**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("Hari Prakash S\n24MCB1022\n\n");

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

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

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

printf("\n");

return 0;

}

**Task 1: Producer-Consumer Problem Code**

#include <stdio.h>

#include <pthread.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <semaphore.h>

sem\_t empty, full, mutex;

char buf[10];

int in = 0;

int out = 0;

void\* producer(void\* arg) {

int i;

printf("Inside producer\n");

for (i = 0; i < 10; i++) {

sem\_wait(&empty);

sem\_wait(&mutex);

buf[in] = i;

printf("Item produced is %d\n", buf[in]);

in = (in + 1) % 10;

sem\_post(&mutex);

sem\_post(&full);

sleep(1);

}

pthread\_exit("Producer\n");

}

void\* consumer(void\* arg) {

int j;

printf("Inside consumer\n");

for (j = 0; j < 10; j++) {

sem\_wait(&full);

sem\_wait(&mutex);

char item = buf[out];

printf("Consumed item is %d\n", item);

out = (out + 1) % 10;

sem\_post(&mutex);

sem\_post(&empty);

sleep(2);

}

pthread\_exit("Consumer\n");

}

int main() {

pthread\_t pid1, pid2;

sem\_init(&empty, 0, 10);

sem\_init(&full, 0, 0);

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

pthread\_create(&pid1, NULL, producer, NULL);

pthread\_create(&pid2, NULL, consumer, NULL);

void\* status;

pthread\_join(pid1, &status);

printf("The exited status of producer is %s\n", (char\*)status);

pthread\_join(pid2, &status);

printf("The exited status of consumer is %s\n", (char\*)status);

sem\_destroy(&empty);

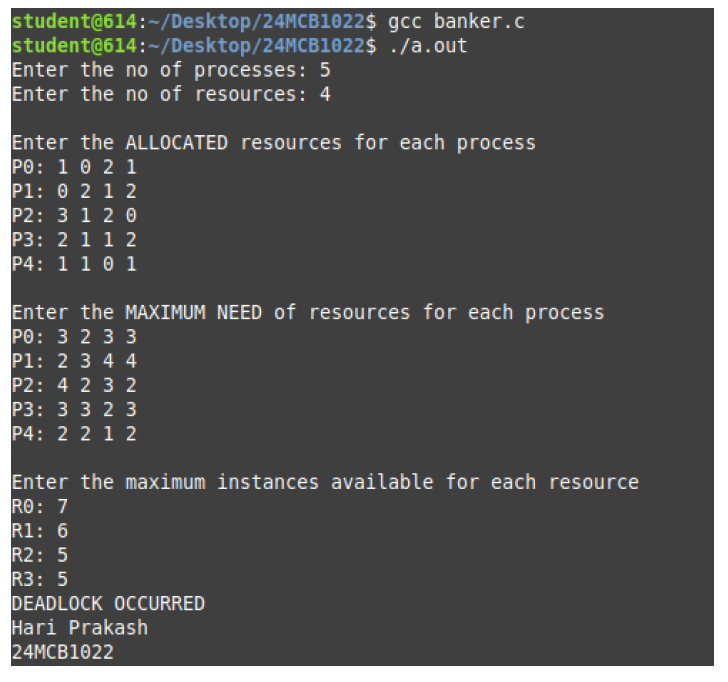
sem\_destroy(&full);

sem\_destroy(&mutex);

printf("\nHari Prakash S\n24MCB1022\n");

return 0;

}



**BANKER ALGo**

#include <stdio.h>

int main() {

int n, m, i, j, k;

int alloc[20][20], max\_need[20][20], max\_avail[20], avail[20];

int sum\_alloc[20] = {0};

printf("Enter the no of processes: ");

scanf("%d", &n);

printf("Enter the no of resources: ");

scanf("%d", &m);

printf("\nEnter the ALLOCATED resources for each process \n");

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

printf("P%d: ", i);

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

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

sum\_alloc[j] = sum\_alloc[j] + alloc[i][j];

}

}

printf("\nEnter the MAXIMUM NEED of resources for each process \n");

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

printf("P%d: ", i);

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

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

}

}

printf("\nEnter the maximum instances available for each resource\n");

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

printf("R%d: ", i);

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

avail[i] = max\_avail[i] - sum\_alloc[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\_need[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("DEADLOCK OCCURRED\n");

break;

}

}

if (flag == 1) {

printf("DEADLOCK NOT OCCURRED \n");

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

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

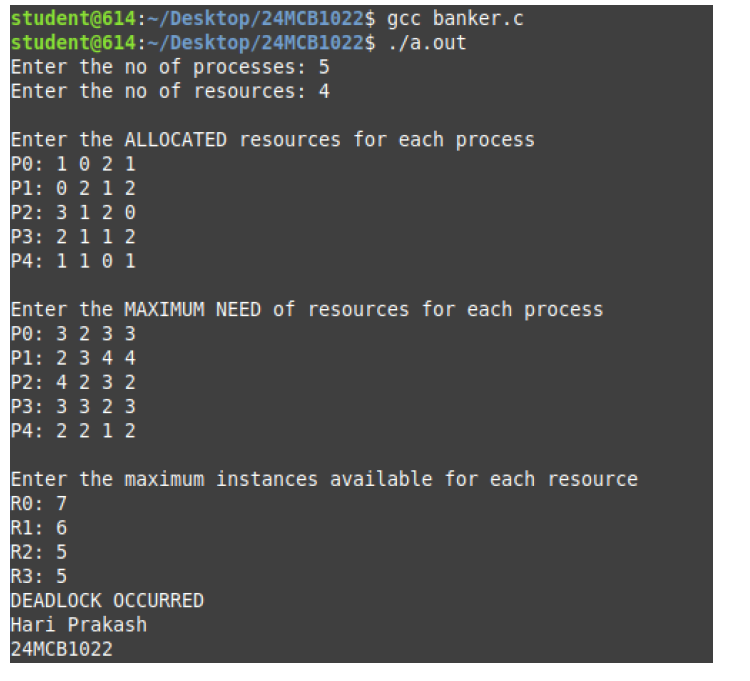
}

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

}

return 0;

}



**Page Replacement – LRU**

**CODE:**

#include <stdio.h>

int findLRU(int time[], int n) {

int i, min = time[0], pos = 0;

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

if(time[i] < min) {

min = time[i];

pos = i;

}

}

return pos;

}

int main() {

int frames, pages, pageFaults = 0, pageHits = 0;

float faultRatio, hitRatio;

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

scanf("%d", &frames);

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

scanf("%d", &pages);

int pageSequence[pages], frame[frames], time[frames];

printf("Enter the sequence of pages: ");

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

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

}

// Initialize frames and time array

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

frame[i] = -1;

time[i] = 0;

}

int counter = 0;

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

int flag1 = 0, flag2 = 0;

for(int j = 0; j < frames; ++j) {

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

counter++;

time[j] = counter;

flag1 = flag2 = 1;

pageHits++;

break;

}

}

if(flag1 == 0) {

for(int j = 0; j < frames; ++j) {

if(frame[j] == -1) {

counter++;

pageFaults++;

frame[j] = pageSequence[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0) {

int pos = findLRU(time, frames);

counter++;

pageFaults++;

frame[pos] = pageSequence[i];

time[pos] = counter;

}

// Display the current frame status

printf("\nFrames after processing page %d: ", pageSequence[i]);

for(int j = 0; j < frames; ++j) {

if(frame[j] != -1) {

printf("%d ", frame[j]);

} else {

printf("- ");

}

}

}

// Calculate ratios

faultRatio = (float)pageFaults / pages;

hitRatio = (float)pageHits / pages;

// Print results

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

printf("\nTotal Page Hits: %d", pageHits);

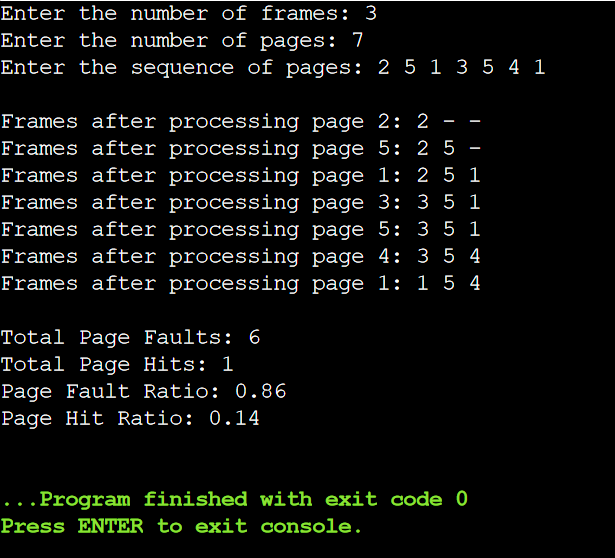
printf("\nPage Fault Ratio: %.2f", faultRatio);

printf("\nPage Hit Ratio: %.2f\n", hitRatio);

return 0;

}

**Output:**



**Disk Scheduling – SCAN**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

void scanDiskScheduling(int requests[], int n, int head, char direction, int disk\_size) {

int seek\_time = 0;

int distance, cur\_track;

int seek\_sequence[n + 1];

int index = 0;

// Sort the request array

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

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

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

int temp = requests[j];

requests[j] = requests[j + 1];

requests[j + 1] = temp;

}

}

}

int pos = 0;

// Find the position to start scanning based on the head position

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

if (requests[i] > head) {

pos = i;

break;

}

}

// Move in the given initial direction

if (direction == 'H') {

// Moving towards the higher end

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

cur\_track = requests[i];

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

distance = abs(cur\_track - head);

seek\_time += distance;

head = cur\_track;

}

// Go to the end and reverse direction

if (head != disk\_size - 1) {

seek\_time += abs(disk\_size - 1 - head);

head = disk\_size - 1;

}

// Move towards the lower end

for (int i = pos - 1; i >= 0; i--) {

cur\_track = requests[i];

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

distance = abs(cur\_track - head);

seek\_time += distance;

head = cur\_track;

}

} else {

// Moving towards the lower end

for (int i = pos - 1; i >= 0; i--) {

cur\_track = requests[i];

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

distance = abs(cur\_track - head);

seek\_time += distance;

head = cur\_track;

}

// Go to the start and reverse direction

if (head != 0) {

seek\_time += abs(head - 0);

head = 0;

}

// Move towards the higher end

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

cur\_track = requests[i];

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

distance = abs(cur\_track - head);

seek\_time += distance;

head = cur\_track;

}

}

// Output seek sequence and seek time

printf("Seek Sequence: ");

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

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

}

printf("\nTotal Seek Time: %d\n", seek\_time);

}

int main() {

int n, head, disk\_size;

char direction;

printf("Enter number of requests: ");

scanf("%d", &n);

int requests[n];

printf("Enter the request array: ");

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

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

}

printf("Enter initial head position: ");

scanf("%d", &head);

printf("Enter disk size: ");

scanf("%d", &disk\_size);

printf("Enter initial direction (L for lower side, H for higher side): ");

scanf(" %c", &direction);

scanDiskScheduling(requests, n, head, direction, disk\_size);

return 0;

}

**Ouptut:**

A computer screen with numbers and letters

Description automatically generated