OS LAB

FCFS

#include <stdio.h>

#include <limits.h>

struct Process {

int pid;

int arrival\_time;

int burst\_time;

int waiting\_time;

int turnaround\_time;

int completion\_time;

};

void displayGanttChart(struct Process processes[], int gantt[], int n, int burst\_time[]) {

printf("\nGantt Chart:\n");

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

printf("| P%d ", gantt[i]);

}

printf("|\n");

int time = 0;

printf("%d", time);

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

time += burst\_time[i];

printf("%5d", time);

}

printf("\n");

}

void fcfsScheduling(struct Process processes[], int n, float\* avg\_waiting\_time, float\* avg\_turnaround\_time) {

int current\_time = 0;

int gantt[100], burst\_time[100], gantt\_len = 0;

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

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

if (processes[i].arrival\_time > processes[j].arrival\_time) {

struct Process temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

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

if (current\_time < processes[i].arrival\_time)

current\_time = processes[i].arrival\_time;

processes[i].completion\_time = current\_time + processes[i].burst\_time;

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

current\_time += processes[i].burst\_time;

gantt[gantt\_len] = processes[i].pid;

burst\_time[gantt\_len] = processes[i].burst\_time;

gantt\_len++;

}

displayGanttChart(processes, gantt, gantt\_len, burst\_time);

\*avg\_waiting\_time = 0;

\*avg\_turnaround\_time = 0;

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

\*avg\_waiting\_time += processes[i].waiting\_time;

\*avg\_turnaround\_time += processes[i].turnaround\_time;

printf("P%d: Waiting Time = %d, Turnaround Time = %d\n", processes[i].pid, processes[i].waiting\_time, processes[i].turnaround\_time);

}

\*avg\_waiting\_time /= n;

\*avg\_turnaround\_time /= n;

printf("Average Waiting Time: %.2f\n", \*avg\_waiting\_time);

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

}

int main() {

int n = 5;

struct Process processes[] = {

{1, 0, 5, 0, 0, 0},

{2, 4, 4, 0, 0, 0},

{3, 3, 7, 0, 0, 0},

{4, 6, 3, 0, 0, 0},

{5, 7, 1, 0, 0, 0}

};

float avg\_waiting\_time\_fcfs, avg\_turnaround\_time\_fcfs;

printf("Shivani J 24MCS1023\n");

printf("\nFCFS Scheduling:\n");

fcfsScheduling(processes, n, &avg\_waiting\_time\_fcfs, &avg\_turnaround\_time\_fcfs);

printf("\nEfficiency Comparison:\n");

printf("FCFS - Average Waiting Time: %.2f, Average Turnaround Time: %.2f\n", avg\_waiting\_time\_fcfs, avg\_turnaround\_time\_fcfs);

printf("\n");

return 0;

}

**Scan algorithm:**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 8

#define DISK\_SIZE 200

void SCAN(int arr[], int head, const char\* direction) {

int seek\_count = 0;

int distance, cur\_track;

int left[SIZE], right[SIZE];

int left\_size = 0, right\_size = 0;

int seek\_sequence[SIZE \* 2];

int seek\_sequence\_index = 0;

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

if (arr[i] < head) {

left[left\_size++] = arr[i];

} else if (arr[i] > head) {

right[right\_size++] = arr[i];

}

}

for (int i = 0; i < left\_size - 1; i++) {

for (int j = 0; j < left\_size - i - 1; j++) {

if (left[j] < left[j + 1]) {

int temp = left[j];

left[j] = left[j + 1];

left[j + 1] = temp;

}

}

}

for (int i = 0; i < right\_size - 1; i++) {

for (int j = 0; j < right\_size - i - 1; j++) {

if (right[j] > right[j + 1]) {

int temp = right[j];

right[j] = right[j + 1];

right[j + 1] = temp;

}

}

}

if (direction[0] == 'l') {

for (int i = 0; i < left\_size; i++) {

cur\_track = left[i];

seek\_sequence[seek\_sequence\_index++] = cur\_track;

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

seek\_sequence[seek\_sequence\_index++] = 0;

seek\_count += head;

head = 0;

for (int i = 0; i < right\_size; i++) {

cur\_track = right[i];

seek\_sequence[seek\_sequence\_index++] = cur\_track;

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

} else if (direction[0] == 'r') {

for (int i = 0; i < right\_size; i++) {

cur\_track = right[i];

seek\_sequence[seek\_sequence\_index++] = cur\_track;

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

seek\_sequence[seek\_sequence\_index++] = DISK\_SIZE - 1;

seek\_count += (DISK\_SIZE - 1 - head);

head = DISK\_SIZE - 1;

for (int i = 0; i < left\_size; i++) {

cur\_track = left[i];

seek\_sequence[seek\_sequence\_index++] = cur\_track;

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

}

printf("Total number of seek operations = %d\n", seek\_count);

printf("Seek Sequence is:\n");

for (int i = 0; i < seek\_sequence\_index; i++) {

printf("%d\n", seek\_sequence[i]);

}

}

int main() {

int arr[SIZE] = {176, 79, 34, 60, 92, 11, 41, 114};

int head = 50;

const char\* direction = "left";

SCAN(arr, head, direction);

return 0;

}

**LRU 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], 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;

time[i] = 0;

}

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

int page = pages[i];

int page\_found = 0;

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

if (frames[j] == page) {

time[j] = counter++;

page\_found = 1;

break;

}

}

if (!page\_found) {

pos = findLRU(time, no\_of\_frames);

frames[pos] = page;

time[pos] = counter++;

faults++;

}

printf("Current frames: ");

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

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

}

printf("\n");

}

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

return 0;

}

Another way:

#include<stdio.h>

int main()

{

int m, n, position, k, l;

int a = 0, b = 0, page\_fault = 0;

int total\_frames = 3;

int frames[total\_frames];

int temp[total\_frames];

int pages[] = {3,1,2,4,2,4,1,5,9,2,1,4,7,8,3};

int total\_pages = sizeof(pages)/sizeof(pages[0]);

for(m = 0; m < total\_frames; m++){

frames[m] = -1;

}

for(n = 0; n < total\_pages; n++)

{

printf("%d: ", pages[n]);

a = 0, b = 0;

for(m = 0; m < total\_frames; m++)

{

if(frames[m] == pages[n])

{

a = 1;

b = 1;

break;

}

}

if(a == 0)

{

for(m = 0; m < total\_frames; m++)

{

if(frames[m] == -1)

{

frames[m] = pages[n];

b = 1;

page\_fault++;

break;

}

}

}

if(b == 0)

{

for(m = 0; m < total\_frames; m++)

{

temp[m] = 0;

}

for(k = n - 1, l = 1; l <= total\_frames - 1; l++, k--)

{

for(m = 0; m < total\_frames; m++)

{

if(frames[m] == pages[k])

{

temp[m] = 1;

}

}

}

for(m = 0; m < total\_frames; m++)

{

if(temp[m] == 0)

position = m;

}

frames[position] = pages[n];

page\_fault++;

}

for(m = 0; m < total\_frames; m++)

{

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

}

printf("\n");

}

printf("\nTotal Number of Page Faults:\t%d\n", page\_fault);

return 0;

}