Tutorial Assignment 2 - Processes
REPORT

PLAGIARISM STATEMENT

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name: Siddharth Saini
Date: 09/11/2020
Signature: SS

ProcEx1.c

```
siddharth@debian:~$ ./ProcEx1.o
commands.txt  example2.ohelloWorld.o   Pictures   ProcEx3.c  ProcEx5.o
Desktop       example3.c   LinkedList.c   ProcEx1.c  ProcEx3.o  Public
Documents     Example3.c   linkedList.o   ProcEx1.o  ProcEx4.c  Songs
Downloads     example3.o   numberPlate.c  ProcEx2.c  ProcEx4.o  Templates
example2.c    helloWorld.c  numberPlate.o  ProcEx2.o  ProcEx5.c  Videos
Child Complete
```

- The fork() creates a new process, creating two identical copies of address spaces: one for child and one for parent
- Since it has a non-zero PID assigned to “pid”, it returns to the parent process.
- There’s no error message, that means the child process is successfully created.
- Since the parent process waits for the child process to complete, the child process starts executing first and shows all the directory and files.
- Once the child process is completed, it again enters the parent process and prints the given text : “Child Complete”.

ProcEx2.c

```
siddharth@debian:~$ ./ProcEx2.o
My pid: 3306 My child’s pid: 3307
siddharth@debian:~$ My pid: 3307 I am a child of ./ProcEx2.o My parent pid: 723
```

- The fork() creates a new process and has a non-zero PID assigned to “cpid” which returns to the parent process.
- The parent process executes first and prints the required things as stated in the c program.
- Since we didn’t wait for the child process to execute in this example, the parent process finishes the execution first, unlike the above example.
- This is the reason why the terminal shows us back to the directory once the parent process gets completed.
- This is also the reason why the parent PID doesn’t match with 3306.
  Since we didn’t wait for the child to execute at first, the parent process executed and hence the child lost its parent PID to some other, since the process no longer existed.
- If we had used “wait(NULL)” in else statement, the child process would have executed first and the terminal would have neither shown the home directory nor does the parent PID would have changed.
- The child process executes after the parent process and since it says to sleep for 5 seconds, the process delays the execution and the child process is executed after 5 seconds.
- The statement of the child process is printed later on after 5 seconds.

ProcEx3.c

```
siddharth@debian:~$ ./ProcEx3.o
parent pid = 3340
In Parent: child pid = 3341
In Parent: waiting for child
In child: Iteration: 0
In child: Iteration: 1
In child: Iteration: 2
In child: Iteration: 3
In child: Iteration: 4
In child: Iteration: 5
In child: Iteration: 6
In child: Iteration: 7
In child: Iteration: 8
In child: Iteration: 9
In child: child exiting
In Parent: Child exit status:0
In Parent: Child exited normally
In Parent: child terminated
In Parent: parent exiting
```

- The “pid” is first assigned to the process id of the parent process which thereby prints the process id.
- The fork() creates a new process and has a non-zero PID assigned to “pid” which returns to the parent process.
- The parent process executes first and prints the required things as stated in the c program.
- It then waits for the child process to execute.
- The child process prints the required things 10 times iterating over the loop.
- It then returns to the parent process to get completed and prints the required things.

Now we un-comment the “sleep(2)” from the code.

```
siddharth@debian:~$ gcc -o ProcEx3.o ProcEx3.c
siddharth@debian:~$ ./ProcEx3.o
parent pid = 3543
In child: Iteration: 0
In child: Iteration: 1
In child: Iteration: 2
In child: Iteration: 3
In child: Iteration: 4
In child: Iteration: 5
In child: Iteration: 6
In child: Iteration: 7
In child: Iteration: 8
In child: Iteration: 9
In child: child exiting
In Parent: child pid = 3544
In Parent: waiting for child
In Parent: Child exit status:0
In Parent: Child exited normally
In Parent: child terminated
In Parent: parent exiting
```

- All the things remain the same as the above except that the child process is executed first as the parent process sleeps for 2 seconds forcing the child to execute.

Now we un-comment the “sleep(2)” and “wait(NULL)” from the code.

```
siddharth@debian:~$ gcc -o ProcEx3.o ProcEx3.c
siddharth@debian./ProcEx3.o
parent pid = 3583
In child: Iteration: 0
In child: Iteration: 1
In child: Iteration: 2
In child: Iteration: 3
In child: Iteration: 4
In child: Iteration: 5
In child: Iteration: 6
In child: Iteration: 7
In child: Iteration: 8
In child: Iteration: 9
In child: child exiting
In Parent: child pid = 3584
In Parent: waiting for child
In Parent: Child exit status:127
In Parent: Child was killed by a signal!!
In Parent: child terminated
In Parent: parent exiting
```

- All the things remain the same as the above except:
  - the child process is executed first as the parent process sleeps for 2 seconds forcing the child to execute.
  - Since we waited for the child process to change state, it’s state changed from 0 to 127 which was then killed by a signal rather than exiting normally.

Now we un-comment the “sleep(2)” , “wait(NULL)”, “wait(&status)” from the code.

```
siddharth@debian:~$ gcc -o ProcEx3.o ProcEx3.c
siddharth@debian:~$ ./ProcEx3.o
parent pid = 3617
```

```
In child: Iteration: 0
In child: Iteration: 1
In child: Iteration: 2
In child: Iteration: 3
In child: Iteration: 4
In child: Iteration: 5
In child: Iteration: 6
In child: Iteration: 7
In child: Iteration: 8
In child: Iteration: 9
In child: child exiting
In Parent: child pid = 3618
In Parent: waiting for child
In Parent: Child exit status:127
In Parent: Child exited for other reasons
In Parent: child terminated
In Parent: parent exiting
```

- All the things remain the same as the above except:
    - the child process is executed first as the parent process sleeps for 2 seconds forcing the child to execute.
    - Since we waited for the child process to change state, it's state changed from 0 to 127 which was then exited due to other reasons rather than exiting normally.

Now we un-comment the "sleep(2)" , "wait(NULL)", "wait(&status)" and waitpid(-1,&status,0) from the code.

```
siddharth@debian:~$ gcc -o ProcEx3.o ProcEx3.c
siddharth@debian:~$ ./ProcEx3.o
parent pid = 3654
In child: Iteration: 0
In child: Iteration: 1
In child: Iteration: 2
In child: Iteration: 3
In child: Iteration: 4
In child: Iteration: 5
In child: Iteration: 6
In child: Iteration: 7
In child: Iteration: 8
In child: Iteration: 9
In child: child exiting
In Parent: child pid = 3655
In Parent: waiting for child
In Parent: Child exit status:127
In Parent: Child was killed by a signal!!
In Parent: child terminated
In Parent: parent exiting
```

- All the things remain the same as the above except:
    - the child process is executed first as the parent process sleeps for 2 seconds forcing the child to execute.
    - Since we waited for the child process to change state, it's state changed from 0 to 127 which was then killed by a signal rather than exiting normally.

ProcEx4.c

```
siddharth@debian:~$ ./ProcEx4.o
Parent PID:27510
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27510 and My Parent PID:3284
My PID:27517 and My Parent PID:27510
.
.
.
.
.
.
infinite
```

- It first prints the parent id.
- Then it enters the infinite loop which keeps on creating new processes in each iteration and printing the value of parent and child process id.
- The process ID of child and parent doesn't change for a considerable amount of time as the processes weren't completed. It changes as soon as both the parent and child process gets completed.
- The program terminates only when we press ctrl+C to exit the infinite loop.

ProcEx5.c

```
siddharth@debian:~$ ./ProcEx5.o
Parent PID:24719
My PID:24720 and My Parent PID:24719
My PID:24721 and My Parent PID:24719
My PID:24722 and My Parent PID:24719
My PID:24723 and My Parent PID:24719
My PID:24724 and My Parent PID:24719
My PID:24725 and My Parent PID:24719
My PID:24726 and My Parent PID:24719
My PID:24727 and My Parent PID:24719
My PID:24728 and My Parent PID:24719
My PID:24729 and My Parent PID:24719
My PID:24730 and My Parent PID:24719
My PID:24731 and My Parent PID:24719
My PID:24732 and My Parent PID:24719
My PID:24733 and My Parent PID:24719
My PID:24734 and My Parent PID:24719
My PID:24735 and My Parent PID:24719
My PID:24736 and My Parent PID:24719
My PID:24737 and My Parent PID:24719
My PID:24738 and My Parent PID:24719
My PID:24739 and My Parent PID:24719
My PID:24740 and My Parent PID:24719
My PID:24741 and My Parent PID:24719
My PID:24742 and My Parent PID:24719
My PID:24743 and My Parent PID:24719
My PID:24744 and My Parent PID:24719
My PID:24745 and My Parent PID:24719
My PID:24746 and My Parent PID:24719
My PID:24747 and My Parent PID:24719
My PID:24748 and My Parent PID:24719
My PID:24749 and My Parent PID:24719
My PID:24750 and My Parent PID:24719
My PID:24751 and My Parent PID:24719
My PID:24752 and My Parent PID:24719
My PID:24753 and My Parent PID:24719
My PID:24754 and My Parent PID:24719
My PID:24755 and My Parent PID:24719
My PID:24756 and My Parent PID:24719
My PID:24757 and My Parent PID:24719
My PID:24758 and My Parent PID:24719
My PID:24759 and My Parent PID:24719
My PID:24760 and My Parent PID:24719
My PID:24761 and My Parent PID:24719
My PID:24762 and My Parent PID:24719
My PID:24763 and My Parent PID:24719
My PID:24764 and My Parent PID:24719
My PID:24765 and My Parent PID:24719
My PID:24766 and My Parent PID:24719
My PID:24767 and My Parent PID:24719
My PID:24768 and My Parent PID:24719
.
.
```

```
.
.
infinite
```

- It first prints the parent id.
- Then it enters the infinite loop which keeps on creating new processes in each iteration till the processes stop generating.
- It keeps on printing the values of Child and Parent process IDs.
- In each iteration, the parent waits for the child to complete it’s process and therefore we observe that the parent PID doesn’t change for a considerable amount of time but the Child PID keeps on changing in each iteration as the Child process gets completed.
- The parent PID changes only when all its children have done the execution.
- The program terminates only when we press ctrl+C to exit the infinite loop.