Project Report #1

Minimum Time to Start a New Process Using fork()

Author: Sujeet Ojha

Date: 1/28/2019

I. Problem Statement

The assigned task was to determine the time it takes to start a new process using the function fork() and gettimeofday() in C/C++. The function fork() allows a process to be copied and be granted its own independent address space. The importance of this method is that child processes are able to run independently while receiving a copy of the parent process' memory segments. As elapsed times, the entity used to determine the time to start a new process, can vary depending on errors, system state, and other criterion, a notable amount of measurements must be taken. These measurements must be portrayed in a histogram which can be used to conclude the most accurate start time for the child processes.
Note: The terms elapsed time and process start time are interchangeable in this report.

II. Approach

The programming language selected for this project is C. The approach taken was to utilize the gettimeofday() function to get the reference start time before calling the fork() function. Right after the fork() function is called, the gettimeofday() function was called in order to get the accurate reference end time after fork() function was called. This ensures that the measured reference start time is kept the same for both the child and the parent processes while the reference end time varies.

The reference start time and end time will be used to generate an elapsed time for each process. The calculated elapsed time will be used to generate a statiscal answer to determine the time it takes to start a new process using the function fork(). The biggest concern with this method is keeping account of all active processes. Therefore, methods such as wait() should be utilized in order to ensure that the heap is not overloaded and further errors are prevented.

All of the measurements gathered for the child and parent processes will be outputted and stored into separate text files in order to create a histogram in EXCEL at a later time. The histogram created using the data points will be used to analyze discrepancies and formulate an answer for the amount of time it takes to start a new process using the function fork().

III. Challenges

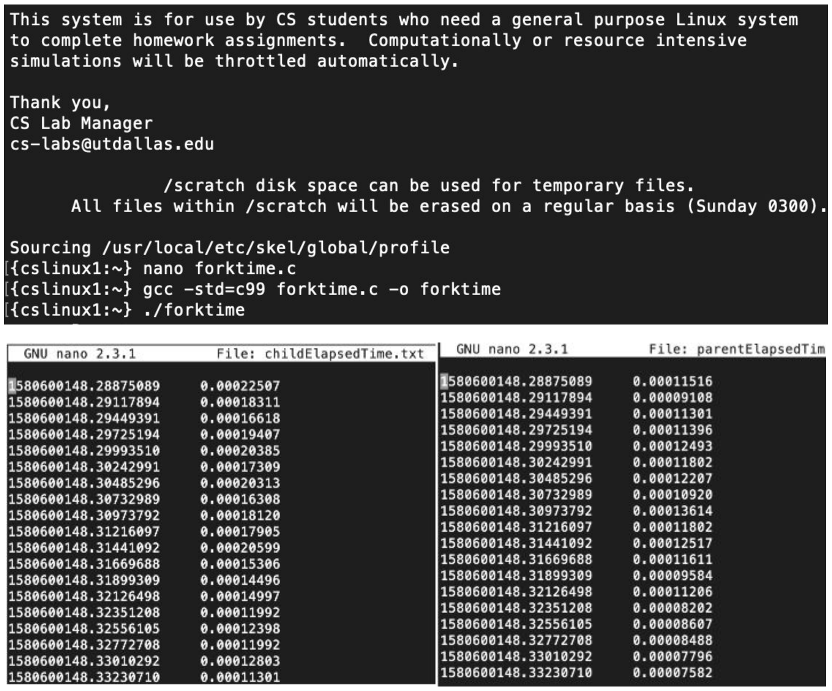
Since the fork() function does not create a new process as soon as it is called, there are challenges present to this problem. Without inspecting the details of the fork() function, it is nearly impossible to find a precise answer. Therefore, although the final determination will not be precise, the approach taken will ensure that the answer is statistically accurate.

IV. Solution

In order to ensure that the elapsed time for each new process is collected without any errors, text files (parentElapsedTime.txt and childElapsedTime.txt) are used to store the required data. The data stored in these text files include the reference start time and the elapsed time for each process which can be used to create a histogram and generate additional plots. Parent process data was collected and stored separately in order to further analyze the discrepancies in elapsed times between child and parent processes. In order to get a statistically accurate representation, the program iterates over 2000 times in order to get an abundant amount of data points. The minimum elapsed time, notated as the minimum amount of time a process takes to start, is then chosen as the final answer since any other times recorded can be considered discrepancies. Error-handling mechanisms are included in order to ensure that the program does not face any preventable bugs.

Figures:

- *Plot (1):* Plot (1) depicts the build and execution of the program in the CS1 linux server.
- *Plot (2):* Plot (2) depicts a histogram of the time it takes to start a new process for all recorded processes.
- *Plot (3):* Plot (3) separately depicts the parent and child process start time (elapsed time) trend with respect to the reference start time.



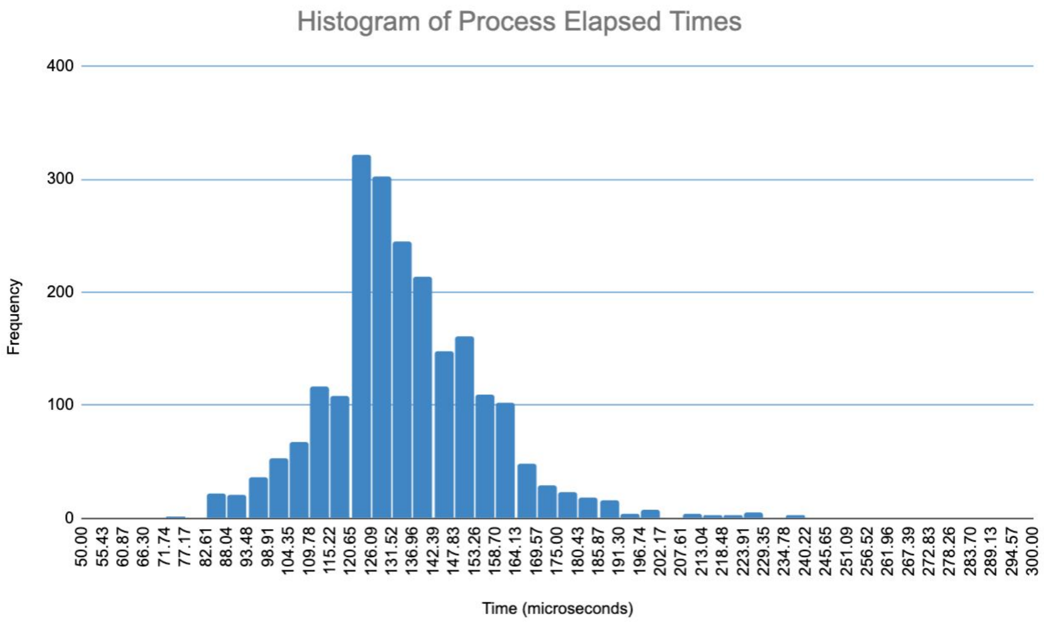
```
This system is for use by CS students who need a general purpose Linux system
to complete homework assignments.  Computationally or resource intensive
simulations will be throttled automatically.

Thank you,
CS Lab Manager
cs-labs@utdallas.edu

               /scratch disk space can be used for temporary files.
      All files within /scratch will be erased on a regular basis (Sunday 0300).

Sourcing /usr/local/etc/skel/global/profile
[{cslinux1:~} nano forktime.c
[{cslinux1:~} gcc -std=c99 forktime.c -o forktime
[{cslinux1:~} ./forktime
```

```
  GNU nano 2.3.1          File: childElapsedTime.txt

1580600148.28875089     0.00022507
1580600148.29117894     0.00018311
1580600148.29449391     0.00016618
1580600148.29725194     0.00019407
1580600148.29993510     0.00020385
1580600148.30242991     0.00017309
1580600148.30485296     0.00020313
1580600148.30732989     0.00016308
1580600148.30973792     0.00018120
1580600148.31216097     0.00017905
1580600148.31441092     0.00020599
1580600148.31669688     0.00015306
1580600148.31899309     0.00014496
1580600148.32126498     0.00014997
1580600148.32351208     0.00011992
1580600148.32556105     0.00012398
1580600148.32772708     0.00011992
1580600148.33010292     0.00012803
1580600148.33230710     0.00011301
```

```
  GNU nano 2.3.1          File: parentElapsedTim

1580600148.28875089     0.00011516
1580600148.29117894     0.00009108
1580600148.29449391     0.00011301
1580600148.29725194     0.00011396
1580600148.29993510     0.00012493
1580600148.30242991     0.00011802
1580600148.30485296     0.00012207
1580600148.30732989     0.00010920
1580600148.30973792     0.00013614
1580600148.31216097     0.00011802
1580600148.31441092     0.00012517
1580600148.31669688     0.00011611
1580600148.31899309     0.00009584
1580600148.32126498     0.00011206
1580600148.32351208     0.00008202
1580600148.32556105     0.00008607
1580600148.32772708     0.00008488
1580600148.33010292     0.00007796
1580600148.33230710     0.00007582
```

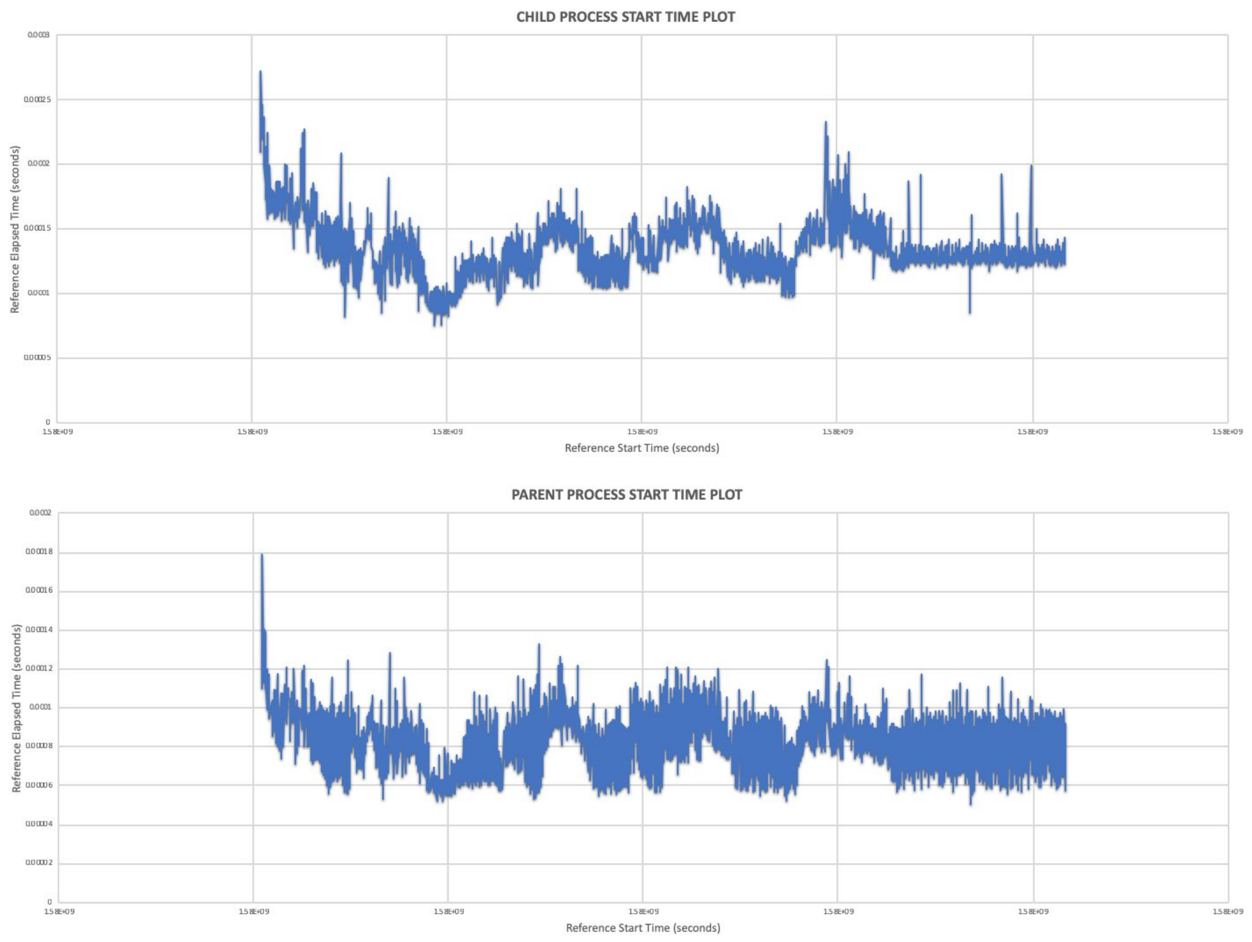
Plot (1): Build and execution of the program

The "cslinux1" server was used to build and execute the code. The files (childElapsedTime.txt and parentElapsedTime.txt) shown above are created by the program and reference start times and elapsed times for each process are stored into these files. The data gathered from these files are then used to create a histogram and related plots.



Plot (2): Histogram of the Process Start Times

This histogram keeps account of the elapsed times for both the parent and the child processes. As the required task does not make a distinction between the amount of time it takes to start parent and child processes, the reason the parent and the child process data is clustered together is because there are times where the parent processes have lower elapsed times than the child processes and vise versa. Therefore, all of the created processes must be considered the same and all data should be analyzed in a similar fashion. Analyzing the histogram, it is clear that the highest frequency of elapsed times was somewhere between 120µs - 140µs. However, the outlier with the smallest elapsed time will determine the minimum amount of time it takes to start a new process.



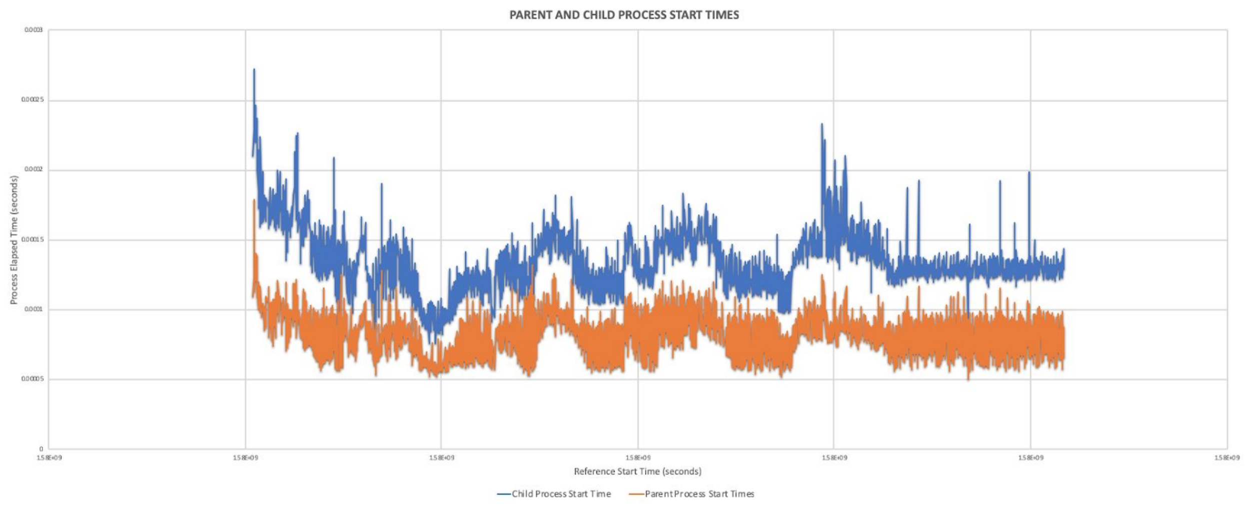
Plot (3): Process Start Times Graphs

Plot (3) clearly illustrates the discrepancies in process start times for the child processes and parent processes with respect to the reference start time. The plot can be used to conclude that the process start time for any given process is unlikely to be the same as other internal and external factors can cause the time to vary. The minimum amount of time taken to start a parent process is 0.00005603 seconds while the minimum amount of time to start a child process is 0.00012088 seconds. As discussed in the section above, parent and child processes should be analyzed without labels and treated the same. Therefore, the total minimum time it takes to start a new process using fork(): 0.00005603 seconds.

V. Observations

Figures:

- *Plot (4):* Plot (4) depicts the elapsed times for the parent processes and the related child processes in a single graph



Plot (4): Parent and Child Process Discrepancies

Upon completing the given task, a few notable observations came to rise. Analyzing *Plot (4)*, it can be asserted that statistically, child processes generally take a longer time to start than parent processes. Although this is not always the case, the figure demonstrates this to be true for the sample illustrated.

Another key aspect of the graph are the discrepancies. Apart from an extended process elapsed times, the child processes seem to follow a similar behavior as the parent processes. The trend illustrates that the child and the parent processes generally face discrepancies at the same times.

As illustrated above, even with 2200 data points, there is an abundant amount of discrepancies. The discrepancies can be explained by the system state and other events that slow down the creation of processes.