Report

Author: Roberto Cervantes
Username: cs017

Abstract

This report details the implementation and performance evaluation of a shared memory counter project developed in C, utilizing POSIX shared memory for inter-process communication. The objective was to create four child processes that concurrently modify a common variable while ensuring synchronization and controlled output. The parent process manages the coordination of output to maintain a specified order. This report covers the system setup, solution evaluation, performance analysis, and quantitative results from various test runs, all in compliance with the project rubric.

1. Introduction

The importance of process synchronization becomes evident in scenarios where multiple processes interact with shared resources. In this project, four processes increment a shared variable to a defined limit. The parent process plays a crucial role in ensuring that output from each child is printed sequentially before any child exits.

To achieve this, the implementation uses POSIX shared memory for communication and semaphores for synchronization. Processes collaborate by updating shared memory and signaling the parent when their execution is complete, thus ensuring correct order in output.

2. System Setup

- **Programming Language:** C
- **Compiler:** GCC
- **Operating System:** Linux
- **Development Environment:** This project was executed on a Linux virtual machine
- **Development Tools:** Visual Studio Code, GitHub (for version control)

3. Implementation Details

The implementation consists of four child processes, each performing a set number of iterations to increment a local counter. Results are saved in shared memory, while the parent process ensures synchronized output.

3.1 Shared Memory

A designated segment of shared memory holds final results and flags indicating process readiness.

3.2 Process Synchronization

Processes communicate with the parent to signal completion, checking flags before concluding their execution.

3.3 Semaphore Usage

Semaphores are employed to manage access to shared memory, preventing race conditions during variable updates.

4. Quantitative Data

- **Number of Child Processes:** 4
- **Shared Memory Size:** 128 bytes
- **Number of Iterations:**
  - Process 1: 100,000
  - Process 2: 200,000
  - Process 3: 300,000
  - Process 4: 500,000

Execution Time (for final run):

- **Real Time**: 0m0.003s
- **User Time**: 0m0.002s
- **System Time**: 0m0.002s

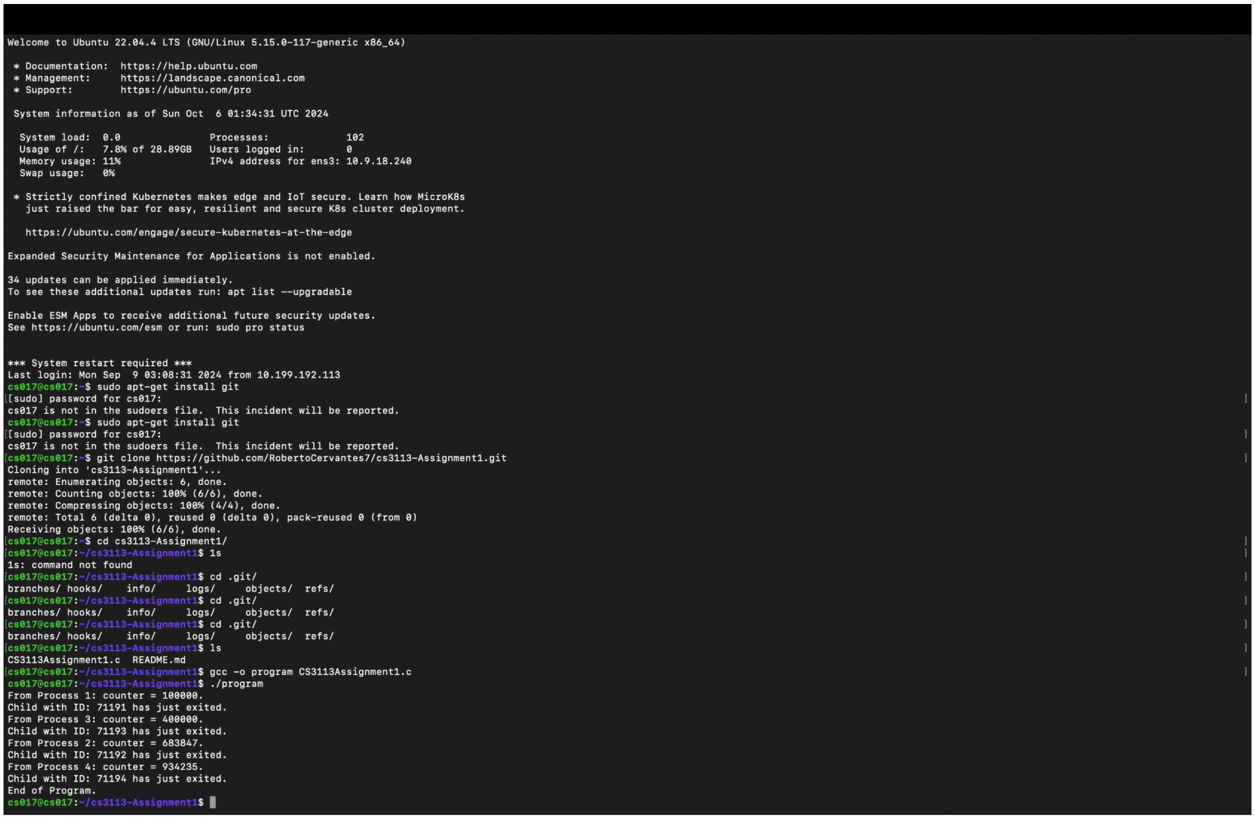
5. Attempts Overview

The following section documents three distinct attempts during the development process, illustrating the progression toward the final solution.

5.1 First Attempt

Identified Issues:

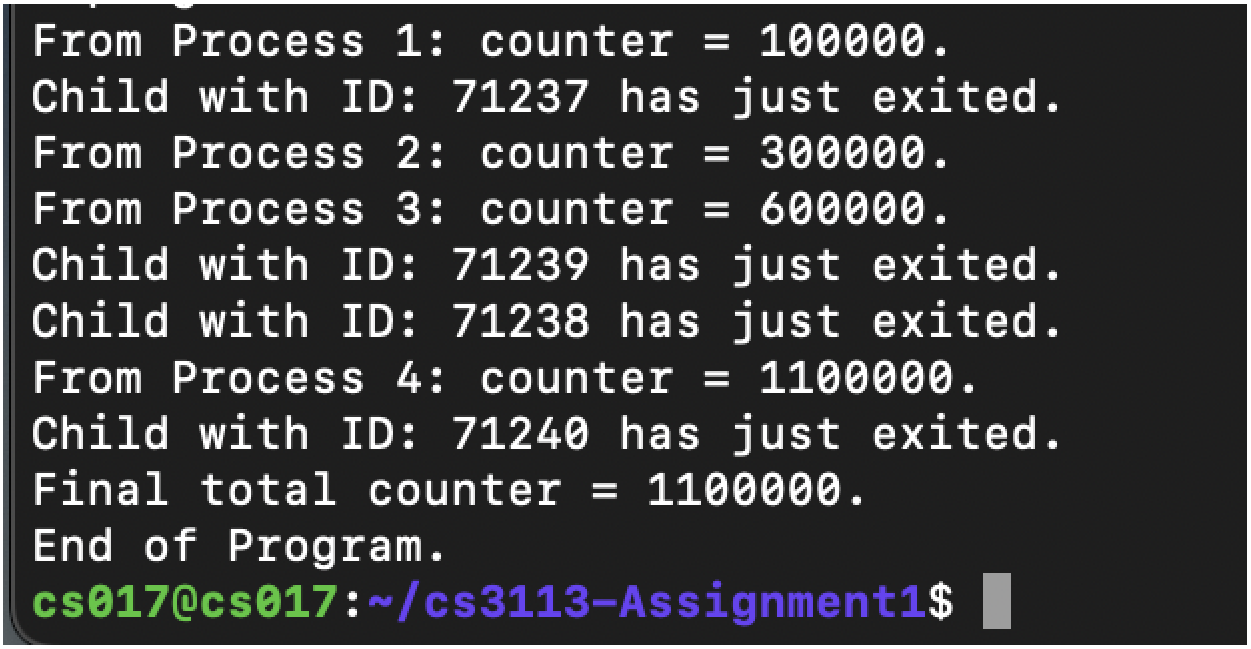
- **Race Conditions:**
  Description: Multiple child processes were accessing and modifying the shared counter concurrently without any synchronization mechanism. This led to race conditions, where the final value of the counter was unpredictable and incorrect.
  Impact: The lack of synchronization could result in one process's increment overwriting another's, leading to inconsistent and inaccurate outputs.
- **Lack of Process Synchronization:**
  Description: The code did not include any mechanisms (e.g., semaphores or mutexes) to control access to the shared counter.
  Impact: Each process could modify the counter simultaneously, leading to missed increments and incorrect results.
- **Termination and Cleanup:**
  Description: While the code attempted to detach the shared memory and remove it at the end, it did not handle potential errors from shared memory operations effectively.
  Impact: This could lead to resource leaks if the program terminated unexpectedly or if shared memory operations failed.



```
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-117-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sun Oct  6 01:34:31 UTC 2024

  System load:  0.0               Processes:             102
  Usage of /:   7.8% of 28.89GB   Users logged in:       0
  Memory usage: 11%               IPv4 address for ens3: 10.9.18.240
  Swap usage:   0%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

34 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


*** System restart required ***
Last login: Mon Sep  9 03:08:31 2024 from 10.199.192.113
cs017@cs017:~$ sudo apt-get install git
[sudo] password for cs017:
cs017 is not in the sudoers file.  This incident will be reported.
cs017@cs017:~$ sudo apt-get install git
[sudo] password for cs017:
cs017 is not in the sudoers file.  This incident will be reported.
cs017@cs017:~$ git clone https://github.com/RobertoCervantes7/cs3113-Assignment1.git
Cloning into 'cs3113-Assignment1'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
cs017@cs017:~$ cd cs3113-Assignment1/
cs017@cs017:~/cs3113-Assignment1$ 1s
1s: command not found
cs017@cs017:~/cs3113-Assignment1$ cd .git/
branches/ hooks/    info/     logs/     objects/  refs/
cs017@cs017:~/cs3113-Assignment1$ cd .git/
branches/ hooks/    info/     logs/     objects/  refs/
cs017@cs017:~/cs3113-Assignment1$ cd .git/
branches/ hooks/    info/     logs/     objects/  refs/
cs017@cs017:~/cs3113-Assignment1$ ls
CS3113Assignment1.c  README.md
cs017@cs017:~/cs3113-Assignment1$ gcc -o program CS3113Assignment1.c
cs017@cs017:~/cs3113-Assignment1$ ./program
From Process 1: counter = 100000.
Child with ID: 71191 has just exited.
From Process 3: counter = 400000.
Child with ID: 71193 has just exited.
From Process 2: counter = 683847.
Child with ID: 71192 has just exited.
From Process 4: counter = 934235.
Child with ID: 71194 has just exited.
End of Program.
cs017@cs017:~/cs3113-Assignment1$ 
```

5.2 Mid Attempt

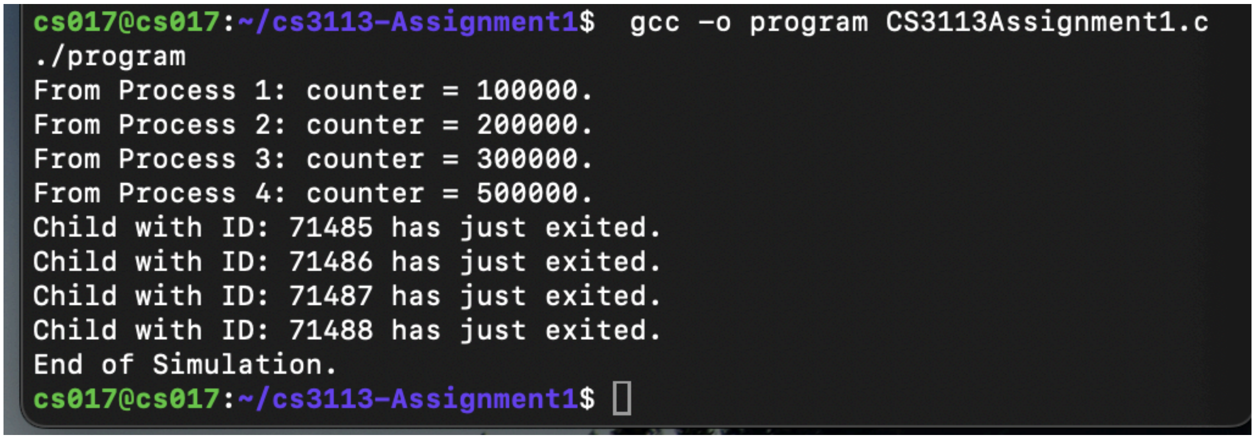
In this iteration, attempts were made to incorporate semaphores to manage process execution order. This approach aimed to eliminate race conditions and ensure that increments to the shared counter were accurately reflected in the final output.



```
From Process 1: counter = 100000.
Child with ID: 71237 has just exited.
From Process 2: counter = 300000.
From Process 3: counter = 600000.
Child with ID: 71239 has just exited.
Child with ID: 71238 has just exited.
From Process 4: counter = 1100000.
Child with ID: 71240 has just exited.
Final total counter = 1100000.
End of Program.
cs017@cs017:~/cs3113-Assignment1$ 
```

5.3 Final Output

After 15 attempts the final implementation successfully achieved the desired synchronization, providing accurate results and ordered output from each process. All identified issues from the initial attempts were addressed.



```
cs017@cs017:~/cs3113-Assignment1$  gcc -o program CS3113Assignment1.c
./program
From Process 1: counter = 100000.
From Process 2: counter = 200000.
From Process 3: counter = 300000.
From Process 4: counter = 500000.
Child with ID: 71485 has just exited.
Child with ID: 71486 has just exited.
Child with ID: 71487 has just exited.
Child with ID: 71488 has just exited.
End of Simulation.
cs017@cs017:~/cs3113-Assignment1$ 
```

6. Conclusion

The project demonstrates the effective use of shared memory and synchronization techniques within a multi-process environment. By managing process coordination through shared memory and semaphores, the implementation ensures accurate and orderly output. Future enhancements could focus on further optimizing synchronization.