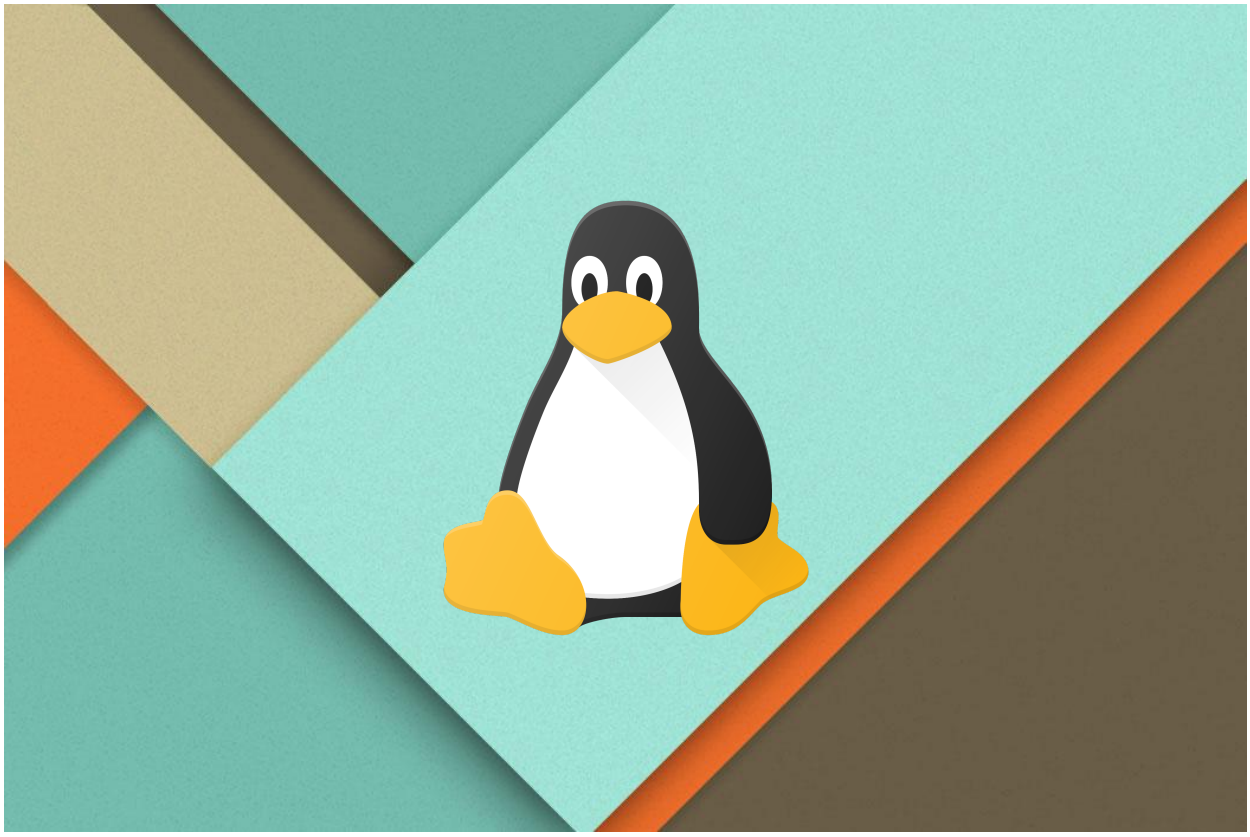


Operating Systems Lab

Experiment No. 4

19.02.2022

Professor - Dr. Shrinivas Khedkar.

Pratham Loya
201080068
IT
prloya_b20@it.vjti.ac.in

Question 1

Using either a UNIX or a Linux system, write a C program that forks a child process that ultimately becomes a zombie process.

This zombie process must remain in the system for at least 10 seconds.

Process states can be obtained from the command

ps –l

The process states are shown below the S column; processes with a state of Z are zombies.

The process identifier (pid) of the child process is listed in the PID column, and that of the parent is listed in the PPID column.

Perhaps the easiest way to determine that the child process is indeed a zombie is to run the program that you have written in the background (using the &) and then run the command ps -l to determine whether the child is a zombie process.

Because you do not want too many zombie processes existing in the system, you will need to remove the one that you have created.

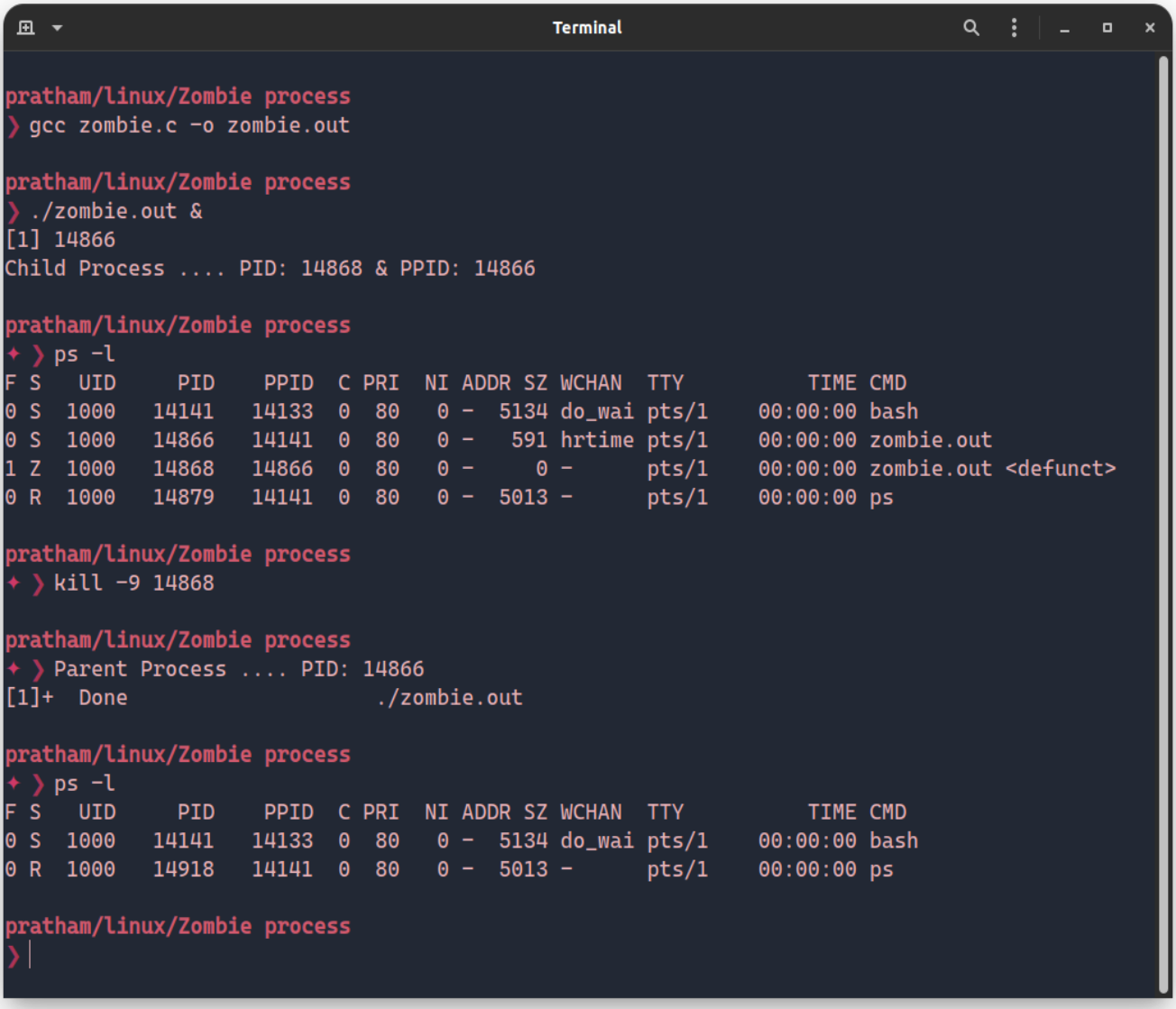
The easiest way to do that is to terminate the parent process using the kill command. For example, if the process id of the parent is 4884, you would enter.

kill -9 4884

Program

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

int main(int argc, char const *argv[])
{
    int p;
    p = fork();

    if (p == 0) // Child Process
    {
        printf("Child Process .... PID: %d & PPID:
%d\n",getpid(),getppid());
    }
    else // Parent Process
    {
        sleep(10);
        printf("Parent Process .... PID: %d",getpid());
    }

    return 0;
}
```

Output


Terminal
pratham/linux/Zombie process
❯ gcc zombie.c -o zombie.out
pratham/linux/Zombie process
❯ ./zombie.out &
[1] 14866
Child Process .... PID: 14868 & PPID: 14866
pratham/linux/Zombie process
✦ ❯ ps -l
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
0 S 1000 14141 14133 0 80 0 - 5134 do_wai pts/1 00:00:00 bash
0 S 1000 14866 14141 0 80 0 - 591 hrtime pts/1 00:00:00 zombie.out
1 Z 1000 14868 14866 0 80 0 - 0 - pts/1 00:00:00 zombie.out <defunct>
0 R 1000 14879 14141 0 80 0 - 5013 - pts/1 00:00:00 ps
pratham/linux/Zombie process
✦ ❯ kill -9 14868
pratham/linux/Zombie process
✦ ❯ Parent Process .... PID: 14866
[1]+ Done ./zombie.out
pratham/linux/Zombie process
✦ ❯ ps -l
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
0 S 1000 14141 14133 0 80 0 - 5134 do_wai pts/1 00:00:00 bash
0 R 1000 14918 14141 0 80 0 - 5013 - pts/1 00:00:00 ps
pratham/linux/Zombie process
❯

Question 2

The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

n = **n/2** if n is **even**

= **3×n + 1** if n is **odd**

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1.

For example, if n = 35, the sequence is 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Write a C program using the fork() system call that generates this sequence in the child process.

The starting number will be provided from the command line.

For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1.

Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence.

Have the parent invoke the **wait()** call to wait for the child process to complete before exiting the program.

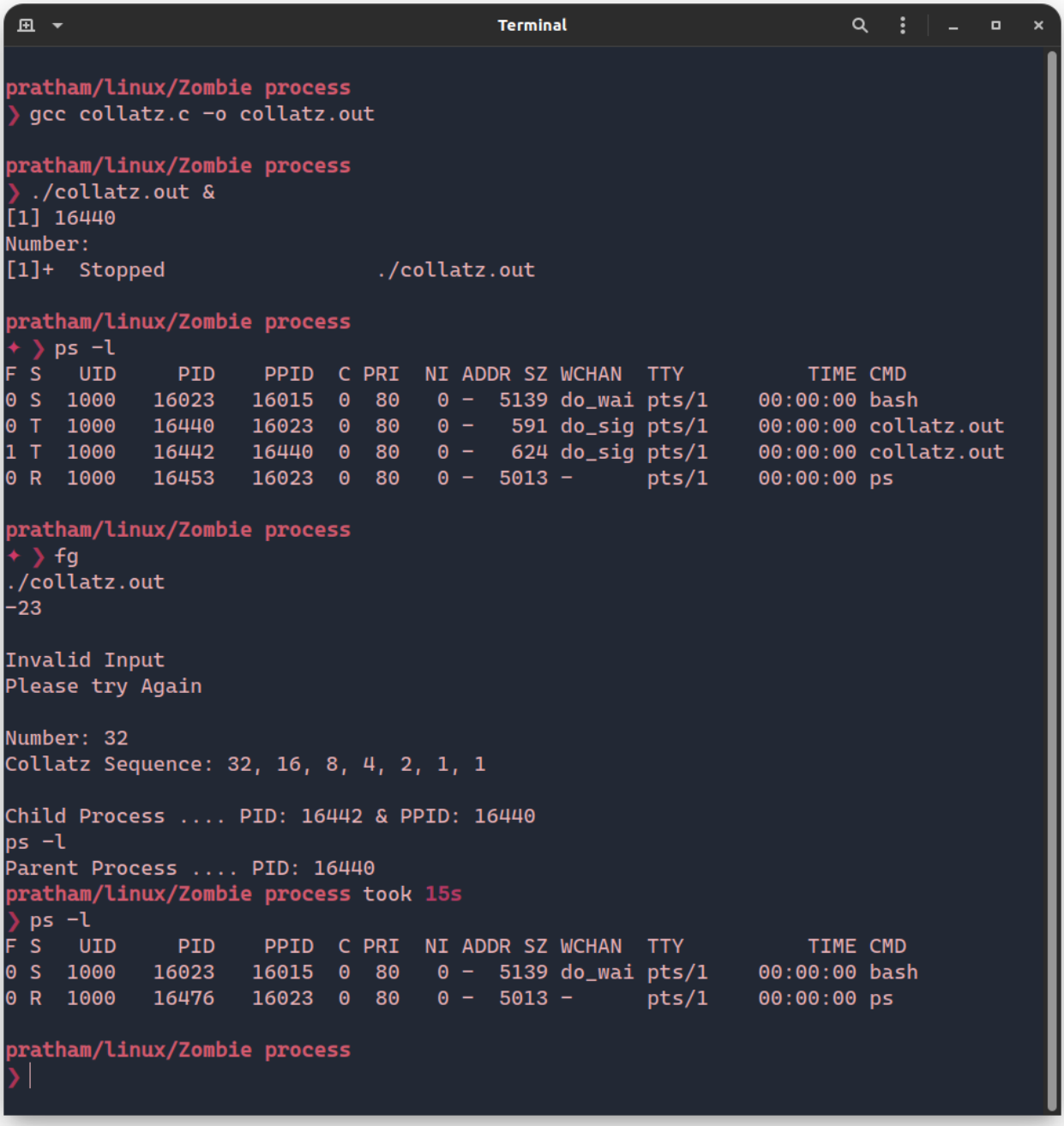
Perform necessary **error checking** to ensure that a positive integer is passed on the command line.

Program

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

int main(int argc, char const *argv[])
{
    int p;
    p = fork();

    if (p == 0) // Child Process
    {
        int n;
        label:
        printf("Number: ");
        scanf("%d",&n);

        if (n<=0)
        {
            printf("\nInvalid Input\n");
            printf("Please try Again\n\n");
            goto label;
        }
```

```
printf("Collatz Sequence: %d, ",n);
        while (n > 1)
        {
            if (n % 2 == 0)
            {
                n/=2;
            }
            else
            {
                n = 3*n + 1;
            }
            printf("%d, ",n);
        }
        printf("1\n");


        printf("\nChild Process .... PID: %d & PPID:
%d\n",getpid(),getppid());
    }
    else // Parent Process
    {
        wait(NULL);
        sleep(10);
        printf("Parent Process .... PID: %d",getpid());
    }

    return 0;
}
```

Output



Thank You