Homework - Chapter 5

Simulation

2.

The height of the process tree increases as fork percentage went from 0.1 to 0.9.

3.

action 1: a forks b

action 2: a forks c

action 3: c exits

action 4: a forks d

action 5: a forks e

4.

1st try: -A a+b,b+c,c+d,c+e,c-

2nd try: -A a+b,b+c,c+d,d+e,b-

3rd try: -A a+b,b+c,c+d,e+p,e-

Finding: when a parent process is killed, its children becomes the children of root process (a or init), so the parent process is allowed to exit before its children exit, and the children becomes independant processes under root, not a child of its grandparent

4th try: -A a+b,b+c,c+d,a-

Finding: root process cannot exit

5.

seed: -1

```
  a
   |____ c
   |     |_____ e
   |_____d
```

seed: -2

```
   a
   |____ c
   |     |_____ d
   |_____e
```

6.

seed: -3

actions 5

Final Process Tree: in this case there are 5 actions, thus we can tell all of them are fork() given the tree.

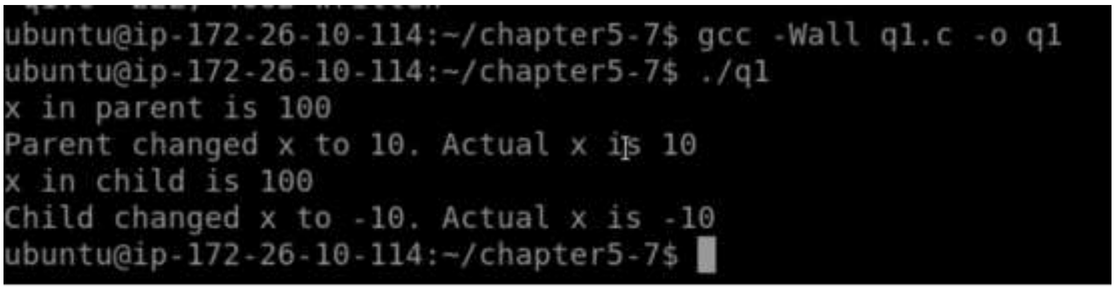
```
a
├── b
│   ├── c
│   └── f
└── d
    └── e
```

seed: -4
actions 5
Final Process Tree: in this case there are 5 actions, but because there are exit() involved, we can't tell the exact actions.
For example, the following tree can be a+b,a+c,a+d,a+e,d- or a+b,a+c,c+d,d+e,d-

```
a
├── b
├── c
└── e
```

Code

1. Write a program that calls fork(). Before calling fork(), have the main process access a variable (e.g., x) and set its value to something (e.g., 100). What value is the variable in the child process? What happens to the variable when both the child and parent change the value of x?

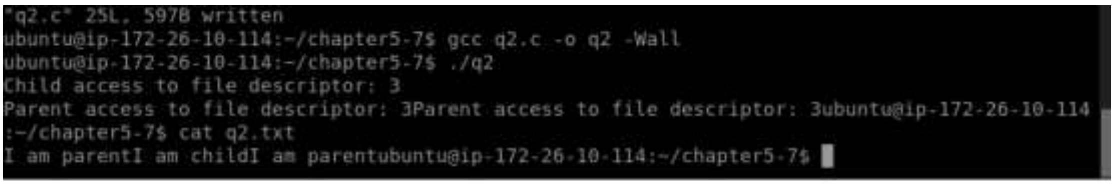


```
ubuntu@ip-172-26-10-114:~/chapter5-7$ gcc -Wall q1.c -o q1
ubuntu@ip-172-26-10-114:~/chapter5-7$ ./q1
x in parent is 100
Parent changed x to 10. Actual x is 10
x in child is 100
Child changed x to -10. Actual x is -10
ubuntu@ip-172-26-10-114:~/chapter5-7$
```

The child can access the value of parent, which is 100. (I think because the child is an exact copy of the current process, including the address space). However, when parent is executed first, and changed x to 10, x is still 100 in child. (I think because from the time child is forked, the current calling process and the child becomes different processes, thus they have their own private address space)

2. Write a program that opens a file (with the open() system call) and then calls fork() to create a new process. Can both the child and parent access the file descriptor returned by open()? What happens when they are writing to the file concurrently, i.e., at the same time?

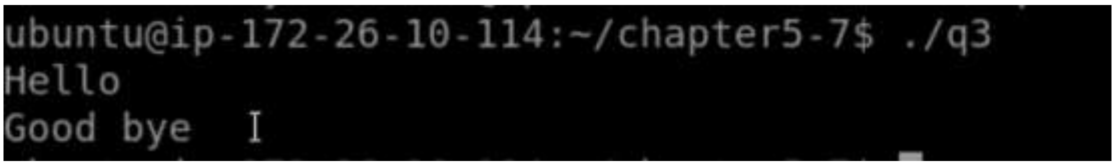
- Yes, child and parent can both access the returned file descriptor. (The child inherits open file descriptors)
- Both of them can write to the same file opened in the parent process.



```
"q2.c" 25L, 597B written
ubuntu@ip-172-26-10-114:~/chapter5-7$ gcc q2.c -o q2 -Wall
ubuntu@ip-172-26-10-114:~/chapter5-7$ ./q2
Child access to file descriptor: 3
Parent access to file descriptor: 3Parent access to file descriptor: 3ubuntu@ip-172-26-10-114
:~/chapter5-7$ cat q2.txt
I am parentI am childI am parentubuntu@ip-172-26-10-114:~/chapter5-7$ 
```

3. Write another program using fork(). The child process should print “hello”; the parent process should print “goodbye”. You should try to ensure that the child process always prints first; can you do this without calling wait() in the parent?

Yes, we can do this by using pipe( ). Pipe provides one-way communication between processes. Pipe also provides a synchronization mechanism, when a process tries to read from the file that hasn’t been written, it will wait until it is written.



```
ubuntu@ip-172-26-10-114:~/chapter5-7$ ./q3
Hello
Good bye
```

4. Write a program that calls fork() and then calls some form of exec() to run the program /bin/ls. See if you can try all of the variants of exec(), including (on Linux) execl(), execle(), execlp(), execv(), execvp(), and execvpe(). Why do you think there are so many variants of the same basic call?

Different variants of exec() family allows for different use cases, and provides convenient interface for user according to their needs.

l stands for location
e stands for environment variable
p stands for PATH
v stands for vector (array)

By combining them, developers have different ways to execute an executable.

5. Now write a program that uses wait() to wait for the child process to finish in the parent. What does wait() return? What happens if you use wait() in the child?

wait( ) returns the pid of child process (if child succeeds)
Call wait( ) in the child returns -1, indicating an error, because the child process doesn’t have a child process.

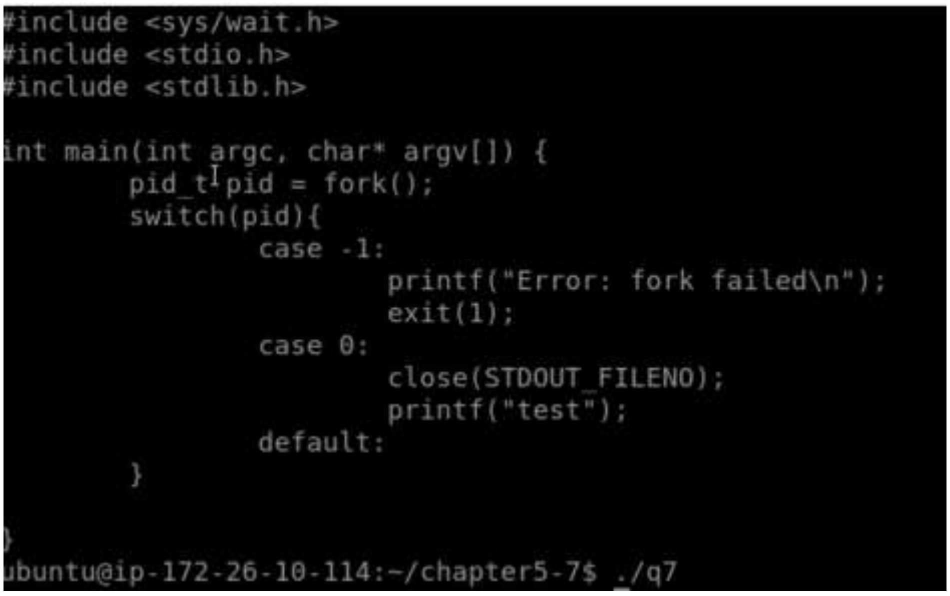
6. Write a slight modification of the previous program, this time using waitpid() instead of wait(). When would waitpid() be useful?

pid_t waitpid(pid_t pid, int *status_ptr, int options);
When the parent process wants to block itself until the child specified by pid has

terminated.This call allows parent process to specify a child to wait by pid, among multiple children.
The different arguments of pid_t pid can specify how the block of parent behaves, e.g. wait for a specific child, or wait for any child.

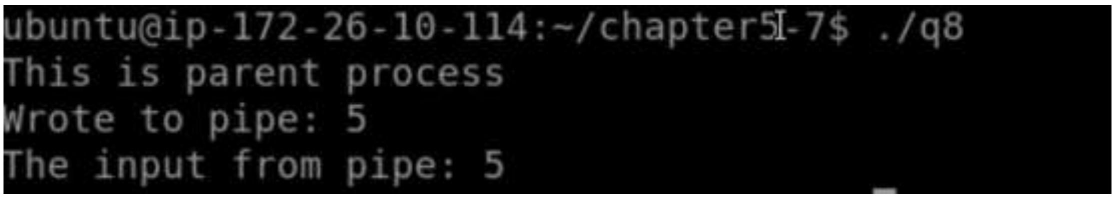
7. Write a program that creates a child process, and then in the child closes standard output (STDOUT FILENO). What happens if the child calls printf() to print some output after closing the descriptor?

Nothing will be printed. (I think because that file descriptor is closed, so we cannot write to it anymore.)



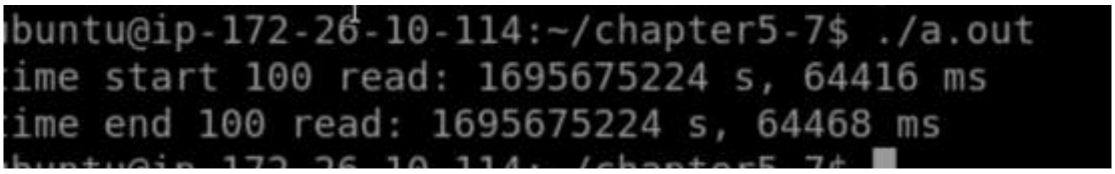
```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
        pid_t pid = fork();
        switch(pid){
                case -1:
                        printf("Error: fork failed\n");
                        exit(1);
                case 0:
                        close(STDOUT_FILENO);
                        printf("test");
                default:
        }

}
ubuntu@ip-172-26-10-114:~/chapter5-7$ ./q7
```

8. Write a program that creates two children, and connects the standard output of one to the standard input of the other, using the pipe() system call.



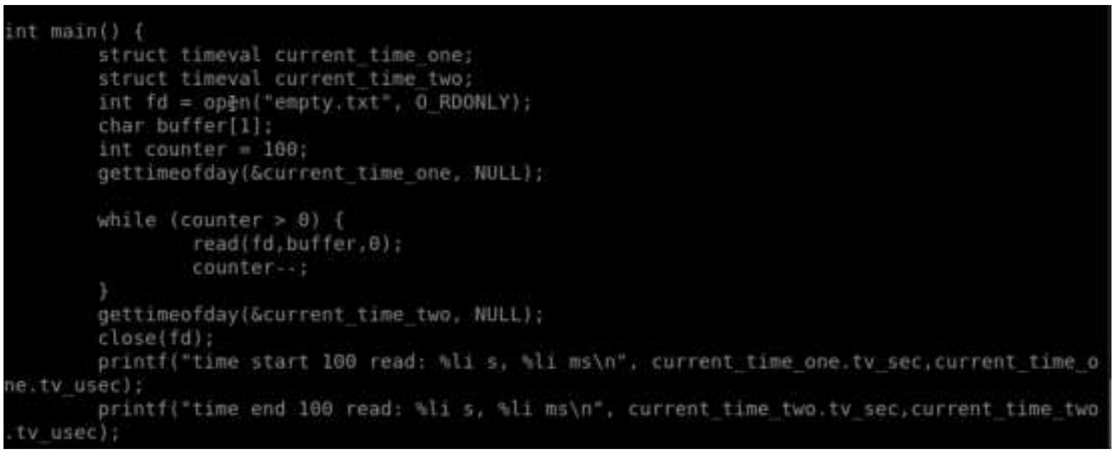
```
ubuntu@ip-172-26-10-114:~/chapter5-7$ ./q8
This is parent process
Wrote to pipe: 5
The input from pipe: 5
```

Homework - Chapter 6
Measure system call by a 0 byte read, result = 0.52ms.



```
ubuntu@ip-172-26-10-114:~/chapter5-7$ ./a.out
ime start 100 read: 1695675224 s, 64416 ms
ime end 100 read: 1695675224 s, 64468 ms
```

Code



```
int main() {
        struct timeval current_time_one;
        struct timeval current_time_two;
        int fd = open("empty.txt", O_RDONLY);
        char buffer[1];
        int counter = 100;
        gettimeofday(&current_time_one, NULL);

        while (counter > 0) {
                read(fd,buffer,0);
                counter--;
        }
        gettimeofday(&current_time_two, NULL);
        close(fd);
        printf("time start 100 read: %li s, %li ms\n", current_time_one.tv_sec,current_time_o
ne.tv_usec);
        printf("time end 100 read: %li s, %li ms\n", current_time_two.tv_sec,current_time_two
.tv_usec);
```

Homework – chapter 7

1. Compute the response time and turnaround time when running three jobs of length 200 with the SJF and FIFO schedulers.

	Average turnaround time	Average response time	Average wait time
FIFO:	400	200	200
SJF:	400	200	200

2. Now do the same but with jobs of different lengths: 100, 200, and 300.

	Average turnaround time	Average response time	Average wait time
FIFO:	333.33	133.33	133.33
SJF:	333.33	133.33	133.33

3. Now do the same, but also with the RR scheduler and a time-slice of 1.

	Average turnaround time	Average response time	Average wait time
FIFO:	400	200	200
SJF:	400	200	200
RR:	~~600~~ 599	1	~~1~~ 399

4. For what types of workloads does SJF deliver the same turnaround times as FIFO?

A, the length of each job are the same;
B, in the queue, the length of job is in increasing order.

5. For what types of workloads and quantum lengths does SJF deliver the same response times as RR?

the length of each job are the same, and quantum length equals the length of each job

6. What happens to response time with SJF as job lengths increase? Can you use the simulator to demonstrate the trend?

Job length	Average response time
30,20,10	13.33
300,200,100	133.33
3000,2000,1000	1333.33
30000,20000,10000	13333.33

7. What happens to response time with RR as quantum lengths increase? Can you write an equation that gives the worst-case response time, given N jobs?

The response time increases as quantum length increases.

The worst case should be the length of each job are longer than quantum length.

Let quantum length be Q, number of jobs N.
Average response time (worst case) = $Q + 2Q + 3Q + \cdots + (N-1)Q = NQ(N-1)/2$