***Name: Aakar Shah***

***Roll No:16BIT046***

***3)FCFS***

#include<iostream>

#include<stdio.h>

#include<vector>

#include<algorithm>

using namespace std;

bool sortcol( const vector<int>& v1,

const vector<int>& v2 ) {

return v1[1] < v2[1];

}

int main()

{

vector< vector <int> > Process;

int n,bt,at,i,j,sum\_TAT=0,TAT,sum\_WT=0,WT,ct=0;

cout<<"Enter number of process: ";

cin>>n;

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

{

vector<int> row;

cout<<"Enter burst time for process "<<i<<" : ";

cin>>bt;

row.push\_back(bt);

cout<<"Enter arrival time for process "<<i<<" : ";

cin>>at;

row.push\_back(at);

row.push\_back(i);

Process.push\_back(row);

}

cout<<endl;

sort(Process.begin(), Process.end(),sortcol);

cout<<"Process id\tArrival time\tBurst time\tCompletion Time\tTAT\tWT\n";

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

{

for (j=2;j>=0;j--)

{

cout << Process[i][j] << "\t\t";

}

ct = Process[i][0] + ct;

TAT = ct - Process[i][1];

WT = TAT - Process[i][0];

sum\_TAT = sum\_TAT + TAT;

sum\_WT = sum\_WT + WT;

cout<<ct<<"\t\t"<<TAT<<"\t\t"<<WT;

//temp\_ct = ct;

cout << endl;

}

cout<<endl<<"Average TAT: "<<sum\_TAT/n;

cout<<endl<<"Average WT: "<<sum\_WT/n;

return 0;

}

***3)PRIORITY***

#include<iostream>

#define n 4

using namespace std;

int main()

{

int AT[n],BT[n],P[n],ID[n],CT[n]={0},WT[n];

int i,j,temp,count=0;

int pro[4],f=0;

int id,a,b,p;

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

{

cout<<endl<<"Enter at, bt, priority";

cin>>AT[i]>>BT[i]>>P[i];

ID[i]=i+1;

}

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

{

for(j = i+1 ; j < 4 ; j++ )

{

if(AT[i] > AT[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;

temp = ID[i];

ID[i] = ID[j];

ID[j] = temp;

}

}

}

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

{

p=0;

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

{

if(AT[j]<=count && CT[j]==0 )

{

if(P[i]>p)

{

id=i;

}

}

}

count = count + BT[id];

CT[id] = count;

}

cout<<endl<<"Final table\n";

cout<<"ID\tAT\tBT\tP\tCT";

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

{

cout<<endl<<ID[i]<<"\t"<<AT[i]<<"\t"<<BT[i]<<"\t"<<P[i]<<"\t"<<CT[i];

}

return 0;

}

***3)ROUNDROBIN***

#include<stdio.h>

#include<stdlib.h>

#include<iostream>

using namespace std;

int main()

{

int at[10],bt[10],ct[10],rt[10],wt[10],id[10],tat[10];

int count,i,j,time\_quant,time=0,n,remains,temp;

cout<<endl<<"Enter number of processes: ";

cin>>n;

// taking input

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

{

cout<<endl<<"Enter process id, arrival time and burst time: ";

cin>>id[i]>>at[i]>>bt[i];

rt[i]=bt[i];

}

// sorting acc to arrival time

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

{

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

{

if(at[i]>at[j])

{

temp = at[i];

at[i] = at[j];

at[j] = temp;

temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

temp = rt[i];

rt[i] = rt[j];

rt[j] = temp;

temp = id[i];

id[i] = id[j];

id[j] = temp;

}

}

}

// logic for round robin

cout<<endl<<"Enter time quantum value: ";

cin>>time\_quant;

remains=n;

count=0;

time=0;

while(remains!=0)

{

if(rt[count] <= time\_quant && rt[count] > 0)

{

time = time + rt[count];

ct[count] = time ;

rt[count] = 0;

remains--;

}

else if( rt[count] > time\_quant )

{

time = time + time\_quant;

rt[count] = rt[count] - time\_quant;

}

if(count == n-1)

{

count = 0;

}

else if(at[count+1]<time)

{

count++;

}

else

{

count = 0;

}

}

// display table

cout<<"\nID\TAT\tBT\tCT\tTAT\tWT\n";

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

{

tat[i] = ct[i] - bt[i];

wt[i] = tat[i] - at[i];

cout<<id[i]<<"\t"<<at[i]<<"\t"<<bt[i]<<"\t"<<ct[i]<<"\t"<<tat[i]<<"\t"<<wt[i]<<endl;

}

return 0;

}

***3)SJF***

#include<stdio.h>

int main()

{

int i,n,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,btime=0;

int bt[10],temp,j,at[10],wt[10],tt[10],ta=0,sum=0;

float wavg=0,tavg=0,tsum=0,wsum=0;

printf(" -------Shortest Job First Scheduling ( NP )-------\n");

printf("\nEnter the No. of processes :");

scanf("%d",&n);

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

{

printf("\tEnter the burst time of %d process :",i+1);

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

printf("\tEnter the arrival time of %d process :",i+1);

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

}

/\*Sorting According to Arrival Time\*/

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

{

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

{

if(at[i]<at[j])

{

temp=p[j];

p[j]=p[i];

p[i]=temp;

temp=at[j];

at[j]=at[i];

at[i]=temp;

temp=bt[j];

bt[j]=bt[i];

bt[i]=temp;

}

}

}

/\*Arranging the table according to Burst time,

Execution time and Arrival Time

Arrival time <= Execution time

\*/

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

{

btime=btime+bt[j];

min=bt[k];

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

{

if (btime>=at[i] && bt[i]<min)

{

temp=p[k];

p[k]=p[i];

p[i]=temp;

temp=at[k];

at[k]=at[i];

at[i]=temp;

temp=bt[k];

bt[k]=bt[i];

bt[i]=temp;

}

}

k++;

}

wt[0]=0;

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

{

sum=sum+bt[i-1];

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

wsum=wsum+wt[i];

}

wavg=(wsum/n);

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

{

ta=ta+bt[i];

tt[i]=ta-at[i];

tsum=tsum+tt[i];

}

tavg=(tsum/n);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n RESULT:-");

printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around" );

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

{

printf("\n p%d\t %d\t %d\t\t %d\t\t\t%d",p[i],bt[i],at[i],wt[i],tt[i]);

}

printf("\n\nAVERAGE WAITING TIME : %f",wavg);

printf("\nAVERAGE TURN AROUND TIME : %f",tavg);

return 0;

}

***3)SRTF***

#include <iostream>

#include <algorithm>

#include <string.h>

using namespace std;

typedef struct proccess

{

int at,bt,ct,ta,wt,btt,pr;

string pro\_id;

/\*

artime = Arrival time,

bt = Burst time,

ct = Completion time,

ta = Turn around time,

wt = Waiting time

\*/

}schedule;

bool compare(schedule a,schedule b)

{

return a.at<b.at;

/\* This schedule will always return TRUE

if above condition comes\*/

}

bool compare2(schedule a,schedule b)

{

return a.bt<b.bt;

/\* This schedule will always return TRUE

if above condition comes\*/

}

int main()

{

schedule pro[10];

int n,i,j,pcom;

cout<<"Enter the number of process::";

cin>>n;

cout<<"Enter the Process id arrival time AND burst time :::";

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

{

cin>>pro[i].pro\_id;

cin>>pro[i].at;

cin>>pro[i].bt;

pro[i].btt=pro[i].bt;

}

sort(pro,pro+n,compare);

/\*sort is a predefined funcion defined in

algorithm.h header file, it will sort the

schedulees according to their arrival time\*/

i=0;

pcom=0;

while(pcom<n)

{

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

{

if(pro[j].at>i)

break;

}

sort(pro,pro+j,compare2);

if(j>0)

{

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

{

if(pro[j].bt!=0)

break;

}

/\* if(pro[j].at>i)

i+=pro[j].at-i;\*/

pro[j].ct=i+1;

pro[j].bt--;

}

i++;

pcom=0;

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

{

if(pro[j].bt==0)

pcom++;

}

}

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

{

pro[i].ta=pro[i].ct-pro[i].btt;

pro[i].wt=pro[i].ta-pro[i].at;

//before executing make it in one statement

cout<<pro[i].pro\_id<<"\t"<<pro[i].at<<"\t"<<pro[i].btt<<"\t"<<pro[i].ct<<"\t"<<pro[i].ta<<"\t"<<pro[i].wt<<"\t"<<pro[i].pr;

cout<<endl;

}

return 0;

}

***4)FORK***

int main( void )

{

printf( "[dad] pid %d\n", getpid() );

for ( int i = 0; i < 3; i++ )

if ( fork() == 0 )

{

// if else with constreaints on i

}

for ( int i = 0; i < 3; i++ )

wait( NULL );

}

// # gcc FCFS.C -o fcfs.o

***5)LS-SORT***

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include<errno.h>

#include<sys/wait.h>

#include <unistd.h>

int main(){

int a[2];

pipe(a);

if(!fork())

{

close(1);

dup(a[1]);

close(a[0]);

execlp("ls","ls",NULL);

}

else

{

close(0);

dup(a[0]);

close(a[1]);

execlp("sort","sort",NULL);

}

}

***6) DEADLOCK***

#include <iostream>

using namespace std;

#define res 1

int Allocation[10][2] , Max[10][2] , Available[2], Need[10][2] , c\_Available[2]; // 2 resources

int Request[10][2], n , i , j , flag=0 , counter=0, pcom=0 , rp;

int safety\_check();

void resource\_request();

int req\_allo();

int req\_need();

int main()

{

cout<<endl<<"Enter number of process: ";

cin>>n;

cout<<endl;

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

{

cout<<"Enter Allocation for Process "<<i+1<<endl;

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

{

cin>>Allocation[i][j];

}

cout<<"Enter Max for Process "<<i+1<<endl;

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

{

cin>>Max[i][j];

}

}

cout<<"Enter total Available resources \n";

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

{

cin>>Available[j];

}

// request and need matrix

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

{

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

{

Need[i][j] = Max[i][j] - Allocation[i][j];

// cout<<endl<<Request[i][j];

}

}

//Currently available resource matrix

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

{

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

{

Available[j] = Available[j] - Allocation[i][j];

}

c\_Available[j] = Available[j];

// cout<<"Available "<<j+1<<" = "<<c\_Available[j];

}

//Deadlock detection

int result = safety\_check();

int select=0;

// Start allocating resources

if(result==1)

{

while(1)

{

cout<<endl<<"Press 1 for resource allocation: ";

cout<<endl<<"Press 2 to exit"<<endl;

cin>>select;

if(select == 2)

{

break;

}

else

{

cout<<endl<<"Enter Requesting Process : ";

cin>>rp;

rp=rp-1;

cout<<endl<<rp;

cout<<"Enter number of instances of resouces requested\n";

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

{

cin>>Request[rp][j];

cout<<Request[rp][j];

}

cout<<endl<<"Reached";

cout<<endl<<Need[rp];

if(req\_need())

{

if(req\_allo())

{

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

{

c\_Available[j] = c\_Available[j] - Request[rp][j];

Allocation[rp][j] = Allocation[rp][j] + Request[rp][j];

Need[rp][j] = Need[rp][j] - Request[rp][j];

}

resource\_request();

}

else

{

cout<<endl<<"Request greater than allocation";

}

}

else

{

cout<<endl<<"Request is greater than need";

}

}

}

}

return 0;

}

int safety\_check()

{

pcom=0;

while(pcom<n)

{

flag=0;

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

{

counter=0;

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

{

if(Need[i][j] > c\_Available[j])

{

flag++;

break;

}

else

{

counter++;

}

}

if(counter == res)

{

c\_Available[j] = c\_Available[j] + Allocation[i][j];

pcom++;

break;

}

}

if(flag == n )

{

cout<<endl<<"Deadlock found!!! It's Unsafe here";

return 0;

}

else

{

// cout<<"Entered for loop";

}

}

cout<<endl<<pcom;

if(pcom == n)

{

cout<<endl<<"It's absolutely safe here : )";

return 1;

}

else

{

//cout<<"ENtered while loop";

}

}

void resource\_request()

{

pcom=0;

while(pcom<n)

{

flag=0;

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

{

counter=0;

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

{

if(Need[i][j] > c\_Available[j])

{

flag++;

break;

}

else

{

counter++;

}

}

if(counter == res)

{

c\_Available[j] = c\_Available[j] + Allocation[i][j];

pcom++;

break;

}

}

if(flag == n )

{

cout<<endl<<"Cannot allocate requested resource!!!";

for(j=0;j<2;j++)

{

c\_Available[j] = c\_Available[j] + Request[rp][j];

Allocation[rp][j] = Allocation[rp][j] - Request[rp][j];

Need[rp][j] = Need[rp][j] + Request[rp][j];

}

break;

}

else

{

// cout<<"Entered for loop";

}

}

cout<<endl<<pcom;

if(pcom == n)

{

cout<<endl<<"Resource successfully allocated : )";

}

else

{

//cout<<"ENtered while loop";

}

}

int req\_need()

{

int success=0;

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

{

if(Request[rp][j]<Need[rp][j])

{

success++;

}

}

if(success==res)

return 1;

else

return 0;

}

int req\_allo()

{

int success=0;

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

{

if(Request[rp][j]<Allocation[rp][j])

{

success++;

}

}

if(success==res)

return 1;

else

return 0;

}

***7) DINING***

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect

// during takefork

// used to wake up hungry philosophers

// during putfork

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum)

{

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

// put down chopsticks

void put\_fork(int phnum)

{

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philospher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

sem\_init(&mutex, 0, 1);

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

sem\_init(&S[i], 0, 0);

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

// create philosopher processes

pthread\_create(&thread\_id[i], NULL,

philospher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

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

pthread\_join(thread\_id[i], NULL);

}

***8)FIFO***

#include<bits/stdc++.h>

using namespace std;

int pageFaults(int pages[], int n, int capacity)

{

set<int> s;

queue<int> indexes;

int page\_faults = 0;

int i;

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

{

if (s.size() < capacity)

{if (s.find(pages[i])==s.end())

{

s.insert(pages[i]);

page\_faults++;

indexes.push(pages[i]);

}

}

else

{

if (s.find(pages[i]) == s.end())

{

int val = indexes.front();

indexes.pop();

s.erase(val);

s.insert(pages[i]);

indexes.push(pages[i]);

page\_faults++;

}

}

}

return page\_faults;

}

int main()

{int m;int capacity;

cout<<"Enter the no. of pages"<<endl;

cin>>m;

int pages[m];

cout<<"Enter the pages"<<endl;

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

{

cin>>pages[i];

}

cout<<"Enter capacity"<<endl;

cin>>capacity;

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

cout << pageFaults(pages, n, capacity);

return 0;

}

***8)LRU***

#include<bits/stdc++.h>

using namespace std;

int pageFaults(int pages[], int n, int capacity)

{

set<int> s;

map<int, int> indexes;

int page\_faults = 0;

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

{

if (s.size() < capacity)

{

if (s.find(pages[i])==s.end())

{

s.insert(pages[i]);

page\_faults++;

}

indexes[pages[i]] = i;

}

else

{

if (s.find(pages[i]) == s.end())

{

int lru = INT\_MAX ,val;

int it;

for ( it=0; it<s.size(); it++)

{

if (indexes[it] < lru)

{

lru = indexes[it];

val = it;

}

}

s.erase(val);

s.insert(pages[i]);

page\_faults++;

}

indexes[pages[i]] = i;

}

}

return page\_faults;

}

int main()

{int m;int capacity;

cout<<"Enter the no. of pages"<<endl;

cin>>m;

int pages[m];

cout<<"Enter the pages"<<endl;

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

{

cin>>pages[i];

}

cout<<"Enter capacity"<<endl;

cin>>capacity;

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

cout << pageFaults(pages, n, capacity);

return 0;

}

***9)CP***

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <errno.h>

#define BUFF\_SIZE 1024

int main(int argc, char\* argv[])

{

int srcFD,destFD,nbread,nbwrite;

char \*buff[BUFF\_SIZE];

/\*Check if both src & dest files are received or --help is received to get usage\*/

if(argc != 3 || argv[1] == "--help")

{

printf("\nUsage: cpcmd source\_file destination\_file\n");

exit(EXIT\_FAILURE);

}

/\*Open source file\*/

srcFD = open(argv[1],O\_RDONLY);

if(srcFD == -1)

{

printf("\nError opening file %s errno = %d\n",argv[1],errno);

exit(EXIT\_FAILURE);

}

/\*Open destination file with respective flags & modes

O\_CREAT & O\_TRUNC is to truncate existing file or create a new file

S\_IXXXX are file permissions for the user,groups & others\*/

destFD = open(argv[2],O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH);

if(destFD == -1)

{

printf("\nError opening file %s errno = %d\n",argv[2],errno);

exit(EXIT\_FAILURE);

}

/\*Start data transfer from src file to dest file till it reaches EOF\*/

while((nbread = read(srcFD,buff,BUFF\_SIZE)) > 0)

{

if(write(destFD,buff,nbread) != nbread)

printf("\nError in writing data to %s\n",argv[2]);

}

if(nbread == -1)

printf("\nError in reading data from %s\n",argv[1]);

if(close(srcFD) == -1)

printf("\nError in closing file %s\n",argv[1]);

if(close(destFD) == -1)

printf("\nError in closing file %s\n",argv[2]);

exit(EXIT\_SUCCESS);

}

***10)NamedPipe***

// C program to implement one side of FIFO

// This side writes first, then reads

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

int fd;

// FIFO file path

char \* myfifo = "/tmp/myfifo";

// Creating the named file(FIFO)

// mkfifo(<pathname>, <permission>)

mkfifo(myfifo, 0666);

char arr1[80], arr2[80];

while (1)

{

// Open FIFO for write only

fd = open(myfifo, O\_WRONLY);

// Take an input arr2ing from user.

// 80 is maximum length

fgets(arr2, 80, stdin);

// Write the input arr2ing on FIFO

// and close it

write(fd, arr2, strlen(arr2)+1);

close(fd);

// Open FIFO for Read only

fd = open(myfifo, O\_RDONLY);

// Read from FIFO

read(fd, arr1, sizeof(arr1));

// Print the read message

printf("User2: %s\n", arr1);

close(fd);

}

return 0;

}

// C program to implement one side of FIFO

// This side reads first, then reads

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

int fd1;

// FIFO file path

char \* myfifo = "/tmp/myfifo";

// Creating the named file(FIFO)

// mkfifo(<pathname>,<permission>)

mkfifo(myfifo, 0666);

char str1[80], str2[80];

while (1)

{

// First open in read only and read

fd1 = open(myfifo,O\_RDONLY);

read(fd1, str1, 80);

// Print the read string and close

printf("User1: %s\n", str1);

close(fd1);

// Now open in write mode and write

// string taken from user.

fd1 = open(myfifo,O\_WRONLY);

fgets(str2, 80, stdin);

write(fd1, str2, strlen(str2)+1);

close(fd1);

}

return 0;

}