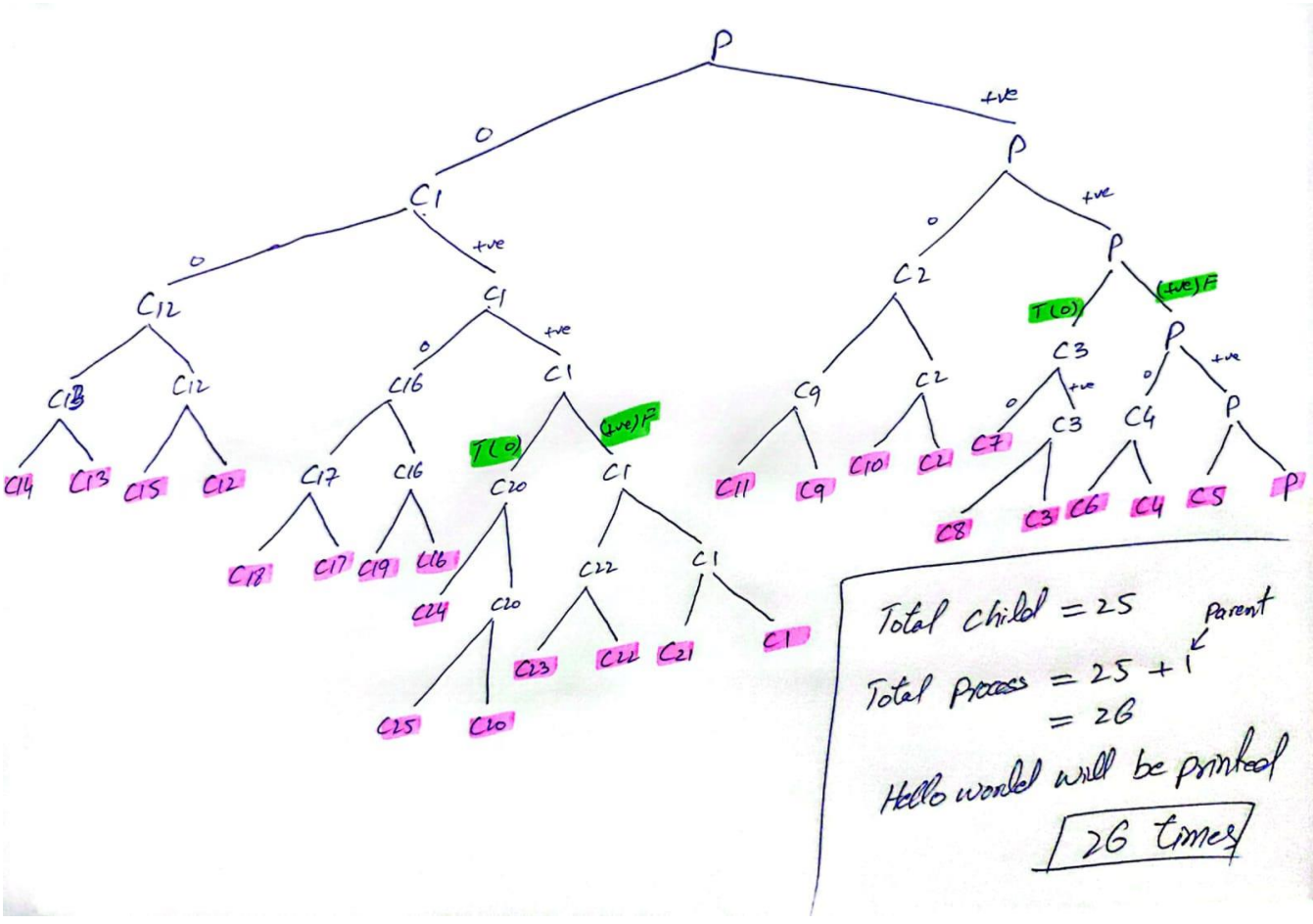
Solution

Quiz 1

Question 1: [5 Marks] Consider the following code:

```
if ((fork() || fork()) && fork() && (!fork()))
{
If(!(fork()&&fork()))
} else {
fork();
fork();
}
printf ("Hello World");
```

How many times "Hello World" will be printed on the screen? Also draw the Process Tree of the given program.



Question 2: [marks 5] Difference between short term, medium term and long term schedulers.

Long-Term Scheduler

The job scheduler is another name for Long-Term scheduler. It selects processes from the pool (or the secondary memory) and then maintains them in the primary memory's ready queue. The Multiprogramming degree is mostly controlled by the Long-Term Scheduler.

Short-Term Scheduler

CPU scheduler is another name for Short-Term scheduler. It chooses one job from the ready queue and then sends it to the CPU for processing. To determine which work will be dispatched for execution, a scheduling method is utilized.

Medium-Term Scheduler

Medium-term scheduler, also called swapper, which removes processes from memory and thus reduces the degree of multiprogramming. At some later time the process can be reintroduced at some later stage, this scheme is called swapping. The process is swapped out and is later swapped in by the medium-term scheduler. Swapping may be necessary to improve the job mix, or because a change in memory requirements has over-committed available memory, requiring memory to be freed up.

Question 2: [5 Marks] Show output of the code below. Write only one sequence if you feel that multiple sequences can be printed.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define BUFFER_SIZE 25

int main()
{
	char msg[BUFFER_SIZE] = "Welcome";
	pid_t pid = fork();

	if (pid > 0)
	{
		strcpy(msg, "Welcome to OS course");
		printf("Parent process waiting for child termination \n");
		wait(NULL);
		printf("Parent Terminating \n");
	}
	else
	{
```

```
                printf("Message: %s \n", msg);
                pid_t pid1 = fork();

                strcpy(msg, "OS course");
                pid_t pid2 = fork();

                if (pid2 == 0)
                {
                        strcpy(msg, "Adv OS course");
                        printf("Child Process called \n");
                }
                else
                {
                        wait(NULL);
                        printf("Message: %s \n", msg);
                }

                if (pid1 > 0)
                {
                        wait(NULL);
                }
        }

        return 0;
}
```

Output:

Parent process waiting for child termination
Message: Welcome
Child Process called
Message: OS course
Child Process called
Message: OS course
Parent Terminating