NAME:S.Nithish

REG :192124146

PROGRAMS

1.Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.

#include<stdio.h>

int main()

{

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

printf("Enter total number of processes(maximum 20):");

scanf("%d",&n);

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

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

{

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

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

}

wt[0]=0;for(i=1;i<n;i++)

{

wt[i]=0;

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

wt[i]+=bt[j];

}

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

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

{

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

avwt+=wt[i];

avtat+=tat[i];

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

}

avwt/=i;

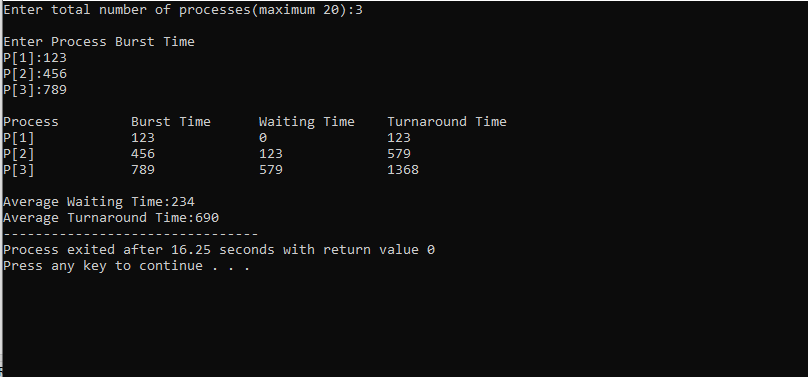
avtat/=i;

printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

return 0;

}

Output: 

2. Identify the system calls to copy the content of one file to another and illustrate the same using a C program.

3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations.

a. All processes are activated at time 0.

b. Assume that no process waits on I/O devices.

#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n,

int bt[], int wt[])

{

wt[0] = 0;

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

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

}

void findTurnAroundTime( int processes[], int n,

int bt[], int wt[], int tat[])

{

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

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

}

void findavgTime( int processes[], int n, int bt[])

{

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

findWaitingTime(processes, n, bt, wt);

findTurnAroundTime(processes, n, bt, wt, tat);

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

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

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

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

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

}

cout << "Average waiting time = "

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

cout << "\nAverage turn around time = "

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

}

int main()

{

int processes[] = { 1, 2, 3};

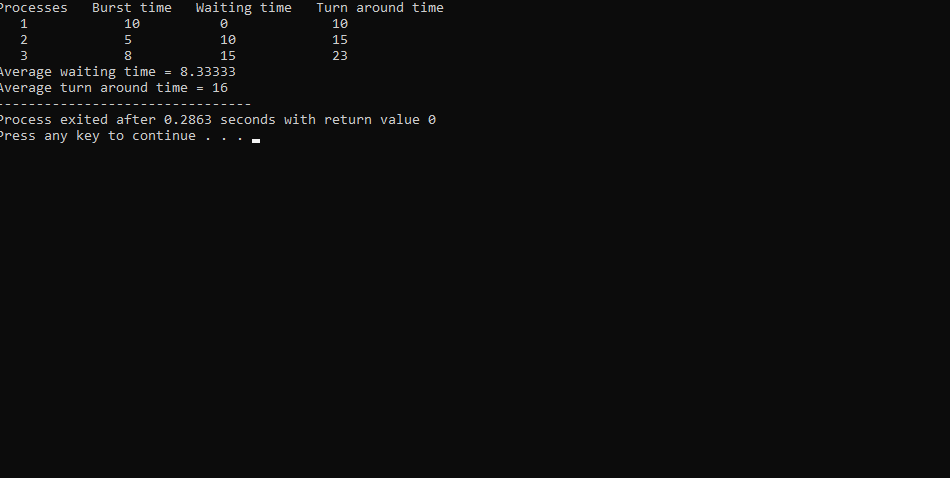
int n = sizeof processes / sizeof processes[0];

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

Output: 

4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.

#include<stdio.h>

#include<stdlib.h>

int main()

{

FILE \*f1,\*f2;

char filename[100],c;

f1=fopen("D:\DEVC++\test2.c","r");

f2=fopen("D:\DEVC++\test1.txt","w");

c=fgetc(f1);

while(c!=EOF)

{

fputc(c,f2);

c=fgetc(f1);

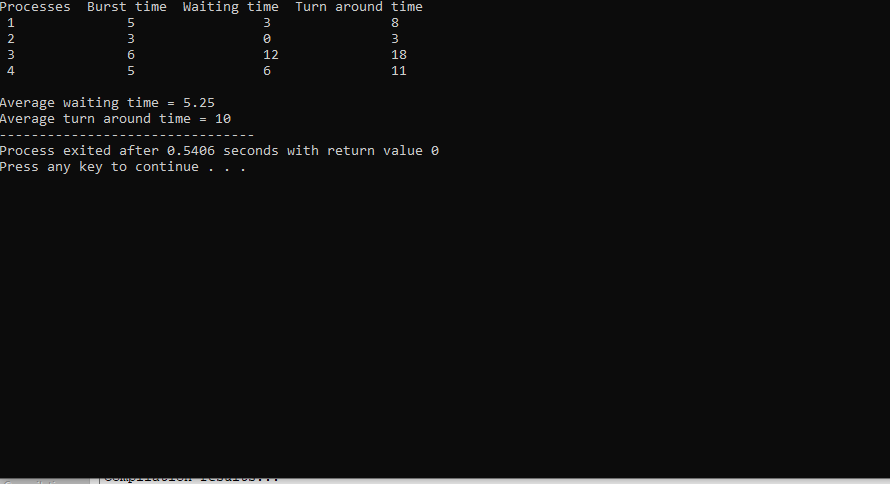
}

printf("CONTENTS COPIED SUCCESSFULLY....");

fclose(f1);

fclose(f2);

}

Output: 

5. Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.

#include <iostream>

using namespace std;

int main()

{

int n, m, i, j, k;

n = 5;

m = 3;

int alloc[5][3] = { { 0, 1, 0 },

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[5][3] = { { 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 } };

int avail[3] = { 3, 3, 2 };

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;

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++)

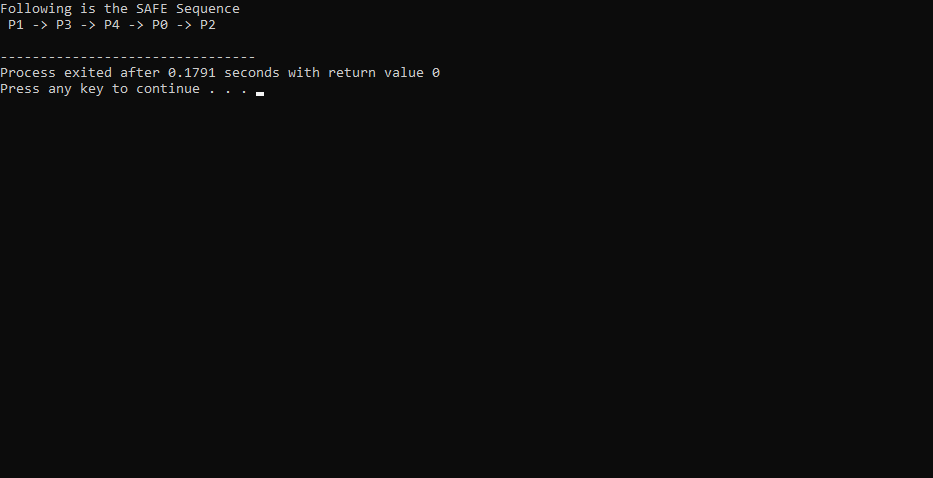
cout << " P" << ans[i] << " ->";

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

}

return (0);

}

Output: 

6. Construct a C program to simulate producer-consumer problem using semaphores.

#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("\n1.Producer\n2.Consumer\n3.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;

}

}

return 0;

}

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("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

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

x--;

mutex=signal(mutex);

}

Output: 

7. Construct a C program to simulate the First in First Out paging technique of memory management.

#include <stdio.h>

int main()

{

int incomingStream[] = {4, 1, 2, 4, 5};

int pageFaults = 0;

int frames = 3;

int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");

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(incomingStream[m] == temp[n])

{

s++;

pageFaults--;

}

}

pageFaults++;

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

{

temp[m] = incomingStream[m];

}

else if(s == 0)

{

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

}

printf("\n");

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

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

{

if(temp[n] != -1)

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

else

printf(" - \t\t\t");

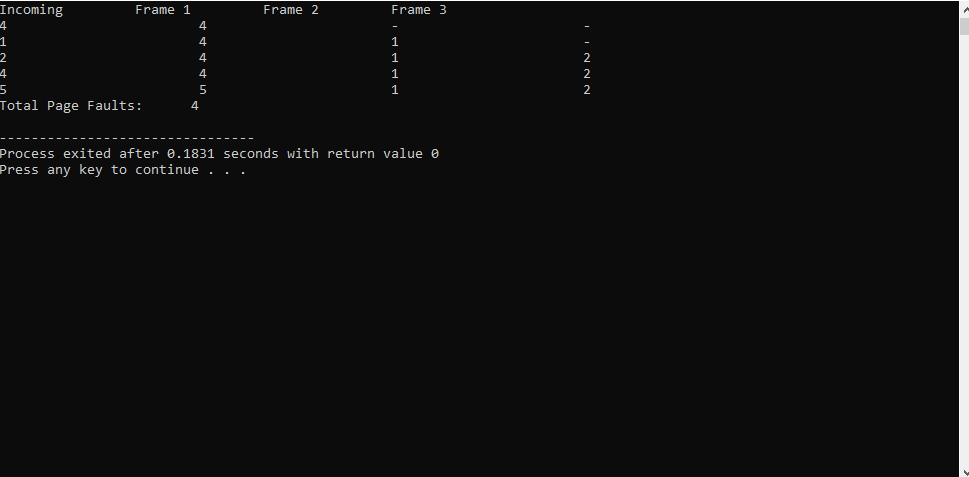
}

}

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

return 0;

}

Output: 

8. Construct a C program to simulate the Least Recently Used paging technique of memory management.

#include<bits/stdc++.h>

using namespace std;

const int N=100005;

int n;

int frame\_size;

int pages[N];

void lru\_page\_replacement()

{

unordered\_set<int> s;

unordered\_map<int, int> indexes;

int page\_faults = 0;

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

{

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

{

cout<<"Reference to page "<<pages[i]<<" did not cause a page fault\n";

}

else

{

if (s.size() < frame\_size)

{

s.insert(pages[i]);

page\_faults++;

}

else

{

int lru = INT\_MAX, val;

for (auto it : s)

{

if (indexes[it] < lru)

{

lru = indexes[it];

val = it;

}

}

s.erase(val);

s.insert(pages[i]);

page\_faults++;

}

cout<<"Reference to page "<<pages[i]<<" caused a page fault\n";

}

indexes[pages[i]] = i;

}

cout<<"\nTotal Page Faults: "<<page\_faults;

}

int main()

{

cout<<"Number of Frames: ";

cin>>frame\_size;

cout<<"Page Reference Stream Length: ";

cin>>n;

cout<<"Page Reference Stream:\n";

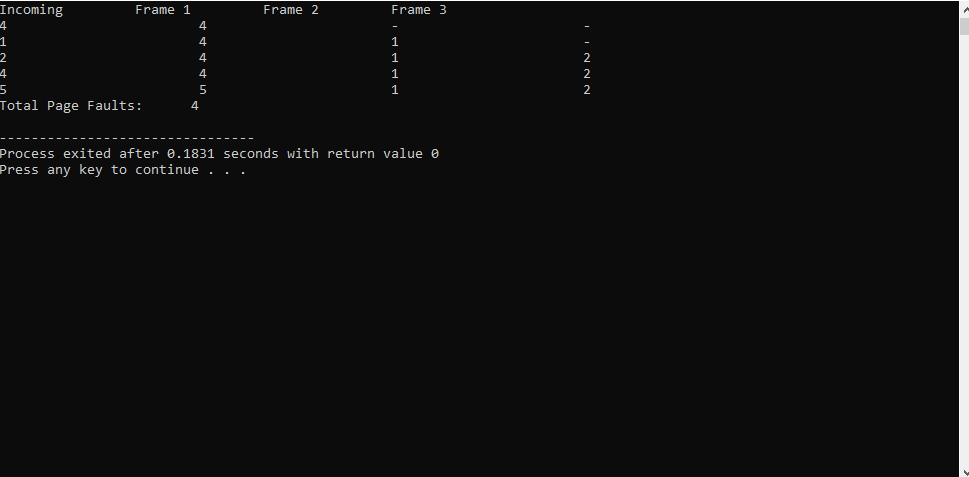
for(int i=0; i<n; i++)

cin>>pages[i];

lru\_page\_replacement();

return 0;

}

Output: 

9. Construct a C program to simulate the optimal paging technique of memory management

#include <bits/stdc++.h>

using namespace std;

int predict(int page[], vector<int>& fr, int pn, int index) {

int res = -1, farthest = index;

for (int i = 0; i < fr.size(); i++) {

int j;

for (j = index; j < pn; j++) {

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

if (j > farthest) {

farthest = j;

res = i;

}

break;

}

}

if (j == pn)

return i;

}

return (res == -1) ? 0 : res;

}

bool search(int key, vector<int>& fr) {

for (int i = 0; i < fr.size(); i++)

if (fr[i] == key)

return true;

return false;

}

void opr(int page[], int pn, int fn) {

vector<int> fr;

int hit = 0;

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

if (search(page[i], fr)) {

hit++;

continue;

}

if (fr.size() < fn)

fr.push\_back(page[i]);

else {

int j = predict(page, fr, pn, i + 1);

fr[j] = page[i];

}

}

cout << "Hits = " << hit << endl;

cout << "Misses = " << pn - hit << endl;

}

int main() {

int page[] = { 1, 7, 8, 3, 0, 2, 0, 3, 5, 4, 0, 6, 1 };

int pn = sizeof(page) / sizeof(page[0]);

int fn = 3;

opr(page, pn, fn);

return 0;

}

Output: 