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;
```

```
    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 {
```

```
    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++) {
```

```
        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[]) {
```

```
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++) {
```

```
        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

```
Enter the number of processes: 6
Enter burst time for process P1: 7
Enter burst time for process P2: 5
Enter burst time for process P3: 3
Enter burst time for process P4: 1
Enter burst time for process P5: 2
1Enter burst time for process P6: 1
Process Burst Time  Waiting Time    Turnaround Time
1   7       0       7
2   5       7       12
3   3       12      15
4   1       15      16
5   2       16      18
6   1       18      19

Gantt Chart:
| P1 | P2 | P3 | P4 | P5 | P6 |
0   7   12  15  16  18  19
Average Waiting Time: 11.33
```

Q2

```
Output                                                    Clear
/tmp/jrD3pDKawa.o
Enter the number of processes: 3
Enter the arrival time for process P1: 0
Enter the burst time for process P1: 9
Enter the priority for process P1: 2
Enter the arrival time for process P2: 0
Enter the burst time for process P2: 4
Enter the priority for process P2: 1
Enter the arrival time for process P3: 0
Enter the burst time for process P3: 9
Enter the priority for process P3: 3
Gantt Chart:
0->P2->22->P1->22->P3->22

TABLE
Process AT BT WT TAT
P2  0   4   0   4
P1  0   9   4   13
P3  0   9   13  22

Average Turnaround Time: 13.000000
Average Waiting Time: 5.666667
```

Q3

```
/tmp/fVfMYJJ7Ei.o
Enter the number of processes: 3
Enter arrival time for process P1: 0
Enter burst time for process P1: 9
Enter priority for process P1: 2
Enter arrival time for process P2: 0
Process Arrival Time    Burst Time  Priority    Waiting Time
    Turnaround Time
1   0       9       2       0       9
2   0       4       1       9       13
3   0       9       3       13      22

Average Waiting Time: 7.33
Average Turnaround Time: 14.67
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

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:

```
1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1
Producer produces item 1
Enter your choice:1
Producer produces item 2
Enter your choice:1
Producer produces item 3
Enter your choice:1
Producer produces item 4
Enter your choice:1
Producer produces item 5
Enter your choice:1
Buffer is full!
Enter your choice:2
Consumer consumes  item 5
Enter your choice:2
Consumer consumes  item 4
Enter your choice:2
Consumer consumes  item 3
Enter your choice:2
Consumer consumes  item 2
Enter your choice:2
Consumer consumes  item 1
Enter your choice:2
Buffer is empty!
Enter your choice:3
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

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

```
Sequence
P0 P3 P4 P1 P2
The process is safe
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

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 ;
}
```

Process No.	Process Size	Block no.
1	20	1
2	60	1
3	70	4
4	40	2

(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 ;
}
```

```
Process No.  Process Size    Block no.
1            40              2
2            10              3
3            30              5
4            60              1
```

(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;
}
```

```
/tmp/Jo1lQIxthb.o
Process No. Process Size    Block no.
1            40             4
2            10             1
3            30             2
4            60             Not Allocated
```

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;
```

}

```
Enter the number of frames: 3
Enter the number of pages: 6
Enter the page reference sequence:
1 2 3 1 5 3
Page Reference Sequence: 1 2 3 1 5 3
Frames at stage [1]: 1 - -
Frames at stage [2]: 1 2 -
Frames at stage [3]: 1 2 3
Frames at stage [4]: 1 2 3
Frames at stage [5]: 5 2 3
Frames at stage [6]: 5 2 3
Total Page Faults: 4
```

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;
}
```

```
Enter the number of pages: 13
Enter the reference string:  7 0 1 2 0 3 0 4 2 3 0 3 2
Enter the number of page frames: 4
Page Frames after page 1: 7
Page Frames after page 2: 7 0
Page Frames after page 3: 7 0 1
Page Frames after page 4: 7 0 1 2
Page Frames after page 5: 7 0 1 2
Page Frames after page 6: 3 0 1 2
Page Frames after page 7: 3 0 1 2
Page Frames after page 8: 3 0 4 2
Page Frames after page 9: 3 0 4 2
Page Frames after page 10: 3 0 4 2
Page Frames after page 11: 3 0 4 2
Page Frames after page 12: 3 0 4 2
Page Frames after page 13: 3 0 4 2

Total Page Faults: 6
```

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.