

Operating Systems Lab (CL2006)

Date: May, 26, 2025

Instructors: All

Sections: All

Final Exam (Lab)

Total Time (H:M): 2:30

Total Marks: 80

Total Questions: 2

Semester: Spring-2025

231-323

Roll No

Section

Student Signature

Submission paths:

\\Exam\Final Exam\Operating Systems

Instructions:

- Open NS3\_1 virtual machine (placed in C drive, password is 123456, alternatively you can use NS2 vm (placed in D drive).
- Check if you are able to copy paste your files from Ubuntu to Windows, if not inform the invigilator at start.
- Submit a zip folder with your code and screenshots named as your rollNumber.

Attempt all the questions.

CLO 1: Services provided by the operating systems

CLO 2: Implement solutions employing concepts of Processes

Q. No 1: Multi-Process Logger with Real-Time Filtering

[25]

You have to write a **real-time system logger**. The system generates logs continuously, and you need to process and store them on-the-fly using a **multi-process pipeline**.

The log flow is as follows:

```
./log_generator | stdbuf -oL grep "ERROR" | stdbuf -oL cut -d ' ' -f2 -> errors.log
```

Your task is to simulate the entire pipeline in C using system calls, without relying on the shell.

1. Write a log\_generator.c file

[2]

- It will simulate real-time log data.
- Write an infinite while loop which should
  - Writes two lines to stdout every second:
    - One INFO line i.e "INFO sample log"
    - One ERROR line i.e "ERROR something failed"
  - Use printf() and fflush(stdout) to ensure real-time output.

Spring 2025

Example output:

```
INFO sample log
ERROR something failed
INFO sample log
ERROR something failed
.....
.....
.....
```

2. Write a program logger.c (Logger Pipeline Controller)

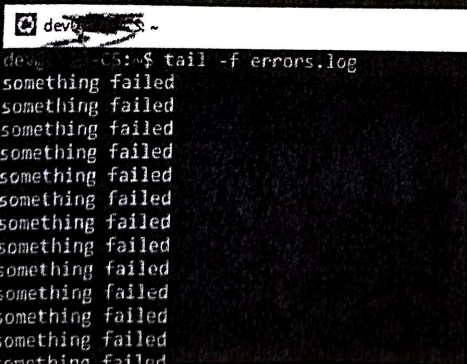
This program will:

1. Create 3 child processes, connected via pipes, forming the full pipeline. [3]
  - a. Use exec to spawn the ./log\_generator process and capture its output via a pipe. [5]
  - b. Pipe that output to a grep "ERROR" process. [5]
  - c. Pipe the result to a cut -d ' ' -f2- process. [5]
  - d. The final output is captured in the parent process and written to a file errors.log using open() and write(). [3]

Note:

- Use stdbuf -oL before grep and cut to ensure unbuffered (line-buffered) output.
  - The output to errors.log should appear in real time, not after the buffer fills. Use tail -f to monitor the errors.log file as in the example screenshot.
  - Use fork(), pipe(), dup2(), and exec\*() you can't use popen().
  - Handle SIGINT (Ctrl+C) (alternatively you can call wait() for each child):
    - a. Kill all child processes
    - b. Close all file descriptors
    - c. Exit cleanly
  - Don't change the file names, and mention any details if needed to run your task.
3. Submit output/ error screenshots [2]

```
-CS:~$ gcc -o logger logger.c
-CS:~$ ./logger
```



```
devb -CS:~$ tail -f errors.log
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
something failed
```

CLO 2: Implement solutions employing concepts of Threads

CLO 3: Mechanisms for scheduling of tasks and implement synchronization mechanisms

CLO 5: Memory management

**[55]**

**Q. No 2: Multithreaded file search with early termination**

Write a multithreaded program that searches for a specific word within all .txt files inside a given folder. The directory path and search word will be provided as command-line arguments.

Multiple threads will be created to search the files concurrently. As soon as any thread finds the word, it should record the file name and line number in shared memory. Once a match is found, all other threads should stop searching immediately.

The program will:

1. Accept two command line arguments (path to folder and word to search) [02]
2. Collect names of all .txt files (just consider the .txt in the given folder and not sub folders) [08]
  - a. Write file names in a shared memory [08]  
[Hint: you can use the result of "ls folderName" system call]
3. Use mmap() to create a region of shared memory accessible to all threads. [05]
  - a. The shared memory should contain:
    - i. The list with status indicators so threads coordinate who processes which file.
    - ii. A flag to indicate that the word has been found.
    - iii. A buffer or structure to store the result file name and line number.  
(you can add more according to your need and understanding)
4. Create a pool of threads i.e. 5 threads simultaneously [02]
  - a. Each thread opens a .txt file [02]
    - i. If a file is opened by one thread then no other thread will open it. (each thread will open an unread file in a critical section) [05]
    - ii. Search for the word in each line. [10]
      1. If found:
        - a. Lock a mutex/semaphore.
        - b. Update SharedResult (filename, line number, word\_found=1).
        - c. Unlock the mutex/semaphore.
        - d. Exit the thread.
    - iii. If the word is not found, close the file and exit. [02]
    - iv. If there are unread files then repeat from step i (threads keep picking new files until none remain or the word is found). [05]
  - b. Threads should periodically check the word\_found flag before starting a new file and also during line scanning to enable early termination. [05]
5. Use semaphores to ensure that [05]



# National University of Computer and Emerging Sciences

Lahore Campus

- a. The shared flag and result structure are updated safely.
- b. Only one thread writes to shared memory.
6. After all threads terminate, the main thread should:
  - a. Check if the word was found.
    - i. If found, print the result from shared memory (file and line number).
    - ii. If not found, display a message: "Word not found."
7. Submit output/ error screenshots

[02]

[02]

Note:

- Given folder contains only .txt files and you can list them by "ls \path\to\folder"
- The search should be **case-sensitive**
- Handle edge cases:
  - No .txt files found.
- Ensure proper cleanup of resources: file descriptors, memory, threads.
- Test your program for scenarios:
  - No of files are less than the number of threads
  - No of files are greater than the number of threads
- Mention any details if needed to run your task.

```
dev@CS:~$ ./task2 f_task2 random
Word not found.
dev@CS:~$ cd f_task2
dev@CS:~/f_task2$ ls
a.txt b.txt c.txt
dev@CS:~/f_task2$ cat c.txt
University
of
Management
Sciences
dev@CS:~/f_task2$ cd ..
dev@CS:~$ ./task2 f_task2 Sciences
Word found in f_task2/c.txt at line 4.
dev@CS:~$
```