Operating System Final Project (strace)

April 2022

1 Introduction

When a program runs into issues such as sudden crash or unexpected incorrect output, a developer can detect where the problem is when looking the program source code and start the debugging process. Ideally a developer should always have access to the source code to detect problem, however in practice this is not always the case. Software, services or libraries made by other team can be deliver as an executable binary file and developer who works on a different product does not authorize to access to the source code of other teams. Linux provides developers a powerful tool called ”strace” with the ability to show what system call or library call a program make at a low level. When running into problems, we can use strace to roughly guess what the program does, what system call it fails to detect problems without digging into the details of source code. Please refer to the manual page: https://man7.org/linux/man-pages/man1/strace.1.html

2 Goal

Unfortunately such useful tool, strace is not built into xv6 operating system. Our goal of the project is to make a version of strace for xv6.

3 Before you start

All implementation requires some small explanations (2-3 sentences) of your approach. You can explain more if you would like but it should be less than or equal to the introduction of the project prompt.

Screenshots show result of each implementation should be provided in your report. Some requirement can be tricky to test. Your points will be deducted if screenshots of result are not provided.

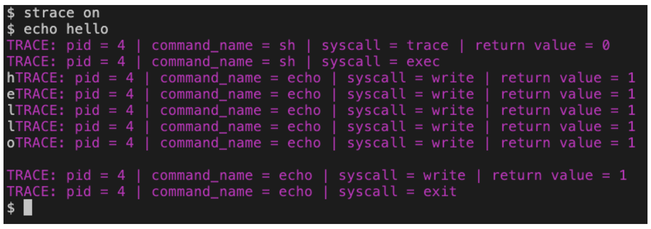
4 Tasks

1. **Get familiar with Linux strace**
   - Use strace in Anubis to learn about echo command or any other commands of your choice. Show a list of system call being made, total number of calls, time for running strace on particular command. (screenshots of your run should be enough for this task)
   - Pick 4 random system call (ex: nmap, write, open, etc) of your choice and explain its functionality for one command that you run (ex: echo, ls, cd, etc) in 2-3 sentences.

2. **Building strace in xv6**

- **Command: strace on**

  When typing "strace on" in the terminal, the mode of strace is on and therefore the next type in command will be traced. The system call list will be printed on screen in format pid (process id), command name, system call name, return value.



```
$ strace on
$ echo hello
TRACE: pid = 4 | command_name = sh | syscall = trace | return value = 0
TRACE: pid = 4 | command_name = sh | syscall = exec
hTRACE: pid = 4 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 4 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 4 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 4 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 4 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 4 | command_name = echo | syscall = write | return value = 1
TRACE: pid = 4 | command_name = echo | syscall = exit
$ 
```

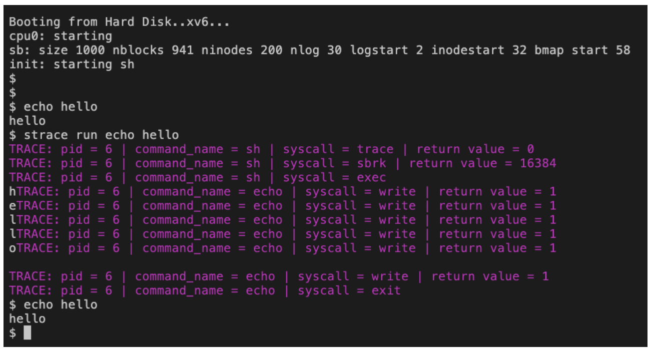
Figure 1: strace on sample

- **Command: strace off**

  When typing "strace off" in the terminal, the mode of strace is off and therefore the next type in command won't be traced and nothing is print on screen besides the result of the commands.

- **Command: strace run <command>**

  Instead of turning on and off strace, we create "strace run" to directly point tracing to the current process that execute the command. For example: when typing "strace run echo hello" in the terminal, we get the output tracing of echo hello.



```
Booting from Hard Disk..xv6...
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
$
$ echo hello
hello
$ strace run echo hello
TRACE: pid = 6 | command_name = sh | syscall = trace | return value = 0
TRACE: pid = 6 | command_name = sh | syscall = sbrk | return value = 16384
TRACE: pid = 6 | command_name = sh | syscall = exec
hTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
TRACE: pid = 6 | command_name = echo | syscall = exit
$ echo hello
hello
$ 
```

Figure 2: strace run sample

- **Command: strace dump**

  Implement a kernel memory that will save N number of latest events. This N number can be configurable by using define in XV6. In order word, it can be hard code but the way to implement is to use define to declare a variable called N with certain value. When "strace dump" command is called, print all events that saved in kernel memory.

- **Trace child process**

  Write a program that uses fork to spawn child processes. Utilize kernel memory that you have previously implemented to store trace output of all processes. Perform strace on this program and print all trace of processes that you have spawn.

- **Extra credits: Formatting more readable output**

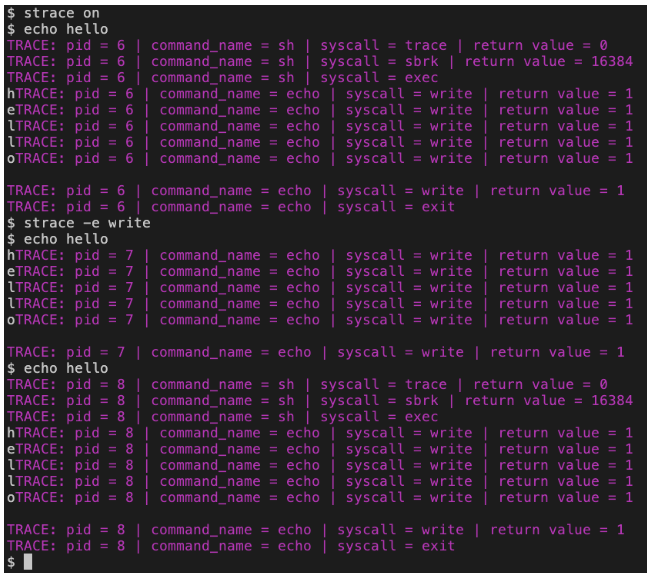
  Depends on your implementation, you might notice your command result is printed along with your strace result which cause tracing to be difficult to read when characters keep mix up between line (ex: ”ls” has a very long list result). Choose one of the follow approach to solve this issue:

  – Find a way to format your output such that result of command is printed first and strace result is printed after.
  – Find a way to suppress the command output and leave only strace output when strace is on.
  – A creative approach of your choice.

3. **Building option for strace**

- **Option: -e <system call name>**

  When option flag -e is provided follow by a system call name (ex: write), we will print only that system call. If no such system call is made in the command, print nothing.



```
$ strace on
$ echo hello
TRACE: pid = 6 | command_name = sh | syscall = trace | return value = 0
TRACE: pid = 6 | command_name = sh | syscall = sbrk | return value = 16384
TRACE: pid = 6 | command_name = sh | syscall = exec
hTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 6 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 6 | command_name = echo | syscall = write | return value = 1
TRACE: pid = 6 | command_name = echo | syscall = exit
$ strace -e write
$ echo hello
hTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 7 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 7 | command_name = echo | syscall = write | return value = 1
$ echo hello
TRACE: pid = 8 | command_name = sh | syscall = trace | return value = 0
TRACE: pid = 8 | command_name = sh | syscall = sbrk | return value = 16384
TRACE: pid = 8 | command_name = sh | syscall = exec
hTRACE: pid = 8 | command_name = echo | syscall = write | return value = 1
eTRACE: pid = 8 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 8 | command_name = echo | syscall = write | return value = 1
lTRACE: pid = 8 | command_name = echo | syscall = write | return value = 1
oTRACE: pid = 8 | command_name = echo | syscall = write | return value = 1

TRACE: pid = 8 | command_name = echo | syscall = write | return value = 1
TRACE: pid = 8 | command_name = echo | syscall = exit
$
```

Figure 3: strace with option -e sample

  Notes: Notice the first strace run is similar to regular ”echo hello” run above in figure 1. Next commands is ”strace -e write” which activate tracking write system call. We run ”echo hello” again, observe that only write system call is printed. One more run ”echo hello”, everything is back to normal without tracing only the write system call. Similar idea should be done in option -s and -f.

- **Option: -s**

  When option flag -s is provided, print only successful system call.

- **Option: -f**

  When option flag -f is provided, print only failed system call.

- **Option runs only once**

  When option is called (ex: ”strace -e write” follow by ”echo hello”), the result of strace with only write system call is printed only once for the following command ”echo hello”. If typing the next follow command for example ”ls”, the option -e is turn off and all system call will be printed. In other word, options is used once per command.

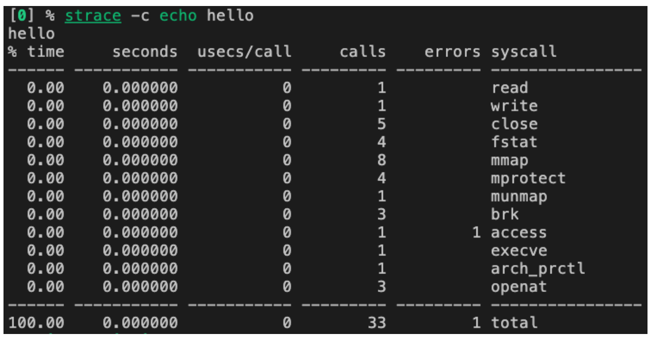
- **Extra credits: Combine options**

  Implement these 2 commands:

  – ”strace -s -e <system call name>”: print only successful system call name.
  – ”strace -f -e <system call name>”: print only failed system call name.

- **Extra credits: Implement -c options**

  Options -c in strace will generate a statistic report of system call regarding the input command such as duration, total call, failed call. Create a similar report table with using option -c.



```
[0] % strace -c echo hello
hello
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         1           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0         5           close
  0.00    0.000000           0         4           fstat
  0.00    0.000000           0         8           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         3           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000           0        33         1 total
```

Figure 4: strace with option -c sample in Linux

  Notes: Even though the sameple is in Linux, a similar report table is expected in XV6 for extra credits. The columns required are: calls, errors, syscall and seconds (measure the total time it takes to execute system call). You should try to test with a command that will take time to execute system call and display result of that command.

4. **Write output of strace to file**

   Find an implementation of choice to write strace output to file.
   For example: by providing option: -o <filename> or editing content of README.

5. **Application of strace**

   Write a small program that produce an unexpected behavior such as race condition, delay output, crash on condition, memory leak or your choice of implementation. Run strace on this program.

   - Explain your program and it's unexpected behavior.
   - (xv6) Does your strace implementation provide anything useful that can tell you something about this unexpected behavior?
   - (Linux) Implement same program on Linux in Anubis (there is slight various of C language between Linux and XV6), does Linux strace provide more information and is it better to help you debug this issue?

5 What to hand in

- An anubis submission (similar to all your homework).
- A zip file that package your entire XV6 repo along with all your strace code.
- A written report that includes all explanations and screenshots.
- A README instruct us TAs on how to run your program.
- A text that state your partner or working alone.
- Notes: Failure to submit one of these item will result in heavy point reduction.

6 Due date

- Extra credit early submission: May 1st 11:59pm
- Regular submission: May 15th 11:59 pm