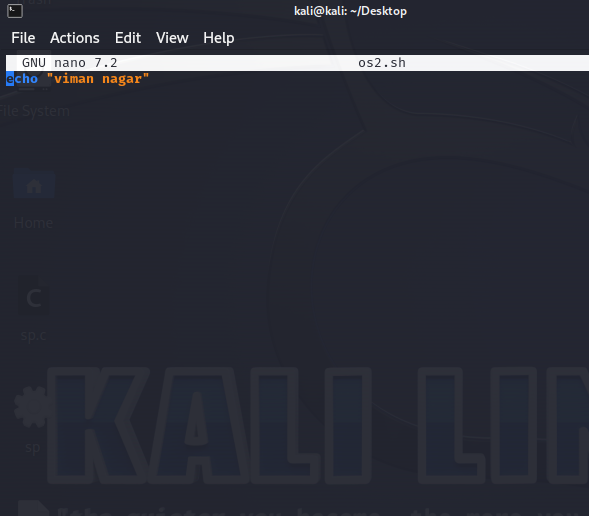
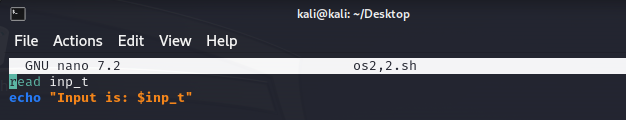
**Assignment No 2**

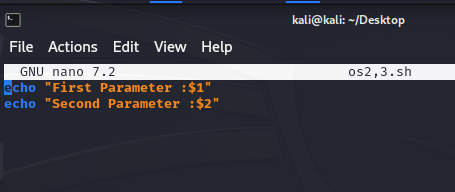
1. **Write a shell script to display your permanent address.**



1. **Write a shell script to take input from the user and display it.**



1. **Write a shell script to demonstrate use of command line arguments.**



1. **Write a shell scrpit to add two numbers where both the numbers are user inputs.**

echo "enter num1:$n1"

read n1

echo "enter num2:$n2"

read n2

sum=$((n1+n2))

echo" SUM = $sum "

1. **Write a shell script to check whether a number n is even or odd, where n is user input.**

echo "enter num: $num"

read num

r=`expr $num % 2`

if [ $r -eq 0 ]

then

echo"even number"

else

echo "odd number"

fi

1. **Write a shell script to input 10 numbers in array and find second largest number in it.**

#!/bin/bash

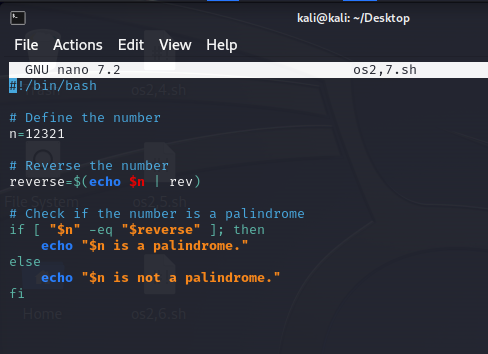
arr=(98 54 21 78 34 22 09 59 11 99)

sorted=($(for i in "${arr[@]}"; do echo $i; done | sort -n))

sec\_lar=${sorted[-1]}

echo "SECOND LARGEST : $sec\_lar"

**7.Write a shell script to check whether a number n is palindrome or not, where n is user input.**



**FOR STRING:**

#!/bin/bash

# Define the string

str="madam"

# Reverse the string

reverse=$(echo "$str" | rev)

# Check if the string is a palindrome

if [ "$str" = "$reverse" ]; then

echo "\"$str\" is a palindrome."

else

echo "\"$str\" is not a palindrome."

Fi

**8.Write a shell script to check grade of a student as follows:**

**Total marks of the student=100**

**If student ‘s mark>= 90,grade is O .**

**If student ‘s mark>=80 and marks<90, grade is A.**

**If student ‘s mark>=70 and marks<80, grade is B.**

**If student ‘s mark>= 60 and marks<70, grade is C.**

**If student ‘s mark>= 50 and marks<60, grade is E.**

**If student ‘s mark < 50, grade is F.**

m=85

if [ $m -ge 90 ]; then

g="A"

elif [ $m -ge 80 ]; then

g="B"

elif [ $m -ge 70 ]; then

g="C"

elif [ $m -ge 60 ]; then

g="D"

elif [ $m -ge 50 ]; then

g="E"

else

g="F"

fi

echo "GRADE FOR $m IS : $g"

**9.Write a shell script to check if input character is vowel or not using case control statement.**

echo "enter character(a-z):"

read char

case $char in

a|e|i|o|u|A|E|I|O|U)

echo" $char is a vowel"

;;

\*)

echo" $char is not a vowel"

;;

esac

**10.Write a shell script to sort an a;rray of n element using bubble sort, where n is user input.**

#!/bin/basH

echo "enter no of elements: "

read n

numbers=()

echo "ENTER THE ELEMENTS: "

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

read number

numbers+=($number)

done

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

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

if (( ${numbers[j]} > ${numbers[j+1]} )); then

temp=${numbers[j]}

numbers[j]=${numbers[j+1]}

numbers[j+1]=$temp

fi

done

done

echo "SORTED ARRAY: ${numbers[@]}"

**11.Write a shell script to search an element from an array of n elements where n is user input.**

#!/bin/bash

echo "Enter no of elements:"

read n

numbers=()

echo "Enter the elements:"

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

read number

numbers+=($number)

done

echo "Enter the search element:"

read sele

found=0

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

if [ ${numbers[$i]} -eq $sele ]; then

echo "Element found at index $i"

found=1

break

fi

done

if [ $found -eq 0 ]; then

echo "Element not found"

fi

**12.Write a shell script to create a file and count number of lines, number of words and number of characters from the file.**

echo "enter the name of file to create:"

read fn

touch $fn

echo "file created.."

echo "enter content in it:(press ctrl+D)"

cat > $fn

nl=$(wc -l < $fn)

nw=$(wc -w < $fn)

nc=$(wc -m < $fn)

echo "number of lines: $nl"

echo "number of words: $nw"

echo "number of characters: $nc"

**13.Write a shell script to search a pattern from a file, where filename is user input.**

echo "enter filename"

read fn

echo "enter pattern to search:"

read pat

grep "$pat" "$fn"

if [ $? -eq 0]; then

echo "no pattern matches"

fi

**Assignment 3**

**Input.txt -**

This is the first line.

Here is the second line.

The third line contains some text.

Another line for testing.

The last line in the file.

**Marks.txt -**

StudentID,StudentName,Telugu,English,Maths,Science,Social

1,John Doe,45,78,65,88,75

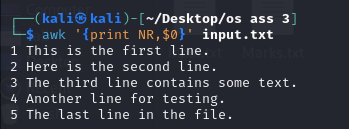
2,Jane Smith,30,45,50,60,40

3,Alice Johnson,75,80,90,88,70

4,Bob Wilson,25,40,35,55,60

5,Charlie Brown,60,70,75,80,85

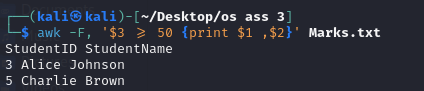
1. **Write an awk command to print the lines and line number in the given input file.**



 NR: Holds the current line number.

 $0: Refers to the entire line.

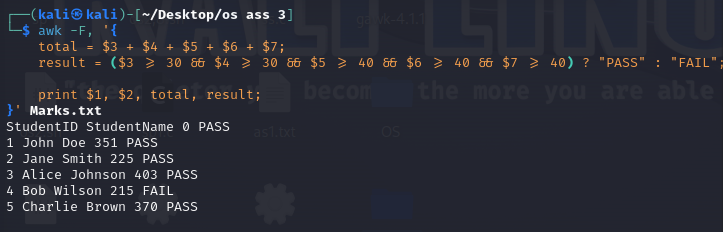
1. **Write an awk command to print first field and second field only if third field value is >=50 in the given input file. (Input field separator is “,” and output field separator is “,”)**



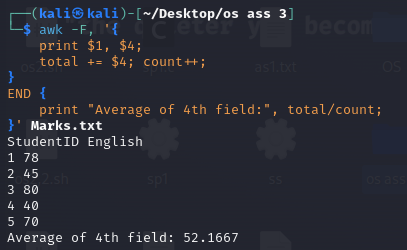
 -F,: Sets the field separator to a comma.

 $3: Refers to the third field.

**3.Consider the marks.txt is a file that contains one record per line( comma separate fields) of the student data in the form of studentid, student name, Telugu marks, English marks, Maths Marks, Science marks, Social Marks. Write an awk script to generate result for every students in the form of studentid, studentname, Total Marks and result. Result is PASS if marks is >=30 in TELUGU and English, and if marks>=40 in other subjects. Result is fail otherwise.**



**4. Write an awk program to print the fields 1 and 4 of a file that is passed as command line argument. The file contains lines of information that is separated by “,” as delimeter. The awk program must print at the end the average of all 4th field data.**



**5. Write an awk program to demonstrate user defined functions and system command.**

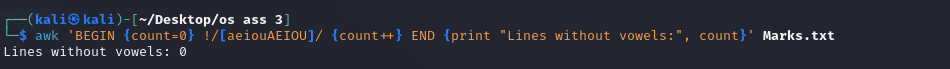


 greet(): A user-defined function.

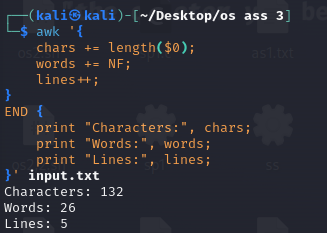
 system(): Executes a system command (here, printing the current date).

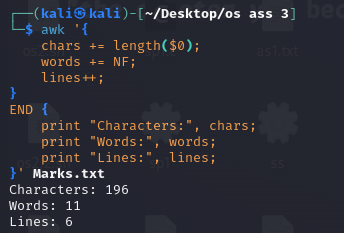
**6. Write an awk script to count the number of lines in a file that do not contain vowels.**





**7. Write an awk script to find the number of characters, words and lines in a file.**





**Assignment 4**

**Q.Implement Reader-Writer Problem using:**

a. using threads and semaphores.

b. using threads and mutex.

1. **using threads and semaphores:**

#!/bin/bash

# Create a temporary file to act as a lock

LOCK\_FILE="/tmp/reader\_writer.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_READS=5

MAX\_WRITES=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

reader() {

local id=$1

for ((i = 1; i <= MAX\_READS; i++)); do

(

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Reader $id: read data = $data (iteration $i)"

) 200<$LOCK\_FILE

sleep 1

done

}

# Function for writer

writer() {

local id=$1

for ((i = 1; i <= MAX\_WRITES; i++)); do

(

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Writer $id: updated data to $data (iteration $i)"

) 200>$LOCK\_FILE

sleep 2

done

}

# Start readers and writers

for i in {1..3}; do

reader $i &

done

for i in {1..2}; do

writer $i &

done

# Wait for all background processes to finish

wait

**b. using threads and mutex:**

#!/bin/bash

# Create a temporary lock file

MUTEX="/tmp/mutex.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_READS=5

MAX\_WRITES=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

reader() {

local id=$1

for ((i = 1; i <= MAX\_READS; i++)); do

# Readers only acquire a shared lock

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Reader $id: read data = $data (iteration $i)"

flock -u 200

sleep 1

done 200<"$MUTEX"

}

# Function for writer

writer() {

local id=$1

for ((i = 1; i <= MAX\_WRITES; i++)); do

# Writers need exclusive access

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Writer $id: updated data to $data (iteration $i)"

flock -u 200

sleep 2

done 200>"$MUTEX"

}

# Start readers and writers

for i in {1..3}; do

reader $i &

done

for i in {1..2}; do

writer $i &

done

# Wait for all background processes to finish

wait

**ASSIGNMENT-5**

IN C:

Q.Solve the Producers-Consumers problem:

a. using threads and semaphores -

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#include <stdlib.h> // Include for rand()

#define BUFFER\_SIZE 5

#define NUM\_ITEMS 10 // Define the number of items to produce/consume

int buffer[BUFFER\_SIZE];

int count = 0;

sem\_t empty;

sem\_t full;

pthread\_mutex\_t mutex;

void \*producer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Produce only 10 items

sleep(1);

item = rand() % 100;

sem\_wait(&empty); // Wait if buffer is full, Waits for an empty slot

pthread\_mutex\_lock(&mutex);

// Produce item

buffer[count++] = item;

printf("Producer produced: %d\n", item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&full); // Signal that buffer is not empty

}

return NULL;

}

void \*consumer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Consume only 10 items

sleep(1);

sem\_wait(&full); // Wait if buffer is empty, Waits for a filled slot

pthread\_mutex\_lock(&mutex);

// Consume item

item = buffer[--count];

printf("Consumer consumed: %d\n", item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty); // Signal that buffer is not full

}

return NULL;

}

int main() {

pthread\_t prod, cons;

sem\_init(&empty, 0, BUFFER\_SIZE); // Initialize semaphore to the size of buffer

sem\_init(&full, 0, 0); // Initialize semaphore to 0 (no items initially)

pthread\_mutex\_init(&mutex, NULL); // Initialize mutex

// Create producer and consumer threads

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

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

// Wait for both threads to complete

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

// Clean up resources

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

b. using threads and mutex-

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <stdlib.h> // Include for rand()

#define BUFFER\_SIZE 5

#define NUM\_ITEMS 10 // Number of items to produce/consume

int buffer[BUFFER\_SIZE];

int count = 0; // Tracks the number of items in the buffer

pthread\_mutex\_t mutex; // Mutex to protect shared buffer

void \*producer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Produce only 10 items

sleep(1);

item = rand() % 100; // Generate a random item

// Critical section

pthread\_mutex\_lock(&mutex); // Lock the mutex before accessing the buffer

if (count < BUFFER\_SIZE) { // Only produce if buffer is not full

buffer[count++] = item;

printf("Producer produced: %d\n", item);

} else {

printf("Buffer is full, producer is waiting...\n");

}

pthread\_mutex\_unlock(&mutex); // Unlock the mutex after producing the item

}

return NULL;

}

void \*consumer(void \*param) {

int item;

for (int i = 0; i < NUM\_ITEMS; i++) { // Consume only 10 items

sleep(1);

// Critical section

pthread\_mutex\_lock(&mutex); // Lock the mutex before accessing the buffer

if (count > 0) { // Only consume if buffer is not empty

item = buffer[--count];

printf("Consumer consumed: %d\n", item);

} else {

printf("Buffer is empty, consumer is waiting...\n");

}

pthread\_mutex\_unlock(&mutex); // Unlock the mutex after consuming the item

}

return NULL;

}

int main() {

pthread\_t prod, cons;

// Initialize the mutex

pthread\_mutex\_init(&mutex, NULL);

// Create producer and consumer threads

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

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

// Wait for both threads to finish

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

// Destroy the mutex

pthread\_mutex\_destroy(&mutex);

return 0;

}

**In Shell-**

**A:** using threads and semaphores -

#!/bin/bash

# Create a temporary file to act as a lock

LOCK\_FILE="/tmp/con\_pro.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_CON=5

MAX\_PRO=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

consumer() {

local id=$1

for ((i = 1; i <= MAX\_CON; i++)); do

(

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Consumer $id: consumed = $data (iteration $i)"

) 200<$LOCK\_FILE

sleep 1

done

}

# Function for writer

producer() {

local id=$1

for ((i = 1; i <= MAX\_PRO; i++)); do

(

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Producer $id: produced = $data (iteration $i)"

) 200>$LOCK\_FILE

sleep 2

done

}

# Start readers and writers

for i in {1..3}; do

consumer $i &

done

for i in {1..2}; do

producer $i &

done

# Wait for all background processes to finish

Wait

**B:** using threads and mutex-

#!/bin/bash

# Create a temporary lock file

MUTEX="/tmp/mutex.lock"

DATA\_FILE="/tmp/shared\_data.txt"

MAX\_CON=5

MAX\_PRO=3

# Initialize shared data file

echo "0" > "$DATA\_FILE"

# Function for reader

consumer() {

local id=$1

for ((i = 1; i <= MAX\_CON; i++)); do

# Readers only acquire a shared lock

flock -s 200

local data

data=$(<"$DATA\_FILE")

echo "Consumer $id: consumed = $data (iteration $i)"

flock -u 200

sleep 1

done 200<"$MUTEX"

}

# Function for writer

producer() {

local id=$1

for ((i = 1; i <= MAX\_PRO; i++)); do

# Writers need exclusive access

flock -x 200

local data

data=$(<"$DATA\_FILE")

data=$((data + 1))

echo "$data" > "$DATA\_FILE"

echo "Producer $id: produced to $data (iteration $i)"

flock -u 200

sleep 2

done 200>"$MUTEX"

}

# Start readers and writers

for i in {1..3}; do

consumer $i &

done

for i in {1..2}; do

producer $i &

done

# Wait for all background processes to finish

wait

**ASSIGNMENT-6**

Q.Implement Banker’s Safety algorithm for Deadlock Avoidance.

CODE-

#include<stdio.h>

int main() {

int p, c, count = 0, i, j;

int alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5] = {0};

printf("Enter the number of processes and resources: ");

scanf("%d %d", &p, &c);

printf("Enter the allocation matrix (%dx%d):\n", p, c);

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

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

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

printf("Enter the max matrix (%dx%d):\n", p, c);

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

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

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

printf("Enter the available resources: ");

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

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

printf("\nNeed matrix:\n");

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

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

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

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

}

printf("\n");

}

while (count < p) {

int found = 0;

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

if (!done[i]) {

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

if (need[i][j] > available[j])

break;

if (j == c) {

safe[count++] = i;

done[i] = 1;

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

available[j] += alc[i][j];

found = 1;

}

}

}

if (!found) {

printf("Safe sequence does not exist.\n");

return 0;

}

}

printf("\nAvailable resources after completion:\n");

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

printf("%d\t", available[i]);

printf("\nSafe sequence:\n");

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

printf("P%d\t", safe[i]);

return 0;

}

OUTPUT-

Enter the number of process and resources

5 3

enter allocation of resource of all process 5x3 matrix

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

enter the max resource process required 5x3 matrix

7 5 3

3 2 2

9 0 2

4 2 2

5 3 3

enter the available resource 3 3 2

need resources matrix are

7 4 3

1 2 2

6 0 0

2 1 1

5 3 1

available resource after completion

10 5 7

safe sequence are

p1 p3 p4 p0 p2

**ASSIGNMENT-7**

Q.Simulate the following CPU scheduling algorithms:  
a. First come First serve b. Shortest Job First (Non-preemptive) c. Shortest Job First (Preemptive) d. Round Robin e. Priority (Non-preemptive) f. Priority (Non-preemptive)

(l)

ALL IN ONE CODE-

import matplotlib.pyplot as plt

import numpy as np

from collections import deque

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time, priority=0):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.remaining\_time = burst\_time

self.priority = priority

self.start\_time = 0

self.completion\_time = 0

self.waiting\_time = 0

self.turnaround\_time = 0

class CPUScheduler:

def \_\_init\_\_(self):

self.processes = []

self.gantt\_chart = []

def add\_process(self, pid, arrival\_time, burst\_time, priority=0):

self.processes.append(Process(pid, arrival\_time, burst\_time, priority))

def calculate\_times(self):

avg\_waiting\_time = sum(p.waiting\_time for p in self.processes) / len(self.processes)

avg\_turnaround\_time = sum(p.turnaround\_time for p in self.processes) / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for i, (pid, start, end) in enumerate(self.gantt\_chart):

ax.barh(0, end - start, left=start, height=0.3,

align='center', color=f'C{pid}', alpha=0.8)

ax.text((start + end)/2, 0, f'P{pid}',

ha='center', va='center')

ax.set\_ylim(-0.5, 0.5)

ax.set\_xlabel('Time')

ax.set\_yticks([])

ax.set\_title(title)

plt.grid(True)

plt.show()

def fcfs(self):

self.gantt\_chart = []

processes = sorted(self.processes, key=lambda x: (x.arrival\_time, x.pid))

current\_time = 0

for process in processes:

if current\_time < process.arrival\_time:

current\_time = process.arrival\_time

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

return self.calculate\_times()

def sjf\_non\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.burst\_time, x.arrival\_time))

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

remaining\_processes.remove(process)

return self.calculate\_times()

def sjf\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

current\_process = None

for p in remaining\_processes:

p.remaining\_time = p.burst\_time

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.remaining\_time, x.arrival\_time))

if current\_process != process.pid:

if current\_process is not None:

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = process.pid

start\_time = current\_time

process.remaining\_time -= 1

current\_time += 1

if process.remaining\_time == 0:

process.completion\_time = current\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

remaining\_processes.remove(process)

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = None

return self.calculate\_times()

def round\_robin(self, quantum):

self.gantt\_chart = []

remaining\_processes = deque(sorted(self.processes, key=lambda x: x.arrival\_time))

current\_time = 0

for p in remaining\_processes:

p.remaining\_time = p.burst\_time

while remaining\_processes:

if not remaining\_processes:

break

process = remaining\_processes.popleft()

if process.arrival\_time > current\_time:

current\_time = process.arrival\_time

execution\_time = min(quantum, process.remaining\_time)

self.gantt\_chart.append((process.pid, current\_time,

current\_time + execution\_time))

current\_time += execution\_time

process.remaining\_time -= execution\_time

if process.remaining\_time > 0:

# Find the position to insert the process back

insert\_pos = 0

for i, p in enumerate(remaining\_processes):

if p.arrival\_time > current\_time:

insert\_pos = i

break

insert\_pos = i + 1

remaining\_processes.insert(insert\_pos, process)

else:

process.completion\_time = current\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

return self.calculate\_times()

def priority\_non\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.priority, x.arrival\_time))

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

remaining\_processes.remove(process)

return self.calculate\_times()

def priority\_preemptive(self):

self.gantt\_chart = []

remaining\_processes = self.processes.copy()

current\_time = 0

current\_process = None

for p in remaining\_processes:

p.remaining\_time = p.burst\_time

while remaining\_processes:

available\_processes = [p for p in remaining\_processes

if p.arrival\_time <= current\_time]

if not available\_processes:

current\_time += 1

continue

process = min(available\_processes, key=lambda x: (x.priority, x.arrival\_time))

if current\_process != process.pid:

if current\_process is not None:

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = process.pid

start\_time = current\_time

process.remaining\_time -= 1

current\_time += 1

if process.remaining\_time == 0:

process.completion\_time = current\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

remaining\_processes.remove(process)

self.gantt\_chart.append((current\_process, start\_time, current\_time))

current\_process = None

return self.calculate\_times()

# Example usage and testing

if \_\_name\_\_ == "\_\_main\_\_":

# Test data

processes\_data = [

(1, 0, 6, 2), # (pid, arrival\_time, burst\_time, priority)

(2, 1, 4, 4),

(3, 2, 2, 1),

(4, 3, 3, 3)

]

algorithms = [

("FCFS", lambda s: s.fcfs()),

("SJF (Non-preemptive)", lambda s: s.sjf\_non\_preemptive()),

("SJF (Preemptive)", lambda s: s.sjf\_preemptive()),

("Round Robin (Q=2)", lambda s: s.round\_robin(2)),

("Priority (Non-preemptive)", lambda s: s.priority\_non\_preemptive()),

("Priority (Preemptive)", lambda s: s.priority\_preemptive())

]

for algo\_name, algo\_func in algorithms:

scheduler = CPUScheduler()

# Add processes

for pid, arrival\_time, burst\_time, priority in processes\_data:

scheduler.add\_process(pid, arrival\_time, burst\_time, priority)

# Run algorithm

avg\_waiting\_time, avg\_turnaround\_time = algo\_func(scheduler)

# Print results

print(f"\n{algo\_name} Results:")

print(f"Average Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

# Plot Gantt chart

scheduler.plot\_gantt\_chart(f"{algo\_name} - Gantt Chart")

**SINGLE ONE’s-**

1. **First come First serve**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.start\_time = 0

self.completion\_time = 0

self.waiting\_time = 0

self.turnaround\_time = 0

class CPUSchedulerFCFS:

def \_\_init\_\_(self):

self.processes = []

self.gantt\_chart = []

def add\_process(self, pid, arrival\_time, burst\_time):

self.processes.append(Process(pid, arrival\_time, burst\_time))

def fcfs(self):

self.gantt\_chart = []

processes = sorted(self.processes, key=lambda x: (x.arrival\_time, x.pid))

current\_time = 0

for process in processes:

if current\_time < process.arrival\_time:

current\_time = process.arrival\_time

process.start\_time = current\_time

process.completion\_time = current\_time + process.burst\_time

process.turnaround\_time = process.completion\_time - process.arrival\_time

process.waiting\_time = process.turnaround\_time - process.burst\_time

self.gantt\_chart.append((process.pid, current\_time, process.completion\_time))

current\_time = process.completion\_time

return self.calculate\_times()

def calculate\_times(self):

avg\_waiting\_time = sum(p.waiting\_time for p in self.processes) / len(self.processes)

avg\_turnaround\_time = sum(p.turnaround\_time for p in self.processes) / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for i, (pid, start, end) in enumerate(self.gantt\_chart):

ax.barh(0, end - start, left=start, height=0.3, color=f'C{pid}', alpha=0.8)

ax.text((start + end) / 2, 0, f'P{pid}', ha='center', va='center')

ax.set\_ylim(-0.5, 0.5)

ax.set\_xlabel('Time')

ax.set\_yticks([])

ax.set\_title(title)

plt.grid(True)

plt.show()

# Example usage

scheduler = CPUSchedulerFCFS()

scheduler.add\_process(1, 0, 6)

scheduler.add\_process(2, 1, 4)

scheduler.add\_process(3, 2, 2)

scheduler.add\_process(4, 3, 3)

avg\_waiting\_time, avg\_turnaround\_time = scheduler.fcfs()

print(f"Average Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("FCFS - Gantt Chart")

IN C:

#include<stdio.h>

int main() {

int i, j, temp;

int bt[10] = {0}, at[10] = {0}, tat[10] = {0}, wt[10] = {0}, ct[10] = {0};

int n;

float totalTAT = 0, totalWT = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

int p[n];

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

p[i] = i + 1;

}

printf("Enter arrival time and burst time for each process\n\n");

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

printf("Arrival time of process[%d]: ", i + 1);

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

printf("Burst time of process[%d]: ", i + 1);

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

printf("\n");

}

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

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

if (at[i] > at[j]) {

temp = at[i];

at[i] = at[j];

at[j] = temp;

temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

int currentTime = 0;

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

if (currentTime < at[i]) {

currentTime = at[i];

}

ct[i] = currentTime + bt[i];

currentTime = ct[i];

tat[i] = ct[i] - at[i];

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

totalTAT += tat[i];

totalWT += wt[i];

}

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

printf(" |");

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

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

}

printf("\n");

printf("0");

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

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

}

printf("\n\n");

printf("Solution: \n\n");

printf("P#\t AT\t BT\t CT\t TAT\t WT\n");

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

printf("P%d\t %d\t %d\t %d\t %d\t %d\n", p[i], at[i], bt[i], ct[i], tat[i], wt[i]);

}

printf("\nAverage Turnaround Time = %.2f\n", totalTAT / n);

printf("Average Waiting Time = %.2f\n\n", totalWT / n);

return 0;

}

1. **Shortest Job First (Non-preemptive)**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

class SJFScheduler:

def \_\_init\_\_(self, processes):

self.processes = processes

self.gantt\_chart = []

def schedule(self):

# Sort processes by arrival time, then by burst time

self.processes.sort(key=lambda p: (p.arrival\_time, p.burst\_time))

time = 0

completed\_processes = 0

while completed\_processes < len(self.processes):

# Filter processes that have arrived and are not completed

available\_processes = [p for p in self.processes if p.arrival\_time <= time and p.completion\_time == 0]

if not available\_processes:

time += 1

continue

# Select the process with the shortest burst time

shortest\_job = min(available\_processes, key=lambda p: p.burst\_time)

start\_time = time

time += shortest\_job.burst\_time

shortest\_job.completion\_time = time

shortest\_job.turnaround\_time = shortest\_job.completion\_time - shortest\_job.arrival\_time

shortest\_job.waiting\_time = shortest\_job.turnaround\_time - shortest\_job.burst\_time

# Add to Gantt chart

self.gantt\_chart.append((shortest\_job.pid, start\_time, time))

completed\_processes += 1

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data

processes = [

Process(pid=1, arrival\_time=0, burst\_time=7),

Process(pid=2, arrival\_time=2, burst\_time=4),

Process(pid=3, arrival\_time=4, burst\_time=1),

Process(pid=4, arrival\_time=5, burst\_time=4)

]

scheduler = SJFScheduler(processes)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("SJF Non-preemptive - Gantt Chart")

IN C:

#include <stdio.h>

int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Burst Time:\n");

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

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

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

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

index = i;

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

if (A[j][1] < A[index][1]) {

index = j;

}

}

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

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

A[i][2] = 0;

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

A[i][2] += A[j][1];

}

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

printf("\nP\tBT\tWT\tTAT\n");

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

A[i][3] = A[i][1] + A[i][2]; // TAT = BT + WT

total += A[i][3];

printf("P%d\t%d\t%d\t%d\n", A[i][0], A[i][1], A[i][2], A[i][3]);

}

avg\_tat = (float)total / n;

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

printf("|");

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

printf(" P%d |", A[i][0]);

}

printf("\n");

int completion\_time = 0;

printf("0");

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

completion\_time += A[i][1];

printf(" %d", completion\_time);

}

printf("\n\nAverage Waiting Time = %.2f", avg\_wt);

printf("\nAverage Turnaround Time = %.2f\n", avg\_tat);

return 0;

}

1. **Shortest Job First (Preemptive)**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.remaining\_time = burst\_time

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

self.start\_time = -1 # To track when the process started execution

class SJFPreemptiveScheduler:

def \_\_init\_\_(self, processes):

self.processes = processes

self.gantt\_chart = []

def schedule(self):

time = 0

completed\_processes = 0

processes = sorted(self.processes, key=lambda x: x.arrival\_time) # Sort by arrival time

ready\_queue = []

current\_process = None

last\_time = 0

while completed\_processes < len(processes):

# Add all processes that have arrived by the current time to the ready queue

for p in processes:

if p.arrival\_time <= time and p not in ready\_queue and p.remaining\_time > 0:

ready\_queue.append(p)

# If there's a process in the ready queue, choose the one with the shortest remaining time

if ready\_queue:

ready\_queue.sort(key=lambda p: p.remaining\_time)

current\_process = ready\_queue[0]

# If the process has not started, set its start time

if current\_process.start\_time == -1:

current\_process.start\_time = time

current\_process.remaining\_time -= 1

time += 1

# If the process has finished execution

if current\_process.remaining\_time == 0:

current\_process.completion\_time = time

current\_process.turnaround\_time = current\_process.completion\_time - current\_process.arrival\_time

current\_process.waiting\_time = current\_process.turnaround\_time - current\_process.burst\_time

completed\_processes += 1

ready\_queue.remove(current\_process)

self.gantt\_chart.append((current\_process.pid, last\_time, time))

else:

time += 1 # If no process is in the ready queue, increment time

return time

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data

processes = [

Process(pid=1, arrival\_time=0, burst\_time=7),

Process(pid=2, arrival\_time=2, burst\_time=4),

Process(pid=3, arrival\_time=4, burst\_time=1),

Process(pid=4, arrival\_time=5, burst\_time=4)

]

scheduler = SJFPreemptiveScheduler(processes)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("SJF Preemptive (SRTF) - Gantt Chart")

IN C:

#include <stdio.h>

#include <limits.h>

struct Process {

int pid;

int bt;

int art;

};

void findWaitingTime(struct Process proc[], int n, int wt[]) {

int rt[n];

int i,j;

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

int check = 0;

while (complete != n) {

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

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = 1;

}

}

if (check == 0) {

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

if (rt[shortest] == 0) {

complete++;

check = 0;

finish\_time = t + 1;

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

t++;

}

}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {

int i;

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

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

}

void findavgTime(struct Process proc[], int n) {

int i;

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

findWaitingTime(proc, n, wt);

findTurnAroundTime(proc, n, wt, tat);

printf(" P\t\t"

"BT\t\t"

"WT\t\t"

"TAT\t\t\n");

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

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

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

printf(" %d\t\t"

"%d\t\t %d"

"\t\t %d\n", proc[i].pid,

proc[i].bt, wt[i], tat[i]);

}

printf("\nAverage waiting time = "

"%f", (float)total\_wt / (float)n);

printf("\nAverage turn around time = "

"%f", (float)total\_tat / (float)n);

}

int main() {

struct Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },

{ 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

1. **Round Robin**

import matplotlib.pyplot as plt

from collections import deque

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.remaining\_time = burst\_time

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

self.start\_time = -1 # To track when the process started execution

class RoundRobinScheduler:

def \_\_init\_\_(self, processes, time\_quantum):

self.processes = processes

self.time\_quantum = time\_quantum

self.gantt\_chart = []

def schedule(self):

time = 0

completed\_processes = 0

processes = sorted(self.processes, key=lambda x: x.arrival\_time) # Sort by arrival time

ready\_queue = deque()

last\_time = 0

while completed\_processes < len(processes):

# Add all processes that have arrived by the current time to the ready queue

for p in processes:

if p.arrival\_time <= time and p not in ready\_queue and p.remaining\_time > 0:

ready\_queue.append(p)

# If there's a process in the ready queue, select the next process to execute

if ready\_queue:

current\_process = ready\_queue.popleft()

# If the process has not started, set its start time

if current\_process.start\_time == -1:

current\_process.start\_time = time

# Calculate the time to execute the process

exec\_time = min(current\_process.remaining\_time, self.time\_quantum)

current\_process.remaining\_time -= exec\_time

time += exec\_time

# If the process has finished execution

if current\_process.remaining\_time == 0:

current\_process.completion\_time = time

current\_process.turnaround\_time = current\_process.completion\_time - current\_process.arrival\_time

current\_process.waiting\_time = current\_process.turnaround\_time - current\_process.burst\_time

completed\_processes += 1

self.gantt\_chart.append((current\_process.pid, last\_time, time))

else:

ready\_queue.append(current\_process)

last\_time = time

else:

time += 1 # If no process is in the ready queue, increment time

return time

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data

processes = [

Process(pid=1, arrival\_time=0, burst\_time=5),

Process(pid=2, arrival\_time=1, burst\_time=3),

Process(pid=3, arrival\_time=2, burst\_time=8),

Process(pid=4, arrival\_time=3, burst\_time=6)

]

time\_quantum = 4 # Set time quantum for Round Robin

scheduler = RoundRobinScheduler(processes, time\_quantum)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("Round Robin Scheduling - Gantt Chart")

IN C:

#include<stdio.h>

#include<conio.h>

void main()

{

int i, NOP, sum = 0, count = 0, y, quant, wt = 0, tat = 0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

int gantt[100], gantt\_time[100], gantt\_index = 0;

printf("Total number of processes in the system: ");

scanf("%d", &NOP);

y = NOP;

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

{

printf("\nEnter the Arrival and Burst time of Process[%d]\n", i + 1);

printf("Arrival time: ");

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

printf("Burst time: ");

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

temp[i] = bt[i];

}

printf("Enter the Time Quantum for the processes: ");

scanf("%d", &quant);

printf("\nProcess No\tBurst Time\tTurnaround Time\tWaiting Time");

for(sum = 0, i = 0; y != 0;)

{

if(temp[i] <= quant && temp[i] > 0)

{

sum += temp[i];

temp[i] = 0;

count = 1;

gantt[gantt\_index] = i + 1;

gantt\_time[gantt\_index] = sum;

gantt\_index++;

}

else if(temp[i] > 0)

{

temp[i] -= quant;

sum += quant;

gantt[gantt\_index] = i + 1;

gantt\_time[gantt\_index] = sum;

gantt\_index++;

}

if(temp[i] == 0 && count == 1)

{

y--;

printf("\nProcess[%d]\t\t%d\t\t%d\t\t%d", i + 1, bt[i], sum - at[i], sum - at[i] - bt[i]);

wt += sum - at[i] - bt[i];

tat += sum - at[i];

count = 0;

}

if(i == NOP - 1)

{

i = 0;

}

else if(at[i + 1] <= sum)

{

i++;

}

else

{

i = 0;

}

}

avg\_wt = (float)wt / NOP;

avg\_tat = (float)tat / NOP;

printf("\nAverage Turnaround Time: %.2f", avg\_tat);

printf("\nAverage Waiting Time: %.2f\n", avg\_wt);

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

for(i = 0; i < gantt\_index; i++)

{

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

}

printf("\n0");

for(i = 0; i < gantt\_index; i++)

{

printf(" %d", gantt\_time[i]);

}

getch();

}

1. **Priority (Non-preemptive)**

import matplotlib.pyplot as plt

class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time, priority):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.priority = priority

self.completion\_time = 0

self.turnaround\_time = 0

self.waiting\_time = 0

self.start\_time = -1 # To track when the process started execution

class PriorityScheduler:

def \_\_init\_\_(self, processes):

self.processes = processes

self.gantt\_chart = []

def schedule(self):

time = 0

completed\_processes = 0

processes = sorted(self.processes, key=lambda x: (x.arrival\_time, x.priority)) # Sort by arrival time and priority

ready\_queue = []

last\_time = 0

while completed\_processes < len(processes):

# Add processes that have arrived by the current time to the ready queue

for p in processes:

if p.arrival\_time <= time and p not in ready\_queue:

ready\_queue.append(p)

# If there's a process in the ready queue, select the one with the highest priority

if ready\_queue:

ready\_queue.sort(key=lambda p: p.priority) # Sort by priority

current\_process = ready\_queue.pop(0) # Process with highest priority

# If the process has not started, set its start time

if current\_process.start\_time == -1:

current\_process.start\_time = time

# Process executes to completion

current\_process.completion\_time = time + current\_process.burst\_time

current\_process.turnaround\_time = current\_process.completion\_time - current\_process.arrival\_time

current\_process.waiting\_time = current\_process.turnaround\_time - current\_process.burst\_time

time = current\_process.completion\_time # Update time to when the current process finishes

completed\_processes += 1

self.gantt\_chart.append((current\_process.pid, last\_time, time))

else:

time += 1 # If no process is in the ready queue, increment time

last\_time = time

return time

def calculate\_averages(self):

total\_waiting\_time = sum(p.waiting\_time for p in self.processes)

total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)

avg\_waiting\_time = total\_waiting\_time / len(self.processes)

avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)

return avg\_waiting\_time, avg\_turnaround\_time

def plot\_gantt\_chart(self, title):

fig, ax = plt.subplots(figsize=(12, 4))

for pid, start, end in self.gantt\_chart:

ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)

ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')

ax.set\_title(title)

ax.set\_xlabel("Time")

ax.set\_yticks([])

ax.legend()

plt.show()

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input process data: (PID, Arrival Time, Burst Time, Priority)

processes = [

Process(pid=1, arrival\_time=0, burst\_time=5, priority=2),

Process(pid=2, arrival\_time=1, burst\_time=3, priority=1),

Process(pid=3, arrival\_time=2, burst\_time=8, priority=4),

Process(pid=4, arrival\_time=3, burst\_time=6, priority=3)

]

scheduler = PriorityScheduler(processes)

scheduler.schedule()

avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()

print("PID\tAT\tBT\tPriority\tCT\tTAT\tWT")

for p in scheduler.processes:

print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.priority}\t\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")

print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")

print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")

scheduler.plot\_gantt\_chart("Priority Scheduling (Non-preemptive) - Gantt Chart")

1. **Priority (preemptive)**

def priority\_scheduling(processes):

n = len(processes)

processes.sort(key=lambda x: (x['arrival\_time'], x['priority'], x['pid']))

time = 0

completed = 0

gantt\_chart = []

ready\_queue = []

current\_process = None

while completed < n:

# Debugging: Display the time and process states

print(f"Time: {time}, Completed: {completed}, Ready Queue: {[p['pid'] for p in ready\_queue]}")

# Add processes to the ready queue based on arrival time

for process in processes:

if process['arrival\_time'] <= time and not process['is\_completed']:

if process not in ready\_queue:

ready\_queue.append(process)

# Check if the current process is completed

if current\_process and current\_process['burst\_time'] == 0:

current\_process['is\_completed'] = True

current\_process['finish\_time'] = time

current\_process['turnaround\_time'] = current\_process['finish\_time'] - current\_process['arrival\_time']

current\_process['waiting\_time'] = current\_process['turnaround\_time'] - current\_process['original\_burst\_time']

completed += 1

current\_process = None # Reset current process

# Select the next process to execute

if current\_process is None and ready\_queue:

ready\_queue.sort(key=lambda x: (x['priority'], x['arrival\_time'], x['pid']))

current\_process = ready\_queue.pop(0)

# Execute the current process or mark the CPU as idle

if current\_process:

gantt\_chart.append(current\_process['pid'])

current\_process['burst\_time'] -= 1

else:

gantt\_chart.append("Idle")

# Increment time

time += 1

# Safeguard: Prevent infinite loop by ensuring progress

if time > 100: # Timeout for debugging

print("ERROR: Timeout reached. Check the logic for infinite loops.")

break

# Optimize Gantt chart to remove consecutive duplicates

optimized\_gantt\_chart = [gantt\_chart[0]]

for i in range(1, len(gantt\_chart)):

if gantt\_chart[i] != gantt\_chart[i - 1]:

optimized\_gantt\_chart.append(gantt\_chart[i])

# Prepare the results for display

results = []

for p in processes:

results.append({

"PID": p["pid"],

"Priority": p["priority"],

"Arrival Time": p["arrival\_time"],

"Burst Time": p["original\_burst\_time"],

"Finish Time": p["finish\_time"],

"Turnaround Time": p["turnaround\_time"],

"Waiting Time": p["waiting\_time"],

})

return results, optimized\_gantt\_chart

# Input

process\_list = [

{"pid": "P1", "arrival\_time": 0, "burst\_time": 7, "priority": 2, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

{"pid": "P2", "arrival\_time": 2, "burst\_time": 4, "priority": 1, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

{"pid": "P3", "arrival\_time": 4, "burst\_time": 1, "priority": 3, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

{"pid": "P4", "arrival\_time": 5, "burst\_time": 4, "priority": 2, "is\_completed": False, "finish\_time": 0, "turnaround\_time": 0, "waiting\_time": 0},

]

# Store original burst time

for p in process\_list:

p["original\_burst\_time"] = p["burst\_time"]

# Run simulation

results, gantt\_chart = priority\_scheduling(process\_list)

# Display Results

print("\nGantt Chart:", " -> ".join(gantt\_chart))

print("\nResults:")

for res in results:

print(res)

**ASSIGNMENT-8**

Q. Simulate the following page replacement algorithms-  
a. FIFO b. LRU c. OPT

ALL IN ONE CODE-

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <limits.h>

// Function to check if a page exists in frame

bool isPagePresent(int page, int\* frames, int frameCount) {

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

if (frames[i] == page)

return true;

}

return false;

}

// Function to find the position of the first occurrence of a page in future

int findOptimalPosition(int\* pages, int pageCount, int\* frames, int frameCount, int currentPos) {

int farthest = -1, replaceIndex = 0;

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

int j;

for (j = currentPos + 1; j < pageCount; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

replaceIndex = i;

}

break;

}

}

if (j == pageCount)

return i;

}

return (farthest == -1) ? 0 : replaceIndex;

}

// FIFO Page Replacement Algorithm

void fifo(int\* pages, int pageCount, int frameCount) {

int\* frames = (int\*)calloc(frameCount, sizeof(int));

int pageFaults = 0;

int currentIndex = 0;

printf("\nFIFO Page Replacement:\n");

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

printf("\nReference to page %d: ", pages[i]);

if (!isPagePresent(pages[i], frames, frameCount)) {

frames[currentIndex] = pages[i];

currentIndex = (currentIndex + 1) % frameCount;

pageFaults++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

}

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

if (frames[j] != 0)

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

else

printf("- ");

}

}

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

free(frames);

}

// LRU Page Replacement Algorithm

void lru(int\* pages, int pageCount, int frameCount) {

int\* frames = (int\*)calloc(frameCount, sizeof(int));

int\* lastUsed = (int\*)calloc(frameCount, sizeof(int));

int pageFaults = 0;

printf("\nLRU Page Replacement:\n");

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

printf("\nReference to page %d: ", pages[i]);

if (!isPagePresent(pages[i], frames, frameCount)) {

int replaceIndex = 0;

int leastUsed = INT\_MAX;

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

if (frames[j] == 0) {

replaceIndex = j;

break;

}

if (lastUsed[j] < leastUsed) {

leastUsed = lastUsed[j];

replaceIndex = j;

}

}

frames[replaceIndex] = pages[i];

lastUsed[replaceIndex] = i;

pageFaults++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

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

if (frames[j] == pages[i]) {

lastUsed[j] = i;

}

}

}

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

if (frames[j] != 0)

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

else

printf("- ");

}

}

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

free(frames);

free(lastUsed);

}

// Optimal Page Replacement Algorithm

void optimal(int\* pages, int pageCount, int frameCount) {

int\* frames = (int\*)calloc(frameCount, sizeof(int));

int pageFaults = 0;

printf("\nOptimal Page Replacement:\n");

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

printf("\nReference to page %d: ", pages[i]);

if (!isPagePresent(pages[i], frames, frameCount)) {

int replaceIndex;

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

if (frames[j] == 0) {

replaceIndex = j;

break;

}

}

if (frames[frameCount - 1] != 0) {

replaceIndex = findOptimalPosition(pages, pageCount, frames, frameCount, i);

}

frames[replaceIndex] = pages[i];

pageFaults++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

}

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

if (frames[j] != 0)

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

else

printf("- ");

}

}

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

free(frames);

}

int main() {

int pageCount, frameCount;

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

scanf("%d", &pageCount);

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

scanf("%d", &frameCount);

int\* pages = (int\*)malloc(pageCount \* sizeof(int));

printf("Enter the page reference string: ");

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

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

}

fifo(pages, pageCount, frameCount);

lru(pages, pageCount, frameCount);

optimal(pages, pageCount, frameCount);

free(pages);

return 0;

}

**SINGLE CODES-**

FIFO-

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Function to check if a page exists in frame

bool isPagePresent(int pg, int\* fs, int fC) {

int i;

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

if (fs[i] == pg)

return true;

}

return false;

}

// FIFO Page Replacement Algorithm

void fifo(int\* pgs, int pC, int fC) {

int i,j;

int\* fs = (int\*)calloc(fC, sizeof(int));

int pF = 0;

int currentIndex = 0;

printf("\nFIFO Page Replacement:\n");

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

printf("\nReference to page %d: ", pgs[i]);

if (!isPagePresent(pgs[i], fs, fC)) {

fs[currentIndex] = pgs[i];

currentIndex = (currentIndex + 1) % fC;

pF++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

}

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

if (fs[j] != 0)

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

else

printf("- ");

}

}

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

free(fs);

}

int main() {

int i;

int pC, fC;

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

scanf("%d", &pC);

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

scanf("%d", &fC);

int\* pgs = (int\*)malloc(pC \* sizeof(int));

printf("Enter the page reference string: ");

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

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

}

fifo(pgs, pC, fC);

free(pgs);

return 0;

}

LRU-

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <limits.h>

// Function to check if a page exists in frame

bool isPagePresent(int page, int\* frames, int frameCount) {

int i;

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

if (frames[i] == page)

return true;

}

return false;

}

// LRU Page Replacement Algorithm

void lru(int\* pages, int pageCount, int frameCount) {

int i,j;

int\* frames = (int\*)calloc(frameCount, sizeof(int));

int\* lastUsed = (int\*)calloc(frameCount, sizeof(int));

int pageFaults = 0;

printf("\nLRU Page Replacement:\n");

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

printf("\nReference to page %d: ", pages[i]);

if (!isPagePresent(pages[i], frames, frameCount)) {

int replaceIndex = 0;

int leastUsed = INT\_MAX;

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

if (frames[j] == 0) {

replaceIndex = j;

break;

}

if (lastUsed[j] < leastUsed) {

leastUsed = lastUsed[j];

replaceIndex = j;

}

}

frames[replaceIndex] = pages[i];

lastUsed[replaceIndex] = i;

pageFaults++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

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

if (frames[j] == pages[i]) {

lastUsed[j] = i;

}

}

}

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

if (frames[j] != 0)

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

else

printf("- ");

}

}

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

free(frames);

free(lastUsed);

}

int main() {

int i;

int pageCount, frameCount;

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

scanf("%d", &pageCount);

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

scanf("%d", &frameCount);

int\* pages = (int\*)malloc(pageCount \* sizeof(int));

printf("Enter the page reference string: ");

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

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

}

lru(pages, pageCount, frameCount);

free(pages);

return 0;

}

OPT-

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Function to check if a page exists in frame

bool isPagePresent(int page, int\* frames, int frameCount) {

int i;

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

if (frames[i] == page)

return true;

}

return false;

}

// Function to find the position of the first occurrence of a page in future

int findOptimalPosition(int\* pages, int pageCount, int\* frames, int frameCount, int currentPos) {

int i,j;

int farthest = -1, replaceIndex = 0;

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

int j;

for (j = currentPos + 1; j < pageCount; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

replaceIndex = i;

}

break;

}

}

if (j == pageCount)

return i;

}

return (farthest == -1) ? 0 : replaceIndex;

}

// Optimal Page Replacement Algorithm

void optimal(int\* pages, int pageCount, int frameCount) {

int i,j;

int\* frames = (int\*)calloc(frameCount, sizeof(int));

int pageFaults = 0;

printf("\nOptimal Page Replacement:\n");

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

printf("\nReference to page %d: ", pages[i]);

if (!isPagePresent(pages[i], frames, frameCount)) {

int replaceIndex;

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

if (frames[j] == 0) {

replaceIndex = j;

break;

}

}

if (frames[frameCount - 1] != 0) {

replaceIndex = findOptimalPosition(pages, pageCount, frames, frameCount, i);

}

frames[replaceIndex] = pages[i];

pageFaults++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

}

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

if (frames[j] != 0)

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

else

printf("- ");

}

}

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

free(frames);

}

int main() {

int pageCount, frameCount;

int i;

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

scanf("%d", &pageCount);

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

scanf("%d", &frameCount);

int\* pages = (int\*)malloc(pageCount \* sizeof(int));

printf("Enter the page reference string: ");

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

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

}

optimal(pages, pageCount, frameCount);

free(pages);

return 0;

}

**PRACTICE QUESTION**

**Q.Implement Banker’s algorithm. Show the safe sequence and print the available**

**vector at each stage. There are 3 resources A, B, C – total instances of each resource are**

**10, 5, 7 respectively.**

**Allocation Max**

**A B C A B C**

**P0 0 1 0 7 5 3**

**P1 2 0 0 3 2 2**

**P2 3 0 2 9 0 2**

**P3 2 1 1 2 2 2**

**P4 0 0 2 4 3 3**

**CODE:**

#include<stdio.h>

int main() {

int p = 5, c = 3, count = 0, i, j;

int alc[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 need[5][3], safe[5], available[3] = {10, 5, 7}, done[5], terminate = 0;

printf("Total resources: A = 10, B = 5, C = 7\n");

printf("\nAllocation Matrix (Alc):\n");

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

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

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

}

printf("\n");

}

printf("\nMax Matrix:\n");

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

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

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

}

printf("\n");

}

printf("\nNeed Matrix (calculated as Max - Alc):\n");

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

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

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

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

}

printf("\n");

}

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

done[i] = 0; // Mark all processes as not done

}

printf("\nAvailable Vector at each stage:\n");

while (count < p) {

int found = 0;

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

if (done[i] == 0) { // If process is not completed

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

if (need[i][j] > available[j])

break;

}

if (j == c) { // If all needs can be satisfied

printf("P%d: ", i); // Show process being executed

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

available[j] += alc[i][j]; // Release resources

}

safe[count++] = i;

done[i] = 1;

found = 1;

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

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

}

printf("\n");

}

}

}

if (!found) { // If no process could be executed

printf("\nSafe sequence does not exist.\n");

return 0;

}

}

printf("\nSafe Sequence: ");

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

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

}

printf("\n");

return 0;

}

**Q.Implement the LRU/FIFO/OPT page replacement algorithm.**

**Calculate the number of page faults for each of the algorithms using 3 page frames and 4**

**page frames for the following reference string:**

**5, 6, 7, 8, 5, 6, 9, 5, 6, 7, 8, 9**

**CODE\_FIFO-**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Function to check if a page exists in frame

bool isPagePresent(int pg, int\* fs, int fC) {

int i;

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

if (fs[i] == pg)

return true;

}

return false;

}

// FIFO Page Replacement Algorithm

void fifo(int\* pgs, int pC, int fC) {

int i,j;

int\* fs = (int\*)calloc(fC, sizeof(int));

int pF = 0;

int currentIndex = 0;

printf("\nFIFO Page Replacement:\n");

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

printf("\nReference to page %d: ", pgs[i]);

if (!isPagePresent(pgs[i], fs, fC)) {

fs[currentIndex] = pgs[i];

currentIndex = (currentIndex + 1) % fC;

pF++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

}

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

if (fs[j] != 0)

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

else

printf("- ");

}

}

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

free(fs);

}

int main() {

int i;

int pC, fC;

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

scanf("%d", &pC);

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

scanf("%d", &fC);

int\* pgs = (int\*)malloc(pC \* sizeof(int));

printf("Enter the page reference string: ");

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

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

}

fifo(pgs, pC, fC);

free(pgs);

return 0;

}

**CODE\_LRU-**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <limits.h>

// Function to check if a page exists in frame

bool isPagePresent(int page, int\* frames, int frameCount) {

int i;

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

if (frames[i] == page)

return true;

}

return false;

}

// LRU Page Replacement Algorithm

void lru(int\* pages, int pageCount, int frameCount) {

int i,j;

int\* frames = (int\*)calloc(frameCount, sizeof(int));

int\* lastUsed = (int\*)calloc(frameCount, sizeof(int));

int pageFaults = 0;

printf("\nLRU Page Replacement:\n");

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

printf("\nReference to page %d: ", pages[i]);

if (!isPagePresent(pages[i], frames, frameCount)) {

int replaceIndex = 0;

int leastUsed = INT\_MAX;

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

if (frames[j] == 0) {

replaceIndex = j;

break;

}

if (lastUsed[j] < leastUsed) {

leastUsed = lastUsed[j];

replaceIndex = j;

}

}

frames[replaceIndex] = pages[i];

lastUsed[replaceIndex] = i;

pageFaults++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

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

if (frames[j] == pages[i]) {

lastUsed[j] = i;

}

}

}

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

if (frames[j] != 0)

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

else

printf("- ");

}

}

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

free(frames);

free(lastUsed);

}

int main() {

int i;

int pageCount, frameCount;

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

scanf("%d", &pageCount);

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

scanf("%d", &frameCount);

int\* pages = (int\*)malloc(pageCount \* sizeof(int));

printf("Enter the page reference string: ");

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

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

}

lru(pages, pageCount, frameCount);

free(pages);

return 0;

}

**CODE\_OPT-**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Function to check if a page exists in frame

bool isPagePresent(int page, int\* frames, int frameCount) {

int i;

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

if (frames[i] == page)

return true;

}

return false;

}

// Function to find the position of the first occurrence of a page in future

int findOptimalPosition(int\* pages, int pageCount, int\* frames, int frameCount, int currentPos) {

int i,j;

int farthest = -1, replaceIndex = 0;

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

int j;

for (j = currentPos + 1; j < pageCount; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

replaceIndex = i;

}

break;

}

}

if (j == pageCount)

return i;

}

return (farthest == -1) ? 0 : replaceIndex;

}

// Optimal Page Replacement Algorithm

void optimal(int\* pages, int pageCount, int frameCount) {

int i,j;

int\* frames = (int\*)calloc(frameCount, sizeof(int));

int pageFaults = 0;

printf("\nOptimal Page Replacement:\n");

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

printf("\nReference to page %d: ", pages[i]);

if (!isPagePresent(pages[i], frames, frameCount)) {

int replaceIndex;

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

if (frames[j] == 0) {

replaceIndex = j;

break;

}

}

if (frames[frameCount - 1] != 0) {

replaceIndex = findOptimalPosition(pages, pageCount, frames, frameCount, i);

}

frames[replaceIndex] = pages[i];

pageFaults++;

printf("Page Fault! Frames: ");

} else {

printf("No Page Fault. Frames: ");

}

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

if (frames[j] != 0)

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

else

printf("- ");

}

}

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

free(frames);

}

int main() {

int pageCount, frameCount;

int i;

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

scanf("%d", &pageCount);

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

scanf("%d", &frameCount);

int\* pages = (int\*)malloc(pageCount \* sizeof(int));

printf("Enter the page reference string: ");

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

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

}

optimal(pages, pageCount, frameCount);

free(pages);

return 0;

}

**Q.Implement the Non preemptive SJT scheduling algorithm using the given test case.**

**Display the Gantt chart and display the turnaround time and waiting time for each**

**process.**

**Process Arrival time Burst time**

P 1 0 7

P 2 1 1

P 3 2 3

P 4 3 4

**CODE-**

import matplotlib.pyplot as plt  
  
  
class Process:  
 def \_\_init\_\_(self, pid, arrival\_time, burst\_time):  
 self.pid = pid  
 self.arrival\_time = arrival\_time  
 self.burst\_time = burst\_time  
 self.completion\_time = 0  
 self.turnaround\_time = 0  
 self.waiting\_time = 0  
  
  
class SJFScheduler:  
 def \_\_init\_\_(self, processes):  
 self.processes = processes  
 self.gantt\_chart = []  
  
 def schedule(self):  
 # Sort processes by arrival time, then by burst time  
 self.processes.sort(key=lambda p: (p.arrival\_time, p.burst\_time))  
 time = 0  
 completed\_processes = 0  
  
 while completed\_processes < len(self.processes):  
 # Filter processes that have arrived and are not completed  
 available\_processes = [p for p in self.processes if p.arrival\_time <= time and p.completion\_time == 0]  
  
 if not available\_processes:  
 time += 1  
 continue  
  
 # Select the process with the shortest burst time  
 shortest\_job = min(available\_processes, key=lambda p: p.burst\_time)  
  
 start\_time = time  
 time += shortest\_job.burst\_time  
 shortest\_job.completion\_time = time  
 shortest\_job.turnaround\_time = shortest\_job.completion\_time - shortest\_job.arrival\_time  
 shortest\_job.waiting\_time = shortest\_job.turnaround\_time - shortest\_job.burst\_time  
  
 # Add to Gantt chart  
 self.gantt\_chart.append((shortest\_job.pid, start\_time, time))  
 completed\_processes += 1  
  
 def calculate\_averages(self):  
 total\_waiting\_time = sum(p.waiting\_time for p in self.processes)  
 total\_turnaround\_time = sum(p.turnaround\_time for p in self.processes)  
 avg\_waiting\_time = total\_waiting\_time / len(self.processes)  
 avg\_turnaround\_time = total\_turnaround\_time / len(self.processes)  
 return avg\_waiting\_time, avg\_turnaround\_time  
  
 def plot\_gantt\_chart(self, title):  
 fig, ax = plt.subplots(figsize=(12, 4))  
 for pid, start, end in self.gantt\_chart:  
 ax.barh(0, end - start, left=start, height=0.3, label=f'P{pid}', edgecolor='black', alpha=0.8)  
 ax.text(start + (end - start) / 2, 0, f'P{pid}', ha='center', va='center', color='white')  
 ax.set\_title(title)  
 ax.set\_xlabel("Time")  
 ax.set\_yticks([])  
 ax.legend()  
 plt.show()  
  
  
# Example Usage  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Input process data  
 processes = [  
 Process(pid=1, arrival\_time=0, burst\_time=7),  
 Process(pid=2, arrival\_time=1, burst\_time=4),  
 Process(pid=3, arrival\_time=2, burst\_time=1),  
 Process(pid=4, arrival\_time=3, burst\_time=3)  
 ]  
  
 scheduler = SJFScheduler(processes)  
 scheduler.schedule()  
 avg\_waiting\_time, avg\_turnaround\_time = scheduler.calculate\_averages()  
  
 print("PID\tAT\tBT\tCT\tTAT\tWT")  
 for p in scheduler.processes:  
 print(f"{p.pid}\t{p.arrival\_time}\t{p.burst\_time}\t{p.completion\_time}\t{p.turnaround\_time}\t{p.waiting\_time}")  
 print(f"\nAverage Waiting Time: {avg\_waiting\_time:.2f}")  
 print(f"Average Turnaround Time: {avg\_turnaround\_time:.2f}")  
  
 scheduler.plot\_gantt\_chart("SJF Non-preemptive - Gantt Chart")

**Q.Implement the Producer Consumer problem using threads and semaphores.**

**Assume 1 Producer and 2 Consumers.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER\_SIZE 5

int buffer[BUFFER\_SIZE];

int in = 0, out = 0; // Buffer indices

sem\_t empty, full; // Semaphores

pthread\_mutex\_t mutex; // Mutex lock for critical section

// Producer function

void \*producer(void \*arg) {

int i;

int item;

for ( i = 1; i <= 10; i++) {

item = rand() % 100; // Producing a random item

sem\_wait(&empty); // Wait for empty slot

pthread\_mutex\_lock(&mutex); // Lock critical section

buffer[in] = item;

printf("Producer produced: %d\n", item);

in = (in + 1) % BUFFER\_SIZE;

pthread\_mutex\_unlock(&mutex); // Unlock critical section

sem\_post(&full); // Signal full slot

sleep(1);

}

pthread\_exit(NULL);

}

// Consumer function

void \*consumer(void \*arg) {

int i ;

int item;

int id = \*(int \*)arg;

free(arg); // Free allocated memory for thread ID

for ( i = 1; i <= 5; i++) { // Each consumer consumes 5 items

sem\_wait(&full); // Wait for full slot

pthread\_mutex\_lock(&mutex); // Lock critical section

item = buffer[out];

printf("Consumer %d consumed: %d\n", id, item);

out = (out + 1) % BUFFER\_SIZE;

pthread\_mutex\_unlock(&mutex); // Unlock critical section

sem\_post(&empty); // Signal empty slot

sleep(1);

}

pthread\_exit(NULL);

}

int main() {

pthread\_t prod, cons1, cons2;

// Initialize semaphores

sem\_init(&empty, 0, BUFFER\_SIZE); // BUFFER\_SIZE empty slots

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

pthread\_mutex\_init(&mutex, NULL); // Initialize mutex

// Create producer thread

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

// Create consumer threads

int \*id1 = malloc(sizeof(int));

int \*id2 = malloc(sizeof(int));

\*id1 = 1;

\*id2 = 2;

pthread\_create(&cons1, NULL, consumer, id1);

pthread\_create(&cons2, NULL, consumer, id2);

// Wait for threads to finish

pthread\_join(prod, NULL);

pthread\_join(cons1, NULL);

pthread\_join(cons2, NULL);

// Destroy semaphores and mutex

sem\_destroy(&empty);

sem\_destroy(&full);

pthread\_mutex\_destroy(&mutex);

return 0;

}

**Q. Write a shell script to check whether entered substring is present in given string or**

**not. Print the number of occurrences and the position of each occurrence.**

**CODE-**

#!/bin/bash

# Prompt user for the main string

echo "Enter the main string:"

read main\_string

# Prompt user for the substring

echo "Enter the substring to search:"

read substring

# Check if the substring exists in the main string

if [[ "$main\_string" == \*"$substring"\* ]]; then

echo "The substring '$substring' is present in the main string."

# Find occurrences and positions

count=0

pos=0

while [[ $pos -lt ${#main\_string} ]]; do

pos=$(expr index "${main\_string:$pos}" "$substring")

if [[ $pos -gt 0 ]]; then

count=$((count + 1))

echo "Occurrence $count found at position $((pos + count - 1))"

pos=$((pos + ${#substring}))

else

break

fi

done

echo "Total occurrences: $count"

else

echo "The substring '$substring' is not present in the main string."

Fi

**Q.Write an awk script to prepare a report in the following format:**

**Roll No. Name BS DA HRA GS**

**The data file contains the Record in the following form:**

**Empno:Name:Basic salary:DA:HRA**

**Use the formula GS = BS + DA + HRA**

**where DA = 0.5\*BS and HRA = 0.2\*BS**

**Calculate the gross salary of employee and display the result.**

**CODE-**

**Example.txt-**

Empno:Name:Basic salary

101:John:40000

102:Jane:35000

103:Alice:45000

**generate\_report.awk-**

BEGIN {

# Print the report header

printf "%-10s %-10s %-10s %-10s %-10s %-10s\n", "Roll No.", "Name", "BS", "DA", "HRA", "GS";

}

{

# Split the input line into fields using ":" as a delimiter

split($0, fields, ":");

# Extract fields

empno = fields[1];

name = fields[2];

bs = fields[3];

# Calculate DA, HRA, and GS

da = 0.5 \* bs;

hra = 0.2 \* bs;

gs = bs + da + hra;

# Print the calculated values in the report format

printf "%-10s %-10s %-10.2f %-10.2f %-10.2f %-10.2f\n", empno, name, bs, da, hra, gs;

}

To run: awk -f generate\_report.awk employee\_data.txt