LAB 3

CS 361: Systems Programming / Spring 2023

Description

In this lab session, you will explore:

- Function `fork`, which allows you to create a new process
- Function `execv`, which allows to you to run a different executable in the current process
- Functions `wait` and `waitpid`, which allow you to wait for a process to terminate
- How to concatenate strings in C with function `snprintf`

Please read this document carefully and follow the instructions on the last section to complete this lab session. When you answered all the questions, please show your work to the TA.

`fork`

Function fork has the following signature:

```
pid_t fork();
```

This function is called by a process (i.e., the parent process), takes no arguments, creates a new process (i.e., the child process), and **returns twice**:

- It returns the integer zero to the child process;
- It returns the an integer with the Process ID (PID) of the child to the parent process.

You can tell which process is which by checking the return of the function. If you save that return in a variable, you can later print the PID of the child process from inside the parent process.

wait / waitpid

Functions `wait` and `waitpid` allow a process to wait for the termination of another process. They have the signature:

```
pid_t wait(int * status);
pid_t waitpid(pid_t pid, int * status, int options);
```

Both functions return the PID of the process that terminated. Both functions take an argument status, which is a pointer to an int. After calling these functions, the int holds the return code of the process that terminated. Example:

```
int pid = … ; // Some valid PID
int return_code;

waitpid(pid, &return_code, 0);
  // Blocks until the process with the given PID terminates
printf("%d\n", return_code);
  // Prints the return code of the terminated process
```

execv

Function `execv` allows to run a different executable in the current process. It has the signature:

```
int execv(const char *pathname, char *const argv[]);
```

It takes the following arguments:

- pathname: the full path to the executable to run (e.g., `/bin/echo` o)
- args: an array of strings with the following contents:
    - Index 0: The full path to the executable to run (e.g., `/bin/echo`)
    - Index 1: The 1st argument to the executable (e.g., `Hello`)
    - Index 2: The 2nd argument to the executable (e.g., `World`)
    - …
    - Index N: The Nth argument to the executable
- The position on this array that indicates the end of the arguments should be NULL

For instance, to execute the command `echo Hello World`, you must first find the full path of the executable (i.e., `/bin/echo`) and call `execv` as follows:

```
char * p = "/bin/echo";
char * a[10];
a[0] = "/bin/echo";
a[1] = "Hello";
a[2] = "World";
a[3] = NULL; // This indicates the end of the arguments
execv(p, a);
```

When `execv` executes successfully, it **returns zero times (i.e., it does not return)**. If `execv` returns, it means that there was a problem (e.g., it could not find the executable to run). The example above executes command `/bin/echo Hello World` in the current process, which simply prints `Hello World` to the terminal and exits with exit code zero.

snprintf

Function `printf` uses a format string to print to the terminal. Function `snprintf` is very similar, but it prints to allocated memory or to a character buffer. Its signature is as follows:

```
int snprintf(char * str, size_t size, const char *restrict fmt, ...);
```

This function prints up to size characters to the memory pointed to by str, and returns how many characters it used. It can be used to concatenate two strings, with a space in between, as follows:

```
char buffer[20];
char * string1 = "Hello";
char * string2 = "World";
snprintf(buffer, 20, "%s %s", string1, string2);
printf("%s\n", buffer); // Prints Hello World
```

exit

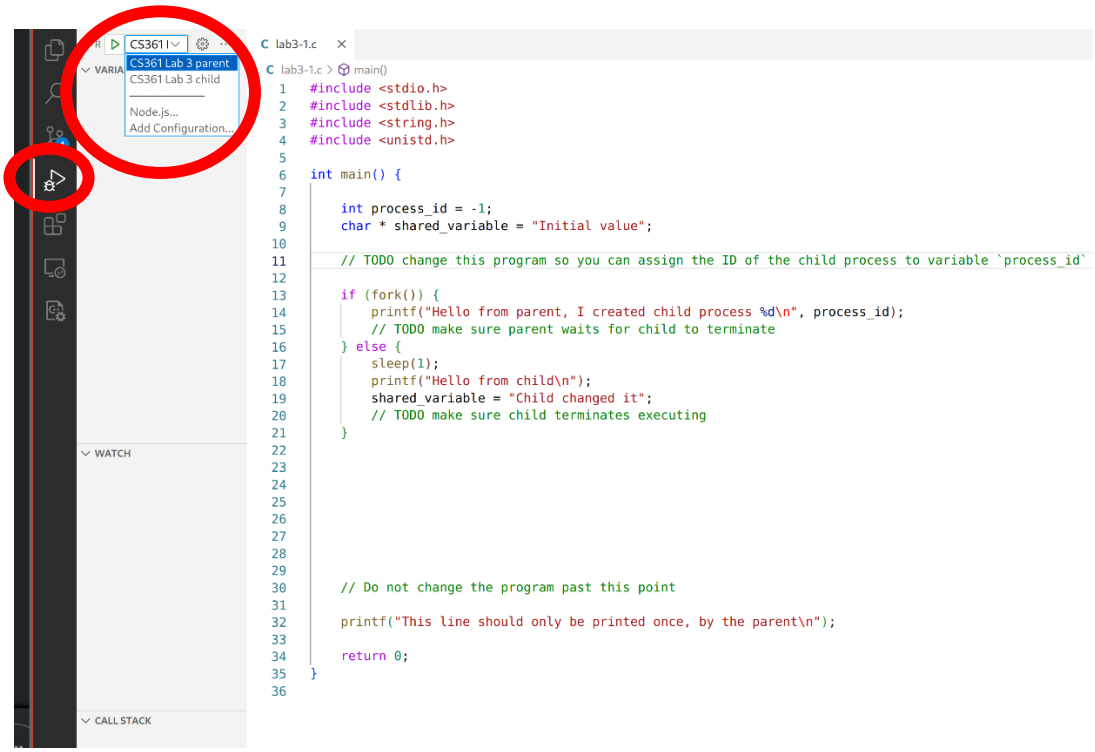
Function exit terminates the execution of a process. Its signature is as follows:

```
void exit(int status);
```

It takes the return code that the processes should exit with, which is a small integer, in which zero means "success" and non-zero means "failure". This function terminates the process immediately, and never returns back to the caller.

Guide

1. Accept the invitation for Lab 3 on Github classroom: https://classroom.github.com/a/PvnonVh_
2. Import the Github repository created to your machine using vscode, as explained in Assignment 0
3. Make sure that you can launch a terminal inside vscode via menus: Terminal > New Terminal
4. The repository has three files that you need: `lab3-1.c, lab3-2.c, lab3-3.c`.
5. Type `make` on the terminal and then run file lab3-1 by typing `./lab3-1` on the terminal. Observe the output. Repeat for files lab3-2 and lab3-3.
6. Run file lab3-1.c under the debugger, using both the parent and child configurations depicted below.




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

int main() {

    int process_id = -1;
    char * shared_variable = "Initial value";

    // TODO change this program so you can assign the ID of the child process to variable `process_id`

    if (fork()) {
        printf("Hello from parent, I created child process %d\n", process_id);
        // TODO make sure parent waits for child to terminate
    } else {
        sleep(1);
        printf("Hello from child\n");
        shared_variable = "Child changed it";
        // TODO make sure child terminates executing
    }

    // Do not change the program past this point

    printf("This line should only be printed once, by the parent\n");

    return 0;
}
```

7. Modify file `lab3-1.c` to answer Question 1.
8. Modify file `lab3-2.c` to answer Question 2.
9. Modify file `lab3-3.c` to answer Question 3.
10. Show your work to the TA, commit and push your changes to the repo created in Step 2.

Questions

You can write the answers to your questions to file `ans.txt` in your repository, it then makes it easy to show your work to the TA.

1. How did you change lab3-1.c so that it prints the following output. You cannot change anything below line 30? Note that the order of each line matters (i.e., the child must print first).

```
Hello from child
Hello from parent, I created child process <POSITIVE INTEGER>
This line should be printed only once, by the parent
```

2. How did you change lab3-2.c so that it prints the following output?

```
I'm about to execute program /bin/date
date (GNU coreutils) 8.30
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<https://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by David MacKenzie.
Program /bin/date returned with exit code <POSITIVE INTEGER OR ZERO>
```

3. How did you change lab3-3.c so that it prints the following output?

```
Trying path /local/bin/uname
Didn't work, trying something else
Trying path /usr/lib/uname
Didn't work, trying something else
Trying path /usr/uname
Didn't work, trying something else
Trying path /usr/bin/core/uname
Didn't work, trying something else
Trying path /usr/ban/uname
Didn't work, trying something else
Trying path /usr/bin/uname
Didn't work, trying something else
Trying path /bin/uname
Linux cs-sys3 5.4.0-126-generic #142-Ubuntu SMP Fri Aug 26 12:12:57 UTC
2022 x86_64 x86_64 x86_64 GNU/Linux
```

Extra / Optional

1. Execute file lab3-1.c inside the debugger. Notice that `shared_variable` starts with contents `Initial value`. The program changes it to `Child changed it`. However, just after printing `This line should only be printed once, by the parent` the value of that variable is `Initial value`. Why?

Grading

Show your UIC card to the TA when you enter the lab, or type your UIN on the chat when joining remotely. Stay in the session until you show your work, or until the TA announces that the lab is over.

- You have to remain present for the whole lab to get attendance, which you can then use to resubmit Assignment 2.
- You can leave early after showing your work to the TA (answers to all questions). In this case, you will get a 5% bonus in Assignment 2.