**Cooperating Processes**

**CSCI411 Lab**

**Adapted from Linux Kernel Projects by Gary Nutt and Operating Systems by Tannenbaum**

**Exercise Goal:** You will learn how to write a LINUX processes that cooperate with each other using message queues.

Contents

[Problem Statement: 1](#_Toc441671353)

[Overview 1](#_Toc441671354)

[Additional Questions: 1](#_Toc441671355)

[Attacking the Problem 1](#_Toc441671356)

[Creating the Processes 1](#_Toc441671357)

[The Message Passing System 2](#_Toc441671358)

[Implementation Hints 5](#_Toc441671359)

[Submission Requirements 5](#_Toc441671360)

[Grading Rubrics 5](#_Toc441671361)

# Problem Statement:

## Overview

This project consists of using IPC POSIX message queues for communicating temperatures between each of four external processes and a central process. Four client processes will cooperate with a server process as follows: . Each client process will communicate temperatures to a central process. The central process will reply to individual clients with its own temperature and indicate whether the entire system has stabilized.

Each client process will receive its initial temperature upon creation and will send that to the server process. Until the system has become stable, the server process will calculate its new temperature and send it to the mailbox for each of the client processes and await their replies. If all four temperatures are exactly the same as those sent by the four processes during the last iteration, the system has stabilized. In this case, the central process will notify each external process that it is now finished (along with the central process itself), and each process will output the final stabilized temperature. The processes will continue to run until the temperature has stabilized.

**The initial temperatures for the 4 external processes should be: 100, 22, 50, and 40.**

New temperatures are calculated according to two formulas:

*new external temp =*

*(myTemp \* 3 + 2 \* centralTemp)/5;*

*new central temp =*

*(2 \* centralTemp + four temps received from external processes)/6;*

Each process should print out the message it is going to send, in the following format:

*Process ID: message*

When the temperatures have stabilized among the processes, the server will terminate all client processes. You will need to determine how to initiate the termination.

# Attacking the Problem

You will write only two programs: a client and a server program. Each client process will execute in a separate process, and separate command window. The server too will execute in a separate command window.

The processes will communicate through POSIX message queues. The POSIC message queues are identified by their names. On Linux, the names of POSIX message queues are limited to 255 characters. POSIX IPC names begin with an initial forward slash / followed by non-slash and ending with the null character. Any process knowing the queue name and having appropriate permissions can send or receive messages from the queue and also do other operations on it.The names of the open queues are listed in the /dev/mqueue virtual file system.

After opening the message queue the program receives a message queue descriptor that can be used in subsequent POSIX APIs.

You will create 8 (yes eight) message queues. That way the client can send on one and wait for a response on another.

## The Message Passing System

At this point, I will direct you to the example on site <https://www.softprayog.in/programming/interprocess-communication-using-posix-message-queues-in-linux>

This example creates a client server application. You can augment this application to complete your assignment.

## Implementation Hints

* It might be best to start by sending one message successfully from the central process to a single outer process, and vice versa, before trying to write all the code to solve this problem.
* It is also wise to check all the return values from the four message queue system calls for possible failed requests and to output a message to the screen after each one successfully completes.
* Finally, extraneous messages residing in a queue could cause a collection of cooperating processes that function correctly to appear erroneous. For that reason, it is wise to remove all mailboxes relevant to this project to ensure that mailboxes are empty before the processes begin. It is important to close and unlink the message queues even while testing. If a queue becomes rogue, you can remove it from /dev/mqueue. But be sure no one else is using the system or you may inadvertently remove their message queue.

# Submission Requirements

* Submit your source code (Shell script file) and makefile if necessary.

## Grading Rubrics

Marking Criteria (100 marks total)

* Warning free compilation and linking of executable with proper name (5 marks)
* Readability, suitability & maintainability of source code and instructions on compiling and executing (makefile, if needed) (20 marks)
* Server correctly coded and functioning 30
* Client correctly codded and functions 30
* Message queues are unlinked and removed for any termination.15