****

**DAYANANDA SAGAR College OF ENGINEERING**

**(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)**

**Department of Computer Science & Engineering**

**2024-25**

**FOURTH SEMESTER**

**OPERATING SYSTEM LAB MANUAL**

**Sub Code: 22CS43**



**DAYANANDA SAGAR COLLEGE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Vision and Mission of the Department**

**Vision**

**To provide a vibrant learning environment in computer science and engineering with Focus on industry needs and research, for the students to be successful global Professionals contributing to the society.**

**Mission**

**\* To adopt a contemporary teaching learning process with emphasis on hands on and Collaborative learning.**

**\* To facilitate skill development through additional training and encourage student forums for enhanced learning.**

**\* To collaborate with industry partners and professional societies and make the students industry ready.**

**\*To encourage innovation through multidisciplinary research and development activities.**

**\*To inculcate human values and ethics to groom the students to be responsible citizens.**



**DAYANANDA SAGAR COLLEGE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Code of Conduct in the Lab**

**Do’s**

**Students shall**

* Come prepared for the program to be developed in the laboratory.
* Report any broken plugs or exposed electrical wires to your faculty/laboratory technician immediately.
* Turn off the machine once you have finished using it.
* Maintain silence while working in the lab.
* Keep the Computer lab premises clean and tidy.
* Place backpacks under the table or computer counters.
* Treat fellow users of the laboratory, and all equipment within the laboratory, with the appropriate level of care and respect.

**Don’ts**

**Students shall not**

* Talk on cell phones in the lab.
* Eat or drink in the laboratory.
* Touch, connect or disconnect any plug or cable without the faculty/laboratory technician’s permission.
* Install or download any software or modify or delete any system files on any lab computers.
* Read or modify other users' files.
* Meddle with other users’ files.
* Leave their personal belongings unattended. We are not responsible for any theft.

**Operating System Lab Programs**

|  |  |  |
| --- | --- | --- |
| **Unit** | **Contents of the Unit** | **Hours** |
| 1 | Write a program to implement a Basic Multi-Process Calculator using system calls | 02 |
| 2 | Write a program that simulates the functionality of UNIX commands like cp, ls, grep. | 02 |
| 3 | Write a program to implement a simulation of the FCFS scheduling algorithm where the program should accept the number of processes, their arrival times, burst times from the user and perform the execution of these processes according to the FCFS algorithm. | 02 |
| 4. | Write a program to implement a simulation of the round robin scheduling algorithm where the program should accept the number of processes, their arrival times, burst times, the time quantum from the user and perform the execution of these processes according to the round robin algorithm | 02 |
| 5. | Write a program to implement the Producer-Consumer problem using semaphores. | 02 |
| 6. | Write a program that demonstrates resource allocation and deadlock avoidance using Bankers algorithm and print the safe sequence. | 02 |
| 7. | Write a program to implement memory allocation method for fixed partition using first fit algorithm. | 02 |
| 8. | Write a program to implement the execution of page replacement algorithm such as LRU with suitable page request and display the number of page faults. | 02 |

**1) Implement a simple Calculator using system calls for multi-processing.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

void performOperation(char operation, int a, int b) {

int result = 0;

switch (operation) {

case '+':

result = a + b;

printf("Result: %d\n", result);

exit(result);

case '-':

result = a - b;

printf("Result: %d\n", result);

exit(result);

case '\*':

result = a \* b;

printf("Result: %d\n", result);

exit(result);

case '/':

if (b != 0) {

result = a / b;

printf("Result: %d\n", result);

exit(result);

} else {

printf("Error: Division by zero\n");

exit(EXIT\_FAILURE);

}

default:

printf("Invalid operation\n");

exit(EXIT\_FAILURE);

}

}

int main() {

char operation;

int operand1, operand2, status;

printf("Enter an operation (+, -, \*, /): ");

scanf(" %c", &operation); // Notice the space before %c to catch any leading whitespace

printf("Enter the first number: ");

scanf("%d", &operand1);

printf("Enter the second number: ");

scanf("%d", &operand2);

pid\_tpid = fork();

if (pid< 0) {

perror("fork");

return EXIT\_FAILURE;

} else if (pid == 0) {

// Child process

performOperation(operation, operand1, operand2);

} else {

// Parent process waits for child to complete

waitpid(pid, &status, 0);

if (WIFEXITED(status)) {

printf("Child process exited with result = %d\n", WEXITSTATUS(status));

} else {

printf("Child process did not terminate successfully\n");

}

}

return EXIT\_SUCCESS;

}

**Practice Questions:**

* Implement a matrix multiplication program where each child process computes a part of the result matrix.
* Use multiple processes to check if numbers in a range are prime and print the primes.

**2. Implement a simulator which simulates the functionality of UNIX commands like cp, ls, grep.**

Program for simulation of cp unix commands

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main(int argc, char \*argv[]) {

FILE \*fp;

char ch;

int sc = 0;

// Check if the file name is provided

if (argc < 2) {

printf("Usage: %s <filename>\n", argv[0]);

return 1;

}

// Open the file in read mode

fp = fopen(argv[1], "r");

if (fp == NULL) {

printf("Unable to open the file: %s\n", argv[1]);

return 1;

} else {

// Read the file character by character

while ((ch = fgetc(fp)) != EOF) {

if (ch == ' ') { // Check for space character

sc++;

}

}

// Print the number of spaces

printf("Number of spaces: %d\n", sc);

fclose(fp); // Close the file

}

return 0;

}

PROGRAM FOR SIMULATION OF LS UNIX COMMANDS

#include <stdio.h>

#include <dirent.h>

int main(int argc, char \*\*argv) {

DIR \*dp;

struct dirent \*link;

// Ensure that the directory argument is provided

if (argc < 2) {

printf("Usage: %s <directory>\n", argv[0]);

return 1;

}

dp = opendir(argv[1]); // Open the directory

if (dp == NULL) {

perror("opendir"); // Print error if directory can't be opened

return 1;

}

printf("\nContents of the directory %s are:\n", argv[1]);

// Read and print all files in the directory

while ((link = readdir(dp)) != NULL) {

printf("%s\n", link->d\_name);

}

closedir(dp); // Close the directory

return 0;

}

PROGRAM FOR SIMULATION OF GREP UNIX COMMANDS

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX 1024

void usage() {

printf("usage:\t ./a.out filename word\n");

}

int main(int argc, char \*argv[]) {

FILE \*fp;

char fline[MAX];

char \*newline;

int count = 0;

int occurrences = 0;

if (argc != 3) {

usage();

exit(1);

}

// Open the file in read mode

if (!(fp = fopen(argv[1], "r"))) {

printf("grep: could not open file: %s\n", argv[1]);

exit(1);

}

// Read file line by line

while (fgets(fline, MAX, fp) != NULL) {

count++;

// Remove the newline character at the end of each line if present

if ((newline = strchr(fline, '\n')) != NULL) {

\*newline = '\0';

}

// Check if the word is found in the line

if (strstr(fline, argv[2]) != NULL) {

printf("%s: %d %s\n", argv[1], count, fline);

occurrences++;

}

}

// If no occurrences were found

if (occurrences == 0) {

printf("No occurrences of '%s' found in the file.\n", argv[2]);

}

fclose(fp);

return EXIT\_SUCCESS;

}

**Practice Questions:**

* Implement a program that simulates basic UNIX ls -l commands using file I/O and system calls
* Implement a program that performs an advanced simulation of the grep UNIX command, which can search for a pattern in files or directories, with support for several features such as case-insensitive search, counting matching lines, inverting the match, and recursive directory search

**3. Implement a simulation of the FCFS scheduling algorithm where the program should accept the number of processes, their arrival times, burst times from the user and perform the execution of these processes according to the FCFS algorithm.**

#include <stdio.h>

int main() {

int bt[20], wt[20], tat[20], i, n;

float wtavg, tatavg;

// Get the number of processes

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

scanf("%d", &n);

// Get burst times for each process

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

printf("Enter Burst Time for Process %d: ", i);

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

}

// Initialize the first process

wt[0] = 0; // Waiting time for the first process is always 0

tat[0] = bt[0]; // Turnaround time for the first process is its burst time

wtavg = 0; // Initialize waiting time average

tatavg = bt[0]; // Initialize turnaround time average for the first process

// Calculate waiting times and turnaround times for the remaining processes

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

wt[i] = wt[i-1] + bt[i-1]; // Waiting time for the current process

tat[i] = tat[i-1] + bt[i]; // Turnaround time for the current process

wtavg += wt[i]; // Add current waiting time to the total

tatavg += tat[i]; // Add current turnaround time to the total

}

// Output the results

printf("\n\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");

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

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

}

// Calculate and display average waiting time and turnaround time

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

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

return 0;

}

INPUT

Enter the number of processes -- 3

Enter Burst Time for Process 0 -- 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 -- 3

OUTPUT

PROCESS BURST TIME WAITING TIME TURNAROUNDTIME

P0 24 0 24

P1 3 24 27

P2 3 27 30

Average Waiting Time-- 17.000000

Average Turnaround Time -- 27.000000

**Practice Questions:**

* Implement a simulation of FCFS Scheduling algorithm with Process Sorting Based on Arrival Time
* Implement a simulation of FCFS Scheduling algorithm that should print the Gantt chart along with user defined arrival and burst time.

**4. Implement a simulation of the round robin scheduling algorithm where the program should accept the number of processes, their arrival times, burst times, the time quantum from the user and perform the execution of these processes according to the round robin algorithm**

#include <stdio.h>

int main() {

int i, j, n, bu[10], wa[10], tat[10], t, ct[10], max;

float awt = 0, att = 0, temp = 0;

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

scanf("%d", &n);

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

printf("\nEnter Burst Time for process %d -- ", i + 1);

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

ct[i] = bu[i]; // Copy burst times to ct for later use

}

printf("\nEnter the size of time slice -- ");

scanf("%d", &t);

max = bu[0]; // Set max to the first process's burst time initially

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

if(max < bu[i]) {

max = bu[i];

}

}

// Round Robin scheduling simulation

for(j = 0; j < (max + t - 1) / t; j++) { // Loop until all processes are completed

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

if(bu[i] > 0) { // Process not completed

if(bu[i] <= t) {

tat[i] = temp + bu[i]; // Turnaround time

temp += bu[i]; // Increment temp with burst time

bu[i] = 0; // Process completed

} else {

bu[i] -= t; // Deduct time slice

temp += t; // Increment temp with time slice

}

}

}

}

// Calculating waiting time and turnaround time

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

wa[i] = tat[i] - ct[i]; // Waiting time = Turnaround time - Burst time

att += tat[i]; // Accumulate total turnaround time

awt += wa[i]; // Accumulate total waiting time

}

printf("\nThe Average Turnaround time is -- %.2f", att / n);

printf("\nThe Average Waiting time is -- %.2f", awt / n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME \t TURNAROUND TIME\n");

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

printf("\t%d \t %d \t\t %d \t\t %d \n", i + 1, ct[i], wa[i], tat[i]);

}

return 0;

}

INPUT:

Enter the no of processes – 3

Enter Burst Time for process 1 – 24

Enter Burst Time for process 2 -- 3

Enter Burst Time for process 3 – 3

Enter the size of time slice – 3

OUTPUT:

PROCESS BURST TIME WAITING TIME TURNAROUNDTIME

1 24 6 30

2 3 4 7

3 3 7 10

The Average Turnaround time is – 15.666667

The Average Waiting time is------- 5.666667

**Practice Questions:**

* Implement a simulation of Round Robin Scheduling algorithm with priority preemption.
* Implement a simulation of Multilevel Queue Scheduling with Round Robin (RR) algorithm.

**5. Implement the Producer-Consumer problem using semaphores.**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1, full = 0, empty = 3, x = 0;

void producer();

void consumer();

int wait(int);

int signal(int);

int main() {

int n;

printf("\n1. PRODUCER\n2. CONSUMER\n3. EXIT\n");

while (1) {

printf("\nENTER YOUR CHOICE\n");

scanf("%d", &n);

switch(n) {

case 1:

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

producer();

} else {

printf("BUFFER IS FULL\n");

}

break;

case 2:

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

consumer();

} else {

printf("BUFFER IS EMPTY\n");

}

break;

case 3:

exit(0);

break;

default:

printf("Invalid choice! Please enter 1, 2, or 3.\n");

}

}

return 0;

}

int wait(int s) {

return (--s); // Decrement semaphore

}

int signal(int s) {

return (++s); // Increment semaphore

}

void producer() {

mutex = wait(mutex); // Enter critical section

full = signal(full); // Increase the number of full slots

empty = wait(empty); // Decrease the number of empty slots

x++; // Produce an item

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

mutex = signal(mutex); // Leave critical section

}

void consumer() {

mutex = wait(mutex); // Enter critical section

full = wait(full); // Decrease the number of full slots

empty = signal(empty); // Increase the number of empty slots

printf("\nConsumer consumes item %d\n", x);

x--; // Consume an item

mutex = signal(mutex); // Leave critical section

}

**Practice Questions:**

* Implement a program that manages multiple buffers with Multiple Producers and Consumers and handle concurrency properly using semaphores to prevent race conditions.
* Implement a simulation of Circular Buffer (Buffer as a Circular Queue) with Single Producer and Consumer.

**6. Implement Bankers algorithm to demonstrate resource allocation and deadlock avoidance. Print the safe sequence.**

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the Process names here

int n = 5; // Number of processes

int m = 3; // Number of resources

int i, j, k, y;

int alloc[5][3] = {

{0, 1, 0}, // P0

{2, 0, 0}, // P1

{3, 0, 2}, // P2

{2, 1, 1}, // P3

{0, 0, 2} // P4

};

int max[5][3] = {

{7, 5, 3}, // P0

{3, 2, 2}, // P1

{9, 0, 2}, // P2

{2, 2, 2}, // P3

{4, 3, 3} // P4

};

int avail[3] = {3, 3, 2}; // Available Resources

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][m];

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

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

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

}

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

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

if (f[i] == 0) {

int flag = 0;

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

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

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

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

if (f[i] == 0) {

flag = 0;

printf("The following system is not safe\n");

break;

}

}

if (flag == 1) {

printf("Following is the SAFE Sequence: \n");

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

printf("P%d -> ", ans[i]);

printf("P%d", ans[n - 1]);

}

return 0;

}

**Output**

Following is the SAFE Sequence:

P0 -> P1 -> P3 -> P4 -> P2

**Practice Questions:**

* Implement a simulation of Banker's Algorithm for Resource Allocation with Dynamic Input.
* Implement a simulation of Banker's Algorithm with Multiple Resource Types and Deadlock Detection.

**7. Implement memory allocation method for fixed partition using first fit algorithm.**

#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

Intfrag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;

static int bf[max],ff[max];

clrscr();

printf("\n\tMemory Management Scheme - first Fit");

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

scanf("%d",&nb);

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

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

for(i=1;i<=nb;i++)

{

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

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

}

printf("Enter the size of the files :-\n");

for(i=1;i<=nf;i++)

{

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

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

}

for(i=1;i<=nf;i++)

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1) //if bf[j] is not allocated

{

temp=b[j]-f[i];

if(temp>=0)

if(highest<temp)

{

}

}

frag[i]=highest; bf[ff[i]]=1; highest=0;

}

ff[i]=j; highest=temp;

}

printf("\nFile\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragement");

for(i=1;i<=nf;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No File Size Block No Block Size Fragment

1 1 3 7 6

2 4 1 5 1

**Practice Questions:**

* Implement a Memory Allocation with Multiple File Types (Fixed and Dynamic Block Sizes) using first fit algorithm.
* Implement a program that compares First Fit, Best Fit, and Worst Fit memory allocation algorithms.

**8. Implement page replacement algorithm LRU with suitable page request and display the number of page faults.**

#include<stdio.h>

#include<conio.h>

int fr[3];

void main()

{

void display();

int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];

int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;

clrscr();

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

{

fr[i]=-1;

}

for(j=0;j<12;j++)

{

flag1=0,flag2=0;

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

{

if(fr[i]==p[j])

{

flag1=1;

flag2=1; break;

}

}

if(flag1==0)

{

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

{

if(fr[i]==-1)

{

fr[i]=p[j]; flag2=1;

break;

}

}

}

if(flag2==0)

{

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

fs[i]=0;

for(k=j-1,l=1;l<=frsize-1;l++,k--)

{

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

if(fr[i]==p[k]) fs[i]=1;

}

}

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

{

if(fs[i]==0)

index=i; }

fr[index]=p[j];

pf++;

}

display();

}

printf("\n no of page faults :%d",pf+frsize);

getch();

}

void display()

{

int i; printf("\n");

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

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

}

OUTPUT:

2 -1 -1

2 3 -1

2 3 -1

2 3 1

2 5 1

2 5 1

2 5 4

2 5 4

3 5 4

3 5 2

3 5 2

3 5 2

No of page faults: 7

**Practice Questions:**

* Implement page replacement algorithm LRU with Queue Implementation.
* Implement page replacement algorithm LRU with a Hash Map and Doubly Linked List