**Operating System – Semester Lab Programs**

# **Shell Programs:**

## Largest of three numbers:

echo "Enter three numbers:"

read num1

read num2

read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]; then

echo "$num1 is the largest number."

elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]; then

echo "$num2 is the largest number."

else

echo "$num3 is the largest number."

fi

## Factorial:

echo "Enter a number"

read num

fact=1

temp=$num

while [ $temp -gt 1 ]

do

fact=$((fact \* temp))

temp=$((temp - 1))

done

echo "Factorial of $num is $fact"

----------------------------------------------------------------------------------------------------------

## Armstrong Number:

echo "Enter A Number: "

read n

arm=0

temp=$n

while [ "$n" -ne 0 ]; do

r=$((n % 10))

arm=$((arm + r \* r \* r))

n=$((n / 10))

done

if [ $arm -eq "$temp" ]; then

echo "$temp is an Armstrong number"

else

echo "$temp is not an Armstrong number"

fi

----------------------------------------------------------------------------------------------------------

## Sums of digits of an integer:

echo "Enter a Number:"

read n

temp=$n

sd=0

sum=0

while [ $n -gt 0 ]

do

sd=$(( $n % 10 ))

n=$(( $n / 10 ))

sum=$(( $sum + $sd ))

done

echo "Sum is $sum"

----------------------------------------------------------------------------------------------------------

## Reverse an integer:

echo "Enter a Number:"

read n

num=0

while [ $n -gt 0 ]

do

num=$(expr $num \\* 10)

k=$(expr $n % 10)

num=$(expr $num + $k)

n=$(expr $n / 10)

done

echo "Reversed number is $num"

----------------------------------------------------------------------------------------------------------

## Fibonacci:

echo "Enter the number of terms: "

read n

a=0

b=1

echo "Fibonacci sequence up to $n terms: "

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

do

echo -n "$a "

fn=$((a + b))

a=$b

b=$fn

done

----------------------------------------------------------------------------------------------------------

## String search:

echo "Enter the main string:"

read mainstring

echo "Enter the substring to search:"

read substring

if [[ "$mainstring" == \*"$substring"\* ]]; then

index=$(awk -v a="$mainstring" -v b="$substring" 'BEGIN{print index(a,b)}')

echo "The substring '$substring' was found in the main string '$mainstring' at index $((index - 1))."

else

echo "The substring '$substring' was not found in the main string."

fi

----------------------------------------------------------------------------------------------------------

## Appending contents of a file:

echo "Enter the initial content:"

read initial\_content

echo "Enter the content to append:"

read append\_content

combined\_content="${initial\_content}${append\_content}"

echo "Combined content:"

echo "$combined\_content"

----------------------------------------------------------------------------------------------------------

# FCFS:

#include <stdio.h>

struct Process {

int id;

int at, bt, wt, tat;

};

void sortByArrivalTime(struct Process p[], int n) {

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

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

if (p[j].at > p[j + 1].at) {

struct Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

}

int main() {

int n;

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

scanf("%d", &n);

struct Process p[n];

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

p[i].id = i + 1;

printf("Enter burst time for process %d: ", i + 1);

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

printf("Enter arrival time for process %d: ", i + 1);

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

}

sortByArrivalTime(p, n);

int curr\_time = 0;

float twt = 0, ttt = 0;

printf("\nFCFS(First Come First Serve) CPU Scheduling:\n");

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

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

if (curr\_time < p[i].at) curr\_time = p[i].at;

p[i].wt = curr\_time - p[i].at;

curr\_time += p[i].bt;

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

twt += p[i].wt;

ttt += p[i].tat;

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

}

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

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

return 0;

}

----------------------------------------------------------------------------------------------------------

# SJF:

#include <stdio.h>

struct Process {

int process\_id;

int burst\_time;

int arrival\_time;

int waiting\_time;

int turnaround\_time;

int completed;

};

void sortByArrivalTime(struct Process processes[], int n) {

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

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

if (processes[j].arrival\_time > processes[j + 1].arrival\_time) {

struct Process temp = processes[j];

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

processes[j + 1] = temp;

}

}

}

}

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].process\_id = i + 1;

processes[i].completed = 0;

printf("Enter burst time for process %d: ", i + 1);

scanf("%d", &processes[i].burst\_time);

printf("Enter arrival time for process %d: ", i + 1);

scanf("%d", &processes[i].arrival\_time);

}

sortByArrivalTime(processes, n);

int current\_time = 0, completed\_processes = 0;

float total\_waiting\_time = 0, total\_turnaround\_time = 0;

while (completed\_processes < n) {

int shortest\_job\_index = -1;

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

if (processes[i].arrival\_time <= current\_time && processes[i].completed == 0) {

if (shortest\_job\_index == -1 || processes[i].burst\_time < processes[shortest\_job\_index].burst\_time) {

shortest\_job\_index = i;

}

}

}

if (shortest\_job\_index == -1) {

current\_time++;

} else {

struct Process \*p = &processes[shortest\_job\_index];

p->waiting\_time = current\_time - p->arrival\_time;

current\_time += p->burst\_time;

p->turnaround\_time = p->waiting\_time + p->burst\_time;

p->completed = 1;

completed\_processes++;

total\_waiting\_time += p->waiting\_time;

total\_turnaround\_time += p->turnaround\_time;

}

}

printf("\nSJF(Shortest Job First) CPU Scheduling:\n");

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

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

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

}

printf("\nAverage Waiting Time: %.2f", total\_waiting\_time / n);

printf("\nAverage Turnaround Time: %.2f", total\_turnaround\_time / n);

return 0;

}

----------------------------------------------------------------------------------------------------------

# Non Preemptive Priority Scheduling:

#include <stdio.h>

struct Process {

int process\_id;

int burst\_time;

int arrival\_time;

int priority;

int waiting\_time;

int turnaround\_time;

};

void sortByArrivalTime(struct Process processes[], int n) {

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

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

if (processes[j].arrival\_time > processes[j + 1].arrival\_time) {

struct Process temp = processes[j];

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

processes[j + 1] = temp;

}

}

}

}

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].process\_id = i + 1;

printf("Enter burst time for process %d: ", i + 1);

scanf("%d", &processes[i].burst\_time);

printf("Enter arrival time for process %d: ", i + 1);

scanf("%d", &processes[i].arrival\_time);

printf("Enter priority for process %d: ", i + 1);

scanf("%d", &processes[i].priority);

}

sortByArrivalTime(processes, n);

int current\_time = 0;

float total\_waiting\_time = 0, total\_turnaround\_time = 0;

int completed = 0;

int is\_completed[n];

for (int i = 0; i < n; i++) is\_completed[i] = 0;

while (completed != n) {

int idx = -1;

int highest\_priority = 100000; // Assuming lower values indicate higher priority

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

if (processes[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

if (processes[i].priority < highest\_priority) {

highest\_priority = processes[i].priority;

idx = i;

}

if (processes[i].priority == highest\_priority) {

if (processes[i].arrival\_time < processes[idx].arrival\_time) {

highest\_priority = processes[i].priority;

idx = i;

}

}

}

}

if (idx != -1) {

current\_time += processes[idx].burst\_time;

processes[idx].turnaround\_time = current\_time - processes[idx].arrival\_time;

processes[idx].waiting\_time = processes[idx].turnaround\_time - processes[idx].burst\_time;

total\_waiting\_time += processes[idx].waiting\_time;

total\_turnaround\_time += processes[idx].turnaround\_time;

is\_completed[idx] = 1;

completed++;

} else {

current\_time++;

}

}

printf("\nNon Preemptive Priority CPU Scheduling:\n");

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

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

printf("%d\t\t%d\t\t\t\t%d\t\t\t\t%d\t\t\t\t%d\t\t\t\t%d\n",

processes[i].process\_id,

processes[i].arrival\_time,

processes[i].burst\_time,

processes[i].priority,

processes[i].waiting\_time,

processes[i].turnaround\_time);

}

printf("\nAverage Waiting Time: %.2f", total\_waiting\_time / n);

printf("\nAverage Turnaround Time: %.2f", total\_turnaround\_time / n);

return 0;

}

----------------------------------------------------------------------------------------------------------

# Producer Consumer C:

#include <stdio.h>

#include <stdlib.h>

int mutex, full, empty, front, rear, buffer\_size;

int queue[100];

void producer(int item)

{

--mutex;

queue[rear] = item;

rear = (rear + 1) % buffer\_size;

++full;

--empty;

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

++mutex;

}

void consumer()

{

--mutex;

printf("\nConsumer consumes item %d", queue[front]);

front = (front + 1) % buffer\_size;

--full;

++empty;

++mutex;

}

int main()

{

int n, i;

printf("Enter the buffer size: ");

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

mutex = 1;

full = 0;

empty = buffer\_size;

front = 0;

rear = 0;

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

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

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n) {

case 1:

if ((mutex == 1)

&& (empty!= 0)) {

int item;

printf("Enter the item to produce: ");

scanf("%d", &item);

producer(item);

}

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;

}

}

}

----------------------------------------------------------------------------------------------------------

# Producer Consumer Java:

import java.util.Scanner;

public class ProducerConsumer {

static int mutex, full, empty, front, rear, buffer\_size;

static int[] queue = new int[100];

static void producer(int item) {

--mutex;

queue[rear] = item;

rear = (rear + 1) % buffer\_size;

++full;

--empty;

System.out.println("\nProducer produces item " + item);

++mutex;

}

static void consumer() {

--mutex;

System.out.println("\nConsumer consumes item " + queue[front]);

front = (front + 1) % buffer\_size;

--full;

++empty;

++mutex;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n;

System.out.print("Enter the buffer size: ");

buffer\_size = scanner.nextInt();

mutex = 1;

full = 0;

empty = buffer\_size;

front = 0;

rear = 0;

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

while (true) {

System.out.print("\nEnter your choice:");

n = scanner.nextInt();

switch (n) {

case 1:

if ((mutex == 1) && (empty!= 0)) {

System.out.print("Enter the item to produce: ");

int item = scanner.nextInt();

producer(item);

} else {

System.out.println("Buffer is full!");

}

break;

case 2:

if ((mutex == 1) && (full!= 0)) {

consumer();

} else {

System.out.println("Buffer is empty!");

}

break;

case 3:

System.exit(0);

break;

}

}

}

}

----------------------------------------------------------------------------------------------------------

# Banker’s Algorithm:

#include <stdio.h>

#define max\_p 10

#define max\_r 10

int avail[max\_r], max[max\_p][max\_r], alloc[max\_p][max\_r], need[max\_p][max\_r], work[max\_r], finish[max\_p], n\_processes, n\_resources;

void initialize() {

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

scanf("%d", &n\_processes);

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

scanf("%d", &n\_resources);

printf("Enter the available resources:\n");

for (int i = 0; i < n\_resources; i++)

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

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

printf("Enter the maximum resources needed for process %d:\n", i);

for (int j = 0; j < n\_resources; j++)

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

}

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

printf("Enter the allocated resources for process %d:\n", i);

for (int j = 0; j < n\_resources; j++)

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

}

for (int i = 0; i < n\_processes; i++)

for (int j = 0; j < n\_resources; j++)

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

}

void printNeedMatrix() {

printf("Need Matrix:\n");

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

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

for (int j = 0; j < n\_resources; j++)

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

printf("\n");

}

}

int safe() {

int work[max\_r];

int safe\_seq[max\_p];

int count = 0;

for (int i = 0; i < n\_resources; i++)

work[i] = avail[i];

for (int i = 0; i < n\_processes; i++)

finish[i] = 0;

while (count < n\_processes) {

int found = 0;

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

if (finish[i] == 0) {

int j;

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

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

break;

}

if (j == n\_resources) {

for (int k = 0; k < n\_resources; k++)

work[k] += alloc[i][k];

safe\_seq[count++] = i;

finish[i] = 1;

found = 1;

}

}

}

if (found == 0)

break;

}

if (count < n\_processes) {

printf("System is unsafe!\n");

return 0;

} else {

printf("System is safe.\nSafe sequence is: ");

for (int i = 0; i < n\_processes; i++)

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

printf("\n");

return 1;

}

}

int main() {

initialize();

printNeedMatrix();

safe();

return 0;

}

----------------------------------------------------------------------------------------------------------

# First fit, Best fit, Worst Fit:

#include <stdio.h>

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

int allocation[n];

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

allocation[i] = -1;

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

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

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

allocation[i] = blockSize[j];

blockSize[j] =0;

break;

}

}

}

printf("\nFirst Fit Allocation:\n");

printf("Process No.\tProcess Size\tBlock No.\n");

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

printf(" %d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

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

int allocation[n];

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

allocation[i] = -1;

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

int bestIdx = -1;

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

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

if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])

bestIdx = j;

}

}

if (bestIdx != -1) {

allocation[i] = blockSize[bestIdx];

blockSize[bestIdx] = 0;

}

}

printf("\nBest Fit Allocation:\n");

printf("Process No.\tProcess Size\tBlock No.\n");

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

printf(" %d \t\t %d \t\t", i+1, processSize[i]);

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

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

int allocation[n];

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

allocation[i] = -1;

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

int worstIdx = -1;

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

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

if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])

worstIdx = j;

}

}

if (worstIdx != -1) {

allocation[i] = blockSize[worstIdx];

blockSize[worstIdx] = 0;

}

}

printf("\nWorst Fit Allocation:\n");

printf("Process No.\tProcess Size\tBlock No.\n");

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

printf(" %d \t\t %d \t\t", i+1, processSize[i]);

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

int main() {

int m, n;

printf("Enter the number of memory blocks: ");

scanf("%d", &m);

int blockSize[m];

printf("Enter the size of each memory block:\n");

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

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

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

}

printf("\nEnter the number of processes: ");

scanf("%d", &n);

int processSize[n];

printf("Enter the size of each process:\n");

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

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

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

}

int choice;

while(1){

printf("Enter your choice:\n1)First Fit Allocation\n2)Best Fit Allocation\n3)Worst Fit Allocation\n");

scanf("%d",&choice);

if(choice == 1){

firstFit(blockSize, m, processSize, n);

break;

}

else if(choice == 2){

bestFit(blockSize, m, processSize, n);

break;

}

else if(choice == 3){

worstFit(blockSize, m, processSize, n);

break;

}

else{

printf("Invalid Choice!\n");

}

}

return 0;

}

----------------------------------------------------------------------------------------------------------

# Page Table:

#include <stdio.h>

#define MAX\_PAGES 100

int main(){

int page\_table [MAX\_PAGES],num\_pages,page\_number, offset;

printf("Enter the number of pages in the page table:");

scanf("%d", &num\_pages);

printf("Enter the frame address for each page:\n");

for (int i=0;i < num\_pages ; i++){

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

scanf("%d", & page\_table[i]);

}

printf ("Enter the Logical address (page numbers and offset): ");

scanf("%d %d", &page\_number, &offset);

if (page\_number >= num\_pages) {

printf("Error: Page number out of bounds \n");

return 1;

}

int frame\_address = page\_table [page\_number];

int physical\_address = frame\_address + offset;

printf("Logical Addres:%d\n", physical\_address);

printf("Page Table: \n");

for (int i=0;i<num\_pages; i++){

printf ("Page %d -> starting address %d\n", page\_table[i]);

}

printf("Physical address:%d", physical\_address);

return 0;

}