Final Project Report

ALI USTUNKOL

COMP 5329: Advanced Operating Systems

North American University

Summer II, 2020

Mahbubur Rahman Ph.D.

Introduction

I would like to explain two system calls. Firstly, I will explain fork and give some detail information. Many multitasking operating systems need to create a new process, like run another program. In process for different program, the fork creates a copy of itself, it called the “child process.” If we take a look detail “child process”, the fork creates a different address space for a child. The child process has exact copy of all memory segments of parent process. When a transaction calls a fork, it is considered a parent transaction, and the newly created transaction is a child. After the fork, both processes do not run the same program, but at the same time, both continue to execute as if they called the system call. They can then examine the “return value” of the call and act accordingly to determine their status, their children, or their parents. Let’s take a look detail of C code.

1. FORK:

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

int main(void)
{
    pid_t pid = fork();

    if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0) {
        printf("Hello from Ali Ustunkol the child process!\n");
        _exit(EXIT_SUCCESS);
    }
    else {
```

```
        int status;
        (void)waitpid(pid, &status, 0);
    }
    return EXIT_SUCCESS;
}
```

```
pid_t pid = fork();
```

This first statement is main call the fork calls to split execution two processes. The recorded in a variable of type pid_t, which is the POSIX type for process identifiers (PIDs).

```
if (pid == -1) {
        perror ("fork failed");
        exit(EXIT_FAILURE);
    }
```

Minus one shows that an error in fork: there is no new process created, so the error message is printed.

```
else if (pid == 0) {
      printf("Hello from Ali Ustunkol the child process!\n");
      _exit(EXIT_SUCCESS);
   }
```

In the child process, if the value comes as zero. The child process prints the desired message, then exit.

```
else {
      int status;
      (void)waitpid(pid, &status, 0);
   }
```

Other process is that; the parent receives from fork the process identifier of the child, which is always a conclusive number. The parent process passes this attribute to the waitpid system call to suspend execution until the child has exited.

Analysis:

I discuss here about the fork code from eclipse. I upload the fork code to eclipse and found some results. I took screenshots for more detail information.

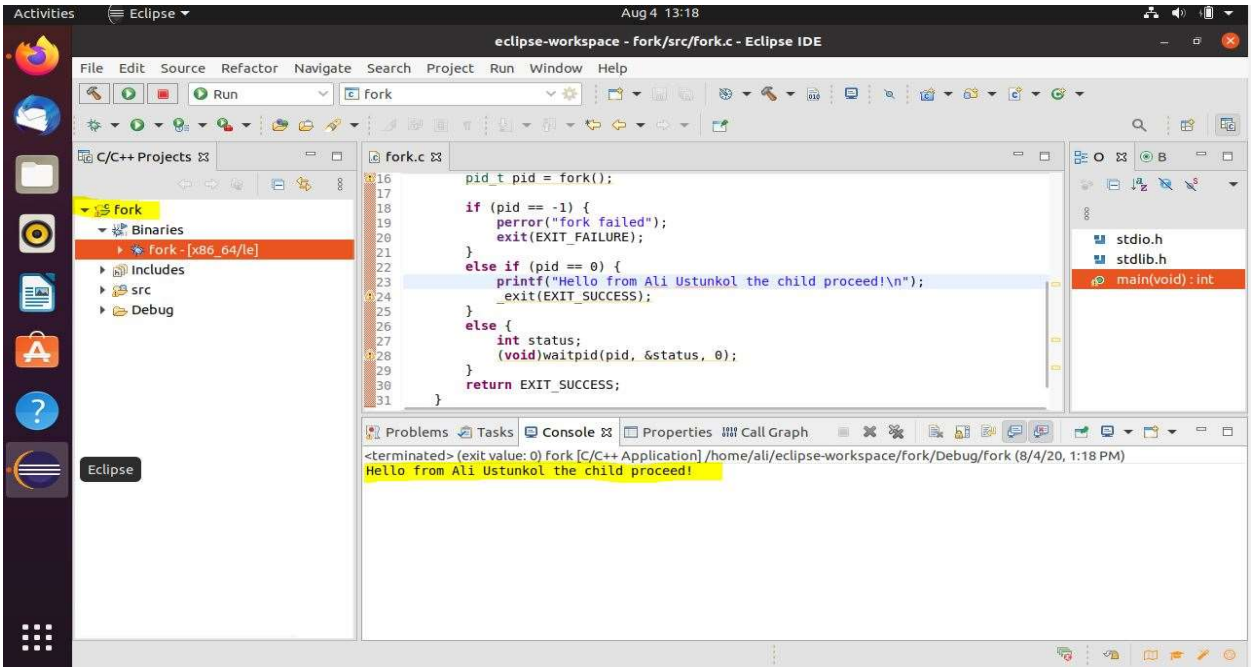

Figure:1

The screenshot shows that I upload the fork code then compile and run the code. The value is zero, therefore it gave a desired message. [Figure:1]

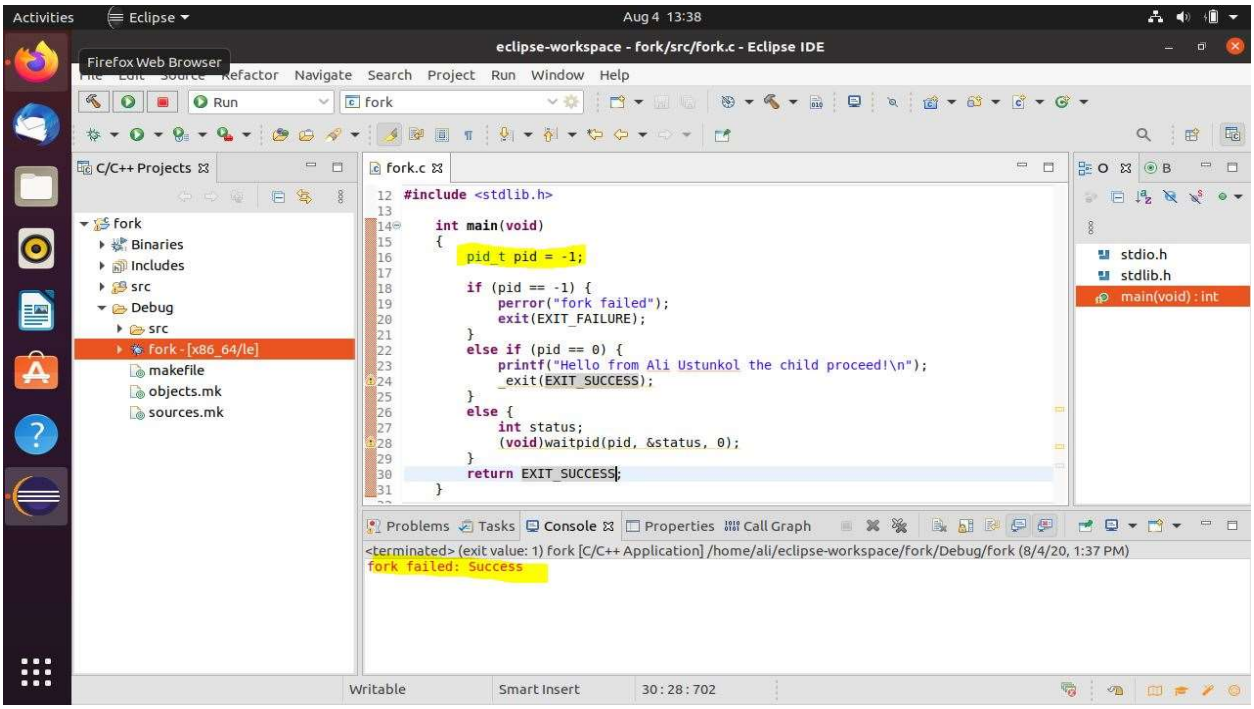

Figure:2

I run the same fork code, but I changed the fork variable (-1) and it gave different message like a fork failed: Success. [Figure:2]

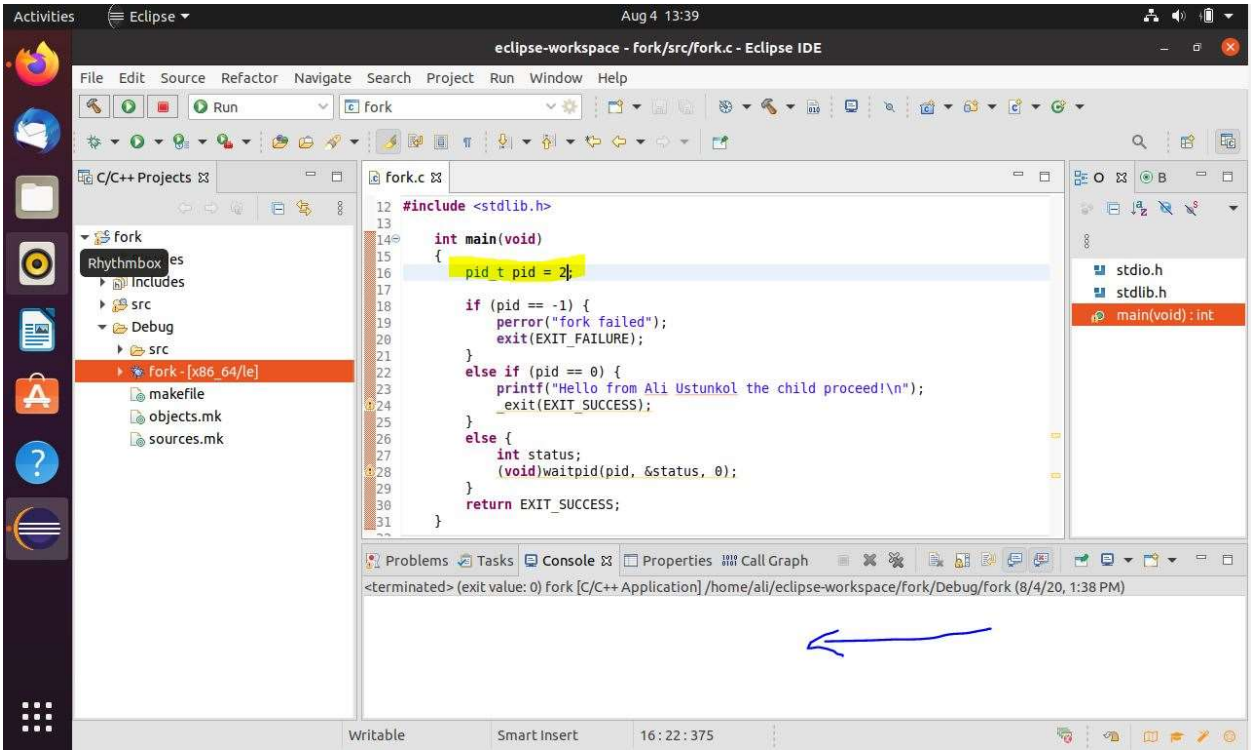

Figure:3

As you can see I gave a real number (2) then it didn't give me any message, because the waitpid system call suspended execution. This process will continue until child has exited. [Figure:3]

Introduction:

I would like to explain my second code is "System Call." In this code, we will call the "getcpu()" call directly our system for a "syscall()." The first part is SYS_ and the second part is the name of the system call. I attached the screenshot from my eclipse.

2. System Call

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

int main() {

    unsigned cpu, node;

    // Get current CPU core and NUMA node via system call
    // Note this has no glibc wrapper so we must call it directly
    syscall(SYS_getcpu, &cpu, &node, NULL);

    // Display information
    printf("This program is running on CPU core %u and NUMA node
%u.\n\n", cpu, node);

    return 0;

}
```

Analysis:

As you can see the code it checks system cpu and print the result. I upload code to eclipse and took screenshot. It prints the result. If you want to get different result or more result, we can run couple times the code.

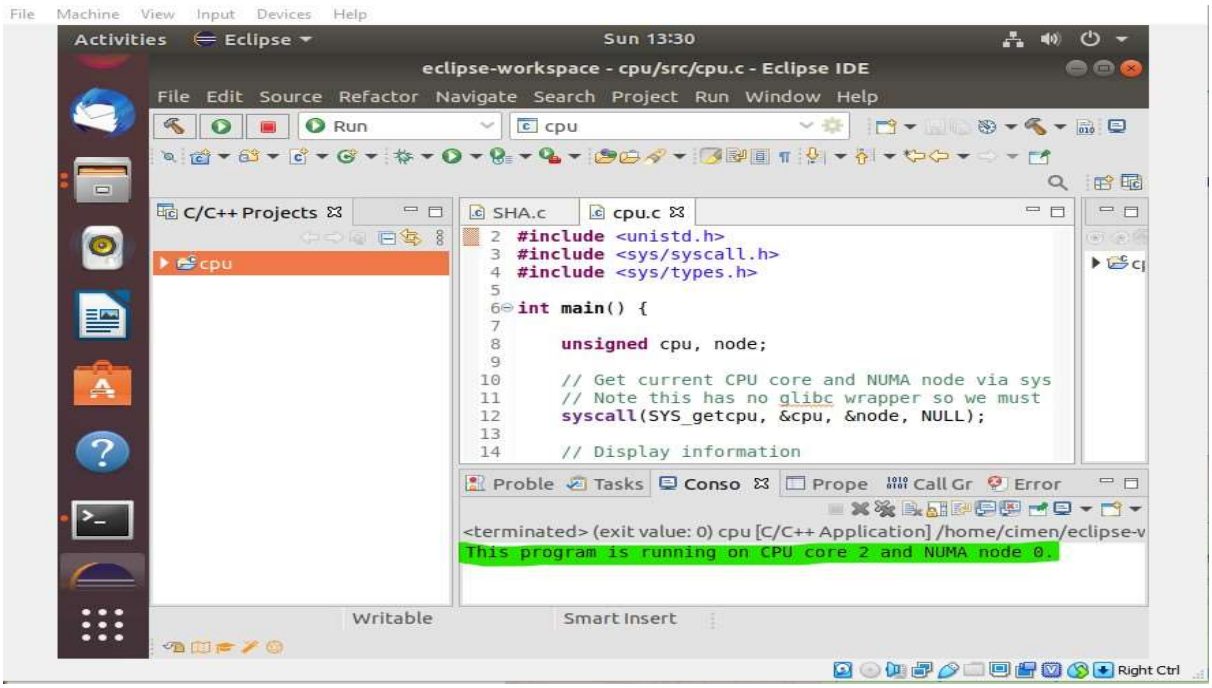

Figure: 4

As you can see, The code checks system's cpu. For more different results, you can spin threads via pthreads library and then call this function to see on which processor is running. [Figure:4]

References

Nyman, Linus (25 August 2016). "Notes on the History of Fork and Join". IEEE Annals of the History of Computing

https://en.wikipedia.org/wiki/Fork_(system_call)

Oliver R. (n.d.) Linux System Call Tutorial with C - 2019

https://linuxhint.com/linux_system_call_tutorial_c/