

Please read this assignment carefully and follow the instructions EXACTLY.

Submission:

Please refer to the lab retrieval and submission instruction. The requirements regarding subdirectory for each part, README.txt and Makefiles remain the same as the previous labs.

Checking memory errors with valgrind

Do not include valgrind output in README.txt. You should keep using valgrind to check your program for memory error (and the TAs will do the same when grading), but you don't have to include the output in README.txt anymore.

Guarding against "fork-bomb":

In this lab, you experiment with forking new processes. It is easy to make a mistake while you're coding and create a "fork-bomb". A fork-bomb refers to a program which continues to fork child processes in an infinite loop. A fork-bomb will eventually drain the system resources and make the machine unresponsive, not only for you, but for all users on that machine.

You can voluntarily limit the number of processes you can fork using the "ulimit" command in bash shell. Please add the following line at the end of the .bashrc file in your CLIC home directory:

```
ulimit -u 256
```

And don't forget to logout and login again, so that the new .bashrc will take effect.

This will prevent you from creating more than 256 processes during your login session, effectively preventing you from accidentally creating a fork-bomb.

It is very important that you do this. If you don't do this, and you accidentally unleash a fork-bomb on a CLIC machine, the CRF will suspend your account for days. You won't be able to submit your lab, and you will get ZERO. No excuse.

Part 1: Remote access to mdb-lookup-cs3157 using netcat (100 points)

---

(a)

You learned in class how to use a named pipe (aka FIFO) and the netcat (nc) program to turn mdb-lookup-cs3157 into a network server.

Write a shell script that executes the pipeline you learned.

- The name of the script is "mdb-lookup-server-nc.sh"

- A shell script starts with the following line (the '#' is the 1st character without any leading space):

```
#!/bin/sh
```

- You must make the file executable using "chmod" command.
- The script takes one parameter, port number, on which nc will listen.
- The script should create a named pipe named mypipe-<pid>, where <pid> indicates the process ID of the shell running the script. The named pipe should be removed at the end of the script.
- See section 3.4 in the Bash Reference Manual (<http://www.gnu.org/software/bash/manual/bashref.html>) for how to refer to the arguments and the process ID from your script.

(b)

Here is a program that runs the shell script from (a) as a child process (mdb-lookup-server-nc-1.c):

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

static void die(const char *s)
{
    perror(s);
    exit(1);
}

int main(int argc, char **argv)
{
    if (argc != 2) {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(1);
    }

    pid_t pid = fork();
    if (pid < 0) {
        die("fork failed");
    } else if (pid == 0) {
        // child process
        fprintf(stderr, "[pid=%d] ", (int)getpid());
        fprintf(stderr, "mdb-lookup-server started on port %s\n", argv[1]);
        execl("./mdb-lookup-server-nc.sh", "mdb-lookup-server-nc.sh",
              argv[1], (char *)0);
        die("execl failed");
    } else {
        // parent process
        if (waitpid(pid,
                    NULL, // no status
```

```

        0) // no options
        != pid)
        die("waitpid failed");
        fprintf(stderr, "[pid=%d] ", (int)pid);
        fprintf(stderr, "mdb-lookup-server terminated\n");
    }

    return 0;
}

```

Study this program and the man pages for the system calls used in it. Run it to make sure it works with your shell script from (a). Make sure you understand how everything works.

While this program is running, run the following command from another terminal window logged into the same machine:

```
ps ajxfww
```

Find the process tree that contains the ancestors and the children of this program, and include it in your README.txt.

Make sure you understand the process relationships in that tree. Identify the files that are shell scripts and list them in your README.txt.

(c)

Write an improved version: mdb-lookup-server-nc-2.c

mdb-lookup-server-nc-1.c from (b) forked once, exec'ed the script, waited until it's done, and then exited.

mdb-lookup-server-nc-2.c will do the following:

- It has a loop in which it displays a prompt, "port number: ". It reads the port number typed in by the user and then fork/exec the script from (a) using that port number. It also prints out a message stating that an instance of mdb-lookup-server has started. The message should include the child's process ID and the port number on which it's listening.
- Hitting ENTER on the prompt should simply display another prompt.
- On every iteration, before it displays a prompt, it should check if any of the child processes have terminated. It should display the process IDs of all terminated mdb-lookup-server processes along with messages saying that they have terminated. For this, you need to use the non-blocking version of waitpid() system call. Here is how you use it:

```
pid = waitpid( (pid_t) -1, NULL, WNOHANG);
```

- Use Ctrl-C to quit.

## Part 2: Orphan and Zombie (0 points)

---

Part 2 of this lab is optional and will not be graded. You don't have to do it if you don't have time. If you have time, do it for learning. Do it for fun!

(a)

When a child process gets orphaned, it gets adopted by the 'init' process (whose pid is 1). Write a program that creates this condition. Capture this condition using ps command and include the output in your README.txt. Explain the output.

(b)

A child process that has terminated, but has not been reaped by its parent (using waitpid()), is called a zombie. Write a program that creates this condition. Capture this condition using ps command and include the output in your README.txt. Explain the output.

--

Good luck!