Assignment 2

Operating Systems | Anthony Nguyen

make_zombie.c = Part 1 source code | **main_program.c** = Part 2 source code

Implementation Details and Explanation:

make_zombie.c (part 1):

```
6   int main()
7   {
8       // Creates child with f
9       int pid = fork();
10
11      sleep(1);   // Used to
12
13      // If fork failed, exit
14      if(pid < 0)
15          exit(EXIT_FAILURE);
16      else if(pid != 0)
17          sleep(10);
18      return 0;
19  }
20
```

This C program is simple. On line 9 fork() is called which creates a new process (child) and the return value is stored into a int pid variable. By line 9 there exists two processes, a child, and a parent each with a different value inside the int pid variable. Both the child and parent processes sleep for 1 second just to give the program 1 second before a zombie is created.

fork() return based on process:
Parent -> Returns the childs process id
child -> Returns 0

Therefore, in the if conditions from line 14 to 17 the parent process will halt for 10 seconds and the child process will pass through all if statement and return 0 (exit/terminate). Though since the parent process did not wait for the child and is busy, then for about 10 seconds the child becomes a zombie until the parent terminates.

In short, this program simply waits for 1 second then creates a zombie process for about 10 seconds.

main_program.c (part 2):

```
5   int main()
6   {
7       system("gcc -o zombie.out make_zombie.c");   // compile the zombie C file and name it zombi
8
9       printf("\n*****Before Zombie Creation*****\n");
10      system("./zombie.out &");    // Run the program in #1 (Create a zombie process)
11      system("ps -l");             // Display active processes in "Long" format
12      sleep(1);                    // Re-sync with zombie.out
13
14      printf("\n*****After Zombie Creation*****\n");
15      system("ps -l");             // Display active processes in "Long" format
16      system("kill -9 $(ps -o ppid,stat | grep 'Z' | sed 's/^ *//g' | cut -d ' ' -f1 | xargs)");
17
18      printf("\n*****After Slaughtering The Zombie(s)*****\n");
19      system("ps -l");             // Display active processes in "Long" format
20
21      return 0;
22  }
```

(I mostly used the system function to execute shell commands)

This program starts by first compiling the make_zombie.c that was made for part 1 and naming the executable zombie.out. After compilation is completed it runs the executable which creates a zombie child for approximately 10 seconds. All the parents of a zombie are sent a kill signal of 9 which is a SIGKILL signal that shuts the process down instantly.

Note: A simple "ps -l" command to show active processes is called three times:
(1) Before the zombie creation
(2) After the zombie creation and before the kill signal
(3) After the kill signal

Note: The sleep on line 12 is used to wait for the zombie.out program to create a zombie since in program 1 we waited for 1 seconds to create a zombie. This is mainly used to show the status of the parent and child processes in part 1 before it becomes a zombie.

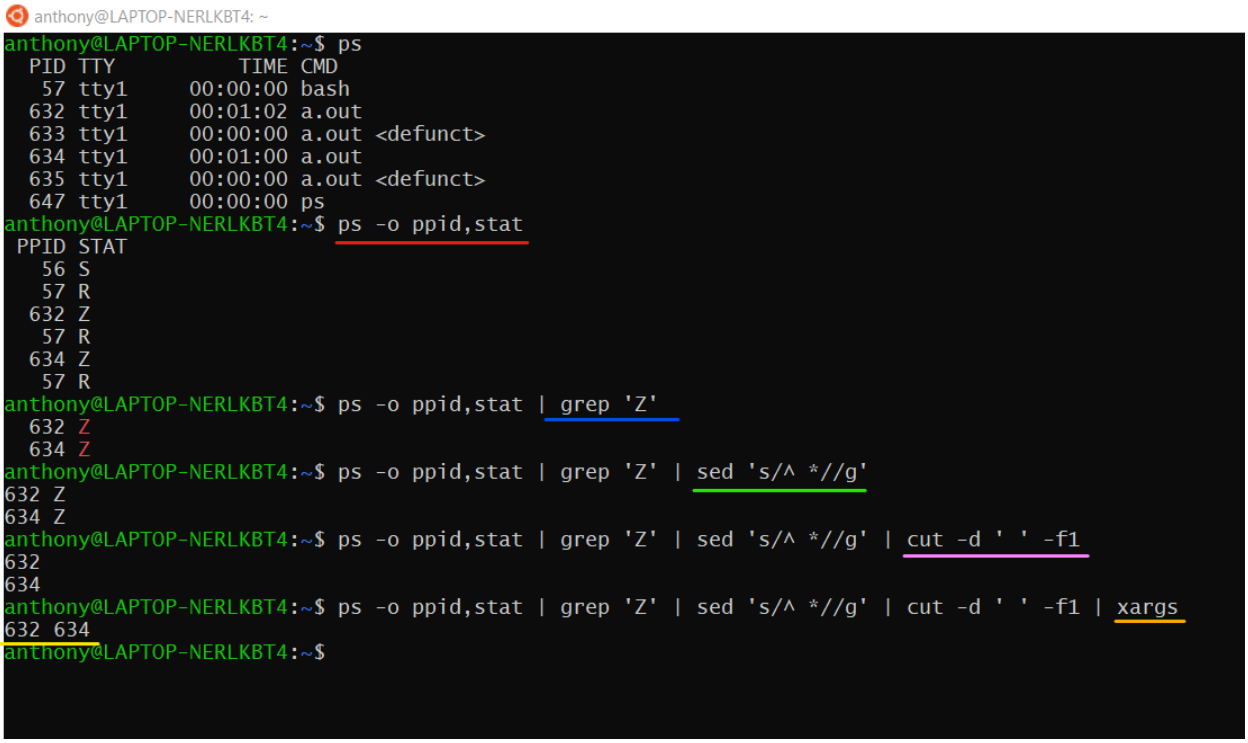
```
system("kill -9 $(ps -o ppid,stat | grep 'Z' | sed 's/^ *//g' | cut -d ' ' -f1 | xargs)");
```

Explanation for Line 16:
I Assumed: ALL/ANY parent(s) of a zombie will get killed (based on assignment wording)

kill -9 $(command) → Sends a kill signal 9 to whatever is inside **$()**
ps -o ppid,stat → outputs only the parent id and status of the current prosseses
grep 'Z' →Will only grab the rows/processes that are zombies
sed 's/^ *//g' → Used to remove leading white spaces (string transformation)
Note: Its actually needed because based on how many digits the ppid has the leading white spaces can vary which causes errors later on in the cut.
cut -d ' ' -f1 → cuts "ppid Z" into multiple parts delimited on white spaces and grabs the 1st section of the cut which is the parent process id (PPID) of the zombie
xargs → it takes standard inputs and arranges them into arguments like "arg1 arg2 arg3 etc"
Note: xargs Is used if there were multiple zombies (if not, then it still runs perfectly fine)

All the commands piped after each other grabs the pid of all the parents that have zombie and sends them a kill signal (kill -9). Below I show a screenshot to understand the multiple commands pipped together.

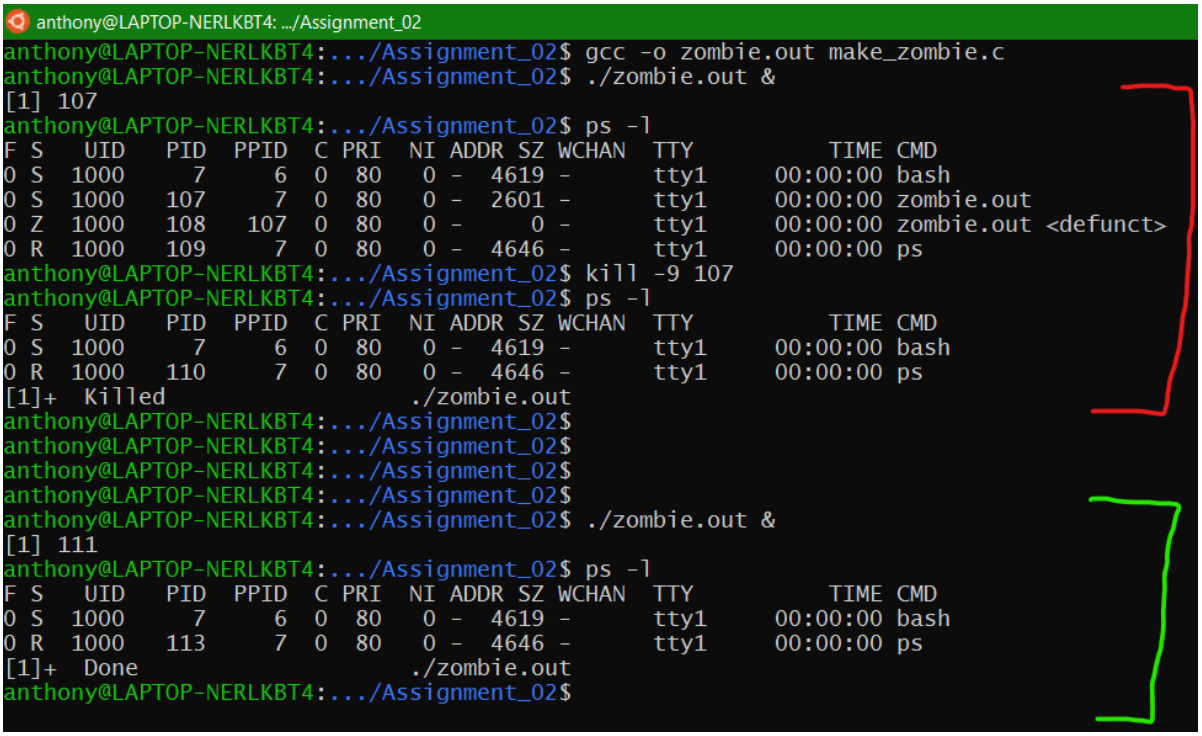
Simpler breakdown of the multiple piped commands with 2 zombies



Note: You can see that the screenshot above shows 2 active zombie processes

Sample Run of Part 1:

Two sample runs of the program **make_zombie.c** with a helpful **ps -l** calls.



First Run

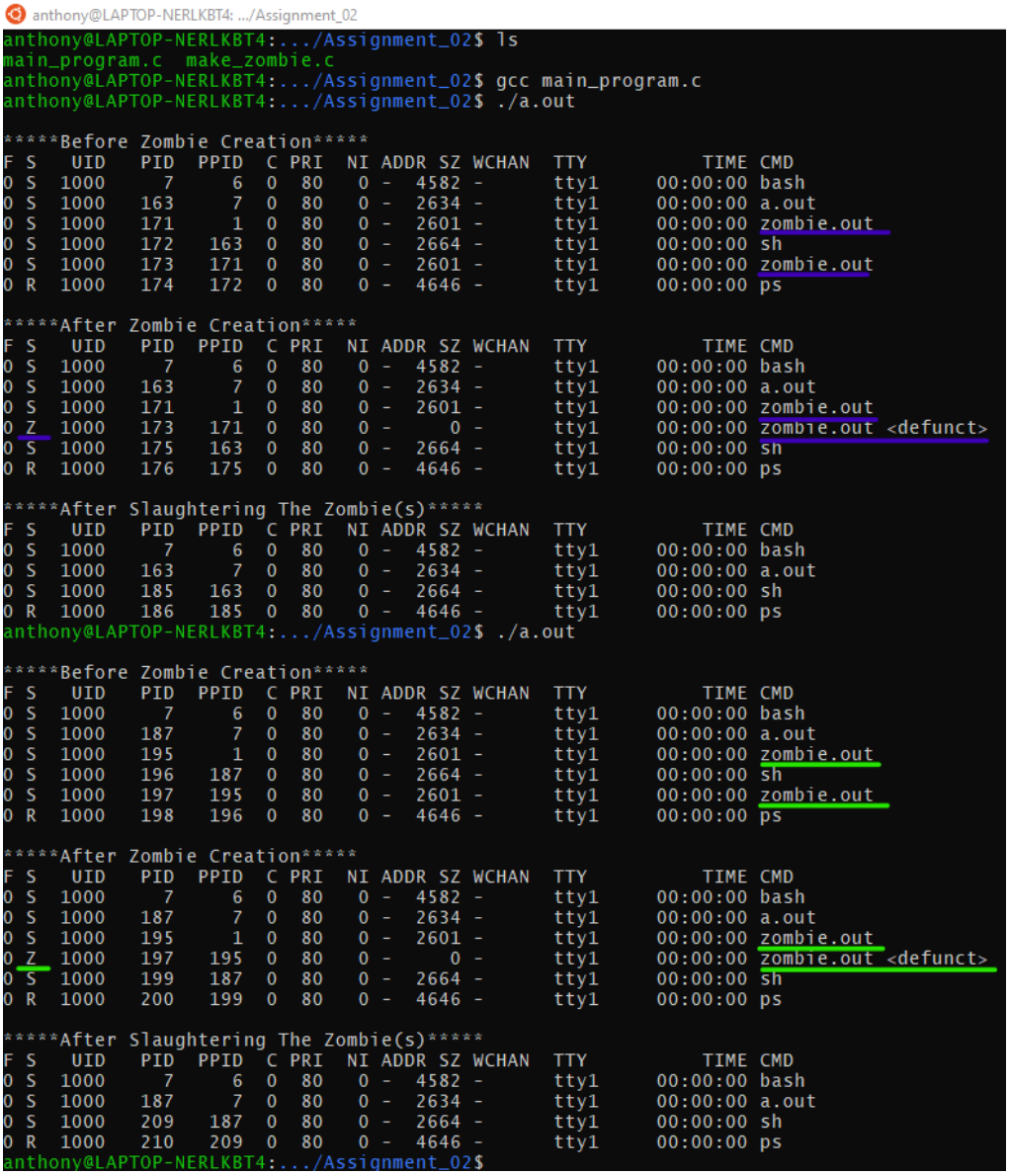
In the first (red) run of the first program I found the zombies parent process id with “ps -l” and I sent a kill signal to the parent. This later shows that the zombie is also killed.

Second Run

You can see that after the second (green) run of the first program I waited 10 seconds and ran “ps -l” which shows the zombie terminated (this is because after 10 seconds the parent program terminates which therefore terminates the child zombie).

Sample Run of Part 2:

Two sample runs of the program **main_program.c**.



```
anthony@LAPTOP-NERLKBT4: .../Assignment_02
anthony@LAPTOP-NERLKBT4:.../Assignment_02$ ls
main_program.c  make_zombie.c
anthony@LAPTOP-NERLKBT4:.../Assignment_02$ gcc main_program.c
anthony@LAPTOP-NERLKBT4:.../Assignment_02$ ./a.out

*****Before Zombie Creation*****
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     7     6  0  80   0 -  4582 -      tty1     00:00:00 bash
0 S  1000   163     7  0  80   0 -  2634 -      tty1     00:00:00 a.out
0 S  1000   171     1  0  80   0 -  2601 -      tty1     00:00:00 zombie.out
0 S  1000   172   163  0  80   0 -  2664 -      tty1     00:00:00 sh
0 S  1000   173   171  0  80   0 -  2601 -      tty1     00:00:00 zombie.out
0 R  1000   174   172  0  80   0 -  4646 -      tty1     00:00:00 ps

*****After Zombie Creation*****
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     7     6  0  80   0 -  4582 -      tty1     00:00:00 bash
0 S  1000   163     7  0  80   0 -  2634 -      tty1     00:00:00 a.out
0 S  1000   171     1  0  80   0 -  2601 -      tty1     00:00:00 zombie.out
0 Z  1000   173   171  0  80   0 -     0 -      tty1     00:00:00 zombie.out <defunct>
0 S  1000   175   163  0  80   0 -  2664 -      tty1     00:00:00 sh
0 R  1000   176   175  0  80   0 -  4646 -      tty1     00:00:00 ps

*****After Slaughtering The Zombie(s)*****
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     7     6  0  80   0 -  4582 -      tty1     00:00:00 bash
0 S  1000   163     7  0  80   0 -  2634 -      tty1     00:00:00 a.out
0 S  1000   185   163  0  80   0 -  2664 -      tty1     00:00:00 sh
0 R  1000   186   185  0  80   0 -  4646 -      tty1     00:00:00 ps
anthony@LAPTOP-NERLKBT4:.../Assignment_02$ ./a.out

*****Before Zombie Creation*****
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     7     6  0  80   0 -  4582 -      tty1     00:00:00 bash
0 S  1000   187     7  0  80   0 -  2634 -      tty1     00:00:00 a.out
0 S  1000   195     1  0  80   0 -  2601 -      tty1     00:00:00 zombie.out
0 S  1000   196   187  0  80   0 -  2664 -      tty1     00:00:00 sh
0 S  1000   197   195  0  80   0 -  2601 -      tty1     00:00:00 zombie.out
0 R  1000   198   196  0  80   0 -  4646 -      tty1     00:00:00 ps

*****After Zombie Creation*****
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     7     6  0  80   0 -  4582 -      tty1     00:00:00 bash
0 S  1000   187     7  0  80   0 -  2634 -      tty1     00:00:00 a.out
0 S  1000   195     1  0  80   0 -  2601 -      tty1     00:00:00 zombie.out
0 Z  1000   197   195  0  80   0 -     0 -      tty1     00:00:00 zombie.out <defunct>
0 S  1000   199   187  0  80   0 -  2664 -      tty1     00:00:00 sh
0 R  1000   200   199  0  80   0 -  4646 -      tty1     00:00:00 ps

*****After Slaughtering The Zombie(s)*****
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     7     6  0  80   0 -  4582 -      tty1     00:00:00 bash
0 S  1000   187     7  0  80   0 -  2634 -      tty1     00:00:00 a.out
0 S  1000   209   187  0  80   0 -  2664 -      tty1     00:00:00 sh
0 R  1000   210   209  0  80   0 -  4646 -      tty1     00:00:00 ps
anthony@LAPTOP-NERLKBT4:.../Assignment_02$
```

Simple breakdown: A zombie was created to live for 10s (since parent is sleeping for 10s) but its parent is killed from a kill signal sent to its pid which results in the death of the zombie.
(The more detailed explanation of each section was above)

You can notice that every process prints there are 3 important processes which are:
a.out – Part 2 program process
zombie.out – Part 1 program parent process
zombie.out – Part 1 program child process

Throughout the prints you will notice that It clearly shows the child **zombie.out** becoming a zombie and then dying after I kill its parent process (Note: Look at the status of the processes)