Misha Dubuc & Dhaval Patel

Dr. C. Bai

COMSC 420

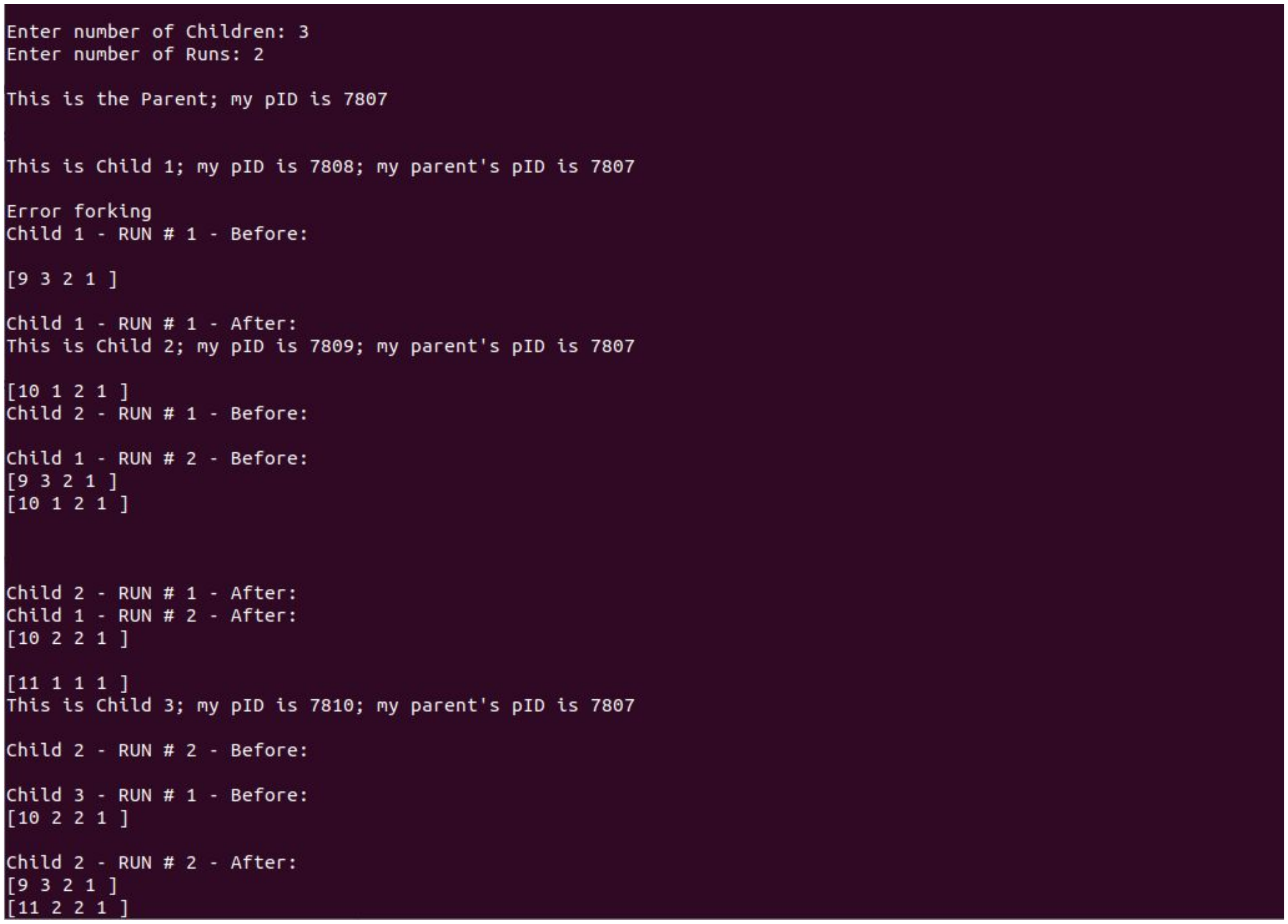
Project 3 Report

Introduction

In project 2, we were given a task to read in a text file with space-separated integers and perform operations on them and write back the changes to the file. The work involved getting the first integer from file (*n*), replacing the *n*-th integer with the process ID of the process performing operations, and incrementing *n* by one. The process ID was obtained with a fork() function and the child processes specified by the users perform the above operations for a certain user-specified number of times. We implemented this by creating an array that was dynamically allocated, depending on how many integers were read. This array was initialized in main() and passed to the work() function to execute operations and then passed back. The work() function was set up so that if there was an IndexOutOfBounds error , it cycled back and skipped the first number. The first number needs to be skipped to maintain consistent results. We planned to write back the array at the end of all child processes. Project 3 is a continuation from this point forward.

Project 2

At the end of project 2, we discovered that the output was all random and the order of commands was mixed up between processes. This was due to the lack of synchronization between the processes. Another flaw we discovered was that the array being passed was the same for all child processes being created. Due to the array being a member of the parent process, the child processes inherited the array and performed operations. Below is an example of the output from Project 2:



```
Enter number of Children: 3
Enter number of Runs: 2

This is the Parent; my pID is 7807


This is Child 1; my pID is 7808; my parent's pID is 7807

Error forking
Child 1 - RUN # 1 - Before:

[9 3 2 1 ]

Child 1 - RUN # 1 - After:
This is Child 2; my pID is 7809; my parent's pID is 7807

[10 1 2 1 ]
Child 2 - RUN # 1 - Before:

Child 1 - RUN # 2 - Before:
[9 3 2 1 ]
[10 1 2 1 ]



Child 2 - RUN # 1 - After:
Child 1 - RUN # 2 - After:
[10 2 2 1 ]

[11 1 1 1 ]
This is Child 3; my pID is 7810; my parent's pID is 7807

Child 2 - RUN # 2 - Before:

Child 3 - RUN # 1 - Before:
[10 2 2 1 ]

Child 2 - RUN # 2 - After:
[9 3 2 1 ]
[11 2 2 1 ]
```

Fig. 1 - Project 2 output (without process synchronization)

Project 3 Modifications

There were a number of changes made in Project 3.

- Semaphore

  A semaphore was used to synchronize the processes. The semaphore functions from SemSupport.c were used to implement the semaphore in the main project file.

- File I/O

  We changed the previous File I/O by including reading, creating array, performing operations and then writing back to file in the work()

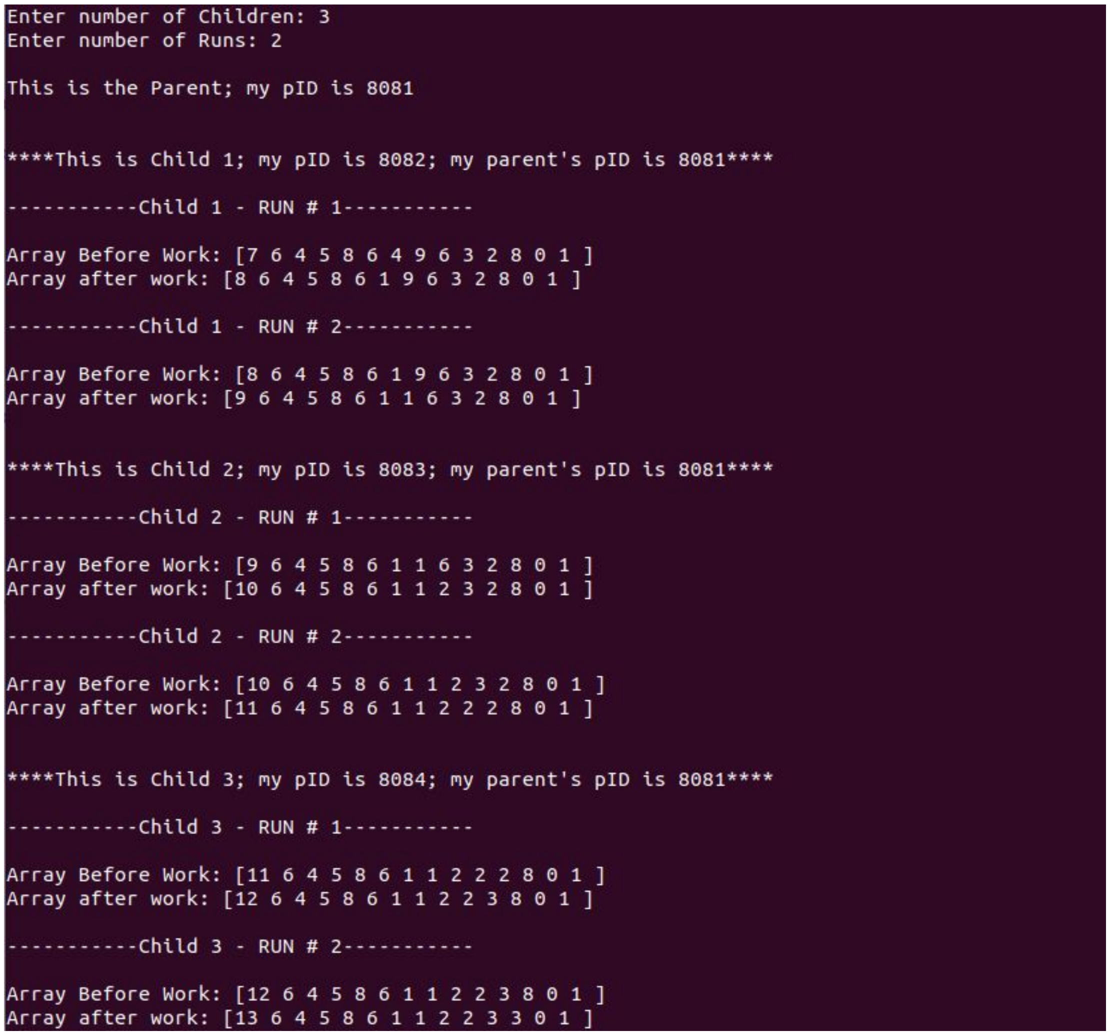
function. The work() function was then called in synchronicity within the semaphore-protected section of the main file by child processes.

- <u>wait() system call</u>

  We ran into a problem where the parent process forked and created child processes and then moved onto remove the semaphore before child processes were done using the semaphore. This was solved by implementing the wait() system call which waited for child processes to terminate before continuing on to remove the semaphore.

In conclusion, we effectively implemented a semaphore for synchronization. As seen in Figure 2, the difference between the outputs is clear in that we are now able to get consistent results from the same input and iterations of work() being done. The wait() system call also removed errors in semaphore calls and the output file reflects the changes made to the original input.

Fig. 2 - Project 3 output (with process synchronization)



```
Enter number of Children: 3
Enter number of Runs: 2

This is the Parent; my pID is 8081


****This is Child 1; my pID is 8082; my parent's pID is 8081****

-----------Child 1 - RUN # 1-----------

Array Before Work: [7 6 4 5 8 6 4 9 6 3 2 8 0 1 ]
Array after work: [8 6 4 5 8 6 1 9 6 3 2 8 0 1 ]

-----------Child 1 - RUN # 2-----------

Array Before Work: [8 6 4 5 8 6 1 9 6 3 2 8 0 1 ]
Array after work: [9 6 4 5 8 6 1 1 6 3 2 8 0 1 ]


****This is Child 2; my pID is 8083; my parent's pID is 8081****

-----------Child 2 - RUN # 1-----------

Array Before Work: [9 6 4 5 8 6 1 1 6 3 2 8 0 1 ]
Array after work: [10 6 4 5 8 6 1 1 2 3 2 8 0 1 ]

-----------Child 2 - RUN # 2-----------

Array Before Work: [10 6 4 5 8 6 1 1 2 3 2 8 0 1 ]
Array after work: [11 6 4 5 8 6 1 1 2 2 2 8 0 1 ]


****This is Child 3; my pID is 8084; my parent's pID is 8081****

-----------Child 3 - RUN # 1-----------

Array Before Work: [11 6 4 5 8 6 1 1 2 2 2 8 0 1 ]
Array after work: [12 6 4 5 8 6 1 1 2 2 3 8 0 1 ]

-----------Child 3 - RUN # 2-----------

Array Before Work: [12 6 4 5 8 6 1 1 2 2 3 8 0 1 ]
Array after work: [13 6 4 5 8 6 1 1 2 2 3 3 0 1 ]
```