CIS 452-10 Lab 2

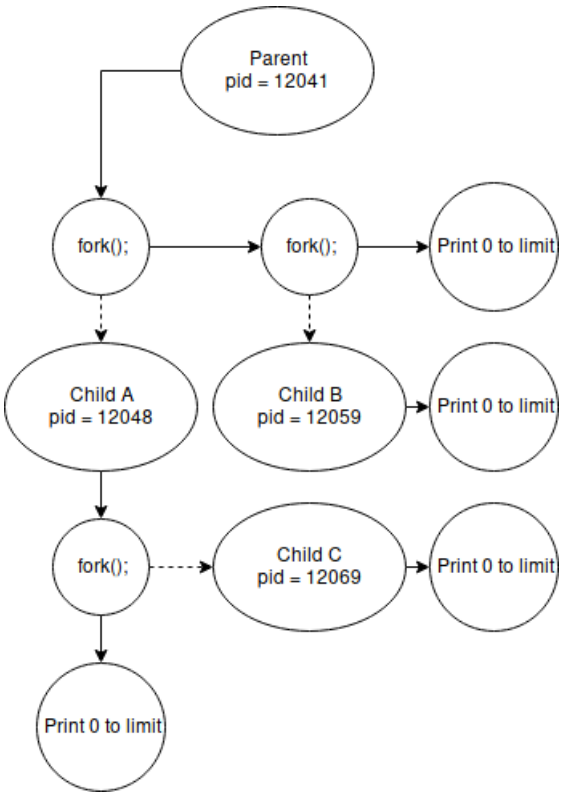
Thomas Bailey
Parker Skarzynski

January 2019

1. Three lines are printed by the program: 'Before fork' and 'After fork' twice.
2. The parent program begins execution. It prints out 'Before fork'. Then, the program forks. A child process is spawned, which is a duplicate of the parent. However, the child process starts on the command after the fork() happened. So, we see both the child and parent 'After fork' output.
3.
```
$ ps -l 17112
  F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
  0 S  8977 17112 13704  0  80   0 -   574 hrtime pts/0      0:00 ./a.out
```

   From this output and the ps man page, I came up with the following information:

   (a) My effective user ID is 8977.
   (b) The process ID is 17112.
   (c) The parent's process ID is 13704.
   (d) The process had 0 percent processor utilization.
   (e) The process has a priority of 80. Lower numbers indicate higher priority.
   (f) The 'nice' value of my process is 0. A nice program is kind to its program peers and doesn't monopolize the CPU.
   (g) The physical size of the core image for the process is 574 bytes.
   (h) The name of the kernel function in which the process is sleeping is called hrtime. The process is probably sleeping here because of the wait(10) call.
   (i) The cumulative CPU time is 0:00, indicating the process was not very intensive.
   (j) The command used to start the process was ./a.out.

4. 

5. I observed that the program prints the integers from 0 to limit four times in total. The output is not always the same, the loops sometimes being printed separately and sometimes concurrently. The concurrent loop happens because sometimes the parent processes are executing at the same time as their children.

6. `child = wait(&status);`

7. The child prints first because the parent doesn't print until wait() returns.

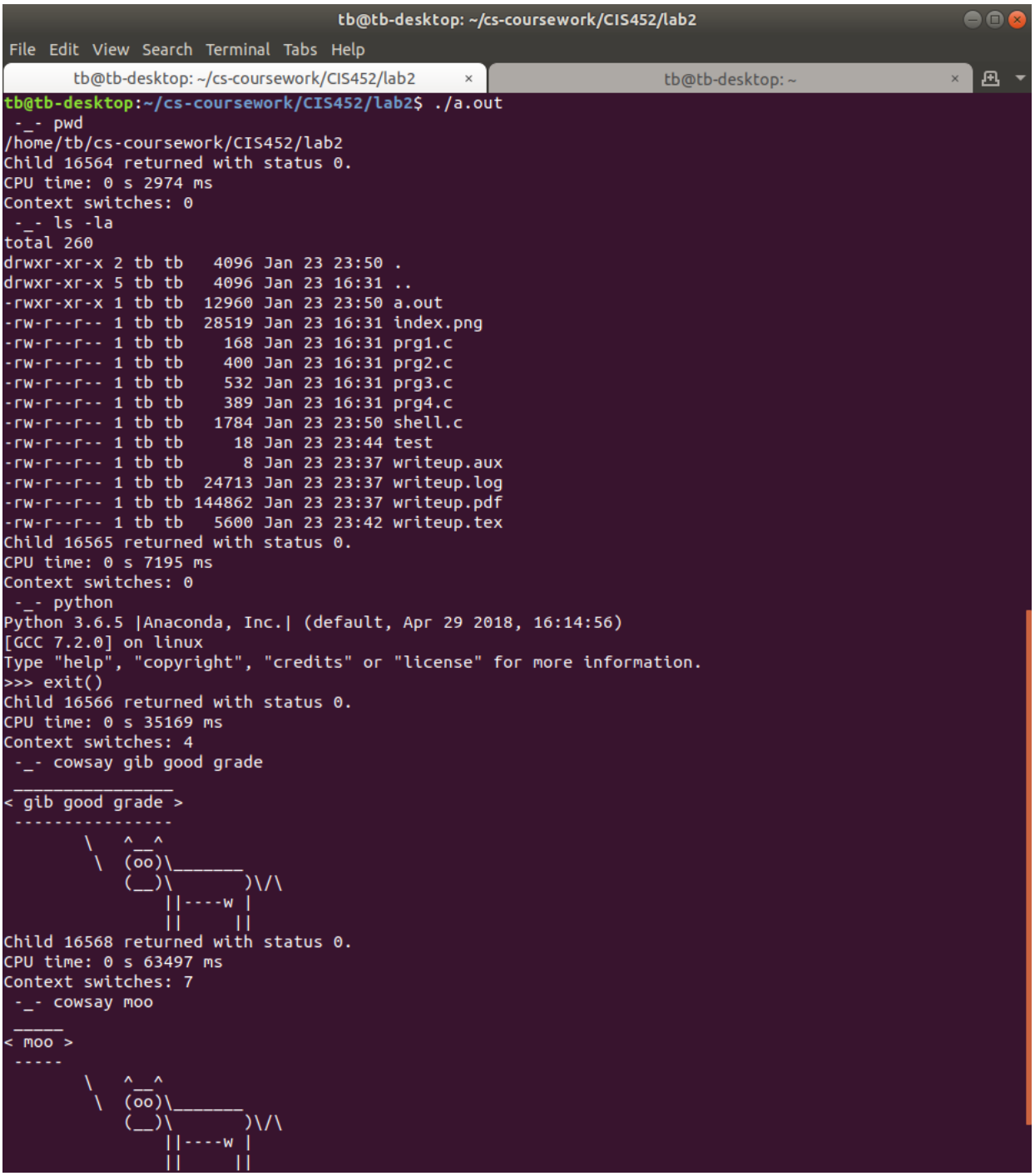
8. The two values that are printed out by the parent in sample program 3 are the process ID of the child, and the status code that was returned. When exit(n) is called with an integer n, n will correspond to a status code. Then, wait(&s) will store the status code in the variable s. Wait returns the process ID of the child, which gets stored in child.

9. The second print line ("After the exec") is **never** printed because it **DOESN'T EXIST**!!!1 The memory that contained that command was overwritten by the execvp() call.

10. The second argument passed to execvp() is the location for the null terminated string containing the arguments for the file to be executed.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <sys/resource.h>

#define MAX_SIZE 1024

int printPrompt() {
  printf(" -_- ");
  return 0;
}

int main(int argc, char **argv) {

  while (1) {
    int childPid;
    char cmdLine[MAX_SIZE];

    // Get the input
    printPrompt();
    fgets(cmdLine, MAX_SIZE, stdin);

    // Quit
    if (strcmp(cmdLine, "quit\n") == 0) {
      exit(0);
    }

    // Parser? I hardly know 'er
    char* token = strtok(cmdLine, " ");
    char* args[64];
    int i = 0;
    while (token != NULL) {
      i += 1;
      args[i] = (char*)malloc(sizeof(char)*MAX_SIZE);
      args[i] = token;
      token = strtok(NULL, " ");
    }

    // Strip a tricky newline
    args[i][strcspn(args[i], "\n")] = 0;

    // Null terminate
    args[i+1] = NULL;

    // Begin forking
    pid_t pid, child;
    int status;
    if ((pid = fork()) < 0) {
      perror("You forked up.");
      exit(1);
    } else if (pid == 0) {
```

```
56        printf("I am a child.");
57        printf("%s", args[0]);
58        if (execvp(args[1], &args[1]) < 0) {
59        perror("Child failed.");
60    exit(1);
61        } else {
62    exit(0);
63        }
64      } else {
65
66        // Get the status
67        child = wait(&status);
68        long seconds;
69        long milsecs;
70        int context_switch;
71        struct rusage resources;
72        if (getrusage(RUSAGE_CHILDREN, &resources) < 0) {
73          printf("Failed.");
74        } else {
75          seconds = resources.ru_utime.tv_sec;
76          milsecs = resources.ru_utime.tv_usec;
77          context_switch = resources.ru_nivcsw;
78          printf("Child %d returned with status %d.\nCPU time: %ld s %ld ms\nContext switches: %d\n", child,
      status, seconds, milsecs, context_switch);
79        }
80      }
81    }
82    return 0;
83  }
```


tb@tb-desktop: ~/cs-coursework/CIS452/lab2
File Edit View Search Terminal Tabs Help
tb@tb-desktop: ~/cs-coursework/CIS452/lab2
tb@tb-desktop: ~
tb@tb-desktop:~/cs-coursework/CIS452/lab2$ ./a.out
-_- pwd
/home/tb/cs-coursework/CIS452/lab2
Child 16564 returned with status 0.
CPU time: 0 s 2974 ms
Context switches: 0
-_- ls -la
total 260
drwxr-xr-x 2 tb tb 4096 Jan 23 23:50 .
drwxr-xr-x 5 tb tb 4096 Jan 23 16:31 ..
-rwxr-xr-x 1 tb tb 12960 Jan 23 23:50 a.out
-rw-r--r-- 1 tb tb 28519 Jan 23 16:31 index.png
-rw-r--r-- 1 tb tb 168 Jan 23 16:31 prg1.c
-rw-r--r-- 1 tb tb 400 Jan 23 16:31 prg2.c
-rw-r--r-- 1 tb tb 532 Jan 23 16:31 prg3.c
-rw-r--r-- 1 tb tb 389 Jan 23 16:31 prg4.c
-rw-r--r-- 1 tb tb 1784 Jan 23 23:50 shell.c
-rw-r--r-- 1 tb tb 18 Jan 23 23:44 test
-rw-r--r-- 1 tb tb 8 Jan 23 23:37 writeup.aux
-rw-r--r-- 1 tb tb 24713 Jan 23 23:37 writeup.log
-rw-r--r-- 1 tb tb 144862 Jan 23 23:37 writeup.pdf
-rw-r--r-- 1 tb tb 5600 Jan 23 23:42 writeup.tex
Child 16565 returned with status 0.
CPU time: 0 s 7195 ms
Context switches: 0
-_- python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
Child 16566 returned with status 0.
CPU time: 0 s 35169 ms
Context switches: 4
-_- cowsay gib good grade
< gib good grade >
\ ^__^
\ (oo)_______
(__)\ )\/\
||----w |
|| ||
Child 16568 returned with status 0.
CPU time: 0 s 63497 ms
Context switches: 7
-_- cowsay moo
< moo >
\ ^__^
\ (oo)_______
(__)\ )\/\
||----w |
|| ||