LAB SHEET

BATCH 03

fork ( ) System call

1.
```c
#include <stdio.h>
#include <unistd.h>
int main () {
    printf("I am Parent\n");

    fork (); // Creates a child process

    printf("Hello World...!\n");

    return 0;
}
```

2.
```c
#include <stdio.h>
#include <unistd.h>
int main() {
    int ret;

    printf("I am Parent\n");

    ret = fork(); // Creates a child process

    printf("Return Value: %d\n", ret);

    return 0;
}
```

3.

```c
#include <stdio.h>
#include <unistd.h>
int main() {
    int ret;

    printf("Hello World\n");

    ret = fork(); // Creates a child process

    if (ret == 0) {
        // Code executed by the child process
        printf("I am Child and Return Value=%d\n", ret);
    } else {
        // Code executed by the parent process
        printf("I am Parent and Return Value=%d\n", ret);
    }

    return 0;
}
```

4.

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    int ret;

    printf("Hello World\n");

    ret = fork(); // Creates a child process

    if (ret == 0) {
        // Code executed by the child process
        printf("I am Child and Return Value=%d\n", ret);
        printf("Child PID: %d\n", getpid());
        printf("Child's Parent PID: %d\n", getppid());
    } else {
        // Code executed by the parent process
        printf("I am Parent and Return Value=%d\n", ret);
        printf("Parent PID: %d\n", getpid());
    }

    sleep(20); // Delay the termination of the program for 20 seconds

    return 0;
}
```

execl ( ) system call

5.

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
printf("Here comes the date. \n");
execl("/bin/date", "date", 0); /*0 means end-of-arguments */
printf("That was the date. \n");
}
```

6.

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("Here comes the date. \n");

    execl("/bin/date", "date", (char *)0);

    // The following code will only be executed if execl fails
    perror("execl"); // Print an error message
    printf("That was the date. \n");

    return 0;
}
```

7.

```c
#include <stdio.h>
#include <unistd.h>
int main()
{ printf("Here comes the date. \n");
fork();
execl("/bin/date", "date", 0);
printf("That was the date. \n");
}
```

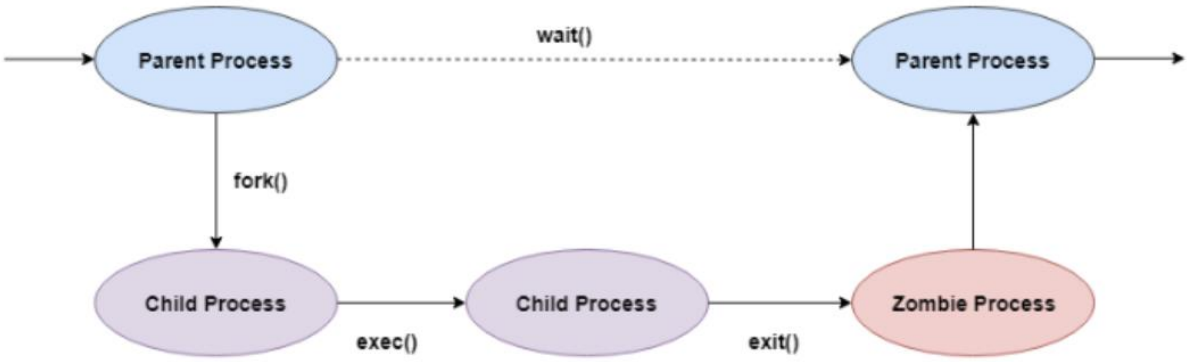
8.

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

I. Why did you get date two times? And why didn't you get first print statement two times? Calculate the number of times hello is printed.

Zombie Process

A zombie process is one that has finished its execution but remains in the process table. Typically seen with child processes, a parent process needs to read the child's exit status, and after using the wait system call, the zombie process is removed from the process table.



Zombie Process in Linux

A zombie is a process which has terminated but its entry still exists in the process table until the parent terminates normally or calls wait(). Suppose you create a child process and the child finishes before the parent process does. If you run the *ps* command before the parent gets terminated the output of *ps* will show the entry of a zombie process. This happens because the child is no longer active but its exit code needs to be stored in case the parent subsequently calls *wait*.

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

int main()
{
int id;
if ((id = fork())== 0)
{
printf("I am child process\n");
}
else
{
        while(1)
        sleep (100);
}
}
```

II. How to avoid creating a zombie process?

Orphan Process

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    int id;

    if ((id = fork()) == 0) {
        // Child process
        printf("I am the child process\n");
        sleep(10); // Sleep for 10 seconds
    } else {
        // Parent process
        printf("I am the parent process\n");
    }

    // The child process may still be running when the parent exits,
    // creating an orphan process if it is not re-parented.

    return 0;
}
```

III. How to avoid creating an orphan process?

CPU Time Slicing

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    int i = 0, j = 0, pid;

    pid = fork();

    if (pid == 0) {
        // Child process
        for (i = 0; i < 500000; i++)
            printf("Child: %d\n", i);
    } else {
        // Parent process
        for (j = 0; j < 500000; j++)
            printf("Parent: %d\n", j);
    }

    return 0;
}
```