**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