**Project 2 - Shared memory**

**Intro to Operating Systems**

**Masiko Mamba**

Contents

[Abstract 3](#_Toc179047894)

[Introduction 3](#_Toc179047895)

[Requirements 3](#_Toc179047896)

[Methodology 3](#_Toc179047897)

[Over of program design 3](#_Toc179047898)

[Process creation and design 4](#_Toc179047899)

[Incrementing total 4](#_Toc179047900)

[Resource management 4](#_Toc179047901)

[Results and Discussion 4](#_Toc179047902)

[Summary of results 5](#_Toc179047903)

[Possible causes 5](#_Toc179047904)

[Recommendations 5](#_Toc179047905)

[Conclusion 5](#_Toc179047906)

# Abstract

This report presents the performance evaluation of the program that is written in C to perform shared memory computations. The program creates a shared memory segment that is accessed by many child processes using semaphores. Each process increments the shared variable and prints the result. The program uses fork() to create processes and shared-memory for communication between processes. This report will evaluate the results of the program, execution and what may have led to the results we collected. The report will provide quantitative data on the results and provide an analysis of the results and provide recommendations for improvement. Analysis shows how efficient memory management and semaphore synchronization to enable reliable concurrent updates to shared variables.

# Introduction

Inter process communication (IPC) is important in processes that have multiple processes and may help by reducing execution time and running multiple processes at the same time. They are also reliable. IPC also means that we need to facilitate access to the data using semaphores. This project implemented a shared memory operation and created multiple processes that all incremented a shared variable using fork() to create the processes. The aim is to evaluate the performance of this program with semaphores maintaining consistency with the data.

# Requirements

* Linux system
* C compiler
* IDE (if preferred)
* Code should be run in a Linux machine

# Methodology

* The program used shared memory to increase a shared variable using four processes.

## Overview of program design

* The Program created four processes where each process increments a shared variable.
* Each child process increments the shared variable total.
* Semaphores are used to ensure that only one process has access to the shared variable at a time avoiding race conditions

## Process creation and design

* The program creates a memory segment to store the integer that will be shared and incremented.
* Once the memory has been created, the processes are created using the fork() method.
* The parent process waits for each child to finish execution using the wait() function.

## Incrementing total

* Each child process increments the child process depending on the increment value for that process. For this program, the increment values were 100000, 200000, 300000 and 500000. The shared variable is protected by a semaphore.
* After the increments, the child processes print their results and terminate.

## Resource management

* After the child’s processes have been completed, the parent process continues execution and detaches and removes the memory segment. Th semaphore is also closed to prevent resource leakage.

To ensure that the program was reliable and produced correct results, it was executed multiple times, and all outputs recorded.

# Results and Discussion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Execution # | Process 1 | Process 2 | Process 3 | Process 4 |
| 1 | 359650 | 566403 | 1088406 | 1100000 |
| 2 | 308650 | 750489 | 1026057 | 1100000 |
| 3 | 180758 | 636676 | 1014462 | 1100000 |
| 4 | 307098 | 755869 | 939811 | 1100000 |

* The results show that all the increments are able to be completed when the program finishes execution showing that the semaphore was able to restrict access to the shared variable.
* Only in process 4 are the increments complete. In the first three processes, the increments can happen in any order as seen with process 1, when it finishes, the results are 359650, 308650, 180758 and 307098 showing that with different runs, the process has different outputs. This can be said with process 2 and 3.
* Process 4 shows the effect of the semaphore in access control as it shows by having the same value that even though the increments may happen in different orders in the previous three processes, they all culminate in the when all increments have occurred with the right numbers.

### Summary of results

* The results show consistent final value as the increments are made. They also show that the increments do not have an order for occurrence seen through processes 1,2 and 3.

### Possible causes

* The semaphore restricts access to the shared variable and prevents race conditions but does not give a specific order of access to the variable.

## Recommendations

* Improved memory synchronization: it will help with preventing inconsistencies by making sure that one process has access to the shared variable at a time and provides an order of access to the variable.

# Conclusion

The results from the program suggest the need for further process synchronization and improved access to shared memory to ensure that a program runs as expected and that an order of access to the shared variable is utilized to ensure that each process always has a consistent output.