CIS 452
Lab 2 Report

Ashley Hendrickson
Muna Gigowski
Fall 2019

Process Creation

Question One

How many lines are printed by the program?

3 lines:

Before fork
After fork
After fork

Question Two

Describe what is happening to produce the answer observed for the above question.

When the program executes, it prints the first line, “Before fork,” before fork() is called. Then, once fork() is called the parent program is still running, so it prints the next line after the call to fork(), which is the line that says “After fork.” At the same time that the parent is printing its “After fork” line though, the duplicate child process that was created is also running and prints its “After fork” line as well.

Question Three

Consult the man pages for the ps (process status) utility; they will help you determine how to display and interpret the various types of information that is reported. Look especially for "verbose mode" or "long format". Then, using the *appropriate* options, observe and report the PIDs and the status (i.e. state info) of your executing program. Provide a brief explanation of your observations.

Using command “ps -l” :

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  8772 18490 13463  0  80   0       -  1431 -          pts/2    00:00:00 bash
0 S  8772 32028 18490  0  80   0       -   572 -          pts/2    00:00:00 a.out
1 S  8772 32029 32028  0  80   0       -   572 -          pts/2    00:00:00 a.out
4 R  8772 32054 18490  0  80   0       -  2411 -          pts/2    00:00:00 ps
```

The results above can be explained fairly simply, one process at a time:

Bash - This process was the first running, since it is the command interpreter we are using to execute all of our other commands. It’s process ID is shown, and it’s state is shown as an “S”

because bash is waiting for the other programs we are running to finish execution, so it can resume.

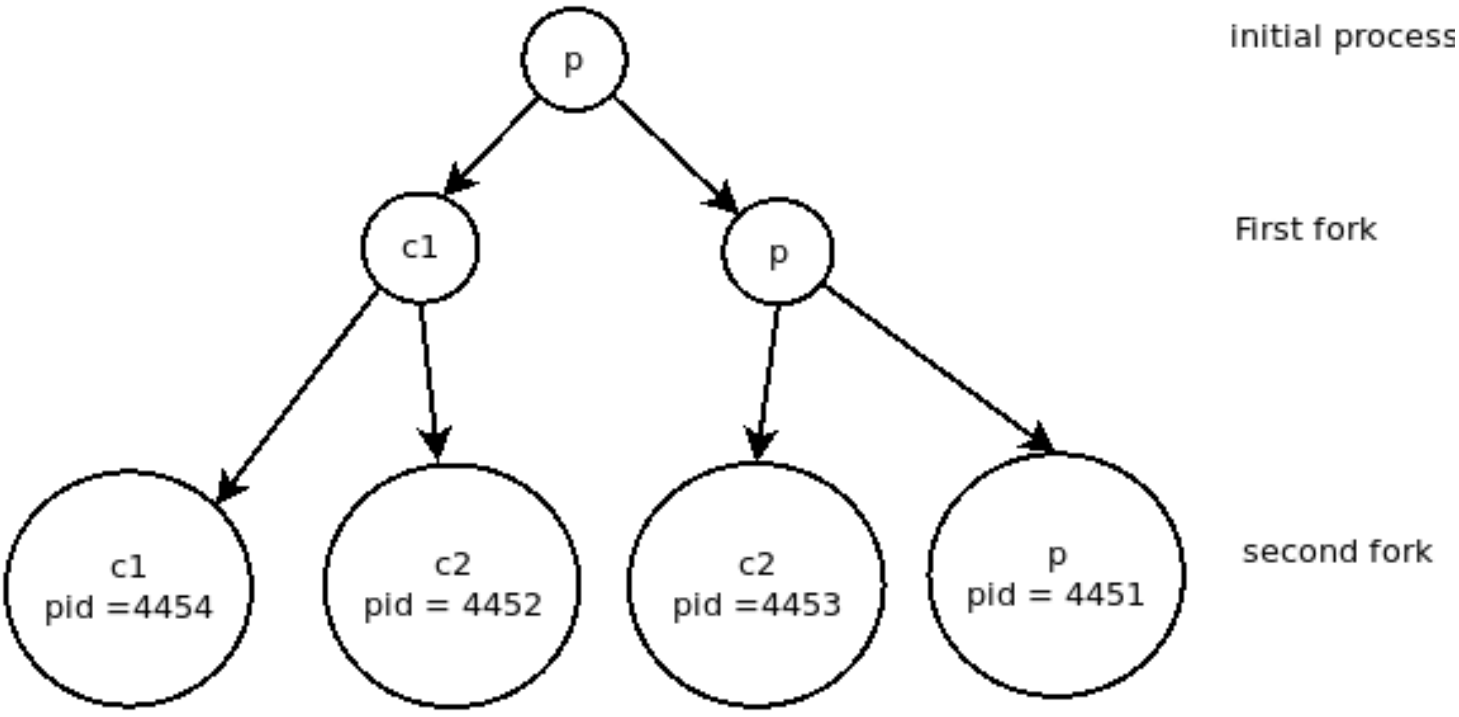
A.out (top one) - This is our parent process, which we can tell by looking at its process ID (32028), which is one lower than the other a.out (meaning that process is the child process because it was created after this one). Upon us executing the ps -l command, this process has a status of "S," which makes sense because we told it to sleep for ten seconds inside the program's code after fork() call.

A.out (bottom one) - This is our child process, which we determined above based on looking at its process ID. The state of this process is also "S" because we added the sleep(10) line after the fork() call in the code, meaning that after the child was created with fork() it picked up execution right at the sleep(10) line - also causing it to sleep for 10 seconds along with its parent process.

Ps - This is our ps -l command, and its status is "R" because it was the only process currently running at the time the results were printed out (and understandably so, because it was the process that was doing the printing out of the results).

Question Four

Create a diagram illustrating how Sample Program 2 executes. I.e. give a process hierarchy diagram (as described in class and in your textbook).



Question Five

In the context of our classroom discussions on process state, process operations, and especially process scheduling, describe what you observed and try to explain what is happening to produce the observed results.

The initial process creates a new process after the first fork making two processes each of those process then hit the second fork resulting in four processes running. One parent and three children. The CPU is then scheduled to make a system call to the print to console function and schedules between all the processes

Process Suspension and Termination

SAMPLE PROGRAM 3 REVISED SOURCE CODE:

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

int main()
{

    // use these variables

    pid_t pid, child;
    int status;

    if ((pid = fork()) < 0) {
        perror("fork failure");
        exit(1);
    }
    else if (pid == 0) {
        printf("I am child PID %ld\n", (long) getpid());
        exit(0);

    }
    else {
        wait((wait(&status)) > 0);

        printf("Child PID %ld terminated with return status %d\n", (long) child, status);
    }
    return 0;
}
~
~
~
                                                                    29,13         All
```

SAMPLE OUTPUT:

```
[gigowskm@eos10 lab2]$ ./a.out
I am child PID 22965
Child PID 32766 terminated with return status 0
[gigowskm@eos10 lab2]$
```

Question Six

Provide the exact line of code that you inserted for the wait() system call.

```
wait((wait(&status)) > 0);
```

Question Seven

Who prints first, the child or the parent? Why?

The child, because the next line in the code after the if() conditional that executes fork() is the elseif that addresses the condition if the pid = 0 (pid previously got assigned the value that was returned from fork()), which would be true within the child process after fork(), but not the parent process (fork() returns twice - the value of 0 to the child process, and the child's pid to the parent process). Because the condition pid = 0 is not true for the parent process, it passes over this conditional and executes the next one down, but not until after the child process already ran their version of the code.

Question Eight

What two values are printed out by the parent in Sample Program 3? (No, not the actual numbers, but what they mean.) In other words, describe the interaction between the exit() function and the wait() system call.

The values printed by the parent in sample program 3 occur in the print line that runs after the child process has terminated. The first number is the process ID of the child process that finished, which was returned by wait() after the child process exited. The second value is the return status of the child that generated upon it finishing, which was 0 - this means that the child process exited successfully.

Process Execution

Question Nine

When is the second print line ("After the exec") printed? Explain your answer.

It never prints. Either execvp() fails and the if conditional gets entered, terminating the program before that line can execute, or execvp() is successful, at which point all of the code in the file gets overridden with the new code that execvp() puts into it – meaning the line that prints "After the exec" will get overridden before it even gets hit.

Question Ten

Explain how the second argument passed to execvp() is used?

The second argument, &argv[1], is a pointer to the starting address of the char array that contains all of the arguments to the program, starting with argv[1]. execvp() passes the array at this starting location to the child process, at which point it then becomes the child process's argv.

Programming Assignment (Simple Shell)

Source Code

SHELLPROGRAM.C

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/times.h>
#include <sys/time.h>
#include <sys/resource.h>
#include "../clib/clib.h"

int main(void) {

        char input [50];
        pid_t pid;
        int status;
        struct rusage buf;
```

```
	char* string[64];
	long lastCpuTotal = 0;
	long lastSwitchTotal = 0;

	do {
		memset(input, 0, 50*sizeof(input[0]));
		printf("Please enter a command with parameters: \n");
		printf(">");
		fgets(input, 50, stdin);
		printf("\n\n");
		strcpy(input,strsegment(input,'\n'));

		execParse(input, string);

		pid = fork();

		if (pid < 0) {
			printf("ERROR: Fork operation failed");
			exit(1);
		} else if (pid) { //parent's code

			wait(&status);

			getrusage(RUSAGE_CHILDREN, &buf); //Monitors total usage for all
children


			printf("\n\n\n\n\n\n");
			printf("user cpu time used %ld \n", (buf.ru_utime.tv_usec -
lastCpuTotal));

			printf("involuntary context switches: %ld\n", buf.ru_nivcsw -
lastSwitchTotal);
			printf("\n\n\n\n\n\n");

			lastCpuTotal = buf.ru_utime.tv_usec;
			lastSwitchTotal = buf.ru_nivcsw;

		} else { //everything in here is the child's code
			printf("Command entered was" );
			if (execvp(*string,string) < 0) {
				perror("exec failed");
				exit(1);
			}
```

```
		}

	}while (strcmp(input,"quit"));

	return 0;
}
```

STRINGLIB

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

//////////////////////////////////////////////////////
//String library
//////////////////////////////////////////////////////

int * getDelimLength(char *str,char del){
//row |^,-> column
	int row=1;
	int col=1;
	int temp=1;
	int length= strlen(str);
	for(int i=0; i<length;i++){
		if(str[i]==del){
			row++;
			col=0;
		}
		if(col>temp){
		temp++;
		}
		col++;
	}
	col=temp;
	int *vector= malloc(2*sizeof(int));
	vector[0]=row;
	vector[1]=col;

	return vector;
}

// parses a string
```

```
char ** splitString(char* str, char delim){
	int * vec=getDelimLength(str, delim);
	char **parString = (char **)malloc(vec[0]*sizeof(char*));
	for(int i =0; i<vec[0]; i++){
		parString[i]=(char *)malloc(vec[1]*sizeof(char));
	}
	int j=0;
	int k=0;
	int length=strlen(str);
	for(int i =0; i <= length; i++)
	{
		if(str[i]==delim){
		j++;
		k=0;
		}else{
		parString[j][k]=str[i];
		k++;
		}
	}
	return parString;
	}


char* strsegment(char *str, char garbage) {
char* s=(char*)malloc(sizeof(char)*2048);
memset(s,0,sizeof(char)*2048);
int i=0;
	while(str[i] != garbage&&str[i] !='\000'){
	s[i]=str[i];
		i++;
	}
	free(s);
	return s;
}


void  execParse(char *line, char **argv)
{
    while (*line != '\0') {       /* if not the end of line ..... */
        while (*line == ' ' || *line == '\t' || *line == '\n')
            *line++ = '\0';     /* replace white spaces with 0    */
        *argv++ = line;          /* save the argument position     */
        while (*line != '\0' && *line != ' ' &&
               *line != '\t' && *line != '\n')
```

```
        line++;          /* skip the argument until ...    */
    }
    *argv = '\0';              /* mark the end of argument list  */
}
```

Sample Output

```
[gigowskm@eos04 lab2]$ make
gcc -std=c11 -g -Wall -Wextra -pedantic -Wwrite-strings  shellProgram.c -o shell
Program
[gigowskm@eos04 lab2]$ ./shellProgram
Please enter a command with parameters:
>ls -a


.          sampleProgram     sampleProgram3.c  shellProgram
..         sampleProgram1.c  sampleProgram4    shellProgram.c
makefile   sampleProgram2.c  sampleProgram4.c




user cpu time used 1014
involuntary context switches: 0
```

```
Please enter a command with parameters:
>vim




user cpu time used 14416
involuntary context switches: 1
```