# Lab Report: Interprocess Communications with Pipes and Java Threads

[Your Name] - [Your Student ID] CSI3131 - Operating Systems 2025-06-08

# 1 Introduction and Objectives

The objective of this lab is to explore interprocess communication (IPC) using UNIX/Linux pipes and to learn about Java threads and thread pools. The specific goals include:

- Understanding IPC mechanisms through the use of pipes in a C program.
- Gaining experience with process management in a Linux environment.
- Learning how to implement and manage threads in Java.
- Comparing the performance of individual threads versus a thread pool.

# 2 Methodology

The lab was conducted in a Linux virtual machine environment. The following steps were taken to achieve the objectives:

- 1. VM Setup: Logged into the Site Linux virtual machine and extracted the provided lab2a.tar file.
- 2. **Modify C Program**: Enhanced the mon.c program to create mon2.c for monitoring processes using pipes.
- 3. Compilation: Compiled the modified mon2.c program using the cc command.

- 4. **Execution**: Ran the mon2 program with the calcloop argument and observed the filtered output.
- 5. **Signal Handling**: Experimented with sending **SIGSTOP** and **SIGCONT** signals to the **calcloop** process.
- 6. **Java Setup**: Compiled the provided Java files for generating the Mandelbrot set.
- 7. Execution of Mandelbrot: Executed the MandelBrot application with various parameters.
- 8. **Thread Implementation**: Modified the Java code to use threads for rendering the Mandelbrot set.
- 9. **Thread Pool Implementation**: Further modified the code to utilize a thread pool with Executors.

## 3 Presentation and Analysis of Results

#### 3.1 C Program Execution

The execution of the mon2 program yielded filtered output for the calcloop process (Fig. 1).

Figure 1: Filtered output of calcloop from mon2.

### 3.2 Signal Handling

We successfully sent signals to the calcloop process, observing the effects on its output. Screenshots of the commands and their effects are shown in Figures 2 and 3.

Figure 2: Commands used to send signals to calcloop.

Figure 3: Effect of signals on calcloop output.

#### 3.3 Java Threads Implementation

The modified Java code effectively utilized threads for rendering the Mandelbrot set. The updated display is shown in Figure 4.

Figure 4: Updated display using Java threads for rendering.

#### 3.4 Thread Pool Implementation

The implementation of a thread pool improved performance by managing thread resources more efficiently. The impact can be seen in Figure 5.

Figure 5: Impact of using a thread pool with Executors.

#### 4 Discussion and Conclusion

This lab provided valuable insights into interprocess communication and threading in Java. Key learnings include:

- The effectiveness of using pipes for IPC in C programs.
- The importance of process management and signal handling in a Linux environment.
- Practical experience in Java threading and the advantages of using thread pools for resource management.

#### Challenges Encountered

One challenge faced was debugging the C program to ensure proper use of pipes and signal handling.

#### Suggestions for Improvement

I would like to further improve my understanding of threading concepts in Java, particularly with regard to performance optimization.

#### Screenshots and Evidence

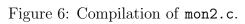


Figure 7: Compilation of Java files for Mandelbrot.

Figure 8: Execution of MandelBrot with parameters.