Project1 (Shared Memory)

Jeffrey Lansford
10/4/2020

Introduction

In this project we are introduced to working with shared memory in a C program and seeing the problems that can occur if shared memory is not protected adequately. I created a C program that creates 4 child process and incremates the shared memory variable “value” in a loop to 100,000, 200,000, 300,000 and 500,000 respectively and print out the total after it is done. Then the parent process waits for each child to finish and release the shard memory.

Code

See project1.c in zip file

Results

After running the program 10 times, here is the captured output:

~~~~~~1~~~~~~
~~~~~~

Form Process1: Counter = 100145

Form Process2: Counter = 200000

Form Process3: Counter = 300052

Form Process4: Counter = 500000

Child 1 24355 pid has just exited.

Child 2 24356 pid has just exited.

Child 3 24357 pid has just exited.

Child 4 24358 pid has just exited.

End of program

~~~~~~2~~~~~~

Form Process1: Counter = 100000

Form Process2: Counter = 200000

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24360 pid has just exited.

Child 2 24361 pid has just exited.

Child 3 24362 pid has just exited.

Child 4 24363 pid has just exited.

End of program

~~~~~~3~~~~~~

Form Process1: Counter = 100000

Form Process2: Counter = 200000

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24365 pid has just exited.

Child 2 24366 pid has just exited.

Child 3 24367 pid has just exited.

Child 4 24368 pid has just exited.

End of program

~~~~~~4~~~~~~

Form Process1: Counter = 100000

Form Process2: Counter = 199886

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24370 pid has just exited.

Child 2 24371 pid has just exited.

Child 3 24372 pid has just exited.

Child 4 24373 pid has just exited.

End of program

~~~~~~5~~~~~~

Form Process1: Counter = 100000

Form Process2: Counter = 200000

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24375 pid has just exited.

Child 2 24376 pid has just exited.

Child 3 24377 pid has just exited.

Child 4 24378 pid has just exited.

End of program

~~~~~~6~~~~~~

Form Process1: Counter = 100226

Form Process2: Counter = 200147

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24380 pid has just exited.

Child 2 24381 pid has just exited.

Child 3 24382 pid has just exited.

Child 4 24383 pid has just exited.

End of program

~~~~~~7~~~~~~

Form Process1: Counter = 100000

Form Process2: Counter = 200000

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24385 pid has just exited.

Child 2 24386 pid has just exited.

Child 3 24387 pid has just exited.

Child 4 24388 pid has just exited.

End of program

~~~~~~8~~~~~~

Form Process1: Counter = 100000

Form Process2: Counter = 200000
~~~~~~

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24390 pid has just exited.

Child 2 24391 pid has just exited.

Child 3 24392 pid has just exited.

Child 4 24393 pid has just exited.

End of program

~~~~~~9~~~~~~

Form Process1: Counter = 100000

Form Process2: Counter = 200000

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24395 pid has just exited.

Child 2 24396 pid has just exited.

Child 3 24397 pid has just exited.

Child 4 24398 pid has just exited.

End of program

~~~~~~10~~~~~~

Form Process1: Counter = 100140

Form Process2: Counter = 199997

Form Process3: Counter = 300000

Form Process4: Counter = 500000

Child 1 24400 pid has just exited.

Child 2 24401 pid has just exited.

Child 3 24402 pid has just exited.

Child 4 24403 pid has just exited.

End of program

Conclusions

After looking at the 10 runs of the program, we can see that on some of the runs do not exactly stop on the respectively value that it is supposed to stop on. For instance, we can see that the first process on run 1 does not print 100000, but 100145 instead and on process 3, it is 300052 and not 300000 as we expect. Even some runs like run 10, the number is less then the amount that it should have stopped at. What is causing the inconsistencies? Well each process is running at the same time and all 4 of them are changing the shared memory variable value. All 4 of them pull value into a register, then add 1 to it, then store it back into the memory location. When each one stores it back, they are not doing right after each other, they are doing at the same time, so when one process is pulling a value out of memory, another process can be storing a new value into the memory location. This can cause inconsistently when the loop my stop at 100,000, put when it prints the value, the value ahs been already changed by another process. This can cause the value being higher or lower than the value it is supposed to print. Though compared to the sample output, it is not as inconsistent as that. This could be that the computers can process the 4 child processes very fast after the print so that it can print the value that we want it to stop on. But again, from 10 runs, 4 runs printed values that are not excepted. 60% chance for it to run correctly is not very reliable especially for an Operating System.