

CS333: Operating Systems Lab

Autumn 2022

Lab Quiz 1, 20 marks

No internet. No phone. No friends (strictly for the duration of the exam only!)

You are provided a cheat sheet along with the auxiliary files.

There are two questions each having two parts that will be graded separately. All the best!

Q1. Secret Sharing (10 marks)

Alice and Bob both have secret codes that when appended give a passcode to crack some puzzle. Alice and Bob can't communicate directly and must firstly communicate the respective secret codes to John, who then reveals the passcode.

You have to write a program that forks two child processes. The parent process corresponds to John and the child processes to Alice and Bob.

- (a) (5 marks) As the first step, Alice and Bob have to read their secret codes from files, Alice reads from the file `secret1.txt` and Bob from the file `secret2.txt`. Both files are provided in the auxiliary files directory.

Next, the two processes print the keys to standard output in a particular order.

You must use system calls (`open`, `read`, `write`, `close`) to read the secret codes.

Sample output for provided `secret.txt`, `secret2.txt`:

```
<pid1>: Secret code for Alice is <secret1>
```

```
<pid2>: Secret code for Bob is <secret2>
```

```
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ gcc q1a.c
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ ./a.out
744: Secret code for Alice is Hello
745: Secret code for Bob is World
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$
```

Note:

1. The secret codes are 5 characters long.
2. `read()` system call doesn't NULL terminate the read string, so make sure to add `'\0'` at the end of the string; else you may get garbage characters.
3. The order of print statements should be maintained, i.e., firstly **secret1** should be printed and then **secret2** should be printed next. Check the output format above.
Note that your program should work even if Alice decides to take a long pause before `secret1` is printed.
4. The parent process must reap both the child processes.

- (b) (5 marks) The child processes must now communicate the secret codes to the parent process. Can you think of how to do so? You must achieve this **without** reading or writing files, i.e., the child processes cannot write to any files which the parent process later reads, or the parent process cannot read the files directly. The parent process should print the passcode (obtained by appending secret2 to secret1, without any spaces) on the terminal.

Sample output:

```
<pid1>: Secret code for Alice is <secret1>
<pid2>: Secret code for Bob is <secret2>
<parent pid>: Passcode is <secret1><secret2>
```

```
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ gcc q1b.c
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ ./a.out
736: Secret code for Alice is Hello
737: Secret code for Bob is World
735: Passcode is HelloWorld
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$
```

The pids in the output may change for you.

Your submission will be tested on different secret1.txt and secret2.txt files, i.e., with different secret codes.

Q2. Control your processes! (10 marks)

- (a) (5 marks) Write a program that takes the name of an executable file (in the same directory) as a command line argument and runs it as a child process. The provided executable **should not run** for more than 5 seconds. Sleep time is considered part of run time.

The child process should print its **pid** before starting the execution of the provided executable file -
"pid <child_pid>: I am child"

If the executable does not exit within 5 seconds, it should be terminated by the parent and the following message should be printed - "Terminating process <child_pid>".

Two auxiliary files onesecc.c and tensescc.c are provided to test your implementation.

With onesecc.c the main code is :

```
int main(){
    printf("I will sleep for one sec\n");
    sleep(1);
    printf("Now I am awake\n");
}
```

Example output:

```
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ gcc onesecc.c -o onesecc
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ gcc q2a.c
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ ./a.out onesecc
pid 688: I am child
I will sleep for one sec
Now I am awake
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$
```

Here the executable completes within 5 seconds and exits normally.

With `tensec.c` the main code is :

```
int main(){  
    printf("I will sleep for ten sec\n");  
    sleep(10);  
    printf("Now I am awake\n");  
}
```

The executable `tensec.c` will not complete within 5 seconds. So the parent terminates the child process executing the `tensec` program.

```
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ gcc tensec.c -o tensec  
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ gcc q2a.c  
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$ ./a.out tensec  
pid 701: I am child  
I will sleep for ten sec  
Terminating process 701  
purvi2001@purvi:/mnt/c/Users/purvi/OneDrive/Documents/OS_2022$
```

Hint: `man 2 kill`, `man 3 sleep`, `man waitpid`

- (b) (5 marks) You are given a program `shell.c`. It prints a prompt ("`>>>` "), takes the name of an executable (present in the current folder) as an input, executes it, and again prints a prompt and reads the name of another executable to execute (indefinitely).

Consider the `helloloop.c` present in the same folder.

(
The source code of the auxiliary file `helloloop.c` is available and can be compiled using :
`gcc helloloop.c -o helloloop`
)

```
λ > gcc shell.c  
λ > ./a.out  
>>> helloloop  
hello  
hello  
hello  
hello  
^C  
λ >
```

In the above program when `Ctrl+C` (`SIGINT`) is used in the terminal it closes the main shell program as well. Try it out.

Next, you need to modify the program such that,

1. When `Ctrl+C` is pressed during execution of the executable given at the custom prompt, only the custom process exits and not the main shell program. The main program keeps on waiting for the next input for execution.
2. Also, on using `Ctrl + \` (`SIGQUIT`) the main program should exit and print (shell is exiting).

Example output:

```
λ > gcc q2b.c -o q2b
λ > ./q2b
>>> helloloop
hello
hello
hello
^C
>>> helloloop
hello
hello
^C
>>> ^\
shell is exiting
λ > 
```

Auxiliary File: shell.c

Hints:

1. After an `exec` system call, a process does not have access to previous code, so it cannot inherit the signal handlers. For this reason after an `exec` call, all signals are reset to default handlers (SIG_DFL), but signals ignored using SIG_IGN remain ignored.

Submission instructions:

- All submissions via moodle. Name your submissions as: `<rollnumber>_labquiz1.tar.gz` (e.g. `21q050005_labquiz1.tar.gz`)
- Auxiliary files are available in the `auxiliary_files` folder.
- The tar should contain the following files in the specified directory structure:

```
<rollnumber>_labquiz1/
| ____q1a.c
| ____q1b.c
| ____q2a.c
| ____q2b.c
```

please adhere to this format strictly

- Command to tar your submission directory ...
`tar -zcvf <rollnumber>_labquiz1.tar.gz <rollnumber>_labquiz1`
- **Due date: 22nd August 2022, 5.15 pm**