William Strouse Jr
CSC 328
HW2

Program Design

Problem Statement

Print a few lines of output from separate processes or threads.

Analysis

Inputs	N/A
Outputs	Each process/thread will have 6 lines of output, the process ID and an incrementing value for that line within the child process:
Formulas	We need to have a logical loop for the lines inside of the process/thread to increment a variable counting the lines.
Constraints	6 lines of output in the logical loop
Assumptions	

Design

Program will declare a variable to capture the return value of the process/thread creation to perform operations.
The program will spawn a new process and the child process created will print out 6 lines of text. We do this by declaring an incrementing logical loop that will display the process ID of the current process and the current value of the incrementing variable used to control the loop.
Example: "ProcessID: ###### Line 00#"
The child process upon completing the loop will return/exit cleanly back to the parent process.
The program must call 5 separate processes to run concurrently, therefore providing enough data for the scheduler to have to switch between each process at least 1x before exiting.
The parent program must wait until all children are returned before exiting, avoiding costly orphaned processes or zombie processes in the process table.

Test Plan

This program is designed to provide the illusion to the end user is that the programs are all running simultaneously. Our program will expose the fact that context switching is occurring at extremely rapid speeds, and we will demonstrate that with our output which will be variations of each processID of the child and associated line number it was able to increment to until the timeslice provided by the scheduler has expired.

William Strouse Jr
CSC 328
HW2

Program Design

The following is a sample, results may vary on actual equipment:

Input	**Expected Output**
(First child process)	ProcessID: 548610 Line 001
	ProcessID: 548610 Line 002
(Second child process is spawned)	ProcessID: 559744 Line 001
	ProcessID: 559744 Line 002
	ProcessID: 559744 Line 003
(Third child process is spawned)	ProcessID: 608013 Line 001
	ProcessID: 608013 Line 002
(First child process continues)	ProcessID: 548610 Line 003
	ProcessID: 548610 Line 004
	ProcessID: 548610 Line 005
(Fourth child process is spawned)	ProcessID: 621748 Line 001
	ProcessID: 621748 Line 002
	ProcessID: 621748 Line 003
	ProcessID: 621748 Line 004
(Fifth child process is spawned)	ProcessID: 638794 Line 001
	ProcessID: 638794 Line 002
(First child completes)	ProcessID: 548610 Line 006
(Second child process completes)	ProcessID: 559744 Line 004
	ProcessID: 559744 Line 005
	ProcessID: 559744 Line 006
(Third child process completes)	ProcessID: 608013 Line 003
	ProcessID: 608013 Line 004
	ProcessID: 608013 Line 005
	ProcessID: 608013 Line 006
(Fifth child process completes)	ProcessID: 638794 Line 003
	ProcessID: 638794 Line 004
	ProcessID: 638794 Line 005
	ProcessID: 638794 Line 006
(Fourth child process completes)	ProcessID: 621748 Line 005
	ProcessID: 621748 Line 006

Time Estimates

Task	Estimated Time	Actual Time
Development/Debugging	3 hours	
Testing	1 hour	
Execution	<10 seconds	

William Strouse Jr
CSC 328
HW2

Program Design

Process vs Thread program comparison

First thing you would need is a baseline of each of the programs recorded on the same system of what the CPU load, memory usage, number of processes, disk I/O, number of users, etc. Once a baseline metric analysis has been recorded using either commands, scripts, or a third party system, it is time to run each program capturing the execution time each program takes. The comparison would have to take place on the same system using similar system conditions. We can verify this by checking the baseline metrics of the system when running each program. Once the programs execute and the execution times are captured, that is when the analysis phase can begin. There are key performance indexes that would have to be determined for the need of the program.

Note: One method of capturing the metrics we need to gather our KPIs is to use the command top and have that write to a txt file - http://www.thegeekstuff.com/2009/10/how-to-capture-unix-top-command-output-to-a-file-in-readable-format/

We want to see which program was faster, which program had lower impact on the CPU, and which had the smallest footprint in our system's memory. Since we timed the two programs, we can simply see which program provided the fastest execution time to completion. We can sift through the captured data from our commands/scripts or third party tools which were monitoring the CPU load to see how much of a percentage each program occupied the CPU, and for how long. Finally we can check the logs to see which program absorbed memory during the context switching process, and get a good picture of all three to tell us which program ran more efficiently on the computer. The other factor is the time it takes to code the programs. Is the difference worth the extra development time? This may depend on the scale of the project since our test is minute compared to other operations spawning thousands of processes and threads to handle the demand.