**Project 1 - 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 written in C to perform shared memory computations. 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.

# 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. This project implemented a shared memory operation and created multiple processes that all incremented a shared variable using fork() to create the processes. The variable to be incremented was stored in the shared memory segment so all the processes had access to it.

# Requirements

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

# Methodology

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

## Over of program design

* The Program created four processes where each process increments a shared variable.
* Each child process increments the shared variable total.

## 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.
* After the increments, the child processes print their results and terminate.

## Resource management

* After the child processes have completed, the parent process continues execution and detaches and removes the memory segment.

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 | 100000 | 400000 | 764215 | 906956 |
| 2 | 100000 | 394830 | 789656 | 886264 |
| 3 | 300000 | 500000 | 1000121 | 1015252 |
| 4 | 100000 | 398436 | 658264 | 782442 |

* Process 1 shows a bit of consistency by completing 100000 increments in executions except for execution 3 where it completes 300000 increments. This shows that the process can complete 100000 increments most times which is what I would expect from the code. The increase to 300000 increments in execution3 suggests that there was a concurrency issue that allowed the process to go over its expected value.
* Process 2 shows different results after execution. This difference may be due to how the processes access and increase the shared variable.
* Process 3 shows the largest difference from the results in the number of increments. This may be due to the how the process accesses the shared variable as this variable is shared among different processes.
* Process 4 also shows some variations in the number of increments also due to the way the shared variable is accessed, all the processes are accessing and incrementing the variable at almost the same time which causes the variations in results.

### Summary of results

* The results show a variation in the number of increments for each execution process. This variation may be due to several reasons.

### Possible causes

* Concurrency issues: all the processes are incrementing the same variable the processes may be fighting to access the variable causing problems
* Process scheduling: The scheduler may not be working correctly to schedule the processes leading to changes as to when processes gain access to the shared variable which may cause differences in the results.

## Recommendations

* Implement synchronization: it will help with preventing inconsistencies by making sure that one process has access to the shared variable at a time.
* Restrict memory access: It will help maintain consistency

# Conclusion

The results from the program suggest the need for process synchronization and improved access to shared memory to ensure that a program runs as expected. Processes may compete for resources leading to inaccurate results or unexpected results. This may be improved with synchronization and restricting memory access.