ABID ALI
#2019380141

Homework-3

1 Using the program shown in below, explain the output will be at LINE A

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int value = 5;
int main()
{
  pid_t pid;
  pid = fork();
  if (pid == 0) { /* child process */
    value += 15;
    return 0;
  }
else if (pid > 0) { /* parent process */
  wait (NULL);
  printf ("PARENT: value = %d", value); /* Line A */
  return 0;
}
```

Ans: As soon as well fork(), both the child & the parent share the same data & codes and the adress space of the parent is exactly copied in child. In fact, only the page table gets occupied & new pages are not written for the child process. When any of the child or parent tries to change the data (the code is placed in the read-only part of the file hence it cann't be changed), an interrupt is generated & the pages are written for the child process before a parent or child can make any change.

We can say, the function `fork()` creates a new process that is a complete copy of the original process. The new process has it's own memory & it's own copy of all variables.

In the new child process the returned `pid` value is "zero". The child adds 15 to it's variable value & exits in the line.

```
if (pid == 0) {value += 15; return 0;}
```

NVR: { The value is '5' in the original process
has pid greater than zero & it goes to

else if (pid>0) { wait(NULL); printf("Value = %d",
return 0;}

Ans: The line prints: Value = 5

1. When a process creates a new process using the fork() operation, which of the following states is shared betw^n the parent process & the child process?

✓ a) Stack
✓ b) Heap
c) Shared Memory segments

Ans: When fork() is called, both stack & heap will be copied into a child process. However, copying will be delayed until chid process makes a change (write) in

those memories. This is called copy on write.
shared memory segments will not be copied.

Result: (a) Stack & (b) Heap will be copied

value);

2 Describe the actions taken by a kernel to context-switch betw processes.

Ans: Whenever the CPU starts executing a new process, the old process's state must be preserved which will later allow the old process to resume it's operations. Context-switching allow multiple processes to share a single CPU. A state save of current state of the CPU is performed & then a state restore to resume operations. The value of the processes's registers, the process state plus memory-management information info. must be saved & another process can then be loaded.

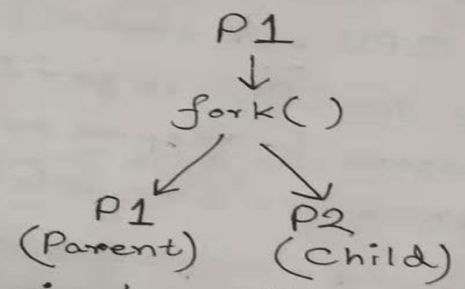
Summary: When a context switch occurs, the kernel saves the context of the old process in it's PCB and loads the saved context of the new process scheduled to run.

[3] What are the zombie & orphan process respectively?

Ans:

Suppose, we have a process P1.

Now, P1 forks itself, creating a chi process P2.



Then, P1 terminates without waiting for P2 to complete.

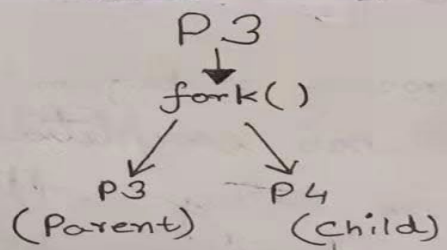
So, P2 is still running, but without it's parent process.

Therefore, P2 is an orphan.

Def:- A orphan process is a computer process whose parent process has finished or terminated, through it (child process) remains running itself.

Now, Consider another process P3.

Then, P3 fork itself, creating a child process P4.



However, P4 terminates before P3 does.

Then, P4's entry will not be removed from the process table until 'P3' read it's

exit() status using the wait() system call.

In other word, P4 is undead, a zombie. It needs to be reaped.

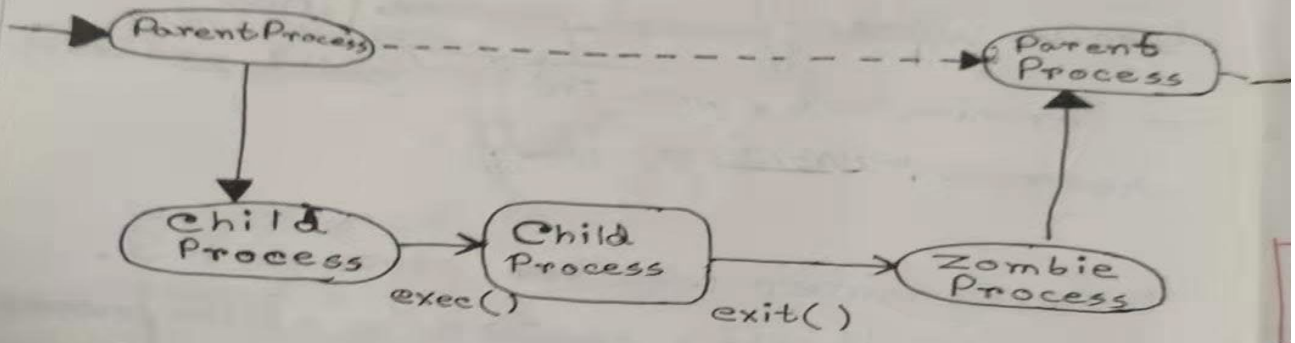


Fig: Zombie Process in Linux

A zombie process or defunct process is a process that has completed execution but still has an entry in the process table as it's parent process didn't invoke an wait() system call.

[4] How many times does this code print "hello"? and why?

```
void main(int argc, char **argv){
    int i;
    for(i=0; i<3; i++){
        fork();
        printf("hello\n");
    }
}
```

Ans:

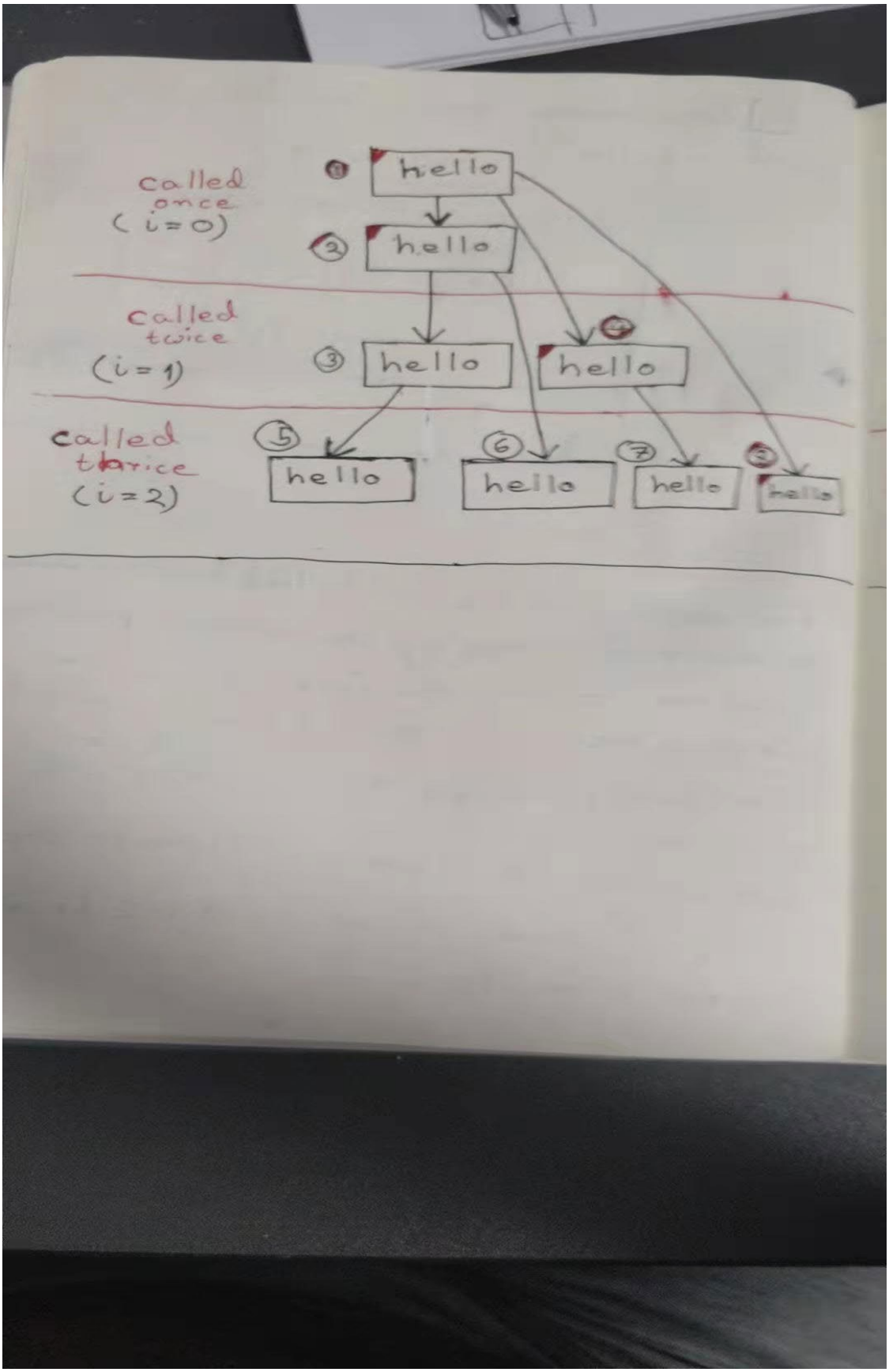
For each fork system called that we ... we create a clone of the current process. So, we are calling the fork() function 3 times which will result 8 process created

```
for(i=0; i<3; i++)
```

$\Rightarrow 2^n$

$\Rightarrow 2^3 = 8$.

We can see that, we will iterate the loop '3' time from i = 0 to 2 because we can't include 3


called once (i=0)
called twice (i=1)
called thrice (i=2)
hello
hello
hello
hello
hello
hello
hello
hello

Ans No: 5

Because, the child is a copy of parent, any changes the child makes will occur in it's copy of the data and won't be reflected in the parent. As a result, the values output by the child at line X are 0, -1, -4, -9, -16. The values output by the parents at line Y are 0, 1, 2, 3, 4.

Answer no. 6

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

int main()
{



        int k=0;

        pid_t pid;

                do
                {
                        printf("Please enter a number greater than 0 to run the Collatz
Conjecture.\n");
                        scanf("%d", &k);
                }while (k <= 0);

                pid = fork();

                if (pid == 0)
                {
                        printf("Child is working...\n");
                        printf("%d\n",k);
                        while (k!=1)
```

```
                {
                        if (k%2 == 0)
                        {
                                k = k/2;
                        }
                        else if (k%2 == 1)
                        {
                                k = 3 * (k) + 1;
                        }

                        printf("%d\n",k);
                }

                printf("Child process is done.\n");
        }
        else
        {
                printf("Parents is waiting on child process...\n");
                wait();
                printf("Parent process is done.\n");
        }
    return 0;
}
```


abid@ubuntu:~/HelloWorld$ ./Hello
Please enter a number greater than 0 to run the Collatz Conjecture.
8
Parents is waiting on child process...
Child is working...
8
4
2
1
Child process is done.
Parent process is done.
abid@ubuntu:~/HelloWorld$