OS Project 1: Experiments In Forking

Marco Salazar
Department Of Computer Science
New Mexico State University
Las Cruces, NM, USA
marcoams@nmsu.edu

ABSTRACT

In this project the purpose was to discover some of the concepts of shared memory and the problems that occur if the shared memory does not have protection from other processes. As such I created a program which creates four children processes and increments a shared piece of data. Each process exits when a certain number is reached and prints out the current value of the number. Finally the main process prints out when each of the four children processes have ended.

From this I found that without adequate protection from mutex locks or semaphores, the shared data would not preserve its ideal state long enough to be printed, since printing is an IO operation that takes longer to accomplish, thus there is more time for the other three children processes to mess up with the data.

KEYWORDS

Processes, Protection, Shared Memory.

1 Process

The shared memory was created through a simple procedure. A struct was created and defined, and a global variable was made of that type. It was initialized in the main function with shmeat, and the result of shmeat. Once this shared memory was made, the four children processes were created. Each process called its own method that would loop and continuously increment the variable, and exit when 100,000, 200,000, 300,000, and 500,000 were reached respectively. Finally the process would exit so as to not use any of the other code.

In the meantime, the parent process waited for each process individually and printed out exactly when the process had finished executing. It finally released the shared memory segments so that they would not persist in the system memory forever. Quick checks with ipcs before and after the execution helped to verify that there were no left over shared memory.

2 Results

The following results were found when executed on the NMSU CS department lab computers that run at 1920 MHz on average based on the lscpu command. Through a trial of ten runs the half executed just as a naïve run would expect printing out:

```
From Process 1: Counter = 100000
Child 31430 pid has just exited.
From Process 2: Counter = 200000
Child 31431 pid has just exited.
From Process 3: Counter = 300000
Child 31432 pid has just exited.
From Process 4: Counter = 500000
Child 31433 pid has just exited.
```

However, the other runs had slight differences, where the counters were not the values they were supposed to be, such as:

```
From Process 1: Counter = 100000
Child 31461 pid has just exited.
From Process 2: Counter = 200085
Child 31462 pid has just exited.
From Process 3: Counter = 300000
Child 31463 pid has just exited.
From Process 4: Counter = 500000
Child 31464 pid has just exited.
```

The time in microseconds to run the program took a total of 4037 microseconds.

I also ran some tests where the breaking numbers were each multiplied by 10, and I found that there was a higher amount of the results that had corrupted or changed data.

3 Analysis

In the second result, Process 2 could not perform the IO operation of printing out the value of Counter fast enough before the other 3 processes caused the value of the counter to increase 85 above its desired value.

This shows that without the proper protection on shared memory, whether it is through semaphores or mutex locks, there is no guarantee that the shared variable will not be corrupted, or unchanged.