1. Any shell scripting program.

**Print Sum of Digits of a given number using command line argument.**

#!/bin/bash

read -p "Enter number: " number

sum=0

while [ $number -gt 0 ]; do

digit=$(( number % 10 ))

sum=$(( sum + digit ))

number=$(( number / 10 ))

done

echo "sum of digits: $sum"

**Output**:

ubuntu@ubuntu:~/Desktop$ ./sum\_of\_digit.sh

Enter number: 123

sum of digits: 6

**Write a shell script using function for following:**

**1) Average of given numbers**

**2) Max digit from given number and**

**3) Min digit from given number.**

#!/bin/bash

average() {

sum=0

count=0

for num in "$@"; do

sum=$((sum + num))

count=$((count + 1))

done

avg=$(echo "scale=2; $sum / $count" | bc)

echo "Average of given numbers: $avg"

}

max\_number() {

max=$1

for num in "$@"; do

if (( num > max )); then

max=$num

fi

done

echo "Maximum: $max"

}

min\_number() {

min=$1

for num in "$@"; do

if (( num < min )); then

min=$num

fi

done

echo "Minimum: $min"

}

echo "Enter Numbers: "

read -a numbers

average "${numbers[@]}"

max\_number "${numbers[@]}"

min\_number "${numbers[@]}"

**Output:**

Enter Numbers:

20 30

Average of given numbers: 25.00

Maximum: 30

Minimum: 20

**Perform sorting on given array elements.**

#!/bin/bash

sort\_array() {

arr=("$@")

n=${#arr[@]}

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

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

if ((arr[i] > arr[j])); then

temp=${arr[i]}

arr[i]=${arr[j]}

arr[j]=$temp

fi

done

done

echo "Sorted Array: ${arr[@]}"

}

echo "Enter Numbers: "

read -a numbers

sort\_array "${numbers[@]}"

**Output:**

Enter Numbers:

9 1 6 3 4 8 2 5 7

Sorted Array: 1 2 3 4 5 6 7 8 9

**Program to find factorial of a given number with and without recursion.**

#!/bin/bash

factorial\_iterative() {

num=$1

factorial=1

for ((i = 2; i <= num; i++)); do

factorial=$((factorial \* i))

done

echo $factorial

}

factorial\_recursive() {

num=$1

if ((num <= 1)); then

echo 1

else

echo $((num \* $(factorial\_recursive $((num - 1)))))

fi

}

echo "Enter Number: "

read number

echo "Factorial (Iterative) of $number: $(factorial\_iterative $number)"

echo "Factorial (Recursive) of $number: $(factorial\_recursive $number)"

**Output**:

Enter Number:

5

Factorial (Iterative) of 5: 120

Factorial (Recursive) of 5: 120

**Program to check file type and permission for a given file.**

#!/bin/bash

check\_file() {

local file=$1

if [ -e "$file" ]; then

file\_type=$(file -b "$file")

permissions=$(stat -c "%A" "$file")

echo "File Type: $file\_type"

echo "Permissions: $permissions"

else

echo "File does not exist"

fi

}

echo "Enter file path: "

read file\_path

check\_file "$file\_path"

**Output**:

Enter file path:

hello.txt

File Type: ASCII text

Permissions: -rw-rw-r—

**Check entered string is palindrome or not?**

#!/bin/bash

is\_palindrome() {

local str="$1"

local reversed\_str=$(echo "$str" | rev)

if [ "$str" = "$reversed\_str" ]; then

echo "The string is palindrome"

else

echo "The string is not palindrome"

fi

}

echo "Enter String: "

read input\_string

is\_palindrome "$input\_string"

**Output:**

Enter String:

madam

The string is palindrome

Enter String:

hello

The string is not palindrome

2. Write a program demonstrating use of different system calls.

#include <iostream>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <cstring>

#include <errno.h>

#include <signal.h>

#include <sys/utsname.h>

using namespace std;

void process\_system\_calls();

void file\_system\_calls();

void communication\_system\_calls();

void information\_system\_calls();

int main() {

int choice;

while (true) {

cout << "\n--- System Call Menu ---\n";

cout << "1. Process Related\n";

cout << "2. File Related\n";

cout << "3. Communication Related\n";

cout << "4. Information Related\n";

cout << "5. Exit\n";

cout << "Choose an option: ";

cin >> choice;

switch (choice) {

case 1: process\_system\_calls(); break;

case 2: file\_system\_calls(); break;

case 3: communication\_system\_calls(); break;

case 4: information\_system\_calls(); break;

case 5: exit(0);

default: cout << "Invalid choice! Try again.\n";

}

}

return 0;

}

void process\_system\_calls() {

int choice;

pid\_t pid;

cout << "\n--- Process Related Calls ---\n";

cout << "1. Fork\n2. Exit\n3. Wait\n4. Kill\n5. Exec\n";

cout << "Choose an option: ";

cin >> choice;

switch (choice) {

case 1:

pid = fork();

if (pid == 0) cout << "Child PID: " << getpid() << endl;

else { cout << "Parent PID: " << getpid() << endl; wait(nullptr); }

break;

case 2:

cout << "Process will exit now.\n";

exit(0);

break;

case 3:

pid = fork();

if (pid == 0) { sleep(2); exit(0); }

else { cout << "Waiting for child...\n"; wait(nullptr); }

break;

case 4:

pid = fork();

if (pid == 0) while (true); // Infinite loop

else { sleep(1); cout << "Killing child process...\n"; kill(pid, SIGKILL); wait(nullptr); }

break;

case 5:

if (fork() == 0) execl("/bin/ls", "ls", nullptr);

else wait(nullptr);

break;

default: cout << "Invalid choice!\n";

}

}

void file\_system\_calls() {

int choice;

int fd;

char buffer[100];

cout << "\n--- File Related Calls ---\n";

cout << "1. Open/Read/Write/Close\n2. Link/Unlink\n3. Stat\n";

cout << "Choose an option: ";

cin >> choice;

switch (choice) {

case 1:

fd = open("testfile.txt", O\_CREAT | O\_RDWR, 0644);

if (fd == -1) { perror("Error opening file"); return; }

write(fd, "Hello, World!\n", 14);

lseek(fd, 0, SEEK\_SET);

read(fd, buffer, sizeof(buffer));

cout << "File content: " << buffer << endl;

close(fd);

break;

case 2:

link("testfile.txt", "testfile\_link.txt");

cout << "Link created.\n";

unlink("testfile\_link.txt");

cout << "Link removed.\n";

break;

case 3: {

struct stat fileStat;

if (stat("testfile.txt", &fileStat) < 0) { perror("Error getting stats"); return; }

cout << "File Size: " << fileStat.st\_size << " bytes\n";

cout << "File Permissions: " << (fileStat.st\_mode & 0777) << endl;

break;

}

default: cout << "Invalid choice!\n";

}

}

void communication\_system\_calls() {

int choice;

cout << "1. Pipe\n2. FIFO\nChoose an option: ";

cin >> choice;

if (choice == 1) {

int fd[2];

pid\_t pid;

char buffer[100];

if (pipe(fd) == -1) {

perror("Pipe failed");

return;

}

pid = fork();

if (pid == 0) {

close(fd[0]);

write(fd[1], "Message via Pipe", 16);

close(fd[1]);

exit(0);

} else {

close(fd[1]);

int bytesRead = read(fd[0], buffer, sizeof(buffer));

buffer[bytesRead] = '\0';

cout << "Received: " << buffer << endl;

close(fd[0]);

wait(nullptr);

}

} else if (choice == 2) {

const char \*fifo = "/tmp/myfifo";

mkfifo(fifo, 0666);

if (fork() == 0) {

int fd = open(fifo, O\_WRONLY);

write(fd, "Message via FIFO", 16);

close(fd);

exit(0);

} else {

char buffer[100];

int fd = open(fifo, O\_RDONLY);

int bytesRead = read(fd, buffer, sizeof(buffer));

buffer[bytesRead] = '\0';

cout << "Received: " << buffer << endl;

close(fd);

wait(nullptr);

}

unlink(fifo);

} else {

cout << "Invalid option!\n";

}

}

void information\_system\_calls() {

int choice;

struct utsname buffer;

cout << "\n--- Information Related Calls ---\n";

cout << "1. Get PID\n2. Get PPID\n3. Uname\n";

cout << "Choose an option: ";

cin >> choice;

switch (choice) {

case 1: cout << "PID: " << getpid() << endl; break;

case 2: cout << "PPID: " << getppid() << endl; break;

case 3:

if (uname(&buffer) < 0) { perror("Uname failed"); return; }

cout << "System: " << buffer.sysname << endl;

cout << "Node: " << buffer.nodename << endl;

cout << "Release: " << buffer.release << endl;

cout << "Version: " << buffer.version << endl;

cout << "Machine: " << buffer.machine << endl;

break;

default: cout << "Invalid choice!\n";

}

}

3. Implement multi threading for Matrix Operations using Pthreads.

#include <iostream>

#include <pthread.h>

#include <vector>

const int MAX = 10; // Max size of matrices for simplicity

int matrixA[MAX][MAX], matrixB[MAX][MAX], result[MAX][MAX];

int rows, cols;

void\* addMatrices(void\* arg) {

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

for (int j = 0; j < cols; j++)

result[i][j] = matrixA[i][j] + matrixB[i][j];

return nullptr;

}

V00oid\* subtractMatrices(void\* arg) {

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

for (int j = 0; j < cols; j++)

result[i][j] = matrixA[i][j] - matrixB[i][j];

return nullptr;

}

void\* multiplyMatrices(void\* arg) {

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

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

result[i][j] = 0;

for (int k = 0; k < cols; k++) {

result[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

return nullptr;

}

void\* divideMatrices(void\* arg) {

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

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

result[i][j] = (matrixB[i][j] != 0) ? matrixA[i][j] / matrixB[i][j] : 0;

}

}

return nullptr;

}

void printMatrix(const std::string& operation) {

std::cout << "Result of " << operation << ":\n";

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

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

std::cout << result[i][j] << " ";

}

std::cout << "\n";

}

}

int main() {

std::cout << "Enter number of rows and columns of matrices: ";

std::cin >> rows >> cols;

std::cout << "Enter elements of Matrix A:\n";

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

for (int j = 0; j < cols; j++)

std::cin >> matrixA[i][j];

std::cout << "Enter elements of Matrix B:\n";

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

for (int j = 0; j < cols; j++)

std::cin >> matrixB[i][j];

pthread\_t threads[4];

pthread\_create(&threads[0], nullptr, addMatrices, nullptr);

pthread\_join(threads[0], nullptr);

printMatrix("Addition");

pthread\_create(&threads[1], nullptr, subtractMatrices, nullptr);

pthread\_join(threads[1], nullptr);

printMatrix("Subtraction");

pthread\_create(&threads[2], nullptr, multiplyMatrices, nullptr);

pthread\_join(threads[2], nullptr);

printMatrix("Multiplication");

pthread\_create(&threads[3], nullptr, divideMatrices, nullptr);

pthread\_join(threads[3], nullptr);

printMatrix("Division");

return 0;

}

4. Implementation of Classical problems (reader writer) using Threads and Mutex

#include <iostream>

#include <thread>

#include <semaphore.h>

#include <unistd.h>

sem\_t resource\_sem;

sem\_t readers\_count\_sem;

int shared\_resource = 0; // Shared resource

int read\_count = 0; // Reader count

void reader(int reader\_id) {

while (true) {

sem\_wait(&readers\_count\_sem); // Lock the read count

read\_count++;

if (read\_count == 1) {

sem\_wait(&resource\_sem); // First reader locks the resource

std::cout << "Reader " << reader\_id << " enters critical section.\n";

}

sem\_post(&readers\_count\_sem); // Unlock the read count

std::cout << "Reader " << reader\_id << " is reading the value " << shared\_resource << std::endl;

sleep(1); // Simulate reading time

sem\_wait(&readers\_count\_sem); // Lock the read count

read\_count--;

if (read\_count == 0) {

std::cout << "Reader " << reader\_id << " exits critical section.\n";

sem\_post(&resource\_sem); // Last reader unlocks the resource

}

sem\_post(&readers\_count\_sem); // Unlock the read count

sleep(2); // Simulate time between reads

}

}

void writer(int writer\_id) {

while (true) {

sem\_wait(&resource\_sem); // Lock the resource for writing

std::cout << "Writer " << writer\_id << " enters critical section.\n";

shared\_resource++; // Writing to shared resource

std::cout << "Writer " << writer\_id << " wrote the value " << shared\_resource << std::endl;

sleep(2); // Simulate writing time

std::cout << "Writer " << writer\_id << " exits critical section.\n";

sem\_post(&resource\_sem); // Unlock the resource

sleep(3); // Simulate time between writes

}

}

int main() {

sem\_init(&resource\_sem, 0, 1);

sem\_init(&readers\_count\_sem, 0, 1);

std::thread readers[3], writers[2];

for (int i = 0; i < 3; ++i) {

readers[i] = std::thread(reader, i);

}

for (int i = 0; i < 2; ++i) {

writers[i] = std::thread(writer, i);

}

for (int i = 0; i < 3; ++i) {

readers[i].join();

}

for (int i = 0; i < 2; ++i) {

writers[i].join();

}

sem\_destroy(&resource\_sem);

sem\_destroy(&readers\_count\_sem);

return 0;

}

5. Implementation of Classical problems( producer consumer) using Threads and Mutex

#include <iostream>

#include <thread>

#include <mutex>

#include <queue>

#include <condition\_variable>

#include <chrono>

const int MAX\_QUEUE\_SIZE = 5;

std::queue<int> buffer;

std::mutex mtx;

std::condition\_variable cond\_empty; // Condition variable for empty buffer

std::condition\_variable cond\_full; // Condition variable for full buffer

void producer(int id) {

for (int i = 0; i < 10; ++i) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(100)); // Simulate work

std::unique\_lock<std::mutex> lock(mtx);

cond\_empty.wait(lock, [] { return buffer.size() < MAX\_QUEUE\_SIZE; }); // Wait if buffer is full

std::cout << "Producer " << id << " is entering critical section." << std::endl;

buffer.push(i);

std::cout << "Producer " << id << " produced: " << i << std::endl;

std::cout << "Producer " << id << " has exited critical section." << std::endl;

lock.unlock();

cond\_full.notify\_one(); // Notify one consumer

}

}

void consumer(int id) {

for (int i = 0; i < 10; ++i) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(150)); // Simulate work

std::unique\_lock<std::mutex> lock(mtx);

cond\_full.wait(lock, [] { return !buffer.empty(); }); // Wait if buffer is empty

std::cout << "Consumer " << id << " is entering critical section." << std::endl;

int item = buffer.front();

buffer.pop();

std::cout << "Consumer " << id << " consumed: " << item << std::endl;

std::cout << "Consumer " << id << " has exited critical section." << std::endl;

lock.unlock();

cond\_empty.notify\_one(); // Notify one producer

}

}

int main() {

std::thread producers[2], consumers[2];

for (int i = 0; i < 2; ++i) {

producers[i] = std::thread(producer, i);

consumers[i] = std::thread(consumer, i);

}

for (int i = 0; i < 2; ++i) {

producers[i].join();

consumers[i].join();

}

return 0;

}

6. Implementation of Classical problems (reader writer) using Threads and Semaphore. .(reader writer, producer consumer, dining philosopher)

#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <unistd.h>

std::mutex resource\_mutex;

std::mutex readers\_count\_mutex;

int shared\_resource = 0; // Shared resource

int read\_count = 0; // Reader count

void reader(int reader\_id) {

while (true) {

{

std::unique\_lock<std::mutex> lock(readers\_count\_mutex);

read\_count++;

if (read\_count == 1) {

resource\_mutex.lock(); // First reader locks the resource

std::cout << "Reader " << reader\_id << " enters critical section.\n";

}

}

std::cout << "Reader " << reader\_id << " is reading the value " << shared\_resource << std::endl;

sleep(1); // Simulate reading time

{

std::unique\_lock<std::mutex> lock(readers\_count\_mutex);

read\_count--;

if (read\_count == 0) {

std::cout << "Reader " << reader\_id << " exits critical section.\n";

resource\_mutex.unlock(); // Last reader unlocks the resource

}

}

sleep(2); // Simulate time between reads

}

}

void writer(int writer\_id) {

while (true) {

std::unique\_lock<std::mutex> lock(resource\_mutex); // Lock the resource for writing

std::cout << "Writer " << writer\_id << " enters critical section.\n";

shared\_resource++; // Writing to shared resource

std::cout << "Writer " << writer\_id << " wrote the value " << shared\_resource << std::endl;

sleep(2); // Simulate writing time

std::cout << "Writer " << writer\_id << " exits critical section.\n";

sleep(3); // Simulate time between writes

}

}

int main() {

std::thread readers[3], writers[2];

for (int i = 0; i < 3; ++i) {

readers[i] = std::thread(reader, i);

}

for (int i = 0; i < 2; ++i) {

writers[i] = std::thread(writer, i);

}

for (int i = 0; i < 3; ++i) {

readers[i].join();

}

for (int i = 0; i < 2; ++i) {

writers[i].join();

}

return 0;

}

7. Implementation of Classical problems (producer consumer,) using Threads and Semaphore.

#include <iostream>

#include <thread>

#include <semaphore.h>

#include <queue>

#include <chrono>

#include <mutex>

const int MAX\_QUEUE\_SIZE = 5;

std::queue<int> buffer;

sem\_t empty; // Semaphore to count empty slots

sem\_t full; // Semaphore to count full slots

std::mutex mtx;

void producer(int id) {

for (int i = 0; i < 10; ++i) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(100)); // Simulate work

sem\_wait(&empty); // Decrease empty slots

std::cout << "Producer " << id << " is entering critical section." << std::endl;

mtx.lock(); // Lock the buffer (critical section)

buffer.push(i);

std::cout << "Producer " << id << " produced: " << i << std::endl;

mtx.unlock(); // Unlock the buffer

std::cout << "Producer " << id << " has exited critical section." << std::endl;

sem\_post(&full); // Increase full slots

}

}

void consumer(int id) {

for (int i = 0; i < 10; ++i) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(150)); // Simulate work

sem\_wait(&full); // Decrease full slots

std::cout << "Consumer " << id << " is entering critical section." << std::endl;

mtx.lock(); // Lock the buffer (critical section)

int item = buffer.front();

buffer.pop();

std::cout << "Consumer " << id << " consumed: " << item << std::endl;

mtx.unlock(); // Unlock the buffer

std::cout << "Consumer " << id << " has exited critical section." << std::endl;

sem\_post(&empty); // Increase empty slots

}

}

int main() {

sem\_init(&empty, 0, MAX\_QUEUE\_SIZE); // Initialize empty slots

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

std::thread producers[2], consumers[2];

for (int i = 0; i < 2; ++i) {

producers[i] = std::thread(producer, i);

consumers[i] = std::thread(consumer, i);

}

for (int i = 0; i < 2; ++i) {

producers[i].join();

consumers[i].join();

}

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

8. Implementation of Classical problems (dining philosopher) using Threads and Semaphore.

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <unistd.h>

#define N 5

sem\_t forks[N];

void \*philosopher(void \*arg) {

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

for (int i = 0; i < 3; i++) {

printf("Philosopher %d is thinking.\n", id);

usleep(100000);

sem\_wait(&forks[id]);

sem\_wait(&forks[(id + 1) % N]);

printf("Philosopher %d is eating.\n", id);

usleep(100000);

sem\_post(&forks[id]);

sem\_post(&forks[(id + 1) % N]);

printf("Philosopher %d finished eating.\n", id);

usleep(100000);

}

return NULL;

}

int main() {

pthread\_t philosophers[N];

int ids[N];

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

sem\_init(&forks[i], 0, 1);

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

ids[i] = i;

pthread\_create(&philosophers[i], NULL, philosopher, &ids[i]);

}

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

pthread\_join(philosophers[i], NULL);

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

sem\_destroy(&forks[i]);

return 0;

}

9. Write a program to compute the finish time, turnaround time and waiting time for the First come First serve

import java.util.Scanner;

public class FCFS\_Simple {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of processes: ");

int n = scanner.nextInt();

int[] arrivalTime = new int[n];

int[] burstTime = new int[n];

int[] finishTime = new int[n];

int[] turnaroundTime = new int[n];

int[] waitingTime = new int[n];

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

System.out.print("Enter arrival time for process " + (i + 1) + ": ");

arrivalTime[i] = scanner.nextInt();

System.out.print("Enter burst time for process " + (i + 1) + ": ");

burstTime[i] = scanner.nextInt();

}

int currentTime = 0;

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

if (currentTime < arrivalTime[i]) {

currentTime = arrivalTime[i];

}

finishTime[i] = currentTime + burstTime[i];

turnaroundTime[i] = finishTime[i] - arrivalTime[i];

waitingTime[i] = turnaroundTime[i] - burstTime[i];

currentTime = finishTime[i];

}

System.out.println("\nProcess\t\tArrival\t\tBurst\t\tFinish\t\tTurnaround\t\tWaiting");

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

System.out.println((i + 1) + "\t\t" + arrivalTime[i] + "\t\t" + burstTime[i] + "\t\t" +

finishTime[i] + "\t\t" + turnaroundTime[i] + "\t\t\t" + waitingTime[i]);

}

scanner.close();

}

}

10. Write a program to compute the finish time, turnaround time and waiting time for the

Shortest Job First (Preemptive and Non Preemptive)

import java.util.Scanner;

class Process {

int arrivalTime;

int burstTime;

int waitingTime;

int turnaroundTime;

int finishTime;

Process(int arrivalTime, int burstTime) {

this.arrivalTime = arrivalTime;

this.burstTime = burstTime;

}

}

public class SimpleSJF {

public static void sjfNonPreemptive(Process[] processes, int n) {

int currentTime = 0;

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

int minIndex = -1;

int minBurst = Integer.MAX\_VALUE;

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

if (processes[j].arrivalTime <= currentTime && processes[j].finishTime == 0 && processes[j].burstTime < minBurst) {

minBurst = processes[j].burstTime;

minIndex = j;

}

}

if (minIndex == -1) {

currentTime++;

} else {

Process p = processes[minIndex];

p.finishTime = currentTime + p.burstTime;

p.turnaroundTime = p.finishTime - p.arrivalTime;

p.waitingTime = p.turnaroundTime - p.burstTime;

currentTime += p.burstTime;

}

}

}

public static void sjfPreemptive(Process[] processes, int n) {

int[] remainingBurst = new int[n];

int complete = 0, currentTime = 0, shortest = -1, minBurst = Integer.MAX\_VALUE;

boolean found;

for (int i = 0; i < n; i++) remainingBurst[i] = processes[i].burstTime;

while (complete < n) {

found = false;

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

if (processes[j].arrivalTime <= currentTime && remainingBurst[j] < minBurst && remainingBurst[j] > 0) {

minBurst = remainingBurst[j];

shortest = j;

found = true;

}

}

if (!found) {

currentTime++;

continue;

}

remainingBurst[shortest]--;

minBurst = remainingBurst[shortest] == 0 ? Integer.MAX\_VALUE : remainingBurst[shortest];

if (remainingBurst[shortest] == 0) {

complete++;

processes[shortest].finishTime = currentTime + 1;

processes[shortest].turnaroundTime = processes[shortest].finishTime - processes[shortest].arrivalTime;

processes[shortest].waitingTime = processes[shortest].turnaroundTime - processes[shortest].burstTime;

}

currentTime++;

}

}

public static void displayResults(Process[] processes, String type) {

System.out.println("\n" + type + " SJF Scheduling:");

System.out.printf("%-15s%-15s%-15s%-15s%-15s%n", "Arrival Time", "Burst Time", "Finish Time", "Turnaround Time", "Waiting Time");

for (Process p : processes) {

System.out.printf("%-15d%-15d%-15d%-15d%-15d%n", p.arrivalTime, p.burstTime, p.finishTime, p.turnaroundTime, p.waitingTime);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of processes: ");

int n = sc.nextInt();

Process[] processes = new Process[n];

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

System.out.print("Enter arrival time and burst time for process " + (i + 1) + ": ");

int arrivalTime = sc.nextInt();

int burstTime = sc.nextInt();

processes[i] = new Process(arrivalTime, burstTime);

}

Process[] processesNonPreemptive = new Process[n];

Process[] processesPreemptive = new Process[n];

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

processesNonPreemptive[i] = new Process(processes[i].arrivalTime, processes[i].burstTime);

processesPreemptive[i] = new Process(processes[i].arrivalTime, processes[i].burstTime);

}

sjfNonPreemptive(processesNonPreemptive, n);

displayResults(processesNonPreemptive, "Non-Preemptive");

sjfPreemptive(processesPreemptive, n);

displayResults(processesPreemptive, "Preemptive");

}

}

11. Write a program to compute the finish time, turnaround time and waiting time for the

Priority (Preemptive and Non Preemptive)

//C++ Implementation of Non-Preemptive Priority Scheduling Algorithm

#include <iostream>

using namespace std;

int main()

{

int n = 4; //Number of Processes

int CPU = 0; //CPU Current time

int arrivaltime[n] = {0, 2, 3, 5};

int bursttime[n] = {10, 5, 2, 20};

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

int ATt[n];

int NoP = n; //number of Processes

int PPt[n];

int waitingTime[n];

int turnaroundTime[n];

int i = 0;

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

{

PPt[i] = priority[i];

ATt[i] = arrivaltime[i];

}

int LAT = 0; //LastArrivalTime

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

if (arrivaltime[i] > LAT)

LAT = arrivaltime[i];

int MAX\_P = 0; //Max Priority

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

if (PPt[i] > MAX\_P)

MAX\_P = PPt[i];

int ATi = 0; //Pointing to Arrival Time indix

int P1 = PPt[0]; //Pointing to 1st priority Value

int P2 = PPt[0]; //Pointing to 2nd priority Value

//finding the First Arrival Time and Highest priority Process

int j = -1;

while (NoP > 0 && CPU <= 1000)

{

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

{

if ((ATt[i] <= CPU) && (ATt[i] != (LAT + 10)))

{

if (PPt[i] != (MAX\_P + 1))

{

P2 = PPt[i];

j = 1;

if (P2 < P1)

{

j = 1;

ATi = i;

P1 = PPt[i];

P2 = PPt[i];

}

}

}

}

if (j == -1)

{

CPU = CPU + 1;

continue;

}

else

{

waitingTime[ATi] = CPU - ATt[ATi];

CPU = CPU + bursttime[ATi];

turnaroundTime[ATi] = CPU - ATt[ATi];

ATt[ATi] = LAT + 10;

j = -1;

PPt[ATi] = MAX\_P + 1;

ATi = 0; //Pointing to Arrival Time index

P1 = MAX\_P + 1; //Pointing to 1st priority Value

P2 = MAX\_P + 1; //Pointing to 2nd priority Value

NoP = NoP - 1;

}

}

cout << "\nProcess\_Number\tBurst\_Time\tPriority\tArrival\_Time\tWaiting\_Time\tTurnaround\_Time\n\n";

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

{

cout << "P" << i + 1 << "\t\t" << bursttime[i] << "\t\t" << priority[i] << "\t\t" << arrivaltime[i] << "\t\t" << waitingTime[i] << "\t\t" << turnaroundTime[i] << endl;

}

float AvgWT = 0; //Average waiting time

float AVGTaT = 0; // Average Turn around time

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

{

AvgWT = waitingTime[i] + AvgWT;

AVGTaT = turnaroundTime[i] + AVGTaT;

}

cout << "Average waiting time = " << AvgWT / n << endl;

cout << "Average turnaround time = " << AVGTaT / n << endl;

}

12. Write a program to compute the finish time, turnaround time and waiting time for the

Round robin

import java.util.Scanner;

public class RoundRobinSimple {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of processes: ");

int numProcesses = scanner.nextInt();

int[] burstTime = new int[numProcesses];

int[] waitingTime = new int[numProcesses];

int[] turnaroundTime = new int[numProcesses];

int[] remainingTime = new int[numProcesses];

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

System.out.print("Enter burst time for process " + (i + 1) + ": ");

burstTime[i] = scanner.nextInt();

remainingTime[i] = burstTime[i];

}

System.out.print("Enter the time quantum: ");

int quantum = scanner.nextInt();

int time = 0;

while (true) {

boolean done = true;

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

if (remainingTime[i] > 0) {

done = false;

if (remainingTime[i] > quantum) {

time += quantum;

remainingTime[i] -= quantum;

} else {

time += remainingTime[i];

waitingTime[i] = time - burstTime[i];

remainingTime[i] = 0;

}

}

}

if (done) break;

}

System.out.println("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");

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

turnaroundTime[i] = burstTime[i] + waitingTime[i];

System.out.println("P" + (i + 1) + "\t\t" + burstTime[i] + "\t\t" + waitingTime[i] + "\t\t" + turnaroundTime[i]);

}

scanner.close();

}

}

13. Write a program to check whether given system is in safe state or not using Banker’s Deadlock Avoidance algorithm.

import java.util.Scanner;

public class BankersAlgorithm {

public static boolean isSafe(int processes, int resources, int[][] allocation, int[][] max, int[] available) {

int[] work = new int[resources];

boolean[] finish = new boolean[processes];

int[][] need = new int[processes][resources];

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

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

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

}

}

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

work[i] = available[i];

}

int count = 0;

while (count < processes) {

boolean found = false;

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

if (!finish[i]) {

boolean canFinish = true;

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

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

canFinish = false;

break;

}

}

if (canFinish) {

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

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

}

finish[i] = true;

found = true;

count++;

}

}

}

if (!found) {

return false; // Deadlock detected

}

}

return true; // Safe state

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of processes: ");

int processes = sc.nextInt();

System.out.print("Enter number of resources: ");

int resources = sc.nextInt();

int[][] allocation = new int[processes][resources];

System.out.println("Enter allocation matrix:");

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

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

allocation[i][j] = sc.nextInt();

}

}

int[][] max = new int[processes][resources];

System.out.println("Enter maximum matrix:");

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

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

max[i][j] = sc.nextInt();

}

}

int[] available = new int[resources];

System.out.println("Enter available resources:");

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

available[i] = sc.nextInt();

}

if (isSafe(processes, resources, allocation, max, available)) {

System.out.println("The system is in a safe state.");

} else {

System.out.println("The system is in a deadlock state.");

}

sc.close();

}

}

14. Write a program for Deadlock detection algorithm //same as 13

15. Write a program to calculate the number of page faults for a reference string for the FIFO page replacement algorithms:

import java.util.Scanner;

public class FIFOPageReplacement {

static final int MAX\_PAGES = 3;

static boolean isInMemory(int page, int[] memory, int numFrames) {

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

if (memory[i] == page) {

return true; // Page is in memory

}

}

return false; // Page is not in memory

}

static void displayMemory(int[] memory, int numFrames) {

System.out.print("Memory: ");

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

System.out.print(memory[i] + " ");

}

System.out.println();

}

static void FIFO(int[] pages, int numPages) {

int[] memory = new int[MAX\_PAGES];

for (int i = 0; i < MAX\_PAGES; i++) {

memory[i] = -1;

}

int front = 0;

int hits = 0, misses = 0;

System.out.print("Referenced String: ");

for (int page : pages) {

System.out.print(page + " ");

}

System.out.println();

System.out.println("FIFO Page Replacement:");

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

int currentPage = pages[i];

if (isInMemory(currentPage, memory, MAX\_PAGES)) {

hits++;

} else {

misses++;

memory[front] = currentPage;

front = (front + 1) % MAX\_PAGES;

}

displayMemory(memory, MAX\_PAGES);

}

System.out.println("Hits: " + hits);

System.out.println("Misses: " + misses);

System.out.println("Page Faults: " + misses);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of pages: ");

int numPages = scanner.nextInt();

int[] pages = new int[numPages];

System.out.println("Enter the page references:");

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

pages[i] = scanner.nextInt();

}

// Call the FIFO function with the inputted pages

FIFO(pages, numPages);

scanner.close();

}

}

16. Write a program to calculate the number of page faults for a reference string for the LRU page replacement algorithms:

import java.util.\*;

public class LRUPageReplacement {

public static int calculatePageFaults(int[] pages, int frameSize) {

Set<Integer> frames = new HashSet<>(frameSize); // Pages currently in frame

Map<Integer, Integer> pageIndices = new HashMap<>(); // Tracks the last used index for each page

int pageFaults = 0;

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

int page = pages[i];

// Check if the page is already in frame (no page fault)

if (!frames.contains(page)) {

pageFaults++; // Page fault occurs

// If frame is full, remove the least recently used page

if (frames.size() == frameSize) {

int lruPage = findLeastRecentlyUsedPage(pageIndices, frames);

frames.remove(lruPage);

pageIndices.remove(lruPage);

}

frames.add(page);

}

// Update the page's most recent index

pageIndices.put(page, i);

}

return pageFaults;

}

// Finds the least recently used page in the current set of frames

private static int findLeastRecentlyUsedPage(Map<Integer, Integer> pageIndices, Set<Integer> frames) {

int lruPage = -1;

int oldestIndex = Integer.MAX\_VALUE;

for (int page : frames) {

int lastUsedIndex = pageIndices.get(page);

if (lastUsedIndex < oldestIndex) {

oldestIndex = lastUsedIndex;

lruPage = page;

}

}

return lruPage;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of pages in the reference string: ");

int n = scanner.nextInt();

int[] pages = new int[n];

System.out.println("Enter the reference string:");

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

pages[i] = scanner.nextInt();

}

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

int frameSize = scanner.nextInt();

int pageFaults = calculatePageFaults(pages, frameSize);

System.out.println("Total Page Faults: " + pageFaults);

}

}

17. Write a program to calculate the number of page faults for a reference string for the Optimal page replacement algorithms:

import java.util.Scanner;

public class OptimalPageReplacement {

static final int MAX\_PAGES = 3;

static boolean isInMemory(int page, int[] memory, int numFrames) {

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

if (memory[i] == page) {

return true; // Page is in memory

}

}

return false; // Page is not in memory

}

static void displayMemory(int[] memory, int numFrames) {

System.out.print("Memory: ");

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

System.out.print(memory[i] + " ");

}

System.out.println();

}

static int findOptimal(int[] pages, int start, int numPages, int[] memory, int numFrames) {

int farthest = -1;

int farthestIndex = -1;

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

int currentPage = memory[i];

int j;

for (j = start; j < numPages; j++) {

if (pages[j] == currentPage) {

if (j > farthest) {

farthest = j;

farthestIndex = i;

}

break;

}

}

if (j == numPages) {

return i; // If a page will not be referenced in the future, replace it

}

}

if (farthestIndex == -1) {

return 0; // Default case: replace the first page if no other page is farthest

}

return farthestIndex;

}

static void OPT(int[] pages, int numPages) {

int[] memory = new int[MAX\_PAGES];

// Initialize memory to 0 to match the C output

for (int i = 0; i < MAX\_PAGES; i++) memory[i] = 0;

int hits = 0;

int misses = 0;

System.out.print("Referenced String: ");

for (int page : pages) {

System.out.print(page + " ");

}

System.out.println();

System.out.println("OPT Page Replacement:");

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

int currentPage = pages[i];

if (isInMemory(currentPage, memory, MAX\_PAGES)) {

hits++;

} else {

misses++;

int replaceIndex = findOptimal(pages, i + 1, numPages, memory, MAX\_PAGES);

memory[replaceIndex] = currentPage;

}

displayMemory(memory, MAX\_PAGES);

}

System.out.println("Hits: " + hits);

System.out.println("Misses: " + misses);

System.out.println("Page Faults: " + misses);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of pages: ");

int numPages = scanner.nextInt();

int[] pages = new int[numPages];

System.out.println("Enter the page references:");

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

pages[i] = scanner.nextInt();

}

OPT(pages, numPages);

scanner.close();

}

}

18. Write a program to simulate FCFS disk scheduling. Calculate total seek time.Print accepted input and output in tabular format

import java.util.Scanner;

public class FCFS\_DiskScheduling {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the starting position of the disk: ");

int start = scanner.nextInt();

System.out.print("Enter the number of disk requests: ");

int n = scanner.nextInt();

int[] requests = new int[n];

System.out.println("Enter the disk positions (separate with spaces): ");

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

requests[i] = scanner.nextInt();

}

int totalSeekTime = 0;

int currentPos = start;

System.out.println("\nRequest No.\tDisk Position\tSeek Time");

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

int seekTime = Math.abs(requests[i] - currentPos); // Seek time for each request

totalSeekTime += seekTime; // Add to total seek time

System.out.println((i + 1) + "\t\t" + requests[i] + "\t\t" + seekTime);

currentPos = requests[i]; // Move disk head to the new position

}

System.out.println("\nTotal Seek Time: " + totalSeekTime);

}

}

19. Write a program to simulate SSTF disk scheduling. Calculate total seek time.Print accepted input and output in tabular format

import java.util.Scanner;

public class SimpleSSTF {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the starting position of the disk: ");

int start = scanner.nextInt();

System.out.print("Enter the number of disk requests: ");

int n = scanner.nextInt();

int[] requests = new int[n];

System.out.println("Enter the disk positions:");

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

requests[i] = scanner.nextInt();

}

int totalSeekTime = 0;

int currentPos = start;

System.out.println("\nRequest No.\tDisk Position\tSeek Time");

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

int minSeekTime = Integer.MAX\_VALUE;

int closestRequest = -1;

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

if (requests[j] != -1) {

int seekTime = Math.abs(requests[j] - currentPos);

if (seekTime < minSeekTime) {

minSeekTime = seekTime;

closestRequest = j;

}

}

}

System.out.println((i + 1) + "\t\t" + requests[closestRequest] + "\t\t" + minSeekTime);

totalSeekTime += minSeekTime;

currentPos = requests[closestRequest];

requests[closestRequest] = -1;

}

System.out.println("\nTotal Seek Time: " + totalSeekTime);

}

}

20. Write a program to simulate SCAN disk scheduling. Calculate total seek time.Print accepted input and output in tabular format

import java.util.Arrays;

import java.util.Scanner;

public class SCANDiskScheduling {

public static void scanScheduling(int[] requests, int start, int totalCylinders) {

Arrays.sort(requests);

System.out.println("\nDisk Request Positions (Sorted):");

System.out.println(Arrays.toString(requests));

int totalSeekTime = 0;

int currentPosition = start;

System.out.println("\nDisk Position\tSeek Time");

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

if (requests[i] >= currentPosition) {

int seekTime = Math.abs(requests[i] - currentPosition);

totalSeekTime += seekTime;

currentPosition = requests[i];

System.out.println(requests[i] + "\t\t" + seekTime);

}

}

for (int i = requests.length - 1; i >= 0; i--) {

if (requests[i] < currentPosition) {

int seekTime = Math.abs(requests[i] - currentPosition);

totalSeekTime += seekTime;

currentPosition = requests[i];

System.out.println(requests[i] + "\t\t" + seekTime);

}

}

System.out.println("\nTotal Seek Time: " + totalSeekTime);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the total number of cylinders: ");

int totalCylinders = scanner.nextInt();

System.out.print("Enter the starting position of the disk: ");

int start = scanner.nextInt();

System.out.print("Enter the number of disk requests: ");

int n = scanner.nextInt();

int[] requests = new int[n];

System.out.println("Enter the disk request positions:");

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

requests[i] = scanner.nextInt();

}

scanScheduling(requests, start, totalCylinders);

}

}

21. Write a program to simulate C-SCAN disk scheduling. Calculate total seek time.Print accepted input and output in tabular format

import java.util.Arrays;

import java.util.Scanner;

public class CScanDiskScheduling {

public static void cscanScheduling(int[] requests, int start, int totalCylinders) {

Arrays.sort(requests);

System.out.println("\nDisk Request Positions (Sorted):");

System.out.println(Arrays.toString(requests));

int totalSeekTime = 0;

int currentPosition = start;

System.out.println("\nDisk Position\tSeek Time");

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

if (requests[i] >= currentPosition) {

int seekTime = Math.abs(requests[i] - currentPosition);

totalSeekTime += seekTime;

currentPosition = requests[i];

System.out.println(requests[i] + "\t\t" + seekTime);

}

}

totalSeekTime += currentPosition; // Seek time from current position to 0

currentPosition = 0;

System.out.println("0\t\t" + currentPosition);

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

if (requests[i] > currentPosition) {

int seekTime = Math.abs(requests[i] - currentPosition);

totalSeekTime += seekTime;

currentPosition = requests[i];

System.out.println(requests[i] + "\t\t" + seekTime);

}

}

System.out.println("\nTotal Seek Time: " + totalSeekTime);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the total number of cylinders: ");

int totalCylinders = scanner.nextInt();

System.out.print("Enter the starting position of the disk: ");

int start = scanner.nextInt();

System.out.print("Enter the number of disk requests: ");

int n = scanner.nextInt();

int[] requests = new int[n];

System.out.println("Enter the disk request positions:");

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

requests[i] = scanner.nextInt();

}

cscanScheduling(requests, start, totalCylinders);

}

}

22. Write a program for following 1) zombie process 2)orphan processes 3)sum of even numbers of an array in parent and odd numbers of an array in child process

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

void createZombieProcess() {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

if (pid > 0) { // Parent process

printf("Parent process: Zombie process created. PID = %d\n", pid);

sleep(10);

}

else { // Child process

printf("Child process exiting to become zombie.\n");

exit(0);

}

}

void createOrphanProcess() {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

if (pid > 0) { // Parent process

printf("Parent process exiting to create orphan process.\n");

exit(0);

}

else { // Child process

sleep(5);

printf("Child process (orphan) continuing after parent termination. PID = %d\n", getpid());

}

}

void sumEvenOdd(int arr[], int size) {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

if (pid > 0) { // Parent process

int evenSum = 0;

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

if (arr[i] % 2 == 0) {

evenSum += arr[i];

}

}

printf("Parent process: Sum of even numbers = %d\n", evenSum);

wait(NULL);

}

else { // Child process

int oddSum = 0;

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

if (arr[i] % 2 != 0) {

oddSum += arr[i];

}

}

printf("Child process: Sum of odd numbers = %d\n", oddSum);

exit(0);

}

}

int main() {

int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int size = sizeof(arr) / sizeof(arr[0]);

printf("Calculating sum of even and odd numbers:\n");

sumEvenOdd(arr, size);

printf("\nCreating a zombie process:\n");

createZombieProcess();

sleep(5);

printf("\nCreating an orphan process:\n");

createOrphanProcess();

return 0;

}

23. Write a shell script to perform following operations on student database.

a) Insert b) Delete c)Update d)Search

#!/bin/bash

DATABASE="student\_db.txt"

# Function to insert a new student record

insert\_student() {

echo "Enter Student ID:"

read student\_id

echo "Enter Name:"

read name

echo "Enter Grade:"

read grade

echo "$student\_id,$name,$grade" >> "$DATABASE"

echo "Student record inserted successfully."

}

# Function to delete a student record by ID

delete\_student() {

echo "Enter Student ID to delete:"

read student\_id

grep -v "^$student\_id," "$DATABASE" > temp\_db.txt && mv temp\_db.txt "$DATABASE"

echo "Student record deleted successfully (if found)."

}

# Function to update a student record by ID

update\_student() {

echo "Enter Student ID to update:"

read student\_id

grep -v "^$student\_id," "$DATABASE" > temp\_db.txt && mv temp\_db.txt "$DATABASE"

echo "Enter new Name:"

read name

echo "Enter new Grade:"

read grade

echo "$student\_id,$name,$grade" >> "$DATABASE"

echo "Student record updated successfully."

}

# Function to search for a student record by ID

search\_student() {

echo "Enter Student ID to search:"

read student\_id

grep "^$student\_id," "$DATABASE" || echo "Student not found."

}

# Main menu

while true; do

echo "Select an option:"

echo "1) Insert Student"

echo "2) Delete Student"

echo "3) Update Student"

echo "4) Search Student"

echo "5) Exit"

read choice

case $choice in

1) insert\_student ;;

2) delete\_student ;;

3) update\_student ;;

4) search\_student ;;

5) exit ;;

\*) echo "Invalid choice. Please try again." ;;

esac

done

24. Write a program to read and copy the contents of file character by character, line by line.

import java.io.\*;

public class FileCopyExample {

public static void main(String[] args) {

String sourceFile = "source.txt";

String destinationFileChar = "destination\_char.txt";

String destinationFileLine = "destination\_line.txt";

copyCharacterByCharacter(sourceFile, destinationFileChar);

copyLineByLine(sourceFile, destinationFileLine);

}

public static void copyCharacterByCharacter(String sourceFile, String destinationFile) {

FileReader reader = null;

FileWriter writer = null;

try {

reader = new FileReader(sourceFile);

writer = new FileWriter(destinationFile);

int character;

while ((character = reader.read()) != -1) {

writer.write(character);

}

System.out.println("File copied character by character to " + destinationFile);

} catch (IOException e) {

System.out.println("Error during character-by-character copy: " + e.getMessage());

} finally {

try {

if (reader != null) reader.close();

if (writer != null) writer.close();

} catch (IOException e) {

System.out.println("Error closing file resources: " + e.getMessage());

}

}

}

public static void copyLineByLine(String sourceFile, String destinationFile) {

BufferedReader reader = null;

BufferedWriter writer = null;

try {

reader = new BufferedReader(new FileReader(sourceFile));

writer = new BufferedWriter(new FileWriter(destinationFile));

String line;

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine(); // Add a new line after each line

}

System.out.println("File copied line by line to " + destinationFile);

} catch (IOException e) {

System.out.println("Error during line-by-line copy: " + e.getMessage());

} finally {

try {

if (reader != null) reader.close();

if (writer != null) writer.close();

} catch (IOException e) {

System.out.println("Error closing file resources: " + e.getMessage());

}

}

}

}

25. Write a program to load ALP program from input file to main memory.

import java.io.\*;

import java.util.\*;

public class ALPLoader {

// Simulating the main memory as an array of strings

static final int MEMORY\_SIZE = 1024; // Memory size

static String[] memory = new String[MEMORY\_SIZE];

public static void main(String[] args) {

String inputFile = "input.alp"; // Input ALP file

try {

loadALPToMemory(inputFile);

System.out.println("ALP Program loaded into memory successfully.");

// Optionally, display the loaded memory

displayMemory();

} catch (IOException e) {

System.err.println("Error loading ALP program: " + e.getMessage());

}

}

// Method to load the ALP program from the input file into memory

public static void loadALPToMemory(String filename) throws IOException {

BufferedReader reader = new BufferedReader(new FileReader(filename));

String line;

int address = 0;

while ((line = reader.readLine()) != null && address < MEMORY\_SIZE) {

memory[address] = line.trim(); // Load the instruction into memory

address++;

}

reader.close();

}

// Method to display the memory content (for debugging purposes)

public static void displayMemory() {

System.out.println("Memory contents:");

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i] != null) {

System.out.println("Address " + i + ": " + memory[i]);

}

}

}

}

LOAD R1, 100

ADD R2, R3

MOV R4, R5

HALT

26. Write a program to check Opcode error in a given job and raise an interrupt.

#include <iostream>

#include <string>

#include <vector>

using namespace std;

// Globals for interrupts

int PI = 0; // Program Interrupt

// Interrupt Service Routine

void MOS() {

if (PI == 4) { // Opcode error

cout << "Interrupt Raised: Opcode Error Detected." << endl;

cout << "Terminating the job due to invalid opcode." << endl;

}

}

// Function to check the validity of an opcode

bool isValidOpcode(const string &opcode) {

// List of valid opcodes

vector<string> validOpcodes = {"GD", "PD", "LR", "SR", "CR", "H", "BT"};

for (const auto &valid : validOpcodes) {

if (opcode == valid) {

return true;

}

}

return false;

}

// Function to simulate opcode execution

void executeJob(const vector<string> &job) {

for (const auto &instruction : job) {

string opcode = instruction.substr(0, 2); // Extract first 2 characters as opcode

if (!isValidOpcode(opcode)) {

cout << "Invalid Opcode Found: " << opcode << endl;

PI = 4; // Set Program Interrupt for Opcode Error

MOS(); // Call interrupt service routine

return; // Terminate job due to error

}

cout << "Executing: " << instruction << endl;

}

cout << "Job executed successfully without errors." << endl;

}

int main() {

// Sample job containing instructions

vector<string> job = {

"GD 100", // Valid opcode

"PD 200", // Valid opcode

"LR 300", // Valid opcode

"SR 400", // Invalid opcode

"H" // Valid opcode

};

cout << "Starting Job Execution..." << endl;

executeJob(job);

return 0;

}

27. Write a program to check Operand error in a given job and raise an interrupt.

import java.util.Scanner;

class OS\_Simulation {

static final int OPERAND\_ERROR\_INTERRUPT = 2;

// Method to simulate operand checking

public static void checkOperandError(String job) {

// Assuming job contains commands in the format "OPCODE OPERAND"

String[] instructions = job.split(";");

for (String instruction : instructions) {

String[] parts = instruction.split(" ");

if (parts.length != 2) {

raiseInterrupt(OPERAND\_ERROR\_INTERRUPT);

System.out.println("Operand Error: Invalid instruction format - " + instruction);

return; // Exit after the first error

}

String opcode = parts[0];

String operand = parts[1];

if (!isValidOperand(operand)) {

raiseInterrupt(OPERAND\_ERROR\_INTERRUPT);

System.out.println("Operand Error: Invalid Operand - " + operand);

return; // Exit after the first error

}

}

System.out.println("All Operands are valid.");

}

// Method to check if an operand is valid

private static boolean isValidOperand(String operand) {

try {

// Operand is valid if it's an integer (simulating the operand being a memory address or register)

Integer.parseInt(operand);

return true;

} catch (NumberFormatException e) {

return false; // Invalid operand if it can't be parsed to an integer

}

}

// Method to raise an interrupt

private static void raiseInterrupt(int interruptCode) {

// In a real OS, this would trigger interrupt handling

System.out.println("Interrupt raised with code: " + interruptCode);

}

public static void main(String[] args) {

// Create a Scanner object to read user input

Scanner scanner = new Scanner(System.in);

// Prompt the user to input instructions

System.out.println("Enter instructions in the format 'OPCODE OPERAND', separated by ';':");

String job = scanner.nextLine();

// Pass the input to check for operand errors

checkOperandError(job);

// Close the scanner

scanner.close();

}

}