**Operating Systems Laboratory**

**B.Tech. 5thSemester**



**Name : Subhendu Maji**

**Roll Number : 18ETCS002121**

**Department : Computer Science and Engineering**

**Faculty of Engineering & Technology**

**Ramaiah University of Applied Sciences**

**Ramaiah University of Applied Sciences**

Private University Established in Karnataka State by Act No. 15 of 2013

|  |  |
| --- | --- |
| Faculty | Engineering & Technology |
| Programme | B. Tech. in Computer Science and Engineering |
| Year/Semester | 2nd Year / 5th Semester |
| Name of the Laboratory | Operating Systems Laboratory |
| Laboratory Code | 19CSL308A |

List of Experiments

1. Programs using process management system calls
2. Programs using file management system calls
3. Programs based on multithreaded programming
4. Programs for process scheduling algorithms
5. Solution to producer consumer problem using Mutex and Semaphore
6. Solutions to Dining philosopher problem using Semaphore
7. Programs for deadlock avoidance algorithm
8. Programs for memory management algorithms

# Index Sheet

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **No.** |  |  |  |  | | **Lab Experiment** | **Performing the experiment**  **(7)** | **Document**  **(7)** | **Viva**  **(6)** | **Total Marks**  **(20)** |
| 1 | Programs using process management system calls |  |  |  |  |
| 2 | Programs using file management system calls |  |  |  |  |
| 3 | Programs based on multithreaded programming |  |  |  |  |
| 4 | Programs for process scheduling algorithms |  |  |  |  |
| 5 | Solution to Producer Consumer Problem using Semaphore and Mutex |  |  |  |  |
| 6 | Solution to Dining Philosopher problem using Semaphore |  |  |  |  |
| 7 | Programs for deadlock avoidance algorithm |  |  |  |  |
| 8 | Programs for memory management algorithms |  |  |  |  |
| 9 | Lab Internal Test conducted along the lines of SEE and valued for 50 Marks and reduced for 20 Marks | | | |  |
|  | **Total Marks** | | | |  |

**Component 1 = Lab Internal Marks =**

**Signature of the Staff In-charge**

# Laboratory 1

Title of the Laboratory Exercise: Programs using process management system calls

1. Introduction and Purpose of Experiment

A system call is a programmatic way in which a [computer program](https://en.wikipedia.org/wiki/Computer_program) requests a service from the [kernel](https://en.wikipedia.org/wiki/Kernel_(computing)) of the [operating system](https://en.wikipedia.org/wiki/Operating_system) it is executed on. There are different types of system calls developed for various purposes. They are mainly classified as process management, file management, directory management. By solving the problems students will be able to apply process management system calls

Aim and Objectives

Aim

* To develop programs involving process management system calls

Objectives

At the end of this lab, the student will be able to

* Use different process management system calls
* Apply different system calls wherever required
* Create C programs using process management system calls

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Questions

Implement the following operations in C

Create four different processes (with different process ID) and assign four different tasks (addition, subtraction, Multiplication, division) to each process. All processes should display the result along with its process ID and parent process ID.

1. Calculations/Computations/Algorithms

STEP 1: Start

STEP 2: x first number, y second number

STEP 3: a fork()

STEP 4: if a is 0, then

4.1: Perform addition, display output.

STEP 5: else

5.1 Perform subtraction, display output

5.2 b fork()

5.3 if b is 0 then

5.3.1 Perform multiplication, display output

5.4 else

5.4.1 c fork()

5.4.1.1 if c is 0, then

5.4.1.1.1 perform division display output.

STEP 6: Stop.

1. Presentation of Results

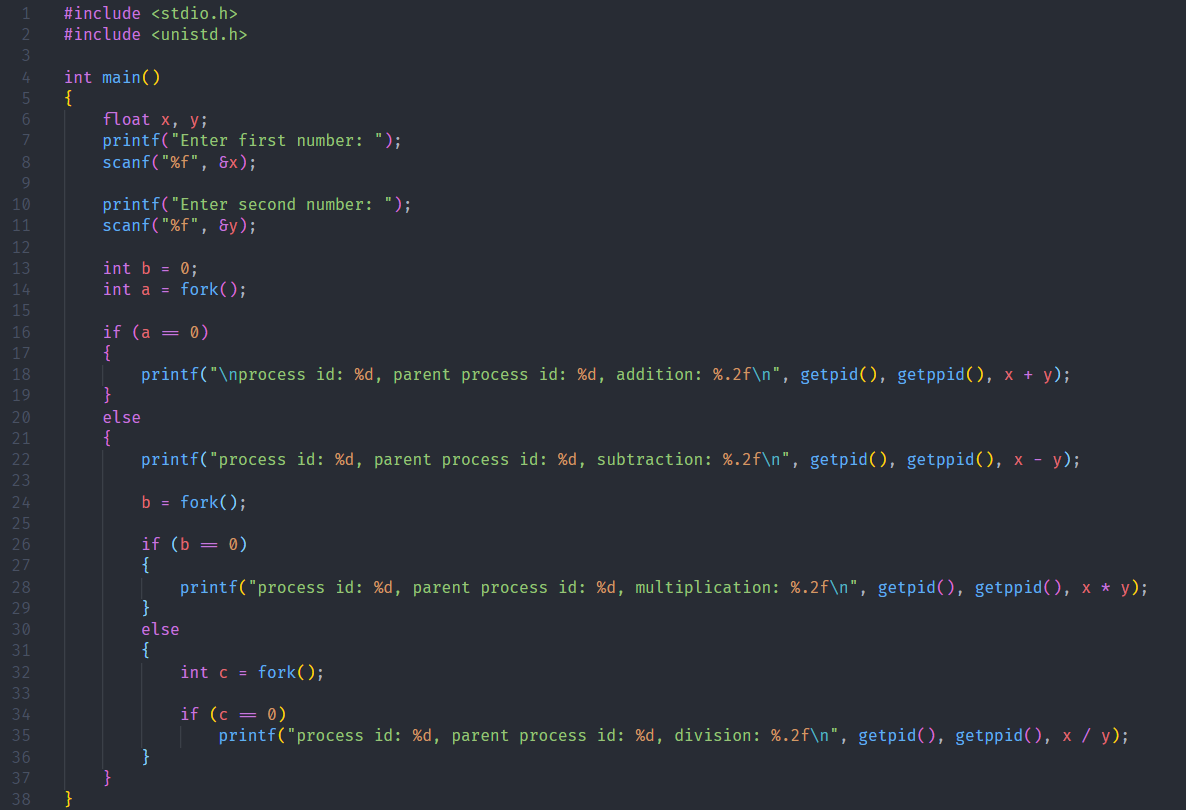


Figure 1 Source Code

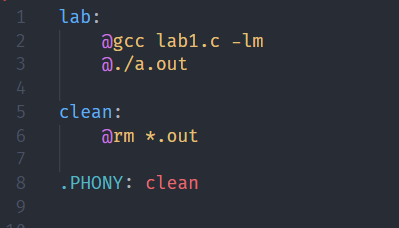


Figure 2 Makefile

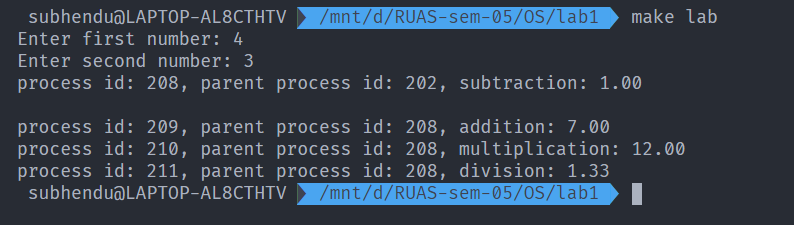


Figure 3 Program Execution

1. Analysis and Discussions

The fork() system call is a system call that is used to create process. This call creates a child process. The child process and the parent process, which called the fork() system call, execute the code that is present after the fork system called.

The parent and child processes do not share the same memory, but they can have multiple threads that share the same memory. The process run concurrently.

To distinguish the parent and the child, the value returned by the fork system call is used. For the parent process, the pid (process id) of the child is returned. For the child, the value returned is 0. If the value returned is negative, that means that there was an error that occurred while creating the child process.

The general intuition behind this program is that the parent processes execute the operations while the child operations call fork().

1. Conclusions

A system call that creates a new process identical to the calling one – Makes a copy of text, data, stack, and heap – Starts executing on that new copy

Uses of fork() – To create a parallel program with multiple processes

1. Comments

1. Limitations of Experiments

The fork() system call creates processes. Each process consumes lot of system resources, such as CPU, memory, I/O etc. if used without care, it can lead to system overloads and other catastrophic failure.

2. Limitations of Results

The child and parent processes do not share the same memory space.

3. Learning happened

The method to create processes using the fork memory call was learned.

# Laboratory 2

Title of the Laboratory Exercise: Programs using file management system calls

1. Introduction and Purpose of Experiment

A system call is a programmatic way in which a [computer program](https://en.wikipedia.org/wiki/Computer_program) requests a service from the [kernel](https://en.wikipedia.org/wiki/Kernel_(computing)) of the [operating system](https://en.wikipedia.org/wiki/Operating_system) it is executed on. There are different types of system calls developed for various purposes. They are mainly classified as process management, file management, directory management. By solving the problems students will be able to apply file management system calls

Aim and Objectives

Aim

* To develop programs involving file management system calls

Objectives

At the end of this lab, the student will be able to

* Use different file management system calls
* Apply different system calls wherever required
* Create C programs using file management system calls

2. Experimental Procedure

* + 1. Analyse the problem statement
    2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
    3. Implement the algorithm in C language
    4. Compile the C program
    5. Test the implemented program
    6. Document the Results
    7. Analyse and discuss the outcomes of your experiment

1. Questions

Implement the following command in C

Implement copy command (cp) to copy a file content to other file using file management system calls

1. Calculations/Computations/Algorithms

STEP 1: Start

STEP 2: buff string of size 100

STEP 3: inFile in\_file.txt file descriptor

STEP 4: outFile out\_file.txt file descriptor

STEP 5: bytesRead 0, bytesWritten 0

STEP 6: while bytesRead = read(inFile) and not EOF do

6.1: bytesWritten write to outFile

STEP 7: if bytesWritten is greater than 0, then

7.1: display success message

STEP 8: Stop

1. Presentation of Results

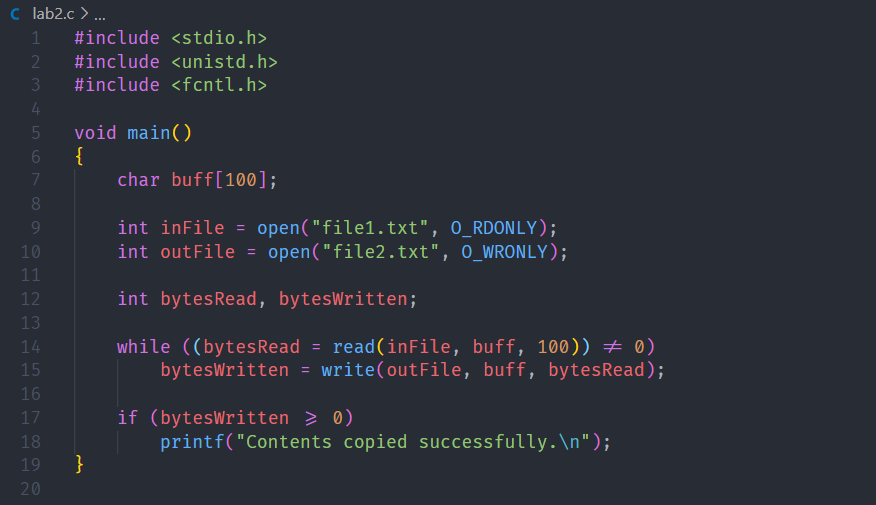


Figure 4 Source Code

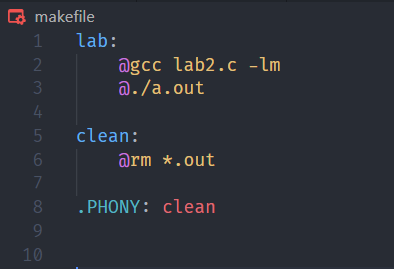


Figure 5 Makefile

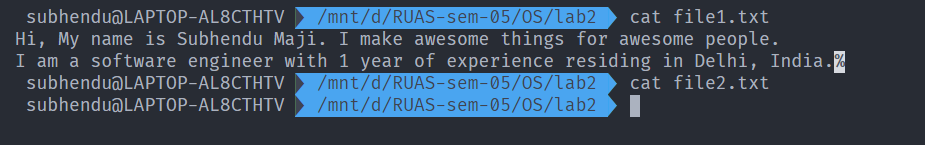


Figure 6 Files before Execution

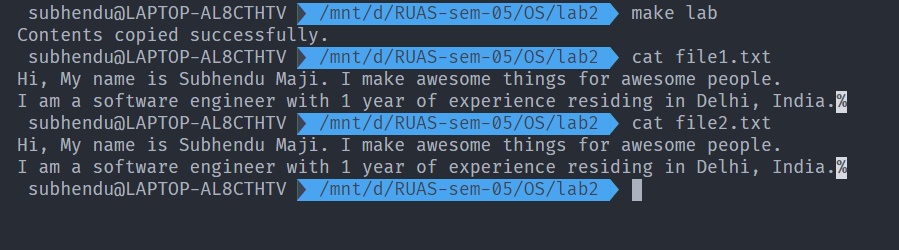


Figure 7 Execution and file copied successfully.

1. Analysis and Discussions

In this experiment, System calls are used to copy the contents from one file to another file. To open a file, the open() system call was used. The open() system call takes the file path and the file mode as arguments. The function returns a file descriptor.

The read() system call is used to read a file that is open in read mode. The read() call takes the file descriptor to be read from, the buffer to store the data that is read and the amount of bytes to read. The function returns the number of bytes that was successfully read.

The write() system call is used to write data to a file that is open in write mode. The write() system call takes the output file descriptor, buffer containing the data and the number of bytes to write. The function returns the number of bytes that was successfully written to the file.

1. Conclusions

In this experiment, system calls were used to open, read and write to a file. These calls were used to make a mock ‘cp’ command. The program was implemented in C.

1. Comments

1. Limitations of Experiments

The read() system call reads from the file sequentially.

2. Limitations of Results

The use of system calls to implement the ‘cp’ command was learned.

3. Recommendations

readv() and writev() can be used, which read and write from multiple buffers at once.

# Laboratory 3

Title of the Laboratory Exercise: Programs based on multithreaded programming

1. Introduction and Purpose of Experiment

Multithreading is the ability of a processor or a single core in a multi-core processor to execute multiple [threads](https://en.wikipedia.org/wiki/Thread_(computer_science)) concurrently, supported by the [operating system](https://en.wikipedia.org/wiki/Operating_system). By solving students will be able to manipulate multiple threads in a program.

1. Aim and Objectives

Aim

* To develop programs using multiple threads.

Objectives

At the end of this lab, the student will be able to

* Identify multiple tasks
* Use threads constructs for creating threads
* Apply threads for different/multiple tasks

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Questions

Create multithreaded programs to implement the following

1. Display “Hello World” message by 3 different threads
2. Create two threads;

* Thread1 adds marks (out of 10) of student1 from subject1 to subject5, Thread2 adds marks of student2 from subject1 to subject 5. Main process takes the sum returned from the Thread1 and Thread2, decides who scored more marks and displays student with its highest score.

1. Calculations/Computations/Algorithms

Algorithm for question a:

STEP 1: Start

STEP 2: define NUM\_THREADS ← 3

STEP 3: ret\_code ← 0, malloc threads array

STEP 4: for i=0 to NUM\_THREADS, do

4.1: ret\_code ← create thread that executes callback function (pthread\_create)

4.2: if ret\_code is positive, then display error message and exit.

STEP 5: Stop

void\* callback(void\* args);

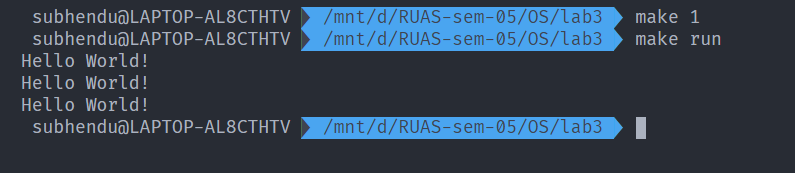
STEP 1: Start

STEP 2: display “hello world”

STEP 3: Stop

1. Presentation of Results





1. Analysis and Discussions
2. Conclusions
3. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations

# Laboratory 4

Title of the Laboratory Exercise: Programs for process scheduling algorithms

1. Introduction and Purpose of Experiment

A Process Scheduler schedules different processes to CPU based on particular scheduling algorithms. There are various scheduling algorithms present in each group of operating system. By solving these problems students will be able use different scheduling algorithms as part of their implementation

1. Aim and Objectives

Aim

* To develop programs to implement scheduling algorithms

Objectives

At the end of this lab, the student will be able to

* Distinguish different scheduling algorithms
* Apply the logic of scheduling algorithms wherever required
* Create C programs to simulate scheduling algorithms

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Questions

Write a multithreaded program to simulate the following process scheduling algorithms. Calculate average waiting time and average turnaround time for processes under each scheduling algorithm by separate threads

Instructions: Assume all the processes arrive at the same time. For round robin scheduling algorithm, read the number of processes in the system, their CPU burst times and the size of the time slice. For priority scheduling algorithm, read the number of processes in the system, their CPU burst times and the priorities.

1. Priority
2. Round Robin
3. Calculations/Computations/Algorithms
4. Presentation of Results
5. Analysis and Discussions
6. Conclusions
7. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations

# Laboratory 5

Title of the Laboratory Exercise: Solution to Producer Consumer Problem using Semaphore and Mutex

1. Introduction and Purpose of Experiment

In multitasking systems, simultaneous use of critical section by multiple processes leads to data inconsistency and several other concurrency issues. By solving this problem students will be able to use Semaphore and Mutex for synchronisation purpose in concurrent programs.

1. Aim and Objectives

Aim

* To implement producer consumer problem using Semaphore and Mutex

Objectives

At the end of this lab, the student will be able to

* Use semaphore and Mutex
* Apply semaphore and Mutex in the required context
* Develop multithreaded programs with Semaphores and Mutex

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Questions

Implement producer consumer problem by using the following

1. Semaphore
2. Mutex
3. Calculations/Computations/Algorithms
4. Presentation of Results
5. Analysis and Discussions
6. Conclusions
7. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations

# Laboratory 6

Title of the Laboratory Exercise: Programs for deadlock avoidance algorithm

1. Introduction and Purpose of Experiment

Deadlocks can be avoided if certain information is available in advance. By solving these problems students will become familiar to avoid deadlock in advance with the available resource information

1. Aim and Objectives

Aim

* To develop Bankers algorithm for multiple resources for deadlock avoidance

Objectives

At the end of this lab, the student will be able to

* Verify a problem to check that whether deadlock will happen or not for the given resources
* Implement the bankers algorithm for multiple resources

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Questions

Implement a Bankers algorithm for deadlock avoidance

1. Calculations/Computations/Algorithms
2. Presentation of Results
3. Analysis and Discussions
4. Conclusions
5. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations

# Laboratory 7

Title of the Laboratory Exercise: Programs for memory management algorithms

1. Introduction and Purpose of Experiment

In a multiprogramming system, the user part of memory must be further subdivided to accommodate multiple processes. This task of subdivision is carried out dynamically done by the operating system known as memory management. By solving these problems students will become familiar with the implementations of memory management algorithms in dynamic memory partitioning scheme of operating system.

1. Aim and Objectives

Aim

* To develop a simulator for memory management algorithms

Objectives

At the end of this lab, the student will be able to

* Apply memory management algorithms wherever required
* Develop simulators for the algorithms

1. Experimental Procedure
   * + Analyse the problem statement
     + Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     + Implement the algorithm in C language
     + Compile the C program
     + Test the implemented program
     + Document the Results
     + Analyse and discuss the outcomes of your experiment
2. Questions

Implement a simulator for the memory management algorithms with the provision of compaction and garbage collection

1. First fit
2. Best fit
3. Worst fit
4. Calculations/Computations/Algorithms
5. Presentation of Results
6. Analysis and Discussions
7. Conclusions
8. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations

# Laboratory 8

Title of the Laboratory Exercise: Solution to Dining Philosopher problem using Semaphore

1. Introduction and Purpose of Experiment

In multitasking systems, simultaneous use of critical section by multiple processes leads to data inconsistency and several other concurrency issues. By solving this problem students will be able to use semaphore for synchronisation purpose in concurrent programs.

1. Aim and Objectives

Aim

* To develop concurrent programs using semaphores

Objectives

At the end of this lab, the student will be able to

* Use semaphore
* Apply appropriate semaphores in different contexts
* Develop concurrent programs using semaphores

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Question

Implement the Dining Philosopher problem using POSIX threads

1. Calculations/Computations/Algorithms
2. Presentation of Results
3. Analysis and Discussions
4. Conclusions
5. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations