COP 4610
Homework Assignment 3

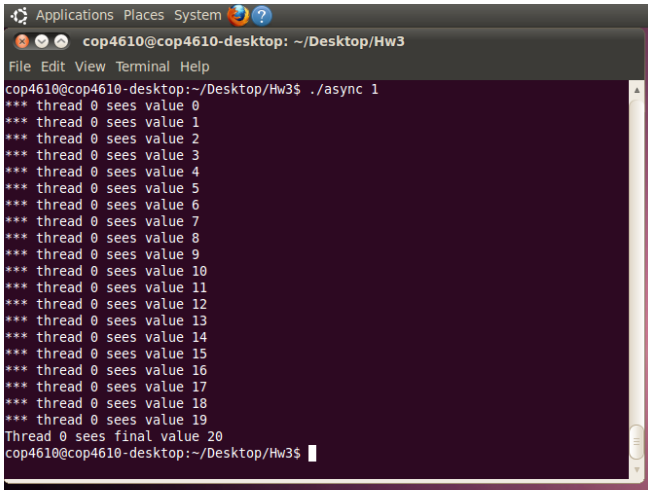
Alejandro Vidal
PID 5913959

FIU
Fall 2017

Part 1: Simple Multithreaded Programming

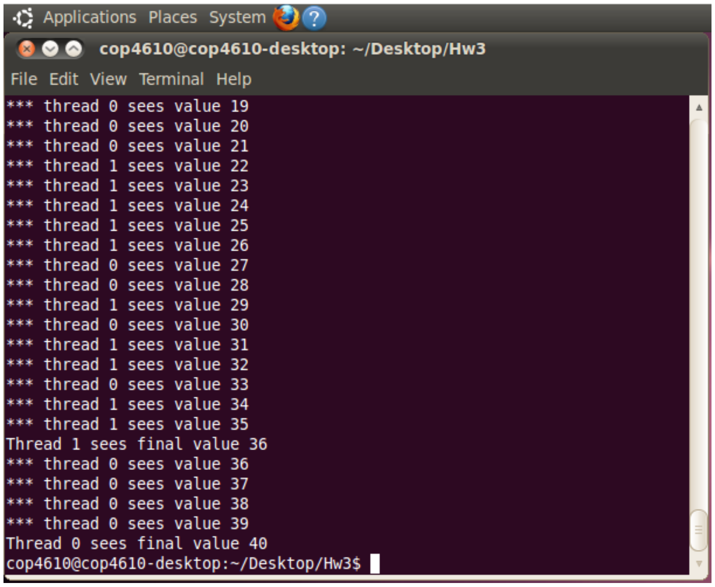
1.1) Simple Multithreaded Programming without Synchronization

When running the simple thread program passing 1 as a parameter it can be seen that the shared variable is modified successively by the created thread. In this case, there's no concurrency issue since there's only one thread modifying the global variable. The result of running the program with one thread can be seen below.

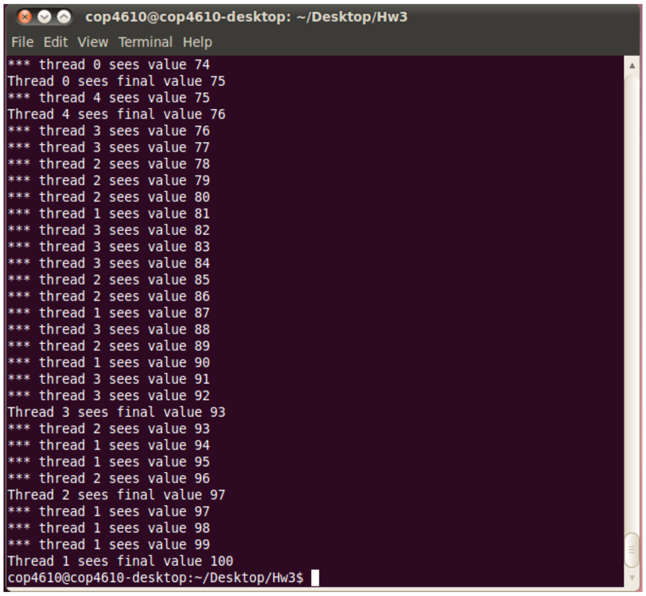

Applications Places System
cop4610@cop4610-desktop: ~/Desktop/Hw3
File Edit View Terminal Help

```
cop4610@cop4610-desktop:~/Desktop/Hw3$ ./async 1
*** thread 0 sees value 0
*** thread 0 sees value 1
*** thread 0 sees value 2
*** thread 0 sees value 3
*** thread 0 sees value 4
*** thread 0 sees value 5
*** thread 0 sees value 6
*** thread 0 sees value 7
*** thread 0 sees value 8
*** thread 0 sees value 9
*** thread 0 sees value 10
*** thread 0 sees value 11
*** thread 0 sees value 12
*** thread 0 sees value 13
*** thread 0 sees value 14
*** thread 0 sees value 15
*** thread 0 sees value 16
*** thread 0 sees value 17
*** thread 0 sees value 18
*** thread 0 sees value 19
Thread 0 sees final value 20
cop4610@cop4610-desktop:~/Desktop/Hw3$
```

However, when running the program with a number of threads greater than 1, issues arise. Since the created threads are accessing the shared variable without any type of concurrency control, the may access and modify the shared variable concurrently, causing all the threads to have different final values for the shared variable. This happened when running the program with 2, 3,4, and 5 threads. The results of running the program with 2 and 5 threads can be seen below.


Applications Places System
cop4610@cop4610-desktop: ~/Desktop/Hw3
File Edit View Terminal Help

```
*** thread 0 sees value 19
*** thread 0 sees value 20
*** thread 0 sees value 21
*** thread 1 sees value 22
*** thread 1 sees value 23
*** thread 1 sees value 24
*** thread 1 sees value 25
*** thread 1 sees value 26
*** thread 0 sees value 27
*** thread 0 sees value 28
*** thread 1 sees value 29
*** thread 0 sees value 30
*** thread 1 sees value 31
*** thread 1 sees value 32
*** thread 0 sees value 33
*** thread 1 sees value 34
*** thread 1 sees value 35
Thread 1 sees final value 36
*** thread 0 sees value 36
*** thread 0 sees value 37
*** thread 0 sees value 38
*** thread 0 sees value 39
Thread 0 sees final value 40
cop4610@cop4610-desktop:~/Desktop/Hw3$
```


cop4610@cop4610-desktop: ~/Desktop/Hw3
File Edit View Terminal Help

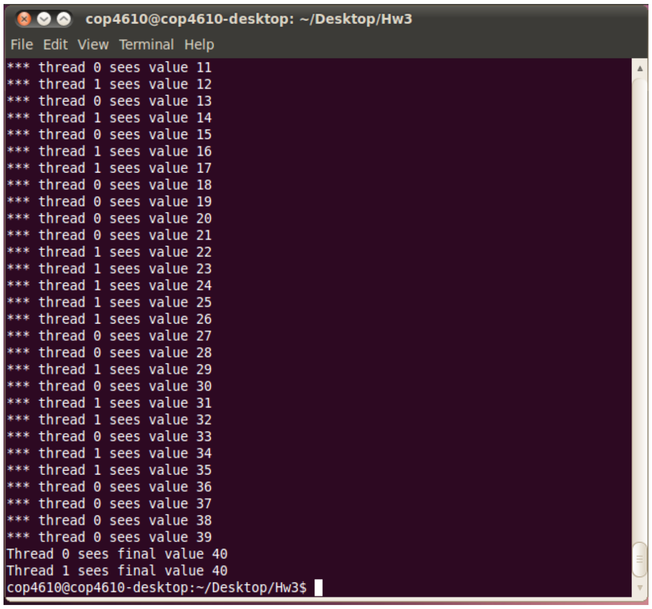
```
*** thread 0 sees value 74
Thread 0 sees final value 75
*** thread 4 sees value 75
Thread 4 sees final value 76
*** thread 3 sees value 76
*** thread 3 sees value 77
*** thread 2 sees value 78
*** thread 2 sees value 79
*** thread 2 sees value 80
*** thread 1 sees value 81
*** thread 3 sees value 82
*** thread 3 sees value 83
*** thread 3 sees value 84
*** thread 2 sees value 85
*** thread 2 sees value 86
*** thread 1 sees value 87
*** thread 3 sees value 88
*** thread 2 sees value 89
*** thread 1 sees value 90
*** thread 3 sees value 91
*** thread 3 sees value 92
Thread 3 sees final value 93
*** thread 2 sees value 93
*** thread 1 sees value 94
*** thread 1 sees value 95
*** thread 2 sees value 96
Thread 2 sees final value 97
*** thread 1 sees value 97
*** thread 1 sees value 98
*** thread 1 sees value 99
Thread 1 sees final value 100
cop4610@cop4610-desktop:~/Desktop/Hw3$
```

(2 Asynchronous Threads)

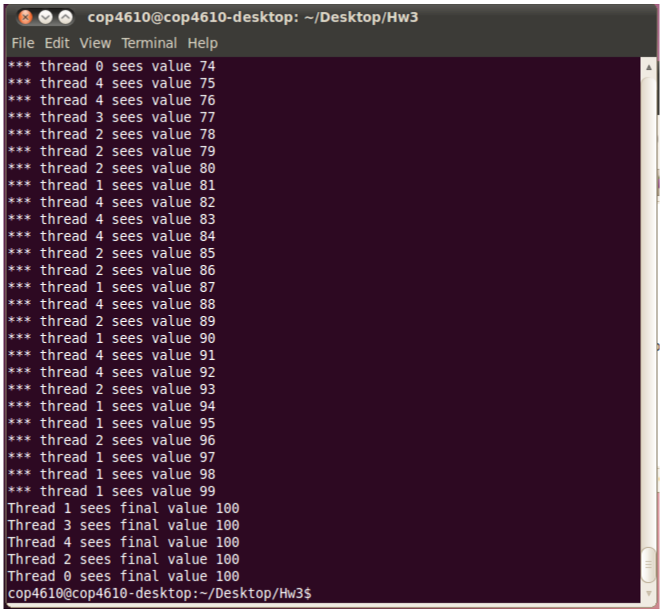
(5 Asynchronous Threads)

1.2) Simple Threads Programming with Proper Synchronization

When adding a mutex variable to the program in task 1.1, the results are different. The mutex allows for synchronized access to shared data, and using one every time the shared variable is going to be modified guarantees that each iteration of the loop increments the shared variable by one, and that all threads see the same final value for the shared variable. Each thread, tries to acquire a lock every time it's going to modify the shared variable, if some other thread has already acquired the lock, it waits. This was verified by running the program with 1, 2, 3, 4, and 5 threads. Results of running the program with 2, and 5 are shown below.


cop4610@cop4610-desktop: ~/Desktop/Hw3
File Edit View Terminal Help

```
*** thread 0 sees value 11
*** thread 1 sees value 12
*** thread 0 sees value 13
*** thread 1 sees value 14
*** thread 0 sees value 15
*** thread 1 sees value 16
*** thread 1 sees value 17
*** thread 0 sees value 18
*** thread 0 sees value 19
*** thread 0 sees value 20
*** thread 0 sees value 21
*** thread 1 sees value 22
*** thread 1 sees value 23
*** thread 1 sees value 24
*** thread 1 sees value 25
*** thread 1 sees value 26
*** thread 0 sees value 27
*** thread 0 sees value 28
*** thread 1 sees value 29
*** thread 0 sees value 30
*** thread 1 sees value 31
*** thread 1 sees value 32
*** thread 0 sees value 33
*** thread 1 sees value 34
*** thread 1 sees value 35
*** thread 0 sees value 36
*** thread 0 sees value 37
*** thread 0 sees value 38
*** thread 0 sees value 39
Thread 0 sees final value 40
Thread 1 sees final value 40
cop4610@cop4610-desktop:~/Desktop/Hw3$
```


cop4610@cop4610-desktop: ~/Desktop/Hw3
File Edit View Terminal Help

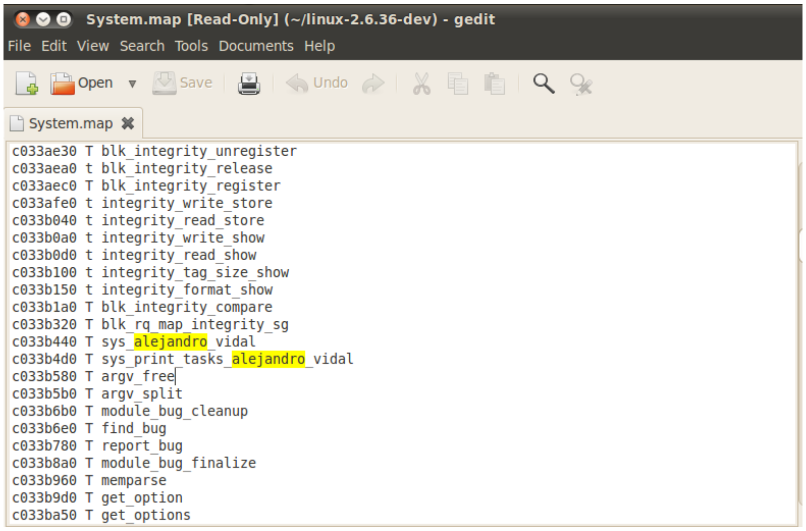
```
*** thread 0 sees value 74
*** thread 4 sees value 75
*** thread 4 sees value 76
*** thread 3 sees value 77
*** thread 2 sees value 78
*** thread 2 sees value 79
*** thread 2 sees value 80
*** thread 1 sees value 81
*** thread 4 sees value 82
*** thread 4 sees value 83
*** thread 4 sees value 84
*** thread 2 sees value 85
*** thread 2 sees value 86
*** thread 1 sees value 87
*** thread 4 sees value 88
*** thread 2 sees value 89
*** thread 1 sees value 90
*** thread 4 sees value 91
*** thread 4 sees value 92
*** thread 2 sees value 93
*** thread 1 sees value 94
*** thread 1 sees value 95
*** thread 2 sees value 96
*** thread 1 sees value 97
*** thread 1 sees value 98
*** thread 1 sees value 99
Thread 1 sees final value 100
Thread 3 sees final value 100
Thread 4 sees final value 100
Thread 2 sees final value 100
Thread 0 sees final value 100
cop4610@cop4610-desktop:~/Desktop/Hw3$
```

(2 Synchronic Threads)

(5 Synchronic Threads)

Part 2: Inspect Tasks in Kernel

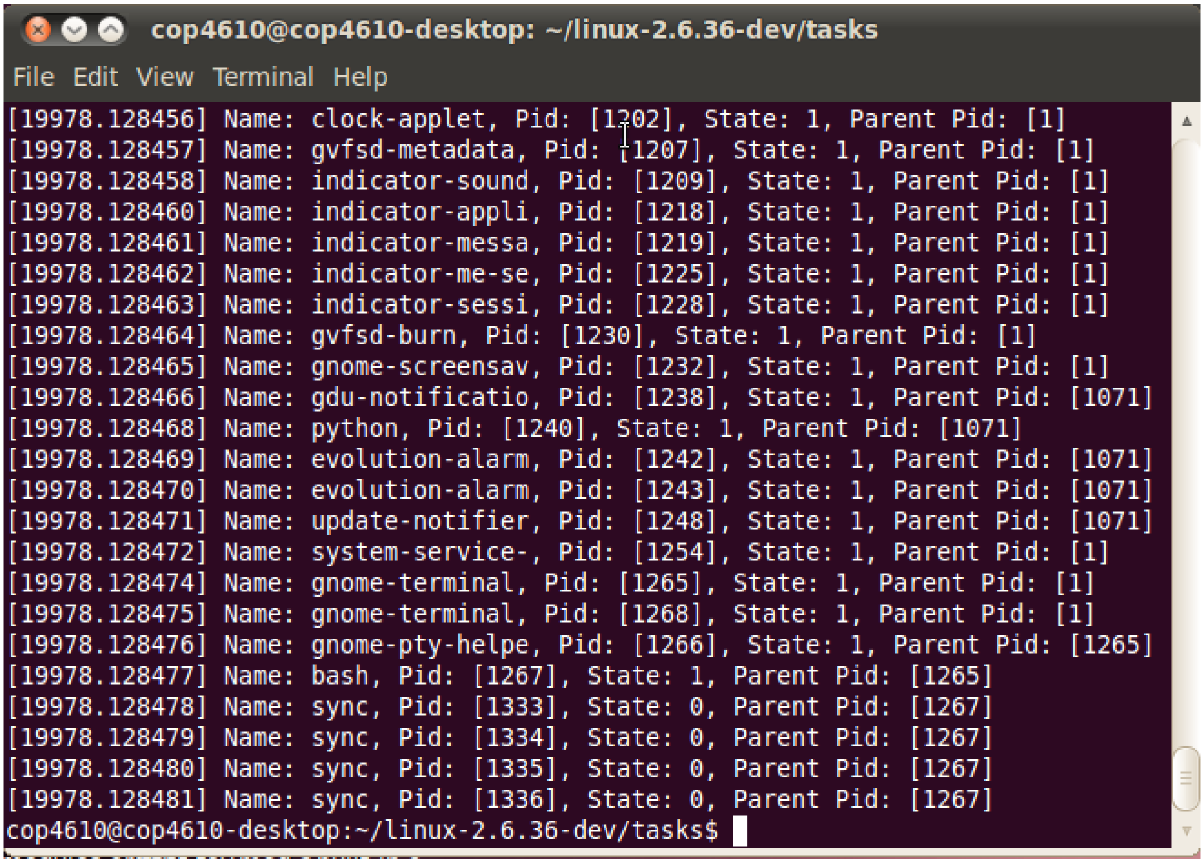
Now I created a new system call (pint_tasks_alejandro_vidal) and add it to the kernel. We can see it in the system.map file along with the previous assignment system call.


System.map [Read-Only] (~/linux-2.6.36-dev) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
System.map

```
c033ae30 T blk_integrity_unregister
c033aea0 t blk_integrity_release
c033aec0 T blk_integrity_register
c033afe0 t integrity_write_store
c033b040 t integrity_read_store
c033b0a0 t integrity_write_show
c033b0d0 t integrity_read_show
c033b100 t integrity_tag_size_show
c033b150 t integrity_format_show
c033b1a0 T blk_integrity_compare
c033b320 T blk_rq_map_integrity_sg
c033b440 T sys_alejandro_vidal
c033b4d0 T sys_print_tasks_alejandro_vidal
c033b580 T argv_free
c033b5b0 T argv_split
c033b6b0 T module_bug_cleanup
c033b6e0 T find_bug
c033b780 T report_bug
c033b8a0 T module_bug_finalize
c033b960 T memparse
c033b9d0 T get_option
c033ba50 T get_options
```

I created a new testmycall.c program to test my new system call. At user level the output of this test program is the same as the one used in Part 1.2 of this assignment including the returning status of the system call.

We can use **dmesg** to see all the information regarding the processes and the threads.



cop4610@cop4610-desktop: ~/linux-2.6.36-dev/tasks
File Edit View Terminal Help

```
[19978.128456] Name: clock-applet, Pid: [1202], State: 1, Parent Pid: [1]
[19978.128457] Name: gvfsd-metadata, Pid: [1207], State: 1, Parent Pid: [1]
[19978.128458] Name: indicator-sound, Pid: [1209], State: 1, Parent Pid: [1]
[19978.128460] Name: indicator-appli, Pid: [1218], State: 1, Parent Pid: [1]
[19978.128461] Name: indicator-messa, Pid: [1219], State: 1, Parent Pid: [1]
[19978.128462] Name: indicator-me-se, Pid: [1225], State: 1, Parent Pid: [1]
[19978.128463] Name: indicator-sessi, Pid: [1228], State: 1, Parent Pid: [1]
[19978.128464] Name: gvfsd-burn, Pid: [1230], State: 1, Parent Pid: [1]
[19978.128465] Name: gnome-screensav, Pid: [1232], State: 1, Parent Pid: [1]
[19978.128466] Name: gdu-notificatio, Pid: [1238], State: 1, Parent Pid: [1071]
[19978.128468] Name: python, Pid: [1240], State: 1, Parent Pid: [1071]
[19978.128469] Name: evolution-alarm, Pid: [1242], State: 1, Parent Pid: [1071]
[19978.128470] Name: evolution-alarm, Pid: [1243], State: 1, Parent Pid: [1071]
[19978.128471] Name: update-notifier, Pid: [1248], State: 1, Parent Pid: [1071]
[19978.128472] Name: system-service-, Pid: [1254], State: 1, Parent Pid: [1]
[19978.128474] Name: gnome-terminal, Pid: [1265], State: 1, Parent Pid: [1]
[19978.128475] Name: gnome-terminal, Pid: [1268], State: 1, Parent Pid: [1]
[19978.128476] Name: gnome-pty-helpe, Pid: [1266], State: 1, Parent Pid: [1265]
[19978.128477] Name: bash, Pid: [1267], State: 1, Parent Pid: [1265]
[19978.128478] Name: sync, Pid: [1333], State: 0, Parent Pid: [1267]
[19978.128479] Name: sync, Pid: [1334], State: 0, Parent Pid: [1267]
[19978.128480] Name: sync, Pid: [1335], State: 0, Parent Pid: [1267]
[19978.128481] Name: sync, Pid: [1336], State: 0, Parent Pid: [1267]
cop4610@cop4610-desktop:~/linux-2.6.36-dev/tasks$
```

We can see at the end, our **sync** process and all the threads information required.