An Understanding of Inter-Process Communication Through Shared Memory and Cooperating Process

Matthew Tran (ID:113606493)  
*Tran0208*  
*CS3113-001   
Dr. Song Fang*  
Matthew.H.Tran-1@ou.edu

A computer screen shot of a program code

Description automatically generated

Figure 1: Key Generation and Program to Shared Memory Link

*Abstract*—Inter-process communication (IPC) is referred to as the mechanisms provided by an operating systems for process to communicate and synchronize their actions with one another. Two specific models are used to implement this mechanism into an operating system: Shared memory, and Message passing. In this paper, we’ll be focusing on Shared memory Inter-Process Communication performed on a Linux system. I’ve created a program that simulates this process using libraries in C and commands executing this file in a Linux environment. This paper seeks to evaluate the performance of my program emulating shared memory inter-process communication in a linux environment.

Keywords—Inter-process communication, Linux OS, Shared memory, cooperating processes

# Project background

Cooperating processes are defined as processes that can affect or be affected by the execution of another process. In short, to be a cooperating process means you interact with data and information of other running processes. This powerful ability to share information with other processes allows for useful advantages such as accelerated computation, better modularity, and a more convenient system for the user. In order to obtain the useful advantages gained from cooperating processes, two specific models are implemented for processes that cooperate: Message Passing, and Shared Memory. To better understand the nature of this mechanism, it’s important to understand how exactly this process functions and what it looks like performance-wise. Message passing and shared memory are optimized for various reasons depending on the type of architecture and data being passed through for each process. In this paper, we are going to be focusing on how shared memory works and its performance on a Linux system.

A computer screen shot of a black screen

Description automatically generated

Figure 2:Print Processes, Release Shared Memory, and Exit

## Shared Memory and Implementation on a Linux System

### Shared Memory is a model of inter-process communication that allows multiple processes to access and manipulate the same section of memory on a computer. Shared memory is implemented on a computer by designating a memory segment for various processes to interact with. Once a process attaches itself to a shared memory segment in the computer, it can read and write to data that’s in the segment that it now has access to. In my program, I incorporated the shared memory segment for the linux environment through generating an IPC key using the ftok(const char \*pathname, int project\_id); command. I attached the program to the shared memory segment using the shmget() and the shmat() functions to prepare the environment for the processes to cooperate.

## Process Communication

Once the shared memory segment was declared and the environment was set up, a total of four processes were declared to access and handle the shared memory between one another. Processes 1-3 would have the function of incrementing or adding by 1 to 10000 to the shared memory segment, however process 4 had the function of incrementing by 1 to 20000 to the shared memory segment. In better words, each process incremented to a certain count for the shared memory, those counts being: 10000 for process 1, 20000 for process 2, 30000 for process 3 and 50000 for process 4. After the process was completed incrementing to their respective amounts, they would print the counter and exit. After each exiting process, the parent process prints the ID of the process that is completed. Once all the processes complete their task, the parent process releases or disconnects from the shared memory segment and ends the program, completing the simulation of shared memory on the Linux system.

# Shared memory performance evaluation

The program was run multiple times (20 documented executions) on the Linux environment to identify various changes and mechanisms that occur during the execution period. One discovery found was the occurrence of race conditions, or unexpected/inconsistent results, that took place with cooperating processes during the execution periods. The table provided details a list of times when a process produced anomalous results not in line with the expectations of what the count should be for the given processes listed in part I.B of the report.

A screen shot of a computer

Description automatically generated

Figure 3: Visual Depiction of Inconsistent Process

| Process  Number | Unexpected Result Count |
| --- | --- |
| Process 1 | 1 |
| Process 2 | 11 |
| Process 3 | 6 |
| Process 4 | 11 |

1. Race Conditions Identified during execution

Figure 4: Process Counter Success and Failure Distribution Between Processes

Additionally, with respect to the unexpected counter outputs, the processes output their results in an inconsistent fashion with processes 2-4 outputting results counted from other processes. Even processes with correct output of 10000,20000,30000, and 50000 often ended up outputting in process orders of (1,3,4,2) or (1,3,2,4), orders that don’t correspond with the expected (1,2,3,4) in terms of output order. An example output of an unexpected and inconsistent process is shown in figure 3 below.

# Conclusion

The execution and implementation of inter-process communication details the mechanism’s advantages in transporting large amounts of data between processes and the ease of access for processes to read and write a shared memory space within a program. However, the program’s inconsistency with outputting desired results and the discovery of race conditions as a result of running the program multiple times highlights the disadvantages or issues that can arise as a result of implementing shared memory between cooperating processes on a system. It is clear shared memory suffers from a few shortcomings as a result of prioritizing transport and accessibility for cooperating processes on a Linux system. However, the implementation of shared memory in the program showcases the advantages it holds as a form of inter-process communication and showcases the mechanism’s methods performance-wise on an operating system such as Linux.