**Program 11:**

**Round Robin CPU Scheduling algorithm.**

#include<stdio.h>

void main()

{

int i,n, 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", &n);

y = n;

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

{

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

printf(" Arrival time:");

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

printf(" \nBurst time:");

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

temp[i] = bt[i];

}

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

scanf("%d", &quant);

printf("\n\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)

{

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--;

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==n-1)

{

i=0;

}

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

{

i++;

}

else

{

i=0;

}

}

avg\_wt = wt \* 1.0/n;

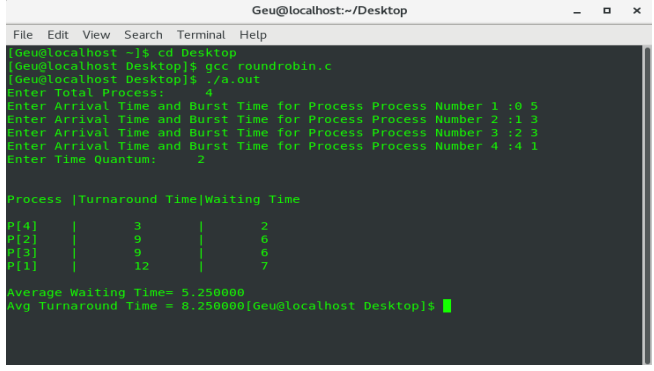
avg\_tat = tat \* 1.0/n;

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

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

}

**OUTPUT:**



**Priority scheduling**

#include<stdio.h>

**int** main()

{

**int** bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

**printf**("Enter Total Number of Process:");

    scanf("%d",&n);

**printf**("\nEnter Burst Time and Priority\n");

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

    {

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

**printf**("Burst Time:");

        scanf("%d",&bt[i]);

**printf**("Priority:");

        scanf("%d",&pr[i]);

        p[i]=i+1;

    }

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

    {

        pos=i;

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

        {

**if**(pr[j]<pr[pos])

                pos=j;

        }

        temp=pr[i];

        pr[i]=pr[pos];

        pr[pos]=temp;

        temp=bt[i];

        bt[i]=bt[pos];

        bt[pos]=temp;

        temp=p[i];

        p[i]=p[pos];

        p[pos]=temp;

    }

    wt[0]=0;

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

    {

        wt[i]=0;

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

            wt[i]+=bt[j];

        total+=wt[i];

    }

    avg\_wt=total/n;

    total=0;

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

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

    {

        tat[i]=bt[i]+wt[i];

        total+=tat[i];

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

    }

    avg\_tat=total/n;     *//average turnaround time*

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

**printf**("\nAverage Turnaround Time=%d\n",avg\_tat);

**return** 0;

}

**Output**

**Text

Description automatically generated**

**Program 12**

#include<stdio.h>

#include<unistd.h>

int main()

{

int pipefds[2];

int returnstatus;

int pid;

char writemessages[2][20]={"Hi", "Hello"};

char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n");

return 1;

}

pid = fork();

if (pid == 0) {

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Child Process - Reading from pipe – Message 1 is %s\n", readmessage);

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Child Process - Reading from pipe – Message 2 is %s\n", readmessage);

}

else

{

printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);

write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);

write(pipefds[1], writemessages[1], sizeof(writemessages[1]));

}

return 0;

}

**OUTPUT:**

Parent Process - Writing to pipe - Message 1 is Hi

Parent Process - Writing to pipe - Message 2 is Hello

Child Process - Reading from pipe – Message 1 is Hi

Child Process - Reading from pipe – Message 2 is Hello

**Program 13**

**FIFO page replacement algorithm**

#include <stdio.h>

int main()

{

int referenceString[10], pageFaults = 0, m, n, s, pages, frames;

printf("\nEnter the number of Pages:\t");

scanf("%d", &pages);

printf("\nEnter reference string values:\n");

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

{

scanf("%d", &referenceString[m]);

}

printf("\n total number of frames:\t");

{

scanf("%d", &frames);

}

int temp[frames];

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

{

temp[m] = -1;

}

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

{

s = 0;

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

{

if(referenceString[m] == temp[n])

{

s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0))

{

temp[m] = referenceString[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = referenceString[m];

}

printf("\n");

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

{

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

}

}

printf("\nTotal Page Faults: %d", pageFaults);

printf("\nTotal Page Hits: %d", pages-pageFaults);

return 0;

}

**OUTPUT:**

Graphical user interface, text

Description automatically generated

**Program 14**

**LRU (Least Recently Used) Page Replacement Algorithm**

#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[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

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

printf("Enter number 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++;

**OUTPUT:**

Text

Description automatically generated

**Program 16**

**optimal page replacement**

#include <stdio.h>

int main()

{

int flag1, flag2, flag3, i, j, k, position, max, faults = 0;

int num\_frames,num\_pages;

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

scanf("%d",&num\_frames);

int frames[num\_frames];

int temp[num\_frames];

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

scanf("%d",&num\_pages);

printf("Enter the reference string: ");

int inputStream[num\_pages];

for(int i=0;i<num\_pages;i++)

{

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

}

for(i = 0; i < num\_frames; i++){

frames[i] = -1;

}

for(i = 0; i < num\_pages; i++){

flag1 = flag2 = 0;

for(j = 0; j < num\_frames; j++){

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

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < num\_frames; j++){

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

faults++;

frames[j] = inputStream[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

for(j = 0; j < num\_frames; j++){

temp[j] = -1;

for(k = i + 1; k < num\_pages; k++){

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

temp[j] = k;

break;

}

}

}

for(j = 0; j < num\_frames; j++){

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

position = j;

flag3 = 1;

break;

}

}

if(flag3 ==0){

max = temp[0];

position = 0;

for(j = 1; j < num\_frames; j++){

if(temp[j] > max){

max = temp[j];

position = j;

}

}

}

frames[position] = inputStream[i];

faults++;

}

printf("\n");

for(j = 0; j < num\_frames; j++){

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

}

}

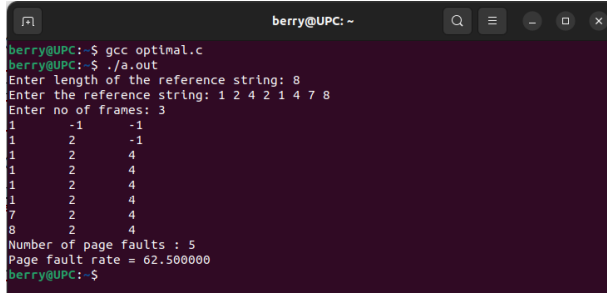
printf("\n\nTotal Page Faults = %d", faults);

printf("\nTotal Hits = %d", num\_pages-faults);

return 0;

}

**OUTPUT:**



**Program 15**

**MRU (Most Recently Used) Page Replacement Algorithm.**

#include <stdio.h>

#include <stdlib.h>

int find(int no\_of\_frames, int \*frames, int tofound)

{

int index = -1;

int i=0;

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

{

if (frames[i] == tofound)

{

index = i;

}

}

return index;

}

int traverse(int no\_of\_pages, int \*pageString, int i, int tofound)

{

int index = -1;

int j=0;

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

{

if (pageString[j] == tofound)

{

return j;

}

}

return index;

}

int find\_MRU(int \*pageString, int i, int no\_of\_pages, int

no\_of\_frames, int \*frames)

{

int \*flag;

flag = (int \*)calloc(no\_of\_frames, sizeof(int));

int j=0;

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

{

flag[j] = 0;

}

int index = -1;

int idx = -1;

int max = -1; //just some high value later to be replaced

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

{

idx = traverse(no\_of\_pages, pageString, i, frames[j]);

if (idx != -1)

{

if (idx > max)

{

max = idx;

index = j;

}

flag[j] = 1;

}

}

free(flag);

return index;

}

int main()

{

int no\_of\_frames, no\_of\_pages;

printf("Enter the no of frames:\n");

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

printf("Enter the no of pages:\n");

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

printf("Enter the pageString\n");

int \*pageString;

pageString = (int \*)calloc(no\_of\_pages, sizeof(int));

int i;

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

{

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

}

int \*frames;

frames = (int \*)calloc(no\_of\_frames, sizeof(int));

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

{

frames[i] = -1;

}

int index = 0;

int no\_of\_page\_faults = 0;

int no\_of\_page\_hits = 0;

int idx;

int count = 0;

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

{

if (count < no\_of\_frames)

{

idx = find(no\_of\_frames, frames, pageString[i]);

if (idx != -1)

{

no\_of\_page\_hits++;

printf("Page Hit : Succesfully found Page %d at %d

Frame\n", pageString[i], idx + 1);

}

else

{

frames[count] = pageString[i];

printf("Page Miss : Storing %d Page no in %d

Frame:\n", pageString[i], count + 1);

count++;

no\_of\_page\_faults++;

}

}

else

{

idx = find(no\_of\_frames, frames, pageString[i]);

if (idx != -1)

{

no\_of\_page\_hits++;

printf("Page Hit : Succesfully found Page %d at %d

Frame\n", pageString[i], idx + 1);

}

else

{

index = find\_MRU(pageString, i, no\_of\_pages,

no\_of\_frames, frames);

printf("Page Miss : Replacing %d Frame Page with %d

Page no:\n", index + 1, pageString[i]);

no\_of\_page\_faults++;

frames[index] = pageString[i];

}

}

}

printf("The total number of page faults are : %d\n",

no\_of\_page\_faults);

printf("The total number of page hits are : %d\n",

no\_of\_page\_hits);

return 0;

}

**OUTPUT**

**Text

Description automatically generated**