**FCFS-CPU SCHEDULING**

//FCFS #include<stdio.h>

int main(void)

{

      int n,at[10],bt[10],ct[10],tat[10],wt[10],sum,i,j,k;

      float totaltat=0,totalwt=0;

      printf("Enter No of processors:");

      scanf(" %d",&n);

      for(i=0;i<n;i++)

      {

            printf("Enter the arrival time of processor %d:",i+1);

            scanf(" %d",&at[i]);

      }

      for(i=0;i<n;i++)

      {

            printf("Enter the burst time of processor %d:",i+1);

            scanf(" %d",&bt[i]);

      }

      //Calculation of completion times for each processor

      sum=at[0];

      for(j=0;j<n;j++)

      {

            sum=sum+bt[j];

            ct[j]=sum;

      }

      //Calculation of turn around time

      for(k=0;k<n;k++)

      {

            tat[k]=ct[k]-at[k];

            totaltat=totaltat+tat[k];

      }

      //Calculation of waiting time

      for(i=0;i<n;i++)

      {

            wt[i]=tat[i]-bt[i];

            totalwt=totalwt+wt[i];

      }

      printf("Process\tAT\tBT\tCT\tTAT\tWt\n");

      for(i=0;i<n;i++)

      {

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

      }

      printf("average of turn around time:%0.1f\n",totaltat/n);

      printf("average of waiting time:%0.1f\n",totalwt/n);

      return 0;

}

**SJF-NON PREEMPTIVE**

#include <stdio.h>

int main() {

int n, i, j, temp, time = 0, count, completed = 0, sum\_wait = 0, sum\_turnaround = 0, start;

float avg\_wait, avg\_turnaround;

int at[10], bt[10], p[10]; // Arrays for arrival times, burst times, and process numbers

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

scanf("%d", &n);

// Input arrival times and burst times for each process

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

printf("Enter the arrival time and burst time for process %d: ", i + 1);

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

p[i] = i + 1;

}

// Sort processes by arrival times, and then by burst times

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

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

if (at[i] > at[j] || (at[i] == at[j] && bt[i] > bt[j])) {

temp = at[i];

at[i] = at[j];

at[j] = temp;

temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

printf("\nProcess\tAT\tBT\tWT\tTAT\n");

// Main loop to calculate waiting times and turnaround times

while (completed < n) {

count = 0;

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

if (at[i] <= time) {

count++;

} else {

break;

}

}

// Sort the ready processes by burst time

if (count > 1) {

for (i = completed; i < completed + count - 1; i++) {

for (j = i + 1; j < completed + count; j++) {

if (bt[i] > bt[j]) {

temp = at[i];

at[i] = at[j];

at[j] = temp;

temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

}

start = time;

time += bt[completed];

printf("P[%d]\t%d\t%d\t%d\t%d\n", p[completed], at[completed], bt[completed], time - at[completed] - bt[completed], time - at[completed]);

sum\_wait += time - at[completed] - bt[completed];

sum\_turnaround += time - at[completed];

completed++;

}

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

avg\_turnaround = (float)sum\_turnaround / (float)n;

printf("Average waiting time is %.2f\n", avg\_wait);

printf("Average turnaround time is %.2f\n", avg\_turnaround);

return 0;

}

**SRTF(SJF-PREEMPTIVE)**

#include <stdio.h>

int main() {

int n, i, j, min\_bt, time = 0, completed = 0, sum\_wait = 0, sum\_turnaround = 0;

int at[10], bt[10], rt[10], p[10]; // Arrays for arrival times, burst times, remaining times, and process numbers

int wait[10], turnaround[10]; // Arrays for waiting times and turnaround times

float avg\_wait, avg\_turnaround;

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

scanf("%d", &n);

// Input arrival times and burst times for each process

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

printf("Enter the arrival time and burst time for process %d: ", i + 1);

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

rt[i] = bt[i]; // Initialize remaining time as burst time

p[i] = i + 1; // Process numbers

}

// Initialize arrays for waiting and turnaround times

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

wait[i] = 0;

turnaround[i] = 0;

}

printf("\nProcess\tAT\tBT\tWT\tTAT\n");

while (completed != n) {

// Find process with minimum remaining time at each time unit

min\_bt = 9999; // A large number to find the minimum burst time

int shortest = -1; // Index of the process with the shortest remaining time

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

if (at[i] <= time && rt[i] > 0 && rt[i] < min\_bt) {

min\_bt = rt[i];

shortest = i;

}

}

if (shortest == -1) {

// No process is currently available to execute

time++;

continue;

}

rt[shortest]--; // Decrement the remaining time of the shortest process

if (rt[shortest] == 0) {

// Process is completed

completed++;

int end\_time = time + 1; // Time at which the process completes

turnaround[shortest] = end\_time - at[shortest]; // Turnaround time

wait[shortest] = turnaround[shortest] - bt[shortest]; // Waiting time

sum\_wait += wait[shortest];

sum\_turnaround += turnaround[shortest];

printf("P[%d]\t%d\t%d\t%d\t%d\n", p[shortest], at[shortest], bt[shortest], wait[shortest], turnaround[shortest]);

}

time++; // Increment time

}

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

avg\_turnaround = (float)sum\_turnaround / (float)n;

printf("Average waiting time is %.2f\n", avg\_wait);

printf("Average turnaround time is %.2f\n", avg\_turnaround);

return 0;

}

**ROUND ROBIN PROGRAM**

#include <stdio.h>

int main() {

int n, tq, i, total\_time = 0, time = 0, flag = 0;

int bt[10], rem\_bt[10], wt[10], tat[10]; // Burst times, remaining burst times, waiting times, turnaround times

float avg\_wt = 0, avg\_tat = 0;

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

scanf("%d", &n);

printf("Enter the burst time for each process:\n");

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

printf("P[%d]: ", i + 1);

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

rem\_bt[i] = bt[i]; // Initialize remaining burst time as burst time

}

printf("Enter the time quantum: ");

scanf("%d", &tq);

while (1) {

flag = 0;

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

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

flag = 1; // There is a pending process

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

time += tq;

rem\_bt[i] -= tq;

} else {

time += rem\_bt[i];

wt[i] = time - bt[i]; // Waiting time is current time minus burst time

rem\_bt[i] = 0;

}

}

}

if (flag == 0) // All processes are done

break;

}

printf("\nProcess\tBT\tWT\tTAT\n");

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

tat[i] = bt[i] + wt[i]; // Turnaround time is burst time plus waiting time

avg\_wt += wt[i];

avg\_tat += tat[i];

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

}

avg\_wt /= n;

avg\_tat /= n;

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

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

return 0;

}

**PRIORITY PROGRAM**

#include <stdio.h>

int main() {

int n, i, j, temp, sum\_wait = 0, sum\_turnaround = 0;

float avg\_wait, avg\_turnaround;

int priority[20], bt[20], wt[20], tat[20];

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

scanf("%d", &n);

// Input burst times and priorities for each process

printf("Enter burst times and priorities for each process:\n");

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

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

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

}

// Sort processes based on priority (ascending order)

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

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

if (priority[i] > priority[j]) {

temp = priority[i];

priority[i] = priority[j];

priority[j] = temp;

temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

}

}

}

// Calculate waiting time for each process

wt[0] = 0; // Waiting time for first process is zero

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

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

sum\_wait += wt[i];

}

// Calculate turnaround time for each process

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

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

sum\_turnaround += tat[i];

}

// Calculate average waiting time and average turnaround time

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

avg\_turnaround = (float)sum\_turnaround / n;

// Print process details

printf("\nProcess\tBT\tPriority\tWT\tTAT\n");

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

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

}

// Print average waiting time and average turnaround time

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

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

return 0;

}

**BANKERS ALGORITHM**

#include <stdio.h>

int main() {

int n, m, i, j, k;

// Get the number of processes and resources from the user

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

scanf("%d", &n);

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

scanf("%d", &m);

int alloc[n][m], max[n][m], avail[m];

// Get the Allocation Matrix from the user

printf("Enter the Allocation Matrix:\n");

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

printf("Process %d:\n", i);

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

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

}

}

// Get the Maximum Matrix from the user

printf("Enter the Maximum Matrix:\n");

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

printf("Process %d:\n", i);

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

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

}

}

// Get the Available Resources from the user

printf("Enter the Available Resources:\n");

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

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

}

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 < n; 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 (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.\nSafe sequence is: ");

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

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

}

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

}

return 0;

}

**PRODUCER-CONSUMER PROBLEM**

#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n 1.Producer \n 2.Consumer \n 3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

case 1:

if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full");

break;

case 2:

if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty");

break;

case 3:

exit(0);

break;

}

}

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("Producer produces the item %d\n",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("Consumer consumes item %d\n",x);

x--;

mutex=signal(mutex);

}

**FCFS-PAGE REPLACEMENT ALGORITHM**

//FIFO PAGE REPLACEMENT ALGORITHM

#include<stdio.h>

#include<conio.h>

int main()

{

int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

printf("\n Enter the length of reference string -- ");

scanf("%d",&n);

printf("\n Enter the reference string -- ");

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

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

printf("\n Enter no. of frames -- ");

scanf("%d",&f);

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

m[i]=-1;

printf("\n The Page Replacement Process is -- \n");

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

{

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

{

if(m[k]==rs[i])

break;

}

if(k==f)

{

m[count++]=rs[i];

pf++;

}

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

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

if(k==f)

printf("\tPF No. %d",pf);

printf("\n");

if(count==f)

count=0;

}

printf("\n The number of Page Faults using FIFO are %d",pf);

getch();

}

**2.LRU PAGE REPLACEMENT ALGORITHM**

//LRU

#include<stdio.h>

#include<conio.h>

void main()

{

int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;

//clrscr();

printf("Enter the length of reference string -- ");

scanf("%d",&n);

printf("Enter the reference string -- ");

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

{

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

flag[i]=0;

}

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

scanf("%d",&f);

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

{

count[i]=0;

m[i]=-1;

}

printf("\nThe Page Replacement process is -- \n");

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

{

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

{

if(m[j]==rs[i])

{

flag[i]=1;

count[j]=next;

next++;

}

}

if(flag[i]==0)

{

if(i<f)

{ m[i]=rs[i];

count[i]=next;

next++;

}

else

{ min=0;

for(j=1;j<f;j++)

if(count[min] > count[j])

min=j;

m[min]=rs[i];

count[min]=next;

next++;

}

pf++;

}

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

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

if(flag[i]==0)

printf("PF No. -- %d" , pf);

printf("\n");

}

printf("\nThe number of page faults using LRU are %d",pf);

getch();

}

**3.OPTIMAL PAGE REPLACEMENT ALGORITHM**

//Optimal page replacement

#include<stdio.h>

int n;

int main()

{

int seq[30],fr[5],pos[5],find,flag,max,i,j,m,k,t,s;

int count=1,pf=0,p=0;

float pfr;

printf("Enter maximum limit of the sequence: ");

scanf("%d",&max);

printf("\nEnter the sequence: ");

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

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

printf("\nEnter no. of frames: ");

scanf("%d",&n);

fr[0]=seq[0];

pf++;

printf("%d\t",fr[0]);

i=1;

while(count<n)

{

flag=1; p++;

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

{

if(seq[i]==seq[j]) flag=0;

}

if(flag!=0)

{

fr[count]=seq[i];

printf("%d\t",fr[count]);

count++;

pf++;

}

i++;

}

printf("\n");

for(i=p;i<max;i++)

{

flag=1;

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

{

if(seq[i]==fr[j])

flag=0;

}

if(flag!=0)

{

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

{

m=fr[j];

for(k=i;k<max;k++)

{

if(seq[k]==m)

{

pos[j]=k;

break;

}

else

pos[j]=1;

}

}

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

{

if(pos[k]==1)

flag=0;

}

if(flag!=0)

s=findmax(pos);

if(flag==0)

{

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

{

if(pos[k]==1)

{

s=k;

break;

}

}

}

fr[s]=seq[i];

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

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

pf++;

printf("\n");

}

}

pfr=(float)pf/(float)max;

printf("\nThe no. of page faults are %d",pf);

printf("\nPage fault rate %f",pfr);

getch();

}

int findmax(int a[])

{

int max,i,k=0;

max=a[0];

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

{

if(max<a[i])

{

max=a[i];

k=i;

}

}

return k;

}