**DATE:**

**EXPERIMENT NO: 1**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

# Experiment 1

1. **To the study the functions of an operating system.**

The operating system (OS) is the core software that manages hardware resources and provides essential services for computer programs. Its functions can be broadly categorized into several key areas:

# Process Management:

* + - Scheduling: Determines which processes get to use the CPU and when, optimizing resource utilization and responsiveness.
    - Process Creation and Termination: Handles the creation, execution, and termination of processes, including memory allocation and cleanup.
    - Inter-process Communication: Facilitates communication and data sharing between processes, often through mechanisms like shared memory or message passing.

# Memory Management:

* + - Allocation: Allocates and de-allocates memory for processes, ensuring efficient utilization and preventing memory leaks.
    - Virtual Memory: Manages the mapping of virtual memory addresses to physical memory, enabling processes to use more memory than physically available through techniques like paging and segmentation.

# File System Management:

* + - File Creation, Deletion, and Manipulation: Provides interfaces for creating, deleting, and manipulating files and directories.
    - File Access Control: Enforces permissions and access controls to ensure data security and integrity.
    - File System Maintenance: Handles tasks such as disk space allocation, fragmentation management, and data backup and recovery.

# Device Management:

* + - Device Drivers: Provides interfaces for communicating with hardware devices, abstracting device-specific details and enabling device-independent access.
    - I/O Management: Manages input and output operations, including buffering, caching, and error handling, to optimize performance and reliability.
    - Device Allocation: Allocates and schedules access to devices among competing processes, preventing conflicts and ensuring fair resource sharing.

# User Interface:

* + - Command Line Interface (CLI): Provides a text-based interface for users to interact with the system through commands.
    - Graphical User Interface (GUI): Presents a visual interface with windows, icons, menus, and pointers (WIMP), making it easier for users to interact with applications and manage files.

1. Write the difference between Linux OS with different OS

|  |  |  |
| --- | --- | --- |
|  | Linux | Windows |
| Kernel Type | Uses a monolithic kernel, which means that the entire operating system (OS)  runs in a single address space for efficiency and performance. | Uses a hybrid kernel, combining aspects of monolithic and microkernel designs to balance performance and modularity. |
| Source Code | Open-source, meaning its source code  is freely available and can be modified and distributed by anyone. | Closed-source, meaning its source code  is proprietary and not available for public modification or distribution. |
| Cost | Generally free to use and distribute, with various distributions available at  no cost. | Commercial software that requires a license to use, typically purchased per  device or user. |
| User Interface | Offers multiple graphical user interfaces (GUIs) like GNOME, KDE, and XFCE, as  well as a powerful command-line interface (CLI). | Has a consistent GUI provided by Microsoft, known for its Start Menu and  taskbar, with PowerShell and Command Prompt available for CLI. |
| File System Support | Supports a wide range of file systems,  including ext4, Btrfs, XFS, and more. | Primarily uses NTFS, with support for  FAT32 and exFAT. |
| Security | Considered to be more secure due to its open-source nature, which allows for faster detection and patching of  vulnerabilities, and its permissions system | More targeted by malware due to its widespread use, though it has robust security features like Windows Defender and BitLocker. |

**DATE:**

**EXPERIMENT NO: 2**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

Experiment 2

# What is System Call

A system call is a programming interface provided by the operating system that allows user-level processes (applications) to request services from the kernel. These services typically involve tasks that require privileged access to system resources or functionality that user-space programs cannot perform directly. System calls act as a bridge between user space and kernel space in an operating system.

# Write the program to program to get the information of system.

#include <stdio.h> #include <stdlib.h> #include <sys/utsname.h> #include <unistd.h>

int main() {

// Structure to store system information struct utsname sys\_info;

// Get the system information

if (uname(&sys\_info) != 0) { perror("uname"); exit(EXIT\_FAILURE);

}

// Print the system information

printf("System Name: %s\n", sys\_info.sysname); printf("Node Name: %s\n", sys\_info.nodename); printf("Release: %s\n", sys\_info.release); printf("Version: %s\n", sys\_info.version); printf("Machine: %s\n", sys\_info.machine);

// Get and print the hostname char hostname[1024];

if (gethostname(hostname, sizeof(hostname)) == 0) { printf("Hostname: %s\n", hostname);

} else {

perror("gethostname");

}

return 0;

}

Output

System Name: Linux

Node Name: 1e9153c09987 Release: 5.15.0-1060-gcp

Version: #68~20.04.1-Ubuntu SMP Wed May 1 14:35:27 UTC 2024 Machine: x86\_64

Hostname: 1e9153c09987

* 1. **Implement Sleep system call.** #include <stdio.h> #include <unistd.h>

int main() {

printf("Sleeping for 7 seconds...\n"); sleep(7);

printf("executed!\n"); return 0;

}

Output

Sleeping for 7 seconds... executed!

* 1. **Fork system call.** #include <stdio.h> #include <unistd.h>

int main() {

pid\_t child\_pid; child\_pid = fork();

if (child\_pid == -1) {

perror("fork"); return 1;

} else if (child\_pid == 0) {

// Child process

printf("Child process: PID=%d\n", getpid());

// Additional child process logic here

} else {

// Parent process

printf("Parent process: PID=%d, Child PID=%d\n", getpid(), child\_pid);

// Additional parent process logic here

}

return 0;

}

Output

Parent process: PID=17157, Child PID=17158 Child process: PID=17158

# Implement Exec() System call.

#include <stdio.h> #include <unistd.h>

int main() {

// Command to execute char \*command = "ls";

// Arguments for the command

char \*args[] = {"ls", "-l", NULL};

// Execute the command

if (execvp(command, args) == -1) { perror("execvp");

return 1;

}

// This line will only be reached if execvp fails printf("Executed successfully!\n");

return 0;

}

**DATE:**

**EXPERIMENT NO: 3**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

# Experiment 3

1. **Explain the need of CPU scheduling algorithm**

CPU scheduling is a process that allows one process to use cpu while other processes are delayed due to unavailability of resources like i/o resources etc. The primary goals of these algorithms are to optimize CPU utilization, ensure fair process execution, reduce waiting time, maximize throughput, and enhance system performance.

**. Efficient CPU Utilization:**Maximize the usage of the CPU by ensuring it is busy as much as possible.

**.Fairness:** Ensure that all processes get a fair share of the CPU.

**. Minimizing Waiting Time:** Reduce the time processes spend waiting in the ready queue.

**. Minimizing Turnaround Time:** Reduce the total time taken from process submission to process completion

**. Minimizing Response Time:** Decrease the time from when a request is submitted until the first response is produced.

**. Preventing Deadlock and Starvation:** Avoid situations where processes are indefinitely waiting for resources (deadlock) and ensure that all processes get a chance to execute (preventing starvation)

# Discuss the various performance parameter.

In CPU scheduling, performance parameters are metrics used to evaluate the effectiveness and efficiency of different scheduling algorithms

# CPU Utilization

* + **Definition:** The percentage of time the CPU is actively working on processes.
  + **Objective:** Maximize CPU utilization to ensure that the CPU is not idle.

# Throughput

* + **Definition:** The number of processes completed per unit of time.
  + **Objective:** Maximize throughput to increase the number of processes that are executed and completed in a given period.

# Turnaround Time

* + **Definition:** The total time taken from the submission of a process to its completion.
  + **Objective:** Minimize turnaround time to ensure quick processing and completion of tasks.

# Waiting Time

* + **Definition:** The total time a process spends waiting in the ready queue before getting the CPU.
  + **Objective:** Minimize waiting time to reduce the time processes spend waiting to be executed.

# Response Time

* + **Definition:** The time from the submission of a process until the first response is produced (i.e., the time it takes from when a process arrives to when it starts execution).
  + **Objective:** Minimize response time to improve the responsiveness of interactive systems.

# Write short notes on CPU Scheduler and Dispatch latency

* **CPU Scheduler:**
  + The CPU scheduler is responsible for selecting and managing process execution based on various scheduling algorithms (e.g., FCFS, SJF, RR). Its decisions directly influence the efficiency of CPU utilization, ensuring processes are executed in a fair and responsive manner. Effective scheduling optimizes performance metrics like throughput, turnaround time, waiting time, and response time.

# Dispatch Latency:

* + Dispatch latency is the time taken to stop one process and start or resume another, encompassing context switch time, scheduler overhead, and interrupt handling. Low dispatch latency is crucial for maintaining system responsiveness, especially in real-time and interactive applications, ensuring that high-priority tasks are promptly addressed.

# Impact on System Performance:

* + Both the CPU scheduler and dispatch latency are critical in designing operating systems that balance performance metrics to provide a smooth and efficient computing experience. The scheduler's efficient process management and minimal dispatch latency contribute to high CPU utilization, quick system responses, and optimal overall system performance.

# Write FCFS CPU scheduling algorithm and Implement in C/C++/JAVA

#include <stdio.h> int main() {

int pid[4], bt[4], wt[4], i;

int n = 4, tat = 0, twt = 0;

printf("Enter process ID of all processes: "); for (int i = 0; i < n; i++) {

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

}

printf("Enter all burst times: "); for (int i = 0; i < n; i++) {

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

}

wt[0] = 0;

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

wt[i] = bt[i - 1] + wt[i - 1];

}

for (i = 0; i < n; i++) { tat += (bt[i] + wt[i]); twt += wt[i];

printf("Waiting time %d = %d\n", i + 1, wt[i]); printf("Turnaround time %d = %d\n", i + 1, bt[i] + wt[i]);

}

printf("Average waiting time = %d\n", twt / n); printf("Average turnaround time = %d\n", tat / n);

return 0;

}

# Output:

Enter process ID of all processes: 1 2 3 4

Enter all burst times: 3 3 4 5

Waiting time 1 = 0

Turnaround time 1 = 3

Waiting time 2 = 3

Turnaround time 2 = 6

Waiting time 3 = 6

Turnaround time 3 = 10

Waiting time 4 = 10

Turnaround time 4 = 15 Average waiting time = 4 Average turnaround time = 8

**DATE:**

**EXPERIMENT NO: 4**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

# Experiment 4

1. **Write SJF CPU scheduling algorithm and Implement in C/C++/JAVA**

import java.util.ArrayList; import java.util.Collections; import java.util.List; public class SJF\_Scheduling {

static class Process { String name;

int burstTime;

public Process(String name, int burstTime) { this.name = name;

this.burstTime = burstTime;

}

}

public static void main(String[] args) { List<Process> processes = new ArrayList<>(); processes.add(new Process("P1", 4));

processes.add(new Process("P2", 3));

processes.add(new Process("P3", 1));

processes.add(new Process("P4", 2));

processes.add(new Process("P5", 5));

Collections.sort(processes, (p1, p2) -> p1.burstTime - p2.burstTime); int currentTime = 0;

double totalWaitingTime = 0; double totalTurnaroundTime = 0;

System.out.println("Process\tBurst Time\tWaiting Time\tTurnaround Time"); for (Process process : processes) {

int waitingTime = Math.max(0, currentTime); totalWaitingTime += waitingTime;

int turnaroundTime = waitingTime + process.burstTime; totalTurnaroundTime += turnaroundTime;

currentTime += process.burstTime;

System.out.println(process.name + "\t\t" + process.burstTime + "\t\t\t"

+

waitingTime + "\t\t\t\t" + turnaroundTime);

}

double avgWaitingTime = totalWaitingTime / processes.size(); double avgTurnaroundTime = totalTurnaroundTime / processes.size(); System.out.println("\nAverage Waiting Time: " + avgWaitingTime);

System.out.println("Average Turnaround Time: " + avgTurnaroundTime);

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| Output:  Process | Burst Time | Waiting Time | Turnaround Time |
| P3 | 1 | 0 | 1 |
| P4 | 2 | 1 | 3 |
| P2 | 3 | 3 | 6 |
| P1 | 4 | 6 | 10 |
| P5 | 5 | 10 | 15 |

Average Waiting Time: 4.0 Average Turnaround Time: 7.0

# Write Round Robin CPU scheduling algorithm and Implement in C/C++/JAVA to calculate waiting time, turnaround time, average waiting time, and average turnaround time for given process. Note time quantum is 1

#include <stdio.h> #include <stdlib.h>

// Function to find the waiting time for all processes

void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum) { int rem\_bt[n]; // Remaining burst times

for (int i = 0; i < n; i++) rem\_bt[i] = bt[i];

int t = 0; // Current time while (1) {

int done = 1;

for (int i = 0; i < n; i++) { if (rem\_bt[i] > 0) {

done = 0; // There is a pending process

if (rem\_bt[i] > quantum) { t += quantum; rem\_bt[i] -= quantum;

} else {

t += rem\_bt[i]; wt[i] = t - bt[i]; rem\_bt[i] = 0;

}

}

}

if (done == 1) break;

}

}

// Function to calculate turnaround time

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) { for (int i = 0; i < n; i++)

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

}

// Function to calculate average time

void findAvgTime(int processes[], int n, int bt[], int quantum) { int wt[n], tat[n];

int total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, bt, wt, quantum); findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes Burst time Waiting time Turnaround time\n"); for (int i = 0; i < n; i++) {

total\_wt += wt[i];

total\_tat += tat[i];

printf(" %d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], wt[i], tat[i]);

}

printf("Average waiting time = %.2f\n", (float)total\_wt / (float)n); printf("Average turnaround time = %.2f\n", (float)total\_tat / (float)n);

}

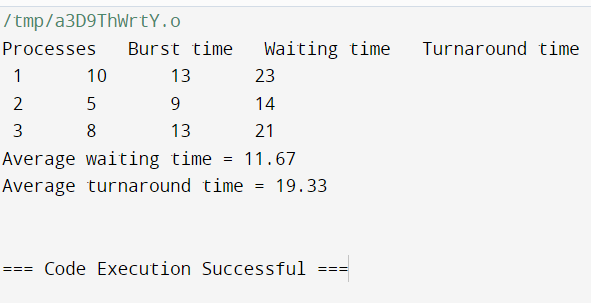
int main() {

int processes[] = {1, 2, 3};

int n = sizeof(processes) / sizeof(processes[0]); int burst\_time[] = {10, 5, 8};

int quantum = 1;

findAvgTime(processes, n, burst\_time, quantum); return 0;



}

**DATE:**

**EXPERIMENT NO: 5**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

# Experiment 5

1. **Implementation of Semaphores**

import java.util.\*; class Semaphores {

public enum Value { ZERO, ONE }

public Queue<Process> q = new LinkedList<>(); public Value value = Value.ONE;

public void P(Semaphores s, Process p) { if (s.value == Value.ONE) {

s.value = Value.ZERO;

} else {

q.add(p);

p.sleep();

}

}

public void V(Semaphores s) { if (s.q.size() == 0) {

s.value = Value.ONE;

} else {

Process p = q.peek(); q.remove();

p.wakeUp();

}

}

}

class Process {

private String name;

public Process(String name) { this.name = name;

}

public void sleep() {

System.out.println(name + " goes to sleep");

}

public void wakeUp() { System.out.println(name + " wakes up");

}

}

public class Main {

public static void main(String[] args) {

Semaphores s = new Semaphores(); Process p1 = new Process("A"); Process p2 = new Process("B");

System.out.println("A requests access to critical section"); s.P(s, p1);

System.out.println("A enters critical section");

System.out.println("B requests access"); s.P(s, p2);

System.out.println("B enters critical section");

System.out.println("A releases access"); s.V(s);

System.out.println("A releases access"); s.V(s);

}

}

Output

A requests access to critical section A enters critical section

B requests access B goes to sleep

B enters critical section A releases access

B wakes up

A releases access

# Creating threads to access shared variables using semaphores

import java.util.concurrent.\*;

class Shared {

static int count = 0;

}

class MyThread extends Thread { Semaphore sem;

String tname;

public MyThread(Semaphore sem, String tname) {

super(tname); this.sem = sem; this.tname = tname;

}

@Override

public void run() {

if (this.getName().equals("A")) { System.out.println("Starting " + tname);

try {

System.out.println(tname + " is waiting for a permit"); sem.acquire();

System.out.println(tname + " gets a permit");

for (int i = 0; i < 5; i++) { Shared.count++;

System.out.println(tname + ": " + Shared.count); Thread.sleep(10);

}

} catch (InterruptedException e) { System.out.println(e);

}

System.out.println(tname + " releases "); sem.release();

} else {

System.out.println("Starting " + tname);

try {

System.out.println(tname + " is waiting for a permit"); sem.acquire();

System.out.println(tname + " gets a permit");

for (int i = 0; i < 5; i++) { Shared.count++;

System.out.println(tname + ": " + Shared.count); Thread.sleep(10);

}

} catch (InterruptedException e) { System.out.println(e);

}

System.out.println(tname + " releases "); sem.release();

}

}

}

public class Demo {

public static void main(String[] args) { Semaphore sem = new Semaphore(1);

MyThread m1 = new MyThread(sem, "A"); MyThread m2 = new MyThread(sem, "B");

m1.start();

m2.start();

try {

m1.join();

m2.join();

} catch (InterruptedException e) { System.out.println(e);

}

System.out.println("count: " + Shared.count);

}

}

Output:

Starting B Starting A

B is waiting for a permit A is waiting for a permit A gets a permit

A: 1

A: 2

A: 3

A: 4

A: 5

A releases

B gets a permit B: 6

B: 7

B: 8

B: 9

B: 10

B releases count: 10

# State Dining Philosophers Problem. Write C program to solve Dining Philosophers Problem using semaphore.

import java.util.concurrent.Semaphore;

public class Main {

static int no\_phil = 2;

static Semaphore[] fork = new Semaphore[no\_phil]; static Semaphore sem = new Semaphore(1);

public static void main(String[] args) { for (int i = 0; i < no\_phil; i++) {

fork[i] = new Semaphore(1);

}

Thread[] p = new Thread[no\_phil]; for (int i = 0; i < no\_phil; i++) {

p[i] = new Thread(new Philosopher(i)); p[i].start();

}

}

static class Philosopher implements Runnable { int id;

public Philosopher(int id) { this.id = id;

}

@Override

public void run() { try {

while (true) { think(); takeFork(); eat(); putFork();

}

} catch (InterruptedException e) { Thread.currentThread().interrupt();

}

}

void think() throws InterruptedException { System.out.println("Philosopher " + id + " is thinking."); Thread.sleep((long) (Math.random() \* 1000));

}

void eat() throws InterruptedException { System.out.println("Philosopher " + id + " is eating."); Thread.sleep((long) (Math.random() \* 1000));

}

void takeFork() throws InterruptedException { sem.acquire();

fork[id].acquire();

fork[(id + 1) % no\_phil].acquire(); sem.release();

}

void putFork() throws InterruptedException { sem.acquire();

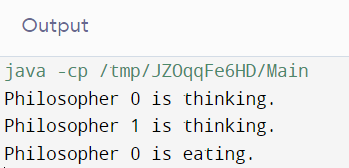
fork[id].release();

fork[(id + 1) % no\_phil].release(); sem.release();

}

}

}



**DATE:**

**EXPERIMENT NO: 6**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

# Experiment 6

* 1. **Write various deadlock condition. Write the Bankers Algorithm for Deadlock Avoidance.**

# Deadlock Conditions:

Deadlock in a computer system is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource held by some other process. Four conditions must hold simultaneously for a deadlock to occur:

# Mutual Exclusion:

* At least one resource must be held in a non-shareable mode; that is, only one process can use the resource at any given instant. If another process requests that resource, the requesting process must be delayed until the resource has been released.

# Hold and Wait:

* A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes**.**

# No Preemption:

* Resources cannot be forcibly removed from the processes holding them until the resources are used to completion.

# Circular Wait:

* A set of processes {P1, P2, ..., Pn} must exist such that P1 is waiting for a resource that is held by P2, P2 is waiting for a resource that is held by P3, and so on until Pn is waiting for a resource that is held by P1

# Banker’s Algorithm:

#include <iostream> #include <vector>

using namespace std; int main() {

int n, m;

cout << "Enter the number of processes: "; cin >> n;

cout << "Enter the number of resources: "; cin >> m;

vector<vector<int>> alloc(n, vector<int>(m)); vector<vector<int>> max(n, vector<int>(m)); vector<int> avail(m);

cout << "Enter allocation matrix:" << endl; for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) { cin >> alloc[i][j];

}

}

cout << "Enter max matrix:" << endl; for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) { cin >> max[i][j];

}

}

cout << "Enter available resources:" << endl; for (int i = 0; i < m; i++) {

cin >> avail[i];

}

vector<vector<int>> need(n, vector<int>(m)); vector<int> finished(n, 0);

vector<int> safe\_sequence(n); int index = 0;

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

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

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

}

}

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

if (finished[i] == 0) { bool flag = true;

for (int j = 0; j < m; j++) { if (need[i][j] > avail[j]) {

flag = false; break;

}

}

if (flag) {

for (int j = 0; j < m; j++) { avail[j] += alloc[i][j];

}

safe\_sequence[index++] = i; finished[i] = 1;

}

}

}

}

bool is\_safe = true;

for (int i = 0; i < n; i++) { if (finished[i] == 0) {

is\_safe = false;

cout << "The system is not in a safe state." << endl; break;

}

}

if (is\_safe) {

cout << "The system is in a safe state." << endl; cout << "Safe sequence is: ";

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

cout << "P" << safe\_sequence[i]; if (i != n - 1) {

cout << " -> ";

}

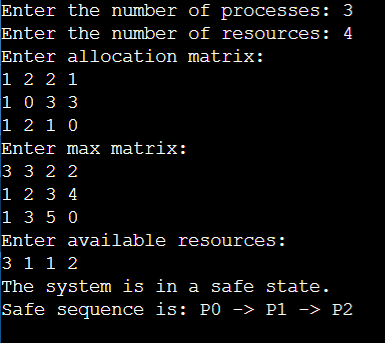
}

cout << endl;

}

return 0;

}



# 2. Implementation of Deadlock Detection Algorithm

public class DeadlockExample {

static class Process implements Runnable { private final int id;

private final Object resource1; private final Object resource2;

public Process(int id, Object resource1, Object resource2) { this.id = id;

this.resource1 = resource1; this.resource2 = resource2;

}

@Override

public void run() { synchronized (resource1) {

System.out.println("Process " + id + " has acquired resource1"); try {

Thread.sleep(100); // Simulate some processing

} catch (InterruptedException e) { e.printStackTrace();

}

synchronized (resource2) {

System.out.println("Process " + id + " has acquired resource2");

}

}

}

}

public static void main(String[] args) { Object resource1 = new Object(); Object resource2 = new Object();

Process process1 = new Process(1, resource1, resource2); Process process2 = new Process(2, resource2, resource1); Thread thread1 = new Thread(process1);

Thread thread2 = new Thread(process2); thread1.start();

thread2.start();

try {

thread1.join(); thread2.join();

} catch (InterruptedException e) { e.printStackTrace();

}

System.out.println("Main thread exiting");

}

}

Output:

Process 1 has acquired resource Process 2 has acquired resource

**DATE:**

**EXPERIMENT NO: 7**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

# Experiment 7

1. **Implementation of the following Memory Allocation Methods for fixed partition a) First Fit b) Worst Fit c) Best Fit**

#include<stdio.h>

void First(int b[],int p[],int n){ for(int i=0;i<n;i++){

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

if(p[i]<=b[j]){

printf("%d->%d\n",p[i],b[j]); b[j]=-1;

flag=1; break;

}

}

if(flag==0)

printf("%d Can't Fit\n",p[i]);

}

printf("\n");

}

void Best(int b[],int p[],int n){ for(int i=0;i<n;i++){

int flag=0,min,h=0,loc; for(int j=0;j<n;j++){

if(p[i]<=b[j]){ flag=1; if(h==0){

min=b[j]-p[i]; loc=j;

}

else{

if(min>b[j]-p[i]){

min=b[j]-p[i]; loc=j;

}

}

h++;

}

}

if(flag==1){

printf("%d->%d\n",p[i],b[loc]); b[loc]=-1;

}

else

printf("%d Can't Fit\n",p[i]);

}

printf("\n");

}

void Worst(int b[],int p[],int n){ for(int i=0;i<n;i++){

int flag=0,max,h=0,loc; for(int j=0;j<n;j++){

if(p[i]<=b[j]){ flag=1; if(h==0){

max=b[j]-p[i]; loc=j;

}

else{

if(max<b[j]-p[i]){

max=b[j]-p[i]; loc=j;

}

}

h++;

}

}

if(flag==1){

printf("%d->%d\n",p[i],b[loc]); b[loc]=-1;

}

else

printf("%d Can't Fit\n",p[i]);

}

printf("\n");

}

int main(){

int n,option; do{

printf("choose\n1.First Fit\n2.Best Fit\n3.Worst fit\n4.Exit\n"); scanf("%d",&option);

if(option==4) break; scanf("%d",&n);

int b[n],p[n]; for(int i=0;i<n;i++) scanf("%d",&b[i]); for(int i=0;i<n;i++) scanf("%d",&p[i]); switch(option){

case 1:printf("First Fit: \n"); First(b,p,n);

break;

case 2:printf("Best Fit: \n"); Best(b,p,n);

break;

case 3:printf("Worst Fit: \n"); Worst(b,p,n);

break;

}

}while(1); return 0;

}

output:

# choose

* 1. **First Fit**

# Best Fit

* 1. **Worst Fit 3**

# 5

**100 500 200 600 300**

# 212 417 112 426 300

Worst Fit 212->600

417->500

112->300

426 can't fit

300 can't fit

# choose

1. **First Fit**

# Best Fit

1. **Worst fit**

# Exit 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | **VNRVJIET**  **NAME OF LAB:**  Operating Systems | **EXPERIMENT NAME:**  **EXPERIMENT NO: 8 DATE:** |  |
| **Experiment 8**  **1. Implementation of the following Page Replacement Algorithms a) FIFO b) LRU c) LFU**  // page replacement using FIFO #include <stdio.h>  #define MAX 50  int a[MAX],f=-1,r=-1; void insert(int d){  if(f==-1&&r==-1)  {  f=0; r=0;  }  else r++;  a[r]=d;  }  int search(int key){ for(int i=f;i<=r;i++){  if(key==a[i])  {  return 1;  }  }  return 0;  }  void replace(int d){ f++; a[++r]=d;}  int main()  {  int n,frames=3,h=0,pf=0; scanf("%d",&n);  int b[n];  for(int i=0;i<n;i++) scanf("%d",&b[i]); for(int i=0;i<n;i++){  if(search(b[i])){ h++;  }  else{  if(r-f+1<frames) insert(b[i]); else  29 | | | |

replace(b[i]); pf++;

}

}

printf("page fault=%d\nhits=%d\nrate=%f\n",pf,h,(float)pf/frames); return 0;

}

output:

9

4 0 3 4 6 0 3 4 6

page fault=5 hits=4 rate=1.666667

//page replacement using LRU #include <iostream> #include <algorithm>

using namespace std;

int checkHit(int incomingPage, int frame\_items[], int frame\_occupied) { for (int i = 0; i < frame\_occupied; i++) {

if (incomingPage == frame\_items[i]) return 1;

}

return 0;

}

int findLRU(int ref\_str[], int frame\_items[], int refStrLen, int index, int frame\_occupied) {

int max\_distance = -1; int replace\_index = -1;

for (int i = 0; i < frame\_occupied; i++) { int distance = 0;

for (int j = index - 1; j >= 0; j--) { distance++;

if (frame\_items[i] == ref\_str[j]) { break;

}

}

if (distance > max\_distance) { max\_distance = distance; replace\_index = i;

}

}

return replace\_index;

}

void lruPage(int ref\_str[], int refStrLen, int max\_frames) { int frame\_items[max\_frames];

fill(frame\_items, frame\_items + max\_frames, -1); int hits = 0;

int pagefault = 0;

int frame\_occupied = 0;

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

if (checkHit(ref\_str[i], frame\_items, frame\_occupied)) { hits++;

} else if (count(frame\_items, frame\_items + max\_frames, -1) > 0) { auto it = find(frame\_items, frame\_items + max\_frames, -1);

\*it = ref\_str[i]; pagefault++; frame\_occupied++;

} else {

int replace\_index = findLRU(ref\_str, frame\_items, refStrLen, i, frame\_occupied);

frame\_items[replace\_index] = ref\_str[i]; pagefault++;

}

}

double hit\_ratio = (double)hits / refStrLen; cout << "\nTotal Hits: " << hits << "\n"; cout << "Page Faults: " << pagefault << "\n"; cout << "Hit Ratio: " << hit\_ratio << endl;

}

int main() {

int max\_frames;

cout << "Enter the number of frames: "; cin >> max\_frames;

int refStrLen;

cout << "Enter the number of pages "; cin >> refStrLen;

int ref\_str[refStrLen];

cout << "Enter the elements";

for (int i = 0; i < refStrLen; ++i) cin >> ref\_str[i];

lruPage(ref\_str, refStrLen, max\_frames); return 0;

}

Output:

Enter the number of frames: 3 Enter the number of pages 9

Enter the elements4 0 3 4 6 0 3 4 6

Total Hits: 1

Page Faults: 8

Hit Ratio: 0.111111

**DATE:**

**EXPERIMENT NO: 9**

**EXPERIMENT NAME:**

**VNRVJIET**

**NAME OF LAB:**

Operating Systems

# Experiment 9

**Simulate the mentioned disk scheduling algorithm using C/C++/Java**

# FCFS

* 1. **SSTF**

# SCAN

* 1. **C-SCAN**

// disk scheduling FCFS #include <stdio.h> #include<stdlib.h>

int main() {

int n=8;

int a[n],head; int seektime=0,i;

printf("enter sequence"); for(i=0;i<n;i++) scanf("%d",&a[i]); printf("enter head position"); scanf("%d",&head); printf("Seek sequence is \n"); for(i=0;i<n;i++)

{

seektime+=abs(a[i]-head); head=a[i];

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

}

printf("\nseek time=%d",seektime);

return 0;

}

Output

enter sequence:176 79 34 60 92 11 41 114 enter head position:50

Seek sequence is

176 79 34 60 92 11 41 114

seek time=510

// disk scheduling SSTF #include <stdio.h> #include<stdlib.h>

int main() { int n=8; int a[n];

for(int i=0;i<n;i++) scanf("%d",&a[i]); int head; scanf("%d",&head); int min=-1,loc; printf("%d ",head); int sum=0;

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

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

min=abs(head-a[i]); loc=i;

}

else{

if(min>abs(head-a[i])){

min=abs(head-a[i]); loc=i;

}

}

}

sum+=min; head=a[loc]; a[loc]=100000;

printf("%d ",head);

}

printf("\nseek time:%d\n",sum); return 0;

}

# Output:

176 79 34 11 60 92 114 41

50

SSTF sequence: 50 41 34 11 60 79 92 114 176

seek time:204

// disk scheduling SCAN #include<iostream> #include<bits/stdc++.h> using namespace std; int n=8;

int a[8],head;

int seek=0; vector<int> l,r;

void seekleft()

{

for(int i=l.size()-1;i>=0;i--)

{ seek+=abs(l[i]-head); head=l[i]; cout<<l[i]<<" ";

}

return ;

}

void seekright()

{

for(int i=0;i<r.size();i++)

{

seek+=abs(r[i]-head); head=r[i]; cout<<r[i]<<" ";

}

return;

}

int main() {

int opt;

printf("enter sequence"); for(int i=0;i<n;i++) scanf("%d",&a[i]); printf("enter head position"); scanf("%d",&head);

cout<<"enter 1:left\n2:right\n"; cin>>opt;

sort(a,a+n);

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

{

if(a[i]>head) r.push\_back(a[i]);

else

l.push\_back(a[i]);

}

if(opt==1){ seekleft(); seekright();} else

{

seekright(); seekleft();

}

cout<<"\n"<<seek; return 0;

}

output

enter sequence 176 79 34 60 92 11 41 114 enter head position 50

enter 1:left 2:right

2

# Right :

Sequence:60 79 92 114 176 41 34 11

Seek time:291

enter sequence176 79 34 60 92 11 41 114 enter head position50

enter 1:left 2:right

1

# Left:

Sequence:41 34 11 60 79 92 114 176

Seek time:204

//disk scheduling using C-SCAN #include <iostream> #include<bits/stdc++.h>

using namespace std; int main() {

int n=8; int a[n+3]; a[0]=0;

for(int i=1;i<n+1;i++) scanf("%d",&a[i]);

int head,i=0; scanf("%d",&head); a[n+1]=head; a[n+2]=199;

sort(a,a+n+3); printf("\n");

int x[n],y[n],k1=0; i=0;

while(a[i]<head){ x[k1++]=a[i]; i++;

}

i++;

int k2=0; while(i<n+3){ y[k2++]=a[i]; i++;

}

printf("enter 1:left\n2:right\n"); int option;

scanf("%d",&option); int seektime=0; if(option==1){

for(i=0;i<k1;i++)

{

seektime+=abs(x[i]-head); head=x[i];

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

}

for(i=0;i<k2;i++)

{

seektime+=abs(y[i]-head); head=y[i];

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

}

}

else{

for(i=0;i<k2;i++)

{

seektime+=abs(y[i]-head); head=y[i];

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

}

for(i=0;i<k1;i++)

{

seektime+=abs(x[i]-head); head=x[i];

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

}

}

printf("\nseektime %d\n",seektime); return 0;

}

output:

176 34 60 92 11 41 114 79

50

enter 1:left 2:right

2

Sequence:60 79 92 114 176 199 0 11 34 41

seektime 389 enter 1:left 2:right

1

Sequence :0 11 34 41 60 79 92 114 176 199

Seek time 249