//PageReplacement - FCFS

#include <iostream>

using namespace std;

int main(){

int page[50],frame[10],n,no,avail,count=0;

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

cin>>n;

cout<<"Enter the page number";

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

cin>>page[i];

}

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

cin>>no;

for(int i=0;i<no;i++){

frame[i]=-1;

}

int k=0;

cout<<"Page\tFrame\n";

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

cout<<page[i]<<"\t\t";

avail=0;

for(int j=0;j<no;j++){

if(frame[j]==page[i]){

avail=1;

}

}

if(avail==0){

frame[k]=page[i];

k=(k + 1)%no;

count++;

for(int j=0;j<no;j++){

cout<<frame[j]<<"\t";

}

}

cout<<endl;

}

cout<<"Total number of page fault: "<<count;

}

//Disk Scheduling - FCFS

#include <iostream>

using namespace std;

int absolutediff(int a,int b){

return abs(a-b);

}

void FCFS(int n,int request[],int head){

int total=0;

int current=head;

cout<<"Request Sequence: "<<head;

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

int next=request[i];

total+=absolutediff(current,next);

current=next;

cout<<" -> "<<next;

}

cout<<endl<<"Total Head Movement: "<<total;

}

int main(){

int n,head;

cout<<"Enter number of process: ";

cin>>n;

int request[n];

cout<<"Enter the requests:";

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

cin>>request[i];

}

cout<<"Enter the initial position: ";

cin>>head;

FCFS(n,request,head);

}

//Stat

#include<stdio.h>

#include<sys/stat.h>

#include<stdlib.h>

int

main ()

{

struct stat s;

int a, b;

if (stat ("SKCET.txt", &s) == -1)

{

perror ("Error: cannot stat file");

exit (0);

}

a = s.st\_blksize;

b = s.st\_size;

printf ("ALLOCATED SIZE OF THE FILE: %d\n", a);

printf ("ACTUAL SIZE OF THE FILE: %d\n", b);

}

//CPU Scheduling - FCFS

#include<bits/stdc++.h>

using namespace std;

void findwt(int wt[],int p[],int n,int bt[]){

wt[0]=0;

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

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

}

}

void findtat(int wt[],int p[],int n,int bt[],int tat[]){

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

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

}

}

void findavg(int p[],int n,int bt[]){

int wt[n],tat[n],twt=0,ttat=0;

findwt(wt,p,n,bt);

findtat(wt,p,n,bt,tat);

cout<<"Process "<<"Burst Time "<<"Waiting Time "<<"Turnaround Time"<<endl;

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

twt = twt + wt[i];

ttat = ttat + tat[i];

cout<<" P"<<i+1<<"\t"<<bt[i]<<"\t"<<wt[i]<<"\t\t"<<tat[i]<<endl;

}

cout<<"Average Waiting Time: "<<(float)twt/(float)n<<endl;

cout<<"Average Turnaround Time: "<<(float)ttat/(float)n;

}

int main(){

int A,B,C,D,E;

cout<<"Enter the jobs: "<<endl;

cin>>A>>B>>C>>D>>E;

int job[]={A,B,C,D,E};

int n=sizeof job/sizeof job[0];

int b1,b2,b3,b4,b5;

cout<<"Enter Burst Time: "<<endl;

cin>>b1>>b2>>b3>>b4>>b5;

int bt[]={b1,b2,b3,b4,b5};

findavg(job,n,bt);

}

//Producer Consumer Problem

#include<bits/stdc++.h>

using namespace std;

struct Buffer{

int mymutex=1;

int full=0;

int Myempty=3;

int x=0;

};

Buffer buffer;

void producer(){

--buffer.mymutex;

++buffer.full;

--buffer.Myempty;

buffer.x++;

cout<<"Producer produces: item";

cout<<buffer.x<<endl;

++buffer.mymutex;

}

void consumer(){

--buffer.mymutex;

--buffer.full;

++buffer.Myempty;

cout<<"Conusumer consumes: item";

cout<<buffer.x<<endl;

buffer.x--;

++buffer.mymutex;

}

int main(){

int n,i;

for(i=1;i>0;i++){

cout<<"Enter the Process:";

cin>>n;

switch(n){

case 1:

if((buffer.mymutex==1) && (buffer.Myempty!=0)){

producer();

}

else{

cout<<"Buffer is full"<<endl;

}

break;

case 2:

if((buffer.mymutex==1) && (buffer.full!=0)){

consumer();

}

else{

cout<<"Buffer is empty"<<endl;

}

break;

case 3:

exit(0);

break;

}

}

}

//Priority Scheduling

#include<bits/stdc++.h>

using namespace std;

class Process{

public:

string p;

int at;

int bt;

int prior;

int rt;

int rest;

int ct;

int wt;

int tat;;

void initialize(){

rt = bt;

}

};

int main(){

cout<<"Enter the process:";

int n;

cin>>n;

Process process[n];

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

cin>>process[i].p;

cin>>process[i].at;

cin>>process[i].bt;

cin>>process[i].prior;

process[i].initialize();

}

cout<<endl;

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

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

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

{

Process s = process[j];

process[j] = process[i];

process[i] = s;

}

}

}

int current=0;

while(true){

int currenthighpriorindex=-1;

int currenthighprior=numeric\_limits<int>::max();

bool isCompleted = true;

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

if(process[i].rt > 0){

isCompleted = false;

if(process[i].at <= current){

if(process[i].prior < currenthighprior){

currenthighprior = process[i].prior;

currenthighpriorindex = i;

}

}

}

}

if(isCompleted){

break;

}

process[currenthighpriorindex].rt=current;

process[currenthighpriorindex].rt =0;

current +=process[currenthighpriorindex].bt;

process[currenthighpriorindex].ct = current;

}

int trt=0,tct=0,twt=0,ttat=0;

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

process[i].tat = process[i].ct - process[i].at;

process[i].wt = process[i].tat - process[i].bt;

trt += process[i].rt;

tct += process[i].ct;

twt += process[i].wt;

ttat += process[i].tat;

}

cout<<"Process\t"<<"Arrival Time\t"<<"Burst Time\t"<<"Priority "<<"CompletionTime"<<endl;

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

cout<<" "<<process[i].p<<"\t"<<process[i].at<<"\t "<<process[i].bt<<"\t "<<process[i].prior<<"\t"<<process[i].ct<<"\t"<<endl;

}

cout<<"Turnaround Time\t"<<"Waiting Time"<<endl;

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

cout<<"\t\t"<<process[i].tat<<"\t\t"<<process[i].wt<<endl;

}

cout<<"Average Waiting Time: "<<twt /n;

cout<<"\nAverage Turnaround Time: "<<ttat /n;

}

//factorial

read -p "Enter a number: " number

fact=1

while [ $number -gt 1 ]

do

fact=$((fact \* number))

number=$((number - 1))

done

echo $fact

//fibonacci

read -p "Enter the number of terms for series: " number

a=0

b=1

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

do

echo "$a "

fn=$((a + b))

a=$b

b=$fn

done

//sum of n numbers

read -p "Enter the number of digits:" number

sum=0

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

do

sum=$((sum+i))

done

echo $sum

//greatest of 3

read n1

read n2

read n3

if((n1 >= n2 && n1>=n3))

then

echo "$n1 is the greatest"

elif((n2 >= n1 && n2 >= n3))

then

echo "$n2 is the greatest"

else

echo "$n3 is the greatest"

fi

//leap year

read year

if(( (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0) ))

then

echo "$year is a leap year"

else

echo "$year is not a leap year"

fi

//odd or even

read number

if((number % 2 == 0))

then

echo "$number is even"

else

echo "$number is odd"

fi

//file operation shell

create\_file() {

echo "Creating a new file..."

touch "$1"

echo "File '$1' created successfully."

}

# Function to copy a file

copy\_file() {

echo "Copying file '$1' to '$2'..."

cp "$1" "$2"

echo "File '$1' copied to '$2' successfully."

}

# Function to delete a file

delete\_file() {

echo "Deleting file '$1'..."

rm "$1"

echo "File '$1' deleted successfully."

}

# Main program

echo "File Operations Program"

# User menu

echo "1. Create a file"

echo "2. Copy a file"

echo "3. Delete a file"

echo "4. Exit"

# Read user choice

read -p "Enter your choice: " choice

case $choice in

1) read -p "Enter file name to create: " filename

create\_file "$filename"

;;

2) read -p "Enter source file name: " source

read -p "Enter destination file name: " destination

copy\_file "$source" "$destination"

;;

3) read -p "Enter file name to delete: " filename

delete\_file "$filename"

;;

4) echo "Exiting program..."

exit 0

;;

\*) echo "Invalid choice. Please enter a valid option."

;;

esac

//edit,rename,copy c program

#include <iostream>

#include <fstream>

#include <string>

#include <cstdio> // For remove() function

using namespace std;

void copyFile(const string& sourceFile, const string& destinationFile) {

ifstream source(sourceFile, ios::binary);

ofstream destination(destinationFile, ios::binary);

destination << source.rdbuf();

source.close();

destination.close();

}

void editFile(const string& filename) {

fstream file(filename, ios::in | ios::out);

if (!file) {

cout << "Error: File " << filename << " not found." << endl;

return;

}

string newContent;

cout << "Enter new content for the file. Press Ctrl+D (Unix) or Ctrl+Z (Windows) to save and exit." << endl;

char ch;

while (cin.get(ch)) {

newContent += ch;

}

file.seekp(0, ios::beg);

file << newContent;

file.close();

}

void renameFile(const string& oldFilename, const string& newFilename) {

if (rename(oldFilename.c\_str(), newFilename.c\_str()) != 0) {

perror("Error renaming file");

} else {

cout << "File renamed successfully." << endl;

}

}

int main() {

string command;

string sourceFile, destinationFile, newFilename;

cout << "Available commands:" << endl;

cout << "1. copy" << endl;

cout << "2. edit" << endl;

cout << "3. rename" << endl;

cout << "Enter command: ";

cin >> command;

if (command == "copy") {

cout << "Enter source file name: ";

cin >> sourceFile;

cout << "Enter destination file name: ";

cin >> destinationFile;

copyFile(sourceFile, destinationFile);

cout << "File copied successfully." << endl;

} else if (command == "edit") {

cout << "Enter file name to edit: ";

cin >> sourceFile;

editFile(sourceFile);

cout << "File edited successfully." << endl;

} else if (command == "rename") {

cout << "Enter old file name: ";

cin >> sourceFile;

cout << "Enter new file name: ";

cin >> newFilename;

renameFile(sourceFile, newFilename);

} else {

cout << "Invalid command. Please enter 'copy', 'edit', or 'rename'." << endl;

}

return 0;

}

//fork and exec

//process system call

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main() {

int pid = fork(); // Create a new process

if (pid < 0) { // Check if fork failed

printf("The fork cannot be created\n");

exit(1);

}

else if (pid == 0) { // In child process

execlp("/bin/ls", "ls", NULL); // Execute command

perror("execlp is failed");

exit(1);

}

else { // In parent process

printf("\n The process id of the Child: %d", getpid());

printf("\n The process id of the Parent: %d", getppid());

}

return 0; // Return from main

}

//given file is in a directory or not

#include <stdio.h>

#include <stdbool.h>

// return true if the file specified by the filename exists

bool file\_exists(const char \*filename)

{

FILE \*fp = fopen(filename, "r");

bool is\_exist = false;

if (fp != NULL)

{

is\_exist = true;

fclose(fp); // close the file

}

return is\_exist;

}

int main()

{

char \*filename = "readme.txt";

if (file\_exists(filename))

printf("File %s exists", filename);

else

printf("File %s doesn't exist.", filename);

return 0;

}

//SJF

#include <iostream>

using namespace std;

int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

cout << "Enter number of process: ";

cin >> n;

cout << "Enter Burst Time:" << endl;

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

{

// cout << "P" << i + 1 << ": ";

cin >> A[i][1];

A[i][0] = i + 1;

}

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

{

index = i;

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

if (A[j][1] < A[index][1])

index = j;

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

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

{

A[i][2] = 0;

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

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

cout << "Process BurstTime WaitingTime TurnaroundTime" << endl;

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

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

cout << " P" << A[i][0] << " " << A[i][1] << " " << A[i][2] << " " << A[i][3] << endl;

}

avg\_tat = (float)total / n;

cout << "Average Waiting Time = " << avg\_wt << endl;

cout << "Average Turnaround Time = " << avg\_tat << endl; }

//Bankers

#include <iostream>

using namespace std;

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;

// To check if sequence is safe or not

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

{

if(f[i]==0)

{

flag = 0;

cout << "The given sequence is not safe";

break;

}

}

if(flag==1)

{

cout << "Following is the SAFE Sequence" << endl;

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

cout << " P" << ans[i] << " ->";

cout << " P" << ans[n - 1] <<endl;

}

return (0);

}