Lab 4 Design Document

Group 2
Aaron Meurer
Oran Wallace
Sheng Lundquist

Introduction

Interprocess communication is essential for a workstation that supports multiple processes. The operating system must provide a tool for processes or managers to communicate with each other. This communication can be used for synchronization, multi-threaded programing, and information sharing.

Design Decisions

Each manager will be essentially a queue, using our queue manager. Each message queue will be length five. This small size will make things easier to test. Also, it should be a reasonable number in a system where we can only have up to 20 processes. It will be easy to change in the future on a per-manager basis if we decide it is too large or too small.

We will have to modify our queue manager to handle messages, rather than just process_control_block. This will be done by copying the queuemanager and modifying it to use message structs instead of process control block structs.

The message will be a struct, making it easier to change the structure of the message in the future. For now, it will contain a variable noting which manager was the source manager, and a char array of size 256 for the message, as well as the necessary variables to be in a queue (prev and next).

Each message queue will be referenced by an integer from 0 to 9. This may later be adapted to use an enum once we know what exactly they are, which would be exactly the same anyway. This method of reference will be used both internally and in the commands to the test manager.

Flow Charts/State Diagrams

IPC Message Flow Chart

This flow chart describes the flow of events a message will experience in our IPC Manager. A send command will signify when a message is to be created. During the message creation if any errors are encountered the manager will discard the message and return the error accordingly. If no errors arise during the message creation process the message is enqueued into the corresponding destination queue. Upon a receive command being issued for the destination queue the message will be dequeued from that queue. In the test manager, the message will be printed to the terminal.

IPC State Diagram

This state diagram describes the states and transitions that are related to our message queues. Following the start state, all of the queues will epsilon transition to the “Init” state. In the “Init” state the queues will be created and initialized for their corresponding managers. From this state the queues perform an epsilon transition into the “Waiting” state. In the “Waiting” state the queues are waiting for a command to be issued to them and depending on that command the queues will transition accordingly.

If a send command is issued the queue will transition to a “Send” state. In this state the message will be enqueued to the destination queue. If an error is produced (queue full or malformed message for example) the queue will transition to an error state and the error will be returned to the test interface. If no error is produced an epsilon transition will be performed back to the “Waiting” state. If a receive command is issued the queues will transition to a “Receive” state. In this state the correct message will be dequeued from the queue for the corresponding manager. If an error is produced (queue empty) the queue will transition to an error state and the error will be returned to the test interface.

If a has_message command is issued the queue will transition “Check Empty” state. In this state the queue just performs a basic check to see if any messages are in it. If a list command is issued the queue will transition to the “List” state. In this state the queue will just list the messages contained inside of it.