

Academic Year: 2023-24 Sem: III

Sub: Operating Systems Laboratory SAP ID: 60003220193

**EXPERIMENT NO. 02**

**Code: Q1**

#include <stdio.h>

void calculateWaitingTime(int processes[], int n, int burst\_time[], int waiting\_time[]) { waiting\_time[0] = 0; // Waiting time for the first process is 0

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

waiting\_time[i] = burst\_time[i-1] + waiting\_time[i-1]; }

}

void calculateTurnaroundTime(int processes[], int n, int burst\_time[], int waiting\_time[], int turnaround\_time[]) {

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

turnaround\_time[i] = burst\_time[i] + waiting\_time[i]; }

}

void displayGanttChart(int processes[], int n, int burst\_time[]) { printf("\nGantt Chart:\n");

for (int i = 0; i < n; i++) { printf("| P%d ", processes[i]);

} printf("|\n");

int current\_time = 0;

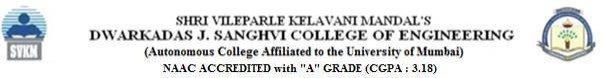
for (int i = 0; i < n; i++) { printf("%d\t", current\_time); current\_time += burst\_time[i];

}

printf("%d\n", current\_time); }

void calculateAverageWaitingTime(int processes[], int n, int burst\_time[], int waiting\_time[]) {

float total\_waiting\_time = 0;

**A.Y.:** 2023-24

for (int i = 0; i < n; i++) { total\_waiting\_time += waiting\_time[i];

}

float avg\_waiting\_time = total\_waiting\_time / n; printf("Average Waiting Time: %.2f\n", avg\_waiting\_time);

}

int main() { int n;

printf("Enter the number of processes: "); scanf("%d", &n);

int processes[n], burst\_time[n], waiting\_time[n], turnaround\_time[n];

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

printf("Enter burst time for process P%d: ", i+1); scanf("%d", &burst\_time[i]);

processes[i] = i + 1; }

calculateWaitingTime(processes, n, burst\_time, waiting\_time); calculateTurnaroundTime(processes, n, burst\_time, waiting\_time, turnaround\_time);

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n"); for (int i = 0; i < n; i++) {

printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst\_time[i], waiting\_time[i], turnaround\_time[i]);

}

displayGanttChart(processes, n, burst\_time);

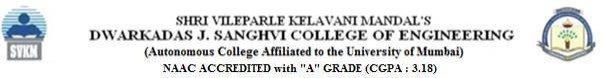
calculateAverageWaitingTime(processes, n, burst\_time, waiting\_time);

return 0; }

**Q2**

#include <stdio.h>

struct Process {

**A.Y.:** 2023-24 int id;

int arrival\_time; int burst\_time; int priority;

int waiting\_time;

int turnaround\_time; };

void sjf\_with\_priority(struct Process processes[], int n) { int total\_waiting\_time = 0;

int total\_turnaround\_time = 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 || (processes[i].arrival\_time == processes[j].arrival\_time &&

(processes[i].priority > processes[j].priority ||

(processes[i].priority == processes[j].priority && processes[i].burst\_time > processes[j].burst\_time)))) {

struct Process temp = processes[i]; processes[i] = processes[j]; processes[j] = temp;

} }

}

int current\_time = 0;

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

if (processes[i].arrival\_time > current\_time) { current\_time = processes[i].arrival\_time;

}

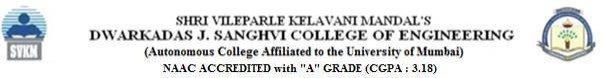
processes[i].waiting\_time = current\_time - processes[i].arrival\_time; processes[i].turnaround\_time = processes[i].waiting\_time + processes[i].burst\_time;

total\_waiting\_time += processes[i].waiting\_time; total\_turnaround\_time += processes[i].turnaround\_time;

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

printf("\nGantt Chart:\n"); printf("0");

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

**A.Y.:** 2023-24

printf("->P%d->%d", processes[i].id, current\_time); }

printf("\n\nTABLE\n"); printf("Process AT BT WT TAT\n"); for (int i = 0; i < n; i++) {

printf("P%d\t%d\t%d\t%d\t%d\n", processes[i].id, processes[i].arrival\_time, processes[i].burst\_time, processes[i].waiting\_time, processes[i].turnaround\_time);

}

double avg\_waiting\_time = (double)total\_waiting\_time / n; double avg\_turnaround\_time = (double)total\_turnaround\_time / n;

printf("\nAverage Turnaround Time: %.6lf\n", avg\_turnaround\_time); printf("Average Waiting Time: %.6lf\n", avg\_waiting\_time);

}

int main() { int n;

printf("Enter the number of processes: "); scanf("%d", &n);

struct Process processes[n];

for (int i = 0; i < n; i++) { processes[i].id = i + 1;

printf("Enter the arrival time for process P%d: ", i + 1); scanf("%d", &processes[i].arrival\_time);

printf("Enter the burst time for process P%d: ", i + 1); scanf("%d", &processes[i].burst\_time);

printf("Enter the priority for process P%d: ", i + 1); scanf("%d", &processes[i].priority);

}

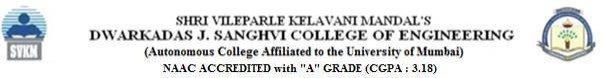
sjf\_with\_priority(processes, n);

return 0; }

**Q3**

#include <stdio.h>

void priorityScheduling(int processes[], int n, int burst\_time[], int priority[], int arrival\_time[]) {

**A.Y.:** 2023-24

int waiting\_time[n], turnaround\_time[n];

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

if(arrival\_time[j] > arrival\_time[j+1]) { int temp = arrival\_time[j]; arrival\_time[j] = arrival\_time[j+1]; arrival\_time[j+1] = temp;

temp = priority[j]; priority[j] = priority[j+1]; priority[j+1] = temp;

temp = burst\_time[j]; burst\_time[j] = burst\_time[j+1]; burst\_time[j+1] = temp;

temp = processes[j]; processes[j] = processes[j+1]; processes[j+1] = temp;

} }

}

waiting\_time[0] = 0;

int current\_time = arrival\_time[0];

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

waiting\_time[i] = burst\_time[i-1] + waiting\_time[i-1]; current\_time += burst\_time[i-1];

}

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

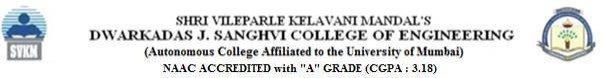
turnaround\_time[i] = burst\_time[i] + waiting\_time[i]; }

printf("\nGantt Chart:\n"); for(int i = 0; i < n; i++) {

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

printf("|\n");

current\_time = arrival\_time[0]; for(int i = 0; i < n; i++) {

**A.Y.:** 2023-24

printf("%d\t", current\_time); current\_time += burst\_time[i];

}

printf("%d\n", current\_time);

printf("\nProcess\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], arrival\_time[i], burst\_time[i], priority[i], waiting\_time[i], turnaround\_time[i]);

}

float avg\_waiting\_time = 0, avg\_turnaround\_time = 0; for(int i = 0; i < n; i++) {

avg\_waiting\_time += waiting\_time[i]; avg\_turnaround\_time += turnaround\_time[i];

}

avg\_waiting\_time /= n; avg\_turnaround\_time /= n;

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

}

int main() { int n;

printf("Enter the number of processes: "); scanf("%d", &n);

int processes[n], burst\_time[n], priority[n], arrival\_time[n]; for(int i = 0; i < n; i++) {

printf("Enter arrival time for process P%d: ", i+1); scanf("%d", &arrival\_time[i]);

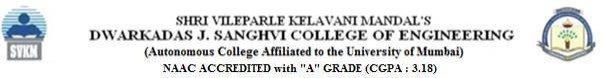
printf("Enter burst time for process P%d: ", i+1); scanf("%d", &burst\_time[i]);

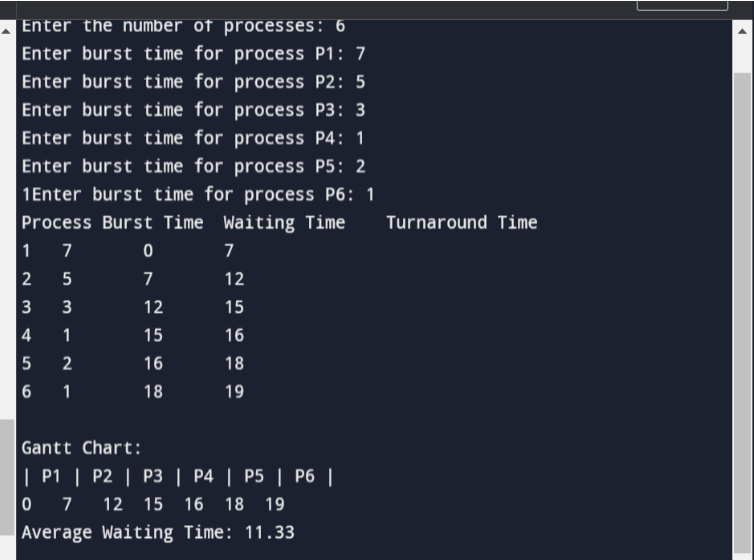
printf("Enter priority for process P%d: ", i+1); scanf("%d", &priority[i]);

processes[i] = i+1; }

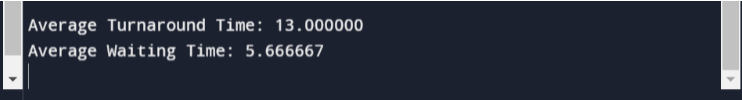
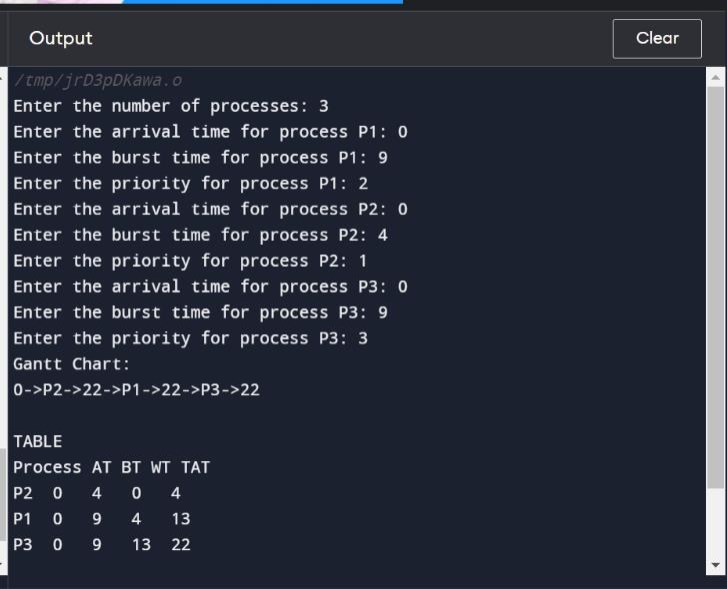
priorityScheduling(processes, n, burst\_time, priority, arrival\_time);

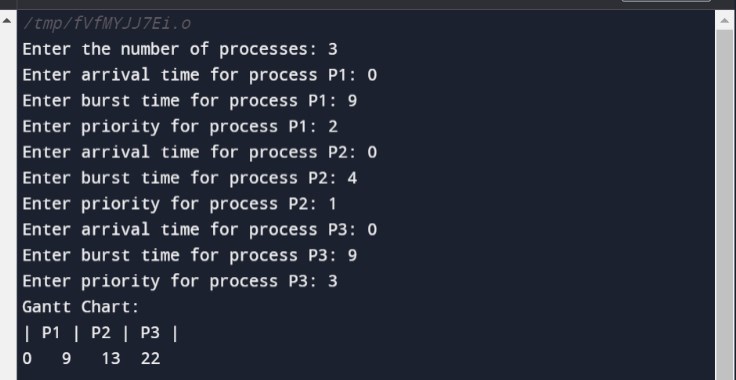
return 0;

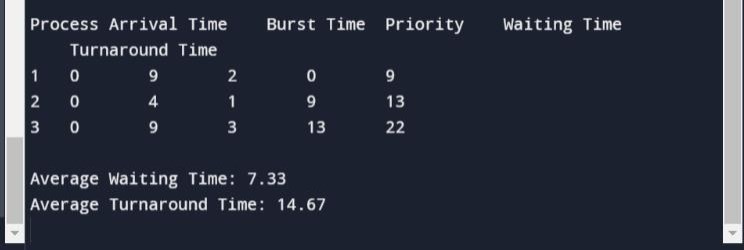
**A.Y.:** 2023-24 }

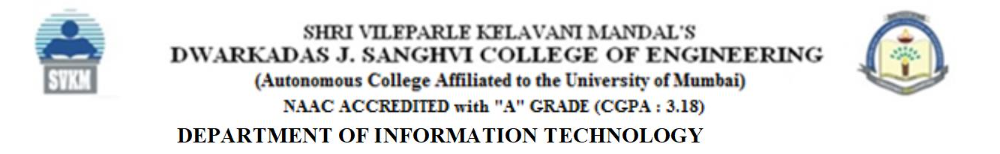
**OUTPUT: Q1**

**Q2**



**Q3**

**A.Y.:** 2023-24



Academic Year: 2023-24 Sem: III

Sub: Operating Systems Laboratory SAP ID: 60003220045

Name: Anish Sharma

**EXPERIMENT NO. 03**

Source Code:

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0; void producer()

{

--mutex; ++full;

--empty; x++;

printf("\nProducer produces" "item %d",

x); ++mutex;

}

void consumer() {

--mutex; --full; ++empty;

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

x--;

++mutex;

}

int main() {

int n, i;

printf("\n1. Press 1 for Producer" "\n2. Press 2 for Consumer" "\n3. Press 3 for Exit");

#pragma omp critical

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

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;

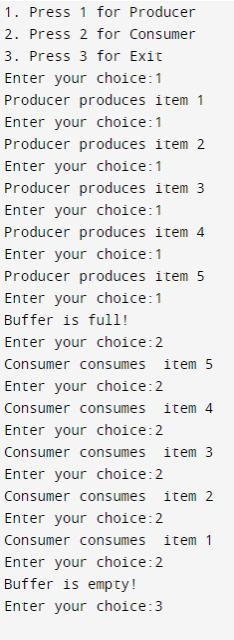
case 3:

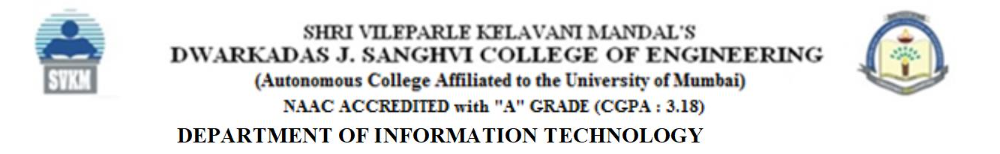
exit(0);

break;

} }

}

**Output:**



Academic Year: 2023-24 Sem: III Sub: Operating Systems Laboratory SAP ID: 60003220045

Name: Anish Sharma

**EXPERIMENT NO. 04**

**Q1)**

**CODE**

import java.util.\*;

class Exp4Q1.java{

public static void main(String[] args) {

int[][] allocation={{1,0,1},{2,1,2},{3,0,0},{1,0,1}};

int[][] max={{2,1,1},{5,4,4},{3,1,1},{1,1,1}};

int[] available={2,1,1};

int[][] need=new int[4][3];

for(int i=0;i<allocation.length;i++){

for(int j=0;j<available.length;j++){

need[i][j]=max[i][j]-allocation[i][j]; }

}

int[] work=available;

boolean[] finish= new boolean[max.length];

for(int i=0;i<work.length;i++){

finish[i]= false; }

int h=0;

int t=0;

System.out.println("Sequence");

while(h<=4){

for(int i=0;i<max.length;i++){

if(finish[i]==false){

if(need[i][0]<=work[0] && need[i][1]<=work[1] && need[i][2]<=work[2]){

for(int j=0;j<work.length;j++){

work[j] = work[j]+allocation[i][j];

}

System.out.print("P"+i+" ");

finish[i]=true;

t++;

}

} }

h++;

}

if(t==max.length){

System.out.println("\nThe process is safe");

}

else{

System.out.println("\nnot safe");

}

} }

***OUTPUT***



**Q2)**

***CODE***

import java.util.\*;

public class Exp4Q2 {

public static void main(String[] args) {

int[][] allocation = { { 0, 1, 1, 0 }, { 1, 2, 3, 1 }, { 1, 3, 6, 5 }, { 0, 6, 3, 2 }, { 0, 0, 1, 4 } };

int[][] max = { { 0, 2, 1, 0 }, { 1, 6, 5, 2 }, { 2, 3, 6, 6 }, { 0, 6, 5, 2 }, { 0, 6, 5, 6 } };

int[] available = { 1, 5, 2, 0 }; int[][] need = new int[5][4];

for (int i = 0; i < allocation.length; i++) {

for (int j = 0; j < available.length; j++) {

need[i][j] = max[i][j] - allocation[i][j];

}

}

int[] work = available;

boolean[] finish = new boolean[max.length];

for (int i = 0; i < max.length; i++) {

finish[i] = false;

}

int h = 0;

int t = 0;

System.out.println("Sequence"); while (h <= 5) {

for (int i = 0; i < max.length; i++) { if (finish[i] == false) {

if (need[i][0] <= work[0] && need[i][1] <= work[1] && need[i][2] <= work[2]

&& need[i][3] <= work[3]) {

for (int j = 0; j < work.length; j++) {

work[j] = work[j] + allocation[i][j];

}

System.out.print("P" + i + " ");

finish[i] = true; t++;

} }

}

h++;

}

if (t == max.length) {

System.out.println("\nThe process is safe");

}

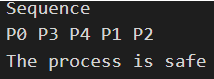
else {

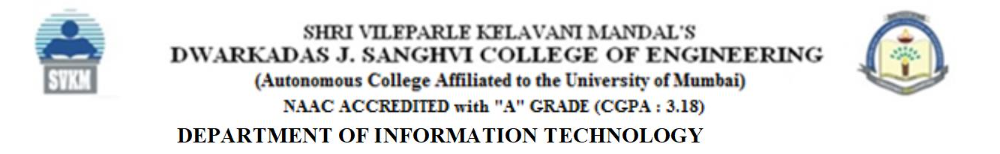
System.out.println("\n not safe");

}

}

***OUTPUT***





Academic Year: 2023-24 Sem: III

Sub: Operating Systems Laboratory SAP ID: 60003220045

Name: Anish Sharma

**EXPERIMENT NO. 05**

**(1) First Fit:**

#include<stdio.h>

void firstFit(int blockSize[], int m, int processSize[], int n) {

int i, j;

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

allocation[i] = -1; }

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

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

if (blockSize[j] >= processSize[i]) {

allocation[i] = j; blockSize[j] -= processSize[i];

break; }

} }

printf("\nProcess No.\tProcess Size\tBlock no.\n"); for (int i = 0; i < n; i++)

{

printf(" %i\t\t\t", i+1); printf("%i\t\t\t\t", processSize[i]); if (allocation[i] != -1)

printf("%i", allocation[i] + 1); else

printf("Not Allocated"); printf("\n");

}

}

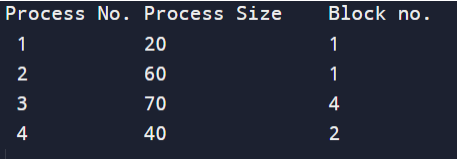
int main() {

int m; int n;

int blockSize[] = {100, 50, 30, 120, 35}; int processSize[] = {20,60,70,40};

m = sizeof(blockSize) / sizeof(blockSize[0]);

n = sizeof(processSize) / sizeof(processSize[0]); firstFit(blockSize, m, processSize, n);

return 0 ; }

**(2) Best Fit** #include <stdio.h>

void implimentBestFit(int blockSize[], int blocks, int processSize[], int proccesses) {

int allocation[proccesses]; int occupied[blocks];

for(int i = 0; i < proccesses; i++){ allocation[i] = -1;

}

for(int i = 0; i < blocks; i++){ occupied[i] = 0;

}

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

int indexPlaced = -1;

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

if (blockSize[j] >= processSize[i] && !occupied[j]) {

if (indexPlaced == -1) indexPlaced = j;

else if (blockSize[j] < blockSize[indexPlaced]) indexPlaced = j;

} }

if (indexPlaced != -1) {

allocation[i] = indexPlaced; occupied[indexPlaced] = 1;

} }

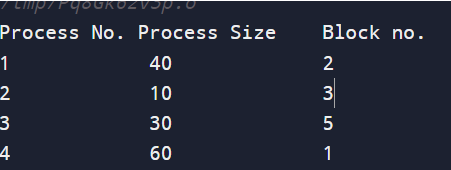
printf("\nProcess No.\tProcess Size\tBlock no.\n"); for (int i = 0; i < proccesses; i++)

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]); if (allocation[i] != -1)

printf("%d\n",allocation[i] + 1); else

printf("Not Allocated\n");



} }

int main() {

int blockSize[] = {100, 50, 30, 120, 35}; int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

int proccesses = sizeof(processSize)/sizeof(processSize[0]); implimentBestFit(blockSize, blocks, processSize, proccesses); return 0 ;

}

**(3) Worst Fit:** #include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes) {

int allocation[processes]; int occupied[blocks];

for(int i = 0; i < processes; i++){ allocation[i] = -1;

}

for(int i = 0; i < blocks; i++){ occupied[i] = 0;

}

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

int indexPlaced = -1; for(int j = 0; j < blocks; j++) {

if(blockSize[j] >= processSize[i] && !occupied[j]) {

if (indexPlaced == -1) indexPlaced = j;

else if (blockSize[indexPlaced] < blockSize[j]) indexPlaced = j;

} }

if (indexPlaced != -1) {

allocation[i] = indexPlaced; occupied[indexPlaced] = 1; blockSize[indexPlaced] -= processSize[i];

} }

printf("\nProcess No.\tProcess Size\tBlock no.\n"); for (int i = 0; i < processes; i++)

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]); if (allocation[i] != -1)

printf("%d\n",allocation[i] + 1); else

printf("Not Allocated\n"); }

}

int main()

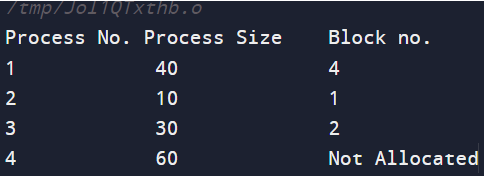
{

int blockSize[] = {100, 50, 30, 120, 35}; int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

int processes = sizeof(processSize)/sizeof(processSize[0]); implimentWorstFit(blockSize, blocks, processSize, processes); return 0;

}



COURSE CODE: DJS22ITL305 Sem: III

COURSE NAME: Operating System Laboratory CLASS:SY. BTech I1-1

# Name :Anish A. Shetty Sap id : 60003220045

**Experiment 6**

**CO/LO:** Apply appropriate process scheduling, memory mapping and disk scheduling methods.

**Aim**: Implementation of Page replacement algorithms (FIFO, LRU)

**FIFO :**

#include <stdio.h>

int main() {

int frames, pages, pageFaults = 0;

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

scanf("%d", &frames);

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

scanf("%d", &pages);

int incomingStream[pages];

printf("Enter the page reference sequence:\n");

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

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

}

int temp[frames];

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

temp[m] = -1;

}

printf("Page Reference Sequence: ");

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

printf("%d ", incomingStream[i]);

}

printf("\n");

int m, n, s;

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("Frames at stage [%d]: ", m + 1);

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

if (temp[i] == -1) {

printf("- ");

} else {

printf("%d ", temp[i]);

}

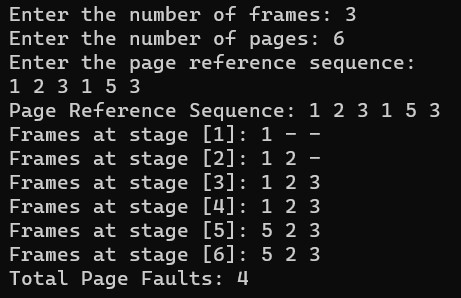
}

printf("\n");

}

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

return 0;

}

**LRU :**

#include <stdio.h>

#include <stdbool.h>

int main() {

int capacity, pages, pageFaults = 0;

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

scanf("%d", &capacity);

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

scanf("%d", &pages);

int incomingStream[pages];

int indexes[capacity];

int set[capacity];

bool pagePresent[capacity];

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

indexes[i] = -1;

pagePresent[i] = false;

}

printf("Enter the page reference sequence:\n");

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

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

}

printf("Page Reference Sequence: ");

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

printf("%d ", incomingStream[i]);

}

printf("\n");

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

if (pagePresent[incomingStream[i]]) {

// Page is already in memory, do nothing

} else {

if (pageFaults < capacity) {

int emptySlot = -1;

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

if (!pagePresent[j]) {

emptySlot = j;

break;

}

}

if (emptySlot != -1) {

set[emptySlot] = incomingStream[i];

pagePresent[emptySlot] = true;

indexes[incomingStream[i]] = i;

}

} else {

int minIndex = pages + 1;

int victimPage;

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

if (indexes[set[j]] < minIndex) {

minIndex = indexes[set[j]];

victimPage = j;

}

}

pagePresent[victimPage] = false;

set[victimPage] = incomingStream[i];

pagePresent[victimPage] = true;

indexes[incomingStream[i]] = i;

pageFaults++;

}

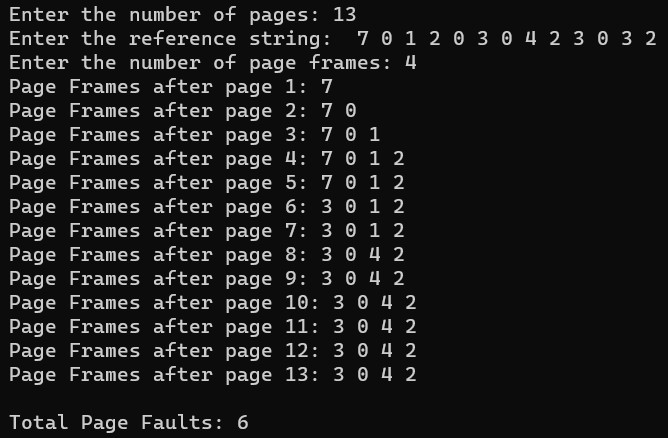
}

}

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

return 0;

}



**BOOKS AND WEB RESOURCES:**

* "Operating System Concepts" by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne

* "Operating Systems: Three Easy Pieces" by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

* GeeksforGeeks

* Tutorialspoint

**Conclusion:**

In this study, we implemented and compared FIFO and LRU page replacement algorithms in C language. Both algorithms were tested with the same reference string "1 2 3 4 1 2 5 1 2". The FIFO algorithm, replacing the oldest page, and the LRU algorithm, replacing the least recently used page, both resulted in 9 page faults. These findings highlight the importance of considering specific application requirements and access patterns when choosing the appropriate page replacement strategy.