for Process 5: 6

Enter the Arrival Time for Process 5: 13

Process Waiting Time Turnaround Time

1 0 6

2 1 11

3 9 22

4 18 20

5 18 24

Average Waiting Time: 9.200000

Average Turnaround Time: 16.600000

**SJF:(without arrival time)**

**Code:**

#include<stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int turn\_time;

} p[10],temp;

void sortProcesses(int n, struct Process p[])

{

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

{

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

{

if (p[j].burst\_time > p[j + 1].burst\_time)

{

temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

}

void main()

{

int n;

float total\_wait = 0;

float total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

p[0].wait\_time = 0;

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

{

printf("Enter the Burst Time for Process %d: ", i + 1);

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

}

sortProcesses(n, p);

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

{

p[i].wait\_time = p[i - 1].wait\_time + p[i - 1].burst\_time;

}

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

{

p[i].turn\_time = p[i].wait\_time + p[i].burst\_time;

total\_wait += p[i].wait\_time;

total\_turn += p[i].turn\_time;

printf("Waiting time: %d\t Turnaround Time: %d\n", p[i].wait\_time, p[i].turn\_time);

}

avg\_wait = total\_wait / n;

avg\_turn = total\_turn / n;

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

}

**Output:**

Enter the number of Processes: 4

Enter the Burst Time for Process 1: 2

Enter the Burst Time for Process 2: 3

Enter the Burst Time for Process 3: 6

Enter the Burst Time for Process 4: 21

Waiting time: 0 TAT Time: 2

Waiting time: 2 TAT Time: 5

Waiting time: 5 TAT Time: 11

Waiting time: 11 TAT Time: 32

Average Waiting Time: 4.500000

Average Turnaround Time: 12.500000

**SJF(with arrival time):**

**(Same as Priority Scheduling (Non pre-emptive) Code just replace priority with burst\_time)**

**Code:**

#include <stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int turn\_time;

int arrival\_time;

int complete\_time;

int status; // to track completion

} p[10];

int main()

{

int n, time = 0, completed = 0, s, i;

int total\_wait = 0;

int total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

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

{

printf("\nEnter the Burst Time for Process %d: ", i + 1);

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

printf("Enter the Arrival Time for Process %d: ", i + 1);

scanf("%d", &p[i].arrival\_time);

p[i].status = 0;

}

while (completed != n)

{

s = -1;

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

{

if (p[i].arrival\_time <= time && p[i].status == 0)

{

s = i;

break;

}

}

if (s != -1)

{

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

{

if (p[i].arrival\_time <= time && p[i].status == 0 && p[i].burst\_time < p[s].burst\_time)

{

s = i;

}

}

p[s].complete\_time = time + p[s].burst\_time;

p[s].turn\_time = p[s].complete\_time - p[s].arrival\_time;

p[s].wait\_time = p[s].turn\_time - p[s].burst\_time;

total\_wait += p[s].wait\_time;

total\_turn += p[s].turn\_time;

p[s].status = 1;

completed++;

time = p[s].complete\_time;

}

else

{

time++;

}

}

printf("\nProcess\tWaiting Time\tTurnaround Time\n");

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

{

printf("%d\t\t%d\t\t%d\n",

i + 1, p[i].wait\_time, p[i].turn\_time);

}

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

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

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

return 0;

}

**Output:**

Enter the number of Processes: 5

Enter the Burst Time for Process 1: 10

Enter the Arrival Time for Process 1: 0

Enter the Burst Time for Process 2: 1

Enter the Arrival Time for Process 2: 0

Enter the Burst Time for Process 3: 2

Enter the Arrival Time for Process 3: 3

Enter the Burst Time for Process 4: 1

Enter the Arrival Time for Process 4: 5

Enter the Burst Time for Process 5: 5

Enter the Arrival Time for Process 5: 10

Process Waiting Time Turnaround Time

1 1 11

2 0 1

3 9 11

4 6 7

5 4 9

Average Waiting Time: 4.000000

Average Turnaround Time: 7.800000

**SRTF:(with Arrival)**

**(Same as Priority Scheduling (Pre-emptive) Code just replace priority with burst\_time)**

**Code:**

#include <stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int temp\_time;

int turn\_time;

int arrival\_time;

int complete\_time;

} p[10];

int main()

{

int n, time, completed = 0, smallest, i, end;

float total\_wait = 0;

float total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

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

{

printf("\nEnter the Burst Time for Process %d: ", i + 1);

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

printf("Enter the Arrival Time for Process %d: ", i + 1);

scanf("%d", &p[i].arrival\_time);

p[i].temp\_time = p[i].burst\_time;

}

for (time = 0; completed != n; time++)

{

smallest = -1;

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

{

if (p[i].arrival\_time <= time && p[i].burst\_time > 0)

{

if (smallest == -1 || p[i].burst\_time < p[smallest].burst\_time)

smallest = i;

}

}

if (smallest != -1)

{

p[smallest].burst\_time--;

if (p[smallest].burst\_time == 0)

{

completed++;

end = time + 1;

p[smallest].complete\_time = end;

p[smallest].wait\_time = end - p[smallest].arrival\_time - p[smallest].temp\_time;

p[smallest].turn\_time = end - p[smallest].arrival\_time;

}

}

}

printf("\nProcess\tWaiting Time\tTurnaround Time\tCompletion Time\n");

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

{

printf("%d\t\t%d\t\t%d\t\t%d\n",

i + 1,p[i].wait\_time,p[i].turn\_time,p[i].complete\_time);

total\_wait += p[i].wait\_time;

total\_turn += p[i].turn\_time;

}

avg\_wait = total\_wait / n;

avg\_turn = total\_turn / n;

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

return 0;

}

**Output:**

Enter the number of Processes: 4

Enter the Burst Time for Process 1: 7

Enter the Arrival Time for Process 1: 0

Enter the Burst Time for Process 2: 4

Enter the Arrival Time for Process 2: 1

Enter the Burst Time for Process 3: 6

Enter the Arrival Time for Process 3: 2

Enter the Burst Time for Process 4: 5

Enter the Arrival Time for Process 4: 3

Process Waiting Time Turnaround Time Completion Time

1 9 16 16

2 0 4 5

3 14 20 22

4 2 7 10

Average Waiting Time: 6.250000

Average Turnaround Time: 11.750000

**Priority (Non pre-emptive):(with Arrival)**

**(Same as SJF with arrival Code just replace burst time with priority)**

#include <stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int turn\_time;

int arrival\_time;

int complete\_time;

int priority;

int status; // to track completion

} p[10];

int main()

{

int n, time = 0, completed = 0, s, i;

int total\_wait = 0;

int total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

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

{

printf("\nEnter the Burst Time for Process %d: ", i + 1);

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

printf("Enter the Arrival Time for Process %d: ", i + 1);

scanf("%d", &p[i].arrival\_time);

printf("Enter the Priority for Process %d: ", i + 1);

scanf("%d", &p[i].priority);

p[i].status = 0;

}

while (completed != n)

{

s = -1;

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

{

if (p[i].arrival\_time <= time && p[i].status == 0)

{

s = i;

break;

}

}

if (s != -1)

{

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

{

if (p[i].arrival\_time <= time && p[i].status == 0 && p[i].priority < p[s].priority)

{

s = i;

}

}

p[s].complete\_time = time + p[s].burst\_time;

p[s].turn\_time = p[s].complete\_time - p[s].arrival\_time;

p[s].wait\_time = p[s].turn\_time - p[s].burst\_time;

total\_wait += p[s].wait\_time;

total\_turn += p[s].turn\_time;

p[s].status = 1;

completed++;

time = p[s].complete\_time;

}

else

{

time++;

}

}

printf("\nProcess\tPriority\tWaiting Time\tTurnaround Time\n");

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

{

printf("%d\t\t%d\t\t%d\t\t%d\n",

i + 1,p[i].priority, p[i].wait\_time,p[i].turn\_time);

}

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

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

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

return 0;

}

**Output:1**

Enter the number of Processes: 5

Enter the Burst Time for Process 1: 10

Enter the Arrival Time for Process 1: 0

Enter the Priority for Process 1: 3

Enter the Burst Time for Process 2: 1

Enter the Arrival Time for Process 2: 0

Enter the Priority for Process 2: 1

Enter the Burst Time for Process 3: 2

Enter the Arrival Time for Process 3: 3

Enter the Priority for Process 3: 3

Enter the Burst Time for Process 4: 1

Enter the Arrival Time for Process 4: 5

Enter the Priority for Process 4: 4

Enter the Burst Time for Process 5: 5

Enter the Arrival Time for Process 5: 10

Enter the Priority for Process 5: 2

Process Priority Waiting Time Turnaround Time

1 3 1 11

2 1 0 1

3 3 13 15

4 4 13 14

5 2 1 6

Average Waiting Time: 5.600000

Average Turnaround Time: 9.400000

**Priority (pre-emptive):(with Arrival)**

**(Same as SRTF Code just replace burst\_time with priority)**

#include <stdio.h>

struct Process

{

int burst\_time;

int wait\_time;

int temp\_time;

int turn\_time;

int arrival\_time;

int complete\_time;

int priority;

} p[10];

int main()

{

int n, time, completed = 0, smallest, i, end;

float total\_wait = 0;

float total\_turn = 0;

float avg\_wait;

float avg\_turn;

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

scanf("%d", &n);

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

{

printf("\nEnter the Burst Time for Process %d: ", i + 1);

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

printf("Enter the Arrival Time for Process %d: ", i + 1);

scanf("%d", &p[i].arrival\_time);

printf("Enter the Priority for Process %d: ", i + 1);

scanf("%d", &p[i].priority);

p[i].temp\_time = p[i].burst\_time;

}

for (time = 0; completed != n; time++)

{

smallest = -1;

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

{

if (p[i].arrival\_time <= time && p[i].burst\_time > 0)

{

if (smallest == -1 || p[i].priority < p[smallest].priority)

smallest = i;

}

}

if (smallest != -1)

{

p[smallest].burst\_time--;

if (p[smallest].burst\_time == 0)

{

completed++;

end = time + 1;

p[smallest].complete\_time = end;

p[smallest].wait\_time = end - p[smallest].arrival\_time - p[smallest].temp\_time;

p[smallest].turn\_time = end - p[smallest].arrival\_time;

}

}

}

printf("\nProcess\tWaiting Time\tTurnaround Time\tCompletion Time\n");

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

{

printf("%d\t\t%d\t\t%d\t\t%d\n",

i + 1,p[i].wait\_time,p[i].turn\_time,p[i].complete\_time);

total\_wait += p[i].wait\_time;

total\_turn += p[i].turn\_time;

}

avg\_wait = total\_wait / n;

avg\_turn = total\_turn / n;

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

return 0;

}

**Output:**

Enter the number of Processes: 5

Enter the Burst Time for Process 1: 10

Enter the Arrival Time for Process 1: 0

Enter the Priority for Process 1: 3

Enter the Burst Time for Process 2: 1

Enter the Arrival Time for Process 2: 0

Enter the Priority for Process 2: 1

Enter the Burst Time for Process 3: 2

Enter the Arrival Time for Process 3: 3

Enter the Priority for Process 3: 3

Enter the Burst Time for Process 4: 1

Enter the Arrival Time for Process 4: 5

Enter the Priority for Process 4: 4

Enter the Burst Time for Process 5: 5

Enter the Arrival Time for Process 5: 10

Enter the Priority for Process 5: 2

Process Waiting Time Turnaround Time Completion Time

1 6 16 16

2 0 1 1

3 13 15 18

4 13 14 19

5 0 5 15

Average Waiting Time: 6.400000

Average Turnaround Time: 10.200000

**Output:2**

Enter the number of Processes: 5

Enter the Burst Time for Process 1: 7

Enter the Arrival Time for Process 1: 0

Enter the Priority for Process 1: 3

Enter the Burst Time for Process 2: 2

Enter the Arrival Time for Process 2: 3

Enter the Priority for Process 2: 2

Enter the Burst Time for Process 3: 3

Enter the Arrival Time for Process 3: 4

Enter the Priority for Process 3: 1

Enter the Burst Time for Process 4: 1

Enter the Arrival Time for Process 4: 4

Enter the Priority for Process 4: 1

Enter the Burst Time for Process 5: 3

Enter the Arrival Time for Process 5: 5

Enter the Priority for Process 5: 3

Process Waiting Time Turnaround Time Completion Time

1 6 13 13

2 4 6 9

3 0 3 7

4 3 4 8

5 8 11 16

Average Waiting Time: 4.200000

Average Turnaround Time: 7.400000

**Round Robin:**

**Code:**

#include<stdio.h>

struct Process {

int burst\_time;

int wait\_time;

int turn\_time;

int remaining\_time;

} p[10];

int main() {

int n;

float total\_wait = 0;

float total\_turn = 0;

float avg\_wait;

float avg\_turn;

int time\_quantum;

int completed = 0;

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

scanf("%d", &n);

printf("Enter the Time Quantum: ");

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

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

printf("Enter the Burst Time for Process %d: ", i + 1);

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

p[i].remaining\_time = p[i].burst\_time;

}

int current\_time = 0;

while (completed < n) {

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

if (p[i].remaining\_time > 0) {

if (p[i].remaining\_time > time\_quantum) {

current\_time += time\_quantum;

p[i].remaining\_time -= time\_quantum;

} else {

current\_time += p[i].remaining\_time;

p[i].wait\_time = current\_time - p[i].burst\_time;

p[i].remaining\_time = 0;

p[i].turn\_time = current\_time;

completed++;

}

}

}

}

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

total\_wait += p[i].wait\_time;

total\_turn += p[i].turn\_time;

}

avg\_wait = total\_wait / n;

avg\_turn = total\_turn / n;

printf("\nProcess\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t%d\t\t%d\n", i + 1, p[i].wait\_time, p[i].turn\_time);

}

printf("\nAverage Waiting Time: %f\n", avg\_wait);

printf("Average Turnaround Time: %f\n", avg\_turn);

return 0;

}

**Output:**

Enter the number of Processes: 4

Enter the Time Quantum: 2

Enter the Burst Time for Process 1: 21

Enter the Burst Time for Process 2: 13

Enter the Burst Time for Process 3: 6

Enter the Burst Time for Process 4: 12

Process Waiting Time Turnaround Time

1 31 52

2 32 45

3 16 22

4 30 42

Average Waiting Time: 27.250000

Average Turnaround Time: 40.250000

**Output:2**

Enter the number of Processes: 5

Enter the Time Quantum: 3

Enter the Burst Time for Process 1: 10

Enter the Burst Time for Process 2: 1

Enter the Burst Time for Process 3: 2

Enter the Burst Time for Process 4: 1

Enter the Burst Time for Process 5: 5

Process Waiting Time Turnaround Time

1 9 19

2 3 4

3 4 6

4 6 7

5 10 15

Average Waiting Time: 6.400000

Average Turnaround Time: 10.200000

1. **Create a C program to implement the Banker’s Algorithm for deadlock Avoidance. Also print the safety sequence. Provide at least two cases, one with deadlock and one without deadlocks.**

**Code:**

#include<stdio.h>

int main() {

int n, m;

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

scanf("%d", &n);

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

scanf("%d", &m);

int allocation[n][m], max[n][m], available[m], need[n][m], finish[n], output[n];

// Input

printf("Enter the values for Allocation:\n");

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

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

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

printf("Enter the values for Max:\n");

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

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

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

printf("Enter the values for Available:\n");

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

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

// Calculate need

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

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

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

// Initialize finish

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

finish[i] = 0;

// Initialize work with available

int work[m];

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

work[j] = available[j];

int index = 0;

while (index < n) {

int found = 0;

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

if (!finish[i]) {

int j;

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

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

break;

if (j == m) {

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

work[k] += allocation[i][k];

output[index++] = i;

finish[i] = 1;

found = 1;

}

}

}

if (!found) {

printf("Unsafe state\n");

return 0;

}

}

printf("Following is the SAFE Sequence:\n");

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

printf(" P%d ", output[i]);

if (i != n - 1)

printf("->");

}

printf("\n");

return 0;

}

**Output:1**

Enter the number of Processes: 5

Enter the number of Resources: 3

Enter the values for Allocation:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter the values for Max:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter the values for Available:

3 3 2

Following is the SAFE Sequence:

P1 -> P3 -> P4 -> P0 -> P2

**Output2:**

Enter the number of Processes: 4

Enter the number of Resources: 3

Enter the values for Allocation:

0 1 0

1 0 0

0 0 1

0 0 1

Enter the values for Max:

1 2 1

1 1 1

1 1 2

1 1 2

Enter the values for Available:

1 0 1

Unsafe state

**First Fit**

**Code:**

#include<stdio.h>

int main() {

int n, m;

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

scanf("%d", &n);

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

scanf("%d", &m);

int block\_size[n], file\_size[m], memory\_allocation[n];

// Input

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

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

scanf("%d", &block\_size[i]);

memory\_allocation[i] = 0;

}

printf("Enter the size of the files:\n");

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

scanf("%d", &file\_size[i]);

int j = 0;

int i = 0;

while (i < n) {

if (block\_size[i] >= file\_size[j] && memory\_allocation[i] == 0) {

memory\_allocation[i] = file\_size[j];

j++;

i = 0;

} else

i++;

}

printf("Block Sizes:\n");

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

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

printf("\nAllocated Memory:\n");

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

if (memory\_allocation[i] == 0)

printf("\t");

else

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

}

return 0;

}

**Output:**

Enter the number of Blocks: 5

Enter the number of files: 4

Enter the size of each block:

100

500

200

300

600

Enter the size of the files:

212

417

112

426

Block Sizes:

100 500 200 300 600

Allocated Memory:

212 112 417

**Best Fit:**

**Code:**

#include<stdio.h>

int main() {

int n, m;

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

scanf("%d", &n);

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

scanf("%d", &m);

int block\_size[n], file\_size[m], memory\_allocation[n];

// Input

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

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

scanf("%d", &block\_size[i]);

memory\_allocation[i] = 0;

}

printf("Enter the size of the files:\n");

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

scanf("%d", &file\_size[i]);

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

int index = -1;

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

if (block\_size[j] >= file\_size[i] && memory\_allocation[j] == 0 && (index == -1 || block\_size[j] < block\_size[index])) {

index = j;

}

}

if (index != -1) {

memory\_allocation[index] = file\_size[i];

}

}

printf("Block Sizes:\n");

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

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

printf("\nAllocated Memory:\n");

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

if (memory\_allocation[i] == 0)

printf("\t");

else

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

}

return 0;

}

**Output:**

Enter the number of Blocks: 5

Enter the number of files: 4

Enter the size of each block:

100

500

200

300

600

Enter the size of the files:

212

417

112

426

Block Sizes:

100 500 200 300 600

Allocated Memory:

417 112 212 426

**Worst Fit:**

**(Same as Best fit just make < to> in** block\_size[j] > block\_size[index]) **)**

**Code:**

#include<stdio.h>

int main() {

int n, m;

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

scanf("%d", &n);

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

scanf("%d", &m);

int block\_size[n], file\_size[m], memory\_allocation[n];

// Input

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

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

scanf("%d", &block\_size[i]);

memory\_allocation[i] = 0;

}

printf("Enter the size of the files:\n");

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

scanf("%d", &file\_size[i]);

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

int index = -1;

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

if (block\_size[j] >= file\_size[i] && memory\_allocation[j] == 0 && (index == -1 || block\_size[j] > block\_size[index])) {

index = j;

}

}

if (index != -1) {

memory\_allocation[index] = file\_size[i];

}

}

printf("Block Sizes:\n");

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

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

printf("\nAllocated Memory:\n");

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

if (memory\_allocation[i] == 0)

printf("\t");

else

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

}

return 0;

}

**Output:**

nter the number of Blocks: 5

Enter the number of files: 4

Enter the size of each block:

100

500

200

300

600

Enter the size of the files:

212

417

112

426

Block Sizes:

100 500 200 300 600

Allocated Memory:

417 112 212

**FIFO Page Replacement:**

**Code:**

#include<stdio.h>

int main() {

int i, j, a[100], frames[100], no\_frames, count = 0, n, page\_faults = 0;

printf("Enter the No of Pages: ");

scanf("%d", &n);

printf("\nEnter the page Numbers: ");

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

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

printf("\nEnter the No of Frames: ");

scanf("%d", &no\_frames);

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

frames[i] = -1;

printf("\nPage Number\tPage Frames\n");

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

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

int present = 0;

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

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

present = 1;

break;

}

}

if(present == 0) {

frames[count] = a[i];

count = (count + 1) % no\_frames;

page\_faults++;

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

{

if(frames[j] == -1)

printf("-\t");

else

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

}

}

printf("\n");

}

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

return 0;

}

**Output:1**

Enter the No of Pages: 12

Enter the page Numbers: 1 2 3 4 1 2 5 1 2 3 4 5

Enter the No of Frames: 3

Page Number Page Frames

1 1 -1 -1

2 1 2 -1

3 1 2 3

4 4 2 3

1 4 1 3

2 4 1 2

5 5 1 2

1

2

3 5 3 2

4 5 3 4

5

Total Page Faults: 9

**Output:2**

Enter the No of Pages: 12

Enter the page Numbers: 1 2 3 4 1 2 5 1 2 3 4 5

Enter the No of Frames: 3

Page Number Page Frames

1 1 - -

2 1 2 -

3 1 2 3

4 4 2 3

1 4 1 3

2 4 1 2

5 5 1 2

1

2

3 5 3 2

4 5 3 4

5

Total Page Faults: 9

**LRU Page Replacement:**

**Code:**

**//almost same as Optimal small changes are there lyk min and max**

#include<stdio.h>

//lru

int main() {

int pages[100], frames[100], temp[100];

int numPages, numFrames, pageFaults = 0;

printf("Enter number of pages: ");

scanf("%d", &numPages);

printf("Enter the page Numbers: ");

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

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

printf("Enter number of frames: ");

scanf("%d", &numFrames);

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

frames[i] = -1;

printf("\nPage Number\tPage Frames\n");

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

int flag = 0;

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

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

flag = 1;

break;

}

}

if(flag == 0) {

int min = 9999, replaceIndex;

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

int found = 0;

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

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

temp[j] = k;

found = 1;

break;

}

}

if(!found) {

temp[j] = -1; // A large negative number to represent least recently used

}

if(temp[j] < min) {

min = temp[j];

replaceIndex = j;

}

}

frames[replaceIndex] = pages[i];

pageFaults++;

}

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

if (flag == 0) {

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

if (frames[j] == -1)

printf("-\t");

else

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

}

}

}

printf("\nTotal page faults: %d", pageFaults);

return 0;

}

**Output:1**

Enter number of pages: 17

Enter the page Numbers: 0 1 2 0 3 0 1 2 3 0 1 2 3 4 5 6 7

Enter number of frames: 4

Page Number Page Frames

0 0 - - -

1 0 1 - -

2 0 1 2 -

0

3 0 1 2 3

0

1

2

3

0

1

2

3

4 4 1 2 3

5 4 5 2 3

6 4 5 6 3

7 4 5 6 7

Total page faults: 8

**Output:2**

Enter number of pages: 20

Enter the page Numbers: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter number of frames: 3

Page Number Page Frames

7 7 - -

0 7 0 -

1 7 0 1

2 2 0 1

0

3 2 0 3

0

4 4 0 3

2 4 0 2

3 4 3 2

0 0 3 2

3

2

1 1 3 2

2

0 1 0 2

1

7 1 0 7

0

1

Total page faults: 12

**Optimal Page Replacemnet:**

**Please Note:**

**If you're replacing a page that hasn't been for a long time, the code works fine. But when it reaches the end, instead of going back to the start it uses FIFO. So, you'll still get the right answer (page faults), but the page numbers might be different. If that's okay with you, you can use this code. Otherwise,you can refer to others code.**

**//almost same as LRU small changes are there lyk min and max**

**Code:**

#include<stdio.h>

int main() {

int pages[100], frames[100], temp[100];

int numPages, numFrames, pageFaults = 0;

printf("Enter number of pages: ");

scanf("%d", &numPages);

printf("Enter the page Numbers: ");

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

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

printf("Enter number of frames: ");

scanf("%d", &numFrames);

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

frames[i] = -1;

printf("\nPage Number\tPage Frames\n");

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

int flag = 0;

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

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

flag = 1;

break;

}

}

if(flag == 0) {

int max = -1, replaceIndex;

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

int found = 0;

for(int k = i + 1; k < numPages; k++) {

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

temp[j] = k;

found = 1;

break;

}

}

if(!found) {

temp[j] = 9999; // A large number to represent infinity

}

if(temp[j] > max) {

max = temp[j];

replaceIndex = j;

}

}

frames[replaceIndex] = pages[i];

pageFaults++;

}

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

if (flag == 0) {

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

if (frames[j] == -1)

printf("-\t");

else

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

}

}

}

printf("\nTotal page faults: %d", pageFaults);

return 0;

}

**Output:**

Enter number of pages: 19

Enter the page Numbers: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0

Enter number of frames: 3

Page Number Page Frames

7 7 - -

0 7 0 -

1 7 0 1

2 2 0 1

0

3 2 0 3

0

4 2 4 3

2

3

0 2 0 3

3

2

1 2 0 1

2

0

1

7 7 0 1

0

Total page faults: 9