COP 4600 Project 1

Shared Memory

Diya Jain[†]
CSE Department
University of South Florida
Tampa FL USA
diyajain@usf.edu

ABSTRACT

This report explores the creation of cooperating processes in C that share memory. Four child processes are implemented, each incrementing a common counter variable up to different maximum values. The cooperation is analyzed between runs, revealing synchronization issues that arise. Though the processes successfully cooperate during some executions, their unsynchronized memory access frequently overwrites values, losing increments. This demonstrates the need for concurrency control when processes share state.

KEYWORDS

Child processes, shared memory, cooperating processes, Concurrency control

ACM Reference format:

Diya Jain. 2023. COP 4600 Project 1: Shared Memory.

1 Introduction

To understand synchronization challenges with shared memory, this project implements cooperating processes in C that share a counter variable. Each process increments the counter to a different maximum value.

2 Design and Implementation

The program is structured as follows:

1. Four child processes (process1, process2, process3, and process4) are created from the main process using the fork() system call.
2. Shared memory is established using the shmget() and shmat() functions, allowing each process to access and update the counter.
3. Each child process has a specific increment limit. Process1 increments the counter by 1,000,000, process2 by 2,000,000, process3 by 4,000,000, and process4 by 5,000,000.
4. Proper synchronization is achieved using the wait() system call, ensuring that the parent process waits for all child processes to complete before proceeding.
5. After each child process finishes its task, it prints the updated counter value.
6. Finally, shared memory is detached and removed using shmdt() and shmctl().

3 Results and Observation

The program was executed multiple times, and the following results were observed:

1. Each child process correctly incremented the counter within its assigned limit.
2. Proper synchronization ensured that all child processes completed before the parent process proceeded.
3. The printed counter values reflected the cumulative incrementations from all child processes.

Sample Output:

```
From Process 1: counter = 1033995
Child with ID:27935 has been exited.
From Process 2: counter = 2306934
Child with ID:27936 has been exited.
From Process 3: counter = 4102787
Child with ID:27937 has be exited.
From Process 4: counter = 5274539
Child with ID:27938 has been exited.
End of Simulation
```

Figure 1: Sample Output after running the ./executable file.

4 Conclusion

The multi-process program successfully demonstrated the concept of concurrent incrementation of a shared variable using shared memory and process synchronization. Each child process had its own increment limit, and the shared memory segments allowed them to update the counter independently. Proper synchronization ensured the correct sequence of execution.