1)

SJF:(With preemption) / SRF:

// C++ program to implement Shortest Remaining Time First

// Shortest Remaining Time First (SRTF)

#include <bits/stdc++.h>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

// Function to find the waiting time for all

// processes

void findWaitingTime(Process proc[], int n,int wt[])

{

int rt[n];

// Copy the burst time into rt[]

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

// Process until all processes gets

// completed

while (complete != n) {

// Find process with minimum

// remaining time among the

// processes that arrives till the

// current time`

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

if ((proc[j].art <= t) &&

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

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current

// process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Function to calculate turn around time

void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

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

}

// Function to calculate average time

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0,

total\_tat = 0;

// Function to find waiting time of all

// processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all

// details

cout << " P\t\t"

<< "BT\t\t"

<< "WT\t\t"

<< "TAT\t\t\n";

// Calculate total waiting time and

// total turnaround time

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t\t " << wt[i]

<< "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },

{ 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };

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

findavgTime(proc, n);

return 0;

}

Priority (non-preemptive):

/\*

\* C program to implement priority scheduling

\*/

#include <stdio.h>

//Function to swap two variables

void swap(int \*a,int \*b)

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

int main()

{

int n;

printf("Enter Number of Processes: ");

scanf("%d",&n);

// b is array for burst time, p for priority and index for process id

int b[n],p[n],index[n];

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

{

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

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

index[i]=i+1;

}

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

{

int a=p[i],m=i;

//Finding out highest priority element and placing it at its desired position

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

{

if(p[j] > a)

{

a=p[j];

m=j;

}

}

//Swapping processes

swap(&p[i], &p[m]);

swap(&b[i], &b[m]);

swap(&index[i],&index[m]);

}

// T stores the starting time of process

int t=0;

//Printing scheduled process

printf("Order of process Execution is\n");

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

{

printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);

t+=b[i];

}

printf("\n");

printf("Process Id Burst Time Wait Time TurnAround Time\n");

int wait\_time=0;

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

{

printf("P%d %d %d %d\n",index[i],b[i],wait\_time,wait\_time + b[i]);

wait\_time += b[i];

}

return 0;

}

Priority: (premption):

#include<stdio.h>

// structure representing a structure

struct priority\_scheduling {

// name of the process

char process\_name;

// time required for execution

int burst\_time;

// waiting time of a process

int waiting\_time;

// total time of execution

int turn\_around\_time;

// priority of the process

int priority;

};

int main() {

// total number of processes

int number\_of\_process;

// total waiting and turnaround time

int total = 0;

// temporary structure for swapping

struct priority\_scheduling temp\_process;

// ASCII numbers are used to represent the name of the process

int ASCII\_number = 65;

// swapping position

int position;

// average waiting time of the process

float average\_waiting\_time;

// average turnaround time of the process

float average\_turnaround\_time;

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

// get the total number of the process as input

scanf("%d", & number\_of\_process);

// initializing the structure array

struct priority\_scheduling process[number\_of\_process];

printf("\nPlease Enter the Burst Time and Priority of each process:\n");

// get burst time and priority of all process

for (int i = 0; i < number\_of\_process; i++) {

// assign names consecutively using ASCII number

process[i].process\_name = (char) ASCII\_number;

printf("\nEnter the details of the process %c \n", process[i].process\_name);

printf("Enter the burst time: ");

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

printf("Enter the priority: ");

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

// increment the ASCII number to get the next alphabet

ASCII\_number++;

}

// swap process according to high priority

for (int i = 0; i < number\_of\_process; i++) {

position = i;

for (int j = i + 1; j < number\_of\_process; j++) {

// check if priority is higher for swapping

if (process[j].priority > process[position].priority)

position = j;

}

// swapping of lower priority process with the higher priority process

temp\_process = process[i];

process[i] = process[position];

process[position] = temp\_process;

}

// First process will not have to wait and hence has a waiting time of 0

process[0].waiting\_time = 0;

for (int i = 1; i < number\_of\_process; i++) {

process[i].waiting\_time = 0;

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

// calculate waiting time

process[i].waiting\_time += process[j].burst\_time;

}

// calculate total waiting time

total += process[i].waiting\_time;

}

// calculate average waiting time

average\_waiting\_time = (float) total / (float) number\_of\_process;

// assigning total as 0 for next calculations

total = 0;

printf("\n\nProcess\_name \t Burst Time \t Waiting Time \t Turnaround Time\n");

printf("------------------------------------------------------------\n");

for (int i = 0; i < number\_of\_process; i++) {

// calculating the turnaround time of the processes

process[i].turn\_around\_time = process[i].burst\_time + process[i].waiting\_time;

// calculating the total turnaround time.

total += process[i].turn\_around\_time;

// printing all the values

printf("\t %c \t\t %d \t\t %d \t\t %d", process[i].process\_name, process[i].burst\_time, process[i].waiting\_time, process[i].turn\_around\_time);

printf("\n-----------------------------------------------------------\n");

}

// calculating the average turn\_around time

average\_turnaround\_time = (float) total / (float) number\_of\_process;

// average waiting time

printf("\n\n Average Waiting Time : %f", average\_waiting\_time);

// average turnaround time

printf("\n Average Turnaround Time: %f\n", average\_turnaround\_time);

return 0;

}

SJF (Without preemptive):

/\*

\* C Program to Implement SJF Scheduling

\*/

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

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

{

printf("p%d:",i+1);

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

p[i]=i+1;

}

//sorting of burst times

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

//finding the waiting time of all the processes

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

{

wt[i]=0;

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

//individual WT by adding BT of all previous completed processes

wt[i]+=bt[j];

//total waiting time

total+=wt[i];

}

//average waiting time

avg\_wt=(float)total/n;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

//turnaround time of individual processes

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

//total turnaround time

totalT+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

//average turnaround time

avg\_tat=(float)totalT/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f",avg\_tat);

}

Round Robin:

#include<stdio.h>

#include<conio.h>

void main()

{

// initlialize the variable name

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

printf(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

printf(" Arrival time is: \t"); // Accept arrival time

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

printf(" \nBurst time is: \t"); // Accept the Burst time

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

temp[i] = bt[i]; // store the burst time in temp array

}

// Accept the Time qunat

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting time

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0) // define the conditions

{

sum = sum + temp[i];

temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant;

sum = sum + quant;

}

if(temp[i]==0 && count==1)

{

y--; //decrement the process no.

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);

wt = wt+sum-at[i]-bt[i];

tat = tat+sum-at[i];

count =0;

}

if(i==NOP-1)

{

i=0;

}

else if(at[i+1]<=sum)

{

i++;

}

else

{

i=0;

}

}

// represents the average waiting time and Turn Around time

avg\_wt = wt \* 1.0/NOP;

avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt);

printf("\n Average Waiting Time: \t%f", avg\_tat);

getch();

}

3)

// Banker's Algorithm

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k;

n = 5; // Number of processes

m = 3; // Number of resources

int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } }; // P4

int avail[3] = { 3, 3, 2 }; // Available Resources

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][m];

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

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

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

}

int y = 0;

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

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

if (f[i] == 0) {

int flag = 0;

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

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

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (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 following system is not safe");

break;

}

}

if(flag==1)

{

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

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

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

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

}

return (0);

// This code is contributed by Deep Baldha (CandyZack)

}

5) LRU:

#include<stdio.h>

main()

{

int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];

printf("Enter no of pages:");

scanf("%d",&n);

printf("Enter the reference string:");

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

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

printf("Enter no of frames:");

scanf("%d",&f);

q[k]=p[k];

printf("\n\t%d\n",q[k]);

c++;

k++;

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

{

c1=0;

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

{

if(p[i]!=q[j])

c1++;

}

if(c1==f)

{

c++;

if(k<f)

{

q[k]=p[i];

k++;

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

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

printf("\n");

}

else

{

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

{

c2[r]=0;

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

{

if(q[r]!=p[j])

c2[r]++;

else

break;

}

}

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

b[r]=c2[r];

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

{

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

{

if(b[r]<b[j])

{

t=b[r];

b[r]=b[j];

b[j]=t;

}

}

}

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

{

if(c2[r]==b[0])

q[r]=p[i];

printf("\t%d",q[r]);

}

printf("\n");

}

}

}

printf("\nThe no of page faults is %d",c);

}

**OUTPUT:**

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1

Enter no of frames:3

7

7 5

7 5 9

4 5 9

4 3 9

4 3 7

9 3 7

9 6 7

9 6 2

1 6 2

The no of page faults is 10

Optimal:

#include<stdio.h>

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;

printf("Enter number of frames: ");

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

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter page 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]){

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

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

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

faults++;

frames[j] = pages[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

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

temp[j] = -1;

for(k = i + 1; k < no\_of\_pages; ++k){

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

temp[j] = k;

break;

}

}

}

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

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

pos = j;

flag3 = 1;

break;

}

}

if(flag3 ==0){

max = temp[0];

pos = 0;

for(j = 1; j < no\_of\_frames; ++j){

if(temp[j] > max){

max = temp[j];

pos = j;

}

}

}

frames[pos] = pages[i];

faults++;

}

printf("\n");

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

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

}

}

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

return 0;

}

Output

Enter number of frames: 3

Enter number of pages: 10

Enter page reference string: 2 3 4 2 1 3 7 5 4 3

2 -1 -1

2 3 -1

2 3 4

2 3 4

1 3 4

1 3 4

7 3 4

5 3 4

5 3 4

5 3 4

Second chance: [Second Chance (or Clock) Page Replacement Policy - GeeksforGeeks](https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/)

// CPP program to find largest in an array

// without conditional/bitwise/ternary/ operators

// and without library functions.

#include<iostream>

#include<cstring>

#include<sstream>

using namespace std;

// If page found, updates the second chance bit to true

static bool findAndUpdate(int x,int arr[],

bool second\_chance[],int frames)

{

int i;

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

{

if(arr[i] == x)

{

// Mark that the page deserves a second chance

second\_chance[i] = true;

// Return 'true', that is there was a hit

// and so there's no need to replace any page

return true;

}

}

// Return 'false' so that a page for replacement is selected

// as he reuested page doesn't exist in memory

return false;

}

// Updates the page in memory and returns the pointer

static int replaceAndUpdate(int x,int arr[],

bool second\_chance[],int frames,int pointer)

{

while(true)

{

// We found the page to replace

if(!second\_chance[pointer])

{

// Replace with new page

arr[pointer] = x;

// Return updated pointer

return (pointer + 1) % frames;

}

// Mark it 'false' as it got one chance

// and will be replaced next time unless accessed again

second\_chance[pointer] = false;

//Pointer is updated in round robin manner

pointer = (pointer + 1) % frames;

}

}

static void printHitsAndFaults(string reference\_string,

int frames)

{

int pointer, i, l=0, x, pf;

//initially we consider frame 0 is to be replaced

pointer = 0;

//number of page faults

pf = 0;

// Create a array to hold page numbers

int arr[frames];

// No pages initially in frame,

// which is indicated by -1

memset(arr, -1, sizeof(arr));

// Create second chance array.

// Can also be a byte array for optimizing memory

bool second\_chance[frames];

// Split the string into tokens,

// that is page numbers, based on space

string str[100];

string word = "";

for (auto x : reference\_string)

{

if (x == ' ')

{

str[l]=word;

word = "";

l++;

}

else

{

word = word + x;

}

}

str[l] = word;

l++;

// l=the length of array

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

{

x = stoi(str[i]);

// Finds if there exists a need to replace

// any page at all

if(!findAndUpdate(x,arr,second\_chance,frames))

{

// Selects and updates a victim page

pointer = replaceAndUpdate(x,arr,

second\_chance,frames,pointer);

// Update page faults

pf++;

}

}

cout << "Total page faults were " << pf << "\n";

}

// Driver code

int main()

{

string reference\_string = "";

int frames = 0;

// Test 1:

reference\_string = "0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4";

frames = 3;

// Output is 9

printHitsAndFaults(reference\_string,frames);

// Test 2:

reference\_string = "2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1";

frames = 4;

// Output is 11

printHitsAndFaults(reference\_string,frames);

return 0;

}