CS 111 - Lab 1c Performance Analysis

Wesley Minner & Christie Mathews
703549234 104404412

Test Cases

Test 1

- Bash: cat z | tr A-Z a-z | tr -sd 'b' 'a' > y
- Simpsh: ./simpsh --profile --rdonly z --pipe --pipe --wronly y --wronly e --command 0 2 6 cat - --command 1 4 6 tr A-Z a-z --command 3 5 6 tr -sd 'b' 'd' --close 2 --close 4 --wait
- Execline (script execline1):

```
redirfd -w 1 y
pipeline { cat z }
pipeline { tr A-Z a-z }
foreground { tr -sd ‘b’ ‘a’ }
```

Test 2

- Bash: cat a | tr A-Z a-z | sort > b
- Simpsh: ./simpsh --profile --rdonly a --creat --wronly b --creat --wronly c --pipe --pipe --command 0 4 2 cat --command 3 6 2 tr A-Z a-z --command 5 1 2 sort --close 4 --close 6 --wait
- Execline (script execline2):

```
redirfd -w 1 b
pipeline { cat a }
pipeline { tr A-Z a-z }
foreground { sort }
```

Test 3

- Bash: head -c 1MB /dev/urandom | tr -s A-Z a-z | cat > b
- Simpsh: ./simpsh --profile --rdonly a --creat --wronly b --creat --wronly c --pipe --pipe --command 0 4 2 head -c 1MB /dev/urandom --command 3 6 2 tr -s A-Z a-z --command 5 1 2 cat --close 4 --close 6 --wait
- Execline (script execline2):

```
redirfd -w 1 b
pipeline { head -c 1MB /dev/urandom }
pipeline { tr -s A-Z a-z }
foreground { cat }
```

Data

Each test case was performed five times on each program. The average of the five trials is listed below for each test case. See the full set of data in the file “time_results”, which was generated from script file “time.sh” (make time).

TEST 1

CRITERIA	BASH	SIMPSH	EXECLINE
User Time (s)	0.001	0.000726	0
System Time (s)	0.003	0.00173	0.003
Max resident set size (KB):	659	671	672
Block Input:	0	0	0
Block Output:	0	8	8
Page Reclaim:	213	696	917
Page Faults:	0	0	0
Voluntary Context Switches:	3	11	12
Involuntary Context Switches:	1	3	6

TEST 2

CRITERIA	BASH	SIMPSH	EXECLINE
User Time (s)	0.001	0.000298	0.001
System Time (s)	0.003	0.002287	0.003
Max resident set size (KB):	659	882	880
Block Input:	0	0	0
Block Output:	0	8	8
Page Reclaim:	213	760	980
Page Faults:	0	0	0
Voluntary Context Switches:	3	11	12
Involuntary Context Switches:	1	3	6

TEST 3

CRITERIA	BASH	SIMPSH	EXECLINE
User Time (s)	0.001	0.033367	0.001
System Time (s)	0.008	0.113074	0.008
Max resident set size (KB):	659	714	659
Block Input:	0	0	0
Block Output:	1952	1952	1952
Page Reclaim:	915	709	915
Page Faults:	0	0	0
Voluntary Context Switches:	184	315	184
Involuntary Context Switches:	21	18	21

Performance Analysis

A quick glance of the user time of bash, simpsh, and execline suggests that simpsh is the superior program, however, a true analysis must take into account more than just the user time or even system time. Just because a call returns quickly doesn't mean it is better than a slower call. The resources it consumes and how efficiently it does so must be taken into account. In our analysis, we recorded the user time, which is the time the processor spends in 'user-mode' whereas system time is the time the processes spends in 'kernel mode'. Max Resident Set Size is roughly the amount of memory in RAM that is associated with it. For Simpsh, we added this metric for the total of the children and the parent to get the overall value. Block Input/Output refers to the amount of I/O operations that were used. Page reclaims refer to how many times old RAM memory was freed because all page frames were used. Page faults occur when some memory mapped by the virtual memory doesn't agree with the memory in the corresponding main memory. Voluntary context switches occur when a thread naturally gives up control to the CPU/scheduler whereas in an involuntary context switch, the scheduler interrupts a thread mid process and lends itself to another thread. Obviously, having more involuntary context switches is bad as the state must be saved and more care must be taken to switch back into it.

Based on the above metrics, we can now better analyze the strength and weaknesses of each type of program to run shell commands: bash, simpsh, and execline. In general:

- Simpsh
    - It appears to perform better on less intensive operations/commands, however, according to our data it does not scale well. It performs better than bash and execline for the first two tests, but for the last test which reads in a MB of data, it takes more than two magnitudes of greater time and almost twice as many context switches, which is not as efficient.The first two tests run some relatively easy commands while the last test is much more resource intensive.
- Bash
    - Bash appears to be slower than simpsh for the first two tests, but we believe this to be the result of timing issues discussed below and that the times of the simpsh and execline were calculated as faster than they really are. However, solely based on our timing data for the first two tests, bash uses fewer context switches and page reclaims than both simpsh and execline. It appears to run in roughly the same time as execline, but we also believe this to be a timing issue.
- Execline
    - Execline runs in roughly the same time as bash. This makes sense because it's implementation appears to be very close to that of bash. It tends to use more page reclaims than bash, but other than that, it's resource usage is rather close to that of simpsh.

*Timing Errors

We believe that the results we found above could not be used to make any real claims about the speed and efficiency of each of the above methods of running shell commands. This is because while time could be used to calculate the timing and resource efficiency of bash commands, it isn't as simple as calculating the time and resource efficiency of simpsh and execline. This is because both simpsh and execline rely on child process whose timing data must be accumulated. For instance, with execline, we couldn't calculate the timing for a command associated with pipe; ie. the time of pipe with a command would be smaller than that of just the command. This is because time couldn't add up the child process time. Thus, the data associated with execline can't be used to make any value statements about its speed. Additionally, getrusage() was used to calculate the data associated with simpsh and it's reasonable to assume that differences in implementations of getrusage() and time() will result in times which cannot be completely compared without note of their differences.