COL331 Operating System : Assignment - 4

Param Khakhar - 2018CS10362

24 March 2021

Introduction

The following is a writeup for the Assignment-4, Operating System (COL331). There are two sections corresponding to part A and part B, and each section contains the answers of the respective tasks in the path.

Part A

Task - 1: Output of t1

After printing "Executing t1", the program terminates with the string printed "Execution of 't1' complete".

Task - 2: MACROS Description

There are generic MACROS defined depending on the number of arguments which are called in the orginal syscall depending on the number of arguments. SYSCALL0 only takes the syscall number as the argument, SYSCALL1 takes the syscall number along with other argument, SYSCALL2 takes syscall number and two other arguments, and SYSCALL3 takes syscall number and three other arguements. All the arguments are pushed onto the stack with the syscall number at the top, along with the other arguments from top to bottom, in the same order in which the syscall is called.

Task - 3: Waiting Function

The function

```
int process_wait (tid_t child_tid);
```

is where the kernel waits for a process to terminate. On using an infinite loop in the function the following message is printed:

"System calls not implemented"

Task - 4:

The testcases for the seek, halt, filesize, and remove are added in the file test_syscalls.c in the lib directory.

Part B

Task - 1: Semaphore Function Interface

Consider two threads A and B, both using a shared sempahore in order to achieve mutual exclusion. Without loss of generality, assume that the initial value of the semaphore is 1. If thread A, first calls, sema_down(), then the value of the semaphore would then be decremented by 1, ultimately to 0. The other thread B, wouldn't complete the execution of the function sema_down(), and would go to sleep. Thread B would sleep until thread A calls sema_up(), which would increment the value of the semaphore back to 1, and also remove thread B from the waiting queue.

Therefore, execution of sema_down() followed by sema_up() by the same thread, would imply entry to the mutual exclusion region for other competing threads.