

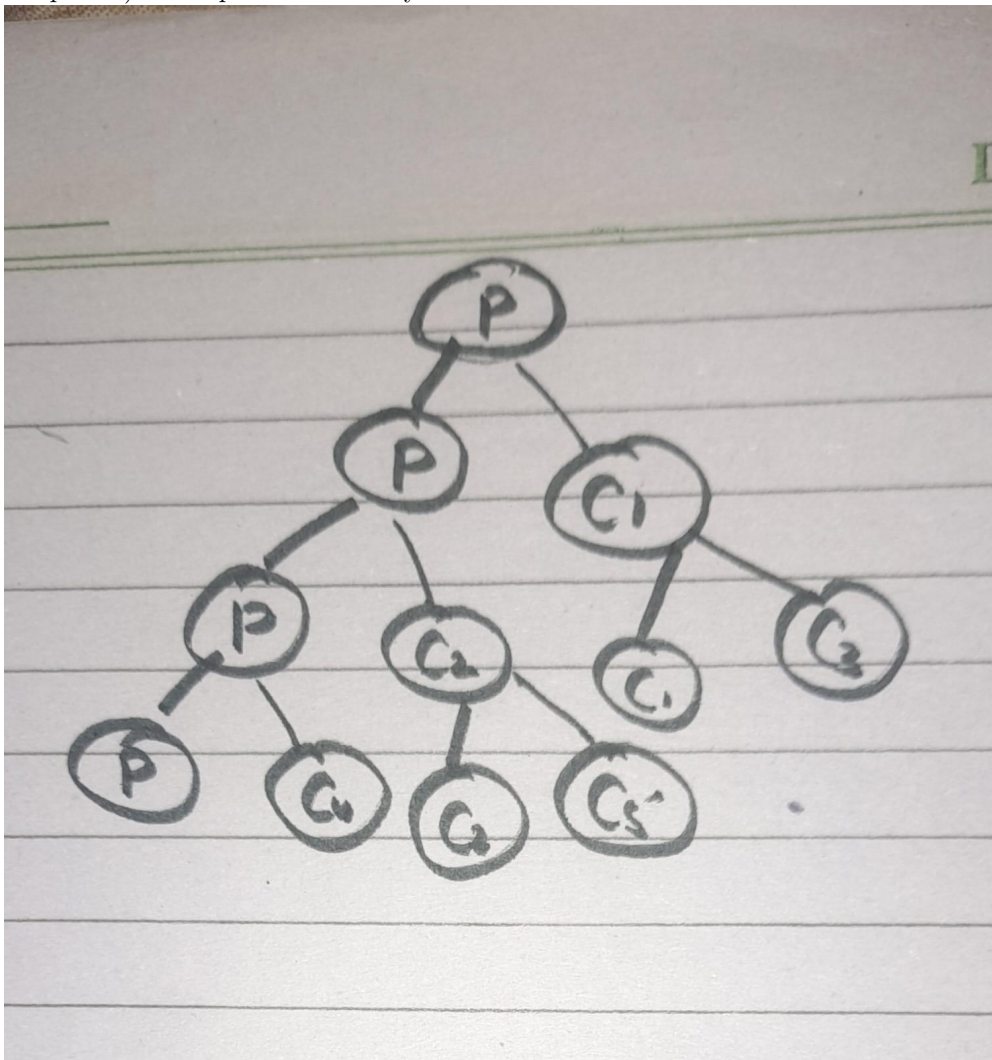
OS-Lab-07

Tazmeen Afroz
Roll No: 22P-9252
BAI-5A

October 4, 2024

1 Exercise 2

Model a `fork()` call in C/C++ so that your program can create a total of EXACTLY 6 processes (including the parent). Your process hierarchy should be as follows:



```
(base) tazneen@afroz:~/os_lab_07_codes$ gcc exercise1.c
(base) tazneen@afroz:~/os_lab_07_codes$ ./a.out
Process: PID = 16183, PPID = 14589
Process: PID = 16184, PPID = 16183
Process: PID = 16185, PPID = 16183
Process: PID = 16187, PPID = 16185
Process: PID = 16186, PPID = 16184
Process: PID = 16188, PPID = 16186
```

2 Running States

2.1 Question 1

What is the 1st argument to the `execv()` call? What is its contents?

The 1st argument to the `execv()` call is the path to the executable file. Its contents is the string representing the full path to the program you want to execute.

2.2 Question 2

What is the 2nd argument to it? What is its contents?

An array of character pointers to NULL-terminated strings. Your application must ensure that the last member of this array is a NULL pointer. These strings constitute the argument list available to the new process image. The value in `argv[0]` must point to a filename that's associated with the process being started. Neither `argv` nor the value in `argv[0]` can be NULL.

2.3 Question 3

What is `arg`?

Arg array of strings. It contains the arguments to be passed to the new process being executed.

2.4 Question 4

Look at the code of the child process (`p==0`). How many times does the statement "Child Process" appear? Why?

One time because once the `exec()` call is made, the current process is gone and a new process starts.

3 Exit Code output

```
int *
(base) tazneen@afroz:~/os_lab_06_part1$ gcc 4.c -o a
(base) tazneen@afroz:~/os_lab_06_part1$ ./a
Enter a Number: 67
Exit 1
(base) tazneen@afroz:~/os_lab_06_part1$ gcc 4.c -o a
(base) tazneen@afroz:~/os_lab_06_part1$ ./a
Enter a Number: 21
Exit 2
(base) tazneen@afroz:~/os_lab_06_part1$
```

If `num > 25`, the program exits using `exit(1)` in the main function.

If `num ≤ 25`, the program calls `anotherExit()`, which also exits using `exit(1)`.

3.1 Question 1

What is the difference between `exit()` and `atexit()`? What do they do?

`exit()` terminates the program immediately. `atexit()` registers functions to be called automatically when the program exits normally. How `atexit` stores functions: `atexit` uses an internal stack to store the functions registered with it. When you register a function using `atexit.register()`, it's added to this stack. Execution order: The functions are called in the reverse order of registration. This means the last function registered is called first, and the first function registered is called last.

3.2 Question 2

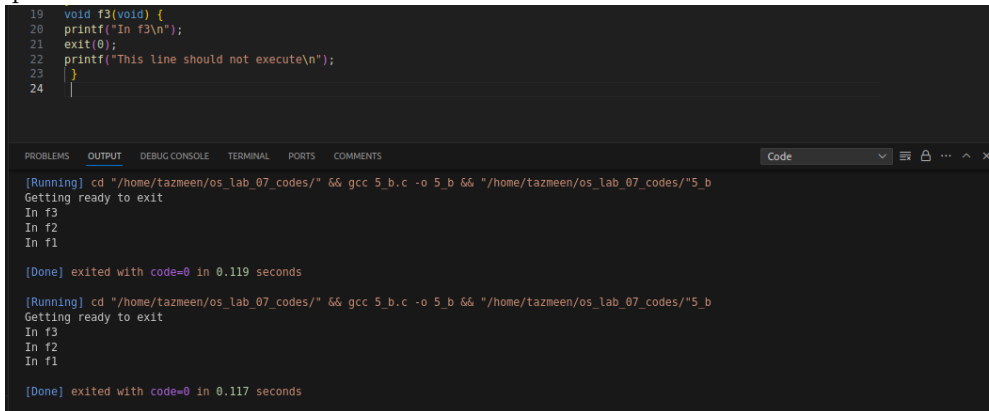
What does the 0 provided in the `exit()` call mean? What will happen if we change it to 1?

0 indicates successful termination. 1 (or any non-zero value) typically indicates an error condition.

3.3 Question 3

If we add an `exit` call to function `f1`, `f2`, or `f3`. What will happen to execution of our program?

If an `exit()` call is added to `f1`, `f2`, or `f3`, it will terminate only the function in which it is called rest of the process will be executed.



```
19 void f3(void) {
20     printf("In f3\n");
21     exit(0);
22     printf("This line should not execute\n");
23 }
24
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Code

```
[Running] cd "/home/tazmeen/os_lab_07_codes/" && gcc 5_b.c -o 5_b && "/home/tazmeen/os_lab_07_codes/"5_b
Getting ready to exit
In f3
In f2
In f1

[Done] exited with code=0 in 0.119 seconds

[Running] cd "/home/tazmeen/os_lab_07_codes/" && gcc 5_b.c -o 5_b && "/home/tazmeen/os_lab_07_codes/"5_b
Getting ready to exit
In f3
In f2
In f1

[Done] exited with code=0 in 0.117 seconds
```

3.4 Question 4

Why do you think we are getting reverse order of execution of `atexit` calls?

`atexit()` functions are called in the reverse order of their registration. This is because they are typically stored in a stack-like structure, where the last registered function is called first.

4 ABORT

4.1 Question 1

Check the man pages for `abort`. How does the `abort` call terminate the program? What is the name of the particular signal?

The `abort()` function terminates the program abnormally by raising the `SIGABRT` signal.

4.2 Question 2

Execute your program. What is the output of our program?

The program will output "Aborted (core dumped)" to the console.

4.3 Question 3

Include the `abort` call in function `f3` in our code provided for `Atexit()` call. How does our program terminate using this?

If `abort()` is called in `f3`, the program will terminate abnormally when `f3` is called during the `atexit` sequence. Any `atexit` functions registered after `f3` will not be executed. Thus `f2`, `f1` will not be executed.

5 KILL Call

In the `kill(getpid(), 9);` call, the number 9 is a signal number that corresponds to the `SIGKILL` signal. Here's what happens in detail: `kill()` function: This function is used to send a signal to a process. The first

argument is the process ID (PID) to which you want to send the signal, and the second argument is the signal number itself. `getpid()`: This function returns the process ID of the currently running process (the program itself in this case). 9: This is the signal number for SIGKILL, which is a signal that immediately terminates the process. It cannot be caught, blocked, or ignored. When SIGKILL is sent to a process, the operating system forcefully kills the process without giving it a chance to perform cleanup actions.

```
My parent is 1288

[Done] exited with code=0 in 5.102 seconds

[Running] cd "/home/tazmeen/os_lab_07_codes/" && gcc 8.c -o 8 && "/home/tazmeen/os_lab_07_codes/"8
My parent is 13841
My parent is 13841
My parent is 1288
My parent is 1288
My parent is 1288

[Done] exited with code=0 in 5.123 seconds
```

5.1 Question 1

What are the PPID values you are receiving from the for loop?

Initially, the child process prints 13841 as its PPID, which is the PID of its parent process. After the parent process terminates (after the second iteration), the PPID changes to 1288, which is the PID of the init process (or systemd) that adopts orphaned processes.

5.2 Question 2

What has happened when the PPID changes?

The change in PPID from 13841 to 1288 indicates that the parent process has exited. When this happens, the child process becomes orphaned, and the operating system assigns the child to the init process (systemd in modern systems). This is why the PPID becomes 1288, which is the PID of the init/systemd process.

5.3 Question 3

What is now the PID of the init process?

The init/systemd process has a PID of 1288. This process takes over orphaned child processes, ensuring they can continue running if necessary.

```
PID TTY TIME CMD
1288 ? 00:00:00 systemd
o (base) tazmeen@afroz:~/os_lab_07_codes$
```

6 Zombie process

```
(base) tazmeen@afroz:~/os_lab_07_codes$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
tazmeen   1349  0.0  0.0 162744 6016 tty2    Ssl+ 15:04   0:00 /usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bt
tazmeen   1352  0.0  0.1 223396 15616 tty2    Sl+  15:04   0:00 /usr/libexec/gnome-session-binary --session=ubuntu
tazmeen   5958  0.0  0.0 11760 4736 pts/0    Ss+  15:14   0:00 bash
tazmeen   5986  0.0  0.0 11760 4992 pts/1    Ss  15:15   0:00 bash
tazmeen  14032  0.0  0.0 11788 5376 pts/3    Ss+  16:44   0:00 /usr/bin/bash --init-file /usr/share/code/resources/app/out/vs/workbench/con
tazmeen  14450  0.0  0.0 2644 1024 pts/1    St+  16:48   0:00 ./a
tazmeen  14451  0.0  0.0 0 0 pts/1    Z+  16:48   0:00 [a] <defunct>
tazmeen  14509  0.0  0.0 11760 5248 pts/4    Ss  16:49   0:00 bash
tazmeen  14559  0.0  0.0 13024 3328 pts/4    R+  16:50   0:00 ps au
(base) tazmeen@afroz:~/os_lab_07_codes$ ps
```

```
tazmeen@afroz: ~/os_lab_07_codes
(base) tazmeen@afroz:~/os_lab_07_codes$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
tazmeen   1349  0.0  0.0 162744  6816 tty2    Ssl+ 15:04   0:00 /usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bl
tazmeen   1352  0.0  0.1 223396 15616 tty2    Sl+  15:04   0:00 /usr/libexec/gnome-session-binary --session=ubuntu
tazmeen   5958  0.0  0.0  11760  4736 pts/0    Ss+  15:14   0:00 bash
tazmeen   5986  0.0  0.0  11760  4992 pts/1    Ss  15:15   0:00 bash
tazmeen  14032  0.0  0.0  11788  5376 pts/3    Ss+  16:44   0:00 /usr/bin/bash --init-file /usr/share/code/resources/app/out/vs/workbench/con
tazmeen  14458  0.0  0.0   2644  1824 pts/1    S+  16:48   0:00 /a
tazmeen  14451  0.0  0.0      0      0 pts/1    Z+  16:48   0:00 [a] <defunct>
tazmeen  14509  0.0  0.0  11760  5248 pts/4    Ss  16:49   0:00 bash
tazmeen  14559  0.0  0.0  13024  3328 pts/4    R+  16:50   0:00 ps au
(base) tazmeen@afroz:~/os_lab_07_codes$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
tazmeen   1349  0.0  0.0 162744  6816 tty2    Ssl+ 15:04   0:00 /usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bl
tazmeen   1352  0.0  0.1 223396 15616 tty2    Sl+  15:04   0:00 /usr/libexec/gnome-session-binary --session=ubuntu
tazmeen   5958  0.0  0.0  11760  4736 pts/0    Ss+  15:14   0:00 bash
tazmeen   5986  0.0  0.0  11760  4992 pts/1    Ss+  15:15   0:00 bash
tazmeen  14032  0.0  0.0  11788  5376 pts/3    Ss+  16:44   0:00 /usr/bin/bash --init-file /usr/share/code/resources/app/out/vs/workbench/con
tazmeen  14509  0.0  0.0  11760  5248 pts/4    Ss  16:49   0:00 bash
tazmeen  14804  0.0  0.0  13024  3328 pts/4    R+  16:53   0:00 ps au
(base) tazmeen@afroz:~/os_lab_07_codes$
```

Both parent and zombie process are gone now