Project 1: Multithreaded Programming and Synchronization

Rowland Sanders

Quan Do

CECS 326 - 04

Prof. Hailu Xu

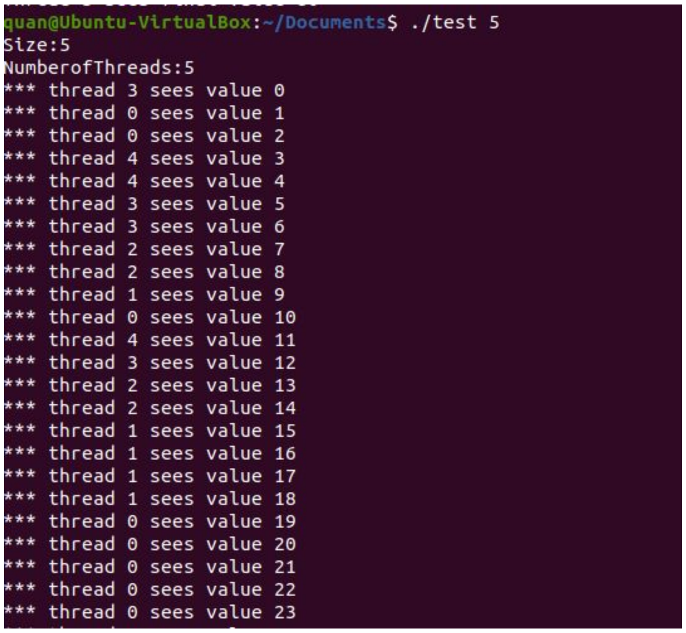
02/21/2021

Abstract

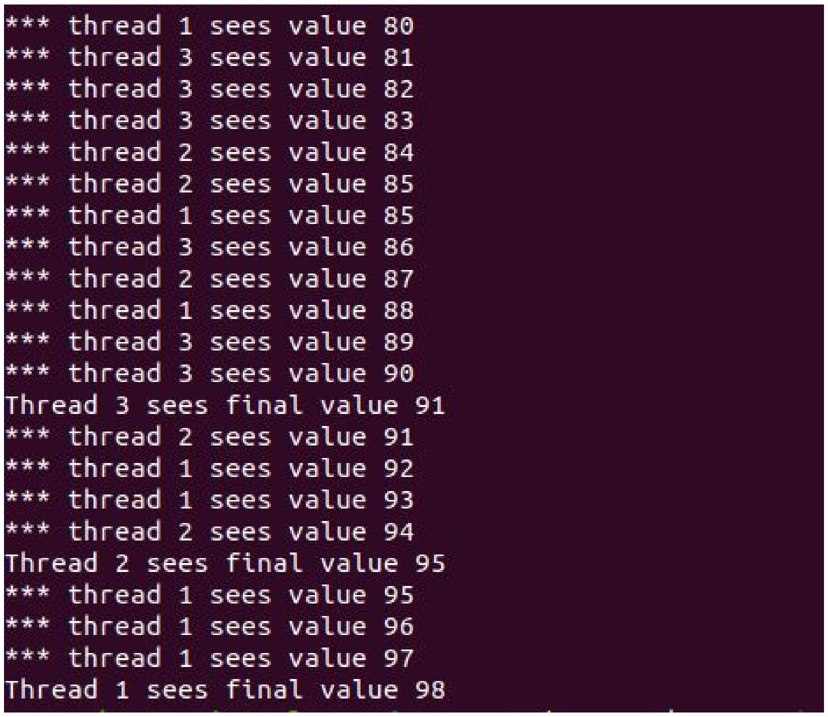
In our first project we are trying to display the importance of Process Management and virtualization of multi-threaded programming. The goal of our program was to reproduce a multi-threaded environment using a Linux virtual machine. We did this using the function Pthreads, this is used to recreate multi-threaded situations and execute them within the same program. POSIX thread or “Pthread” refers to a standard API and allows us to create unique and concurrent process flows. In the projects there were two parts one with synchronization and one without synchronization, Process synchronization provides a time slice for each process so that they can get the required time for execution. We had to create this simulation in linux, compile and then execute our code and record the output results. This allowed us to see the threads being created and the corresponding values.

Results

Without 5 threads synchronization

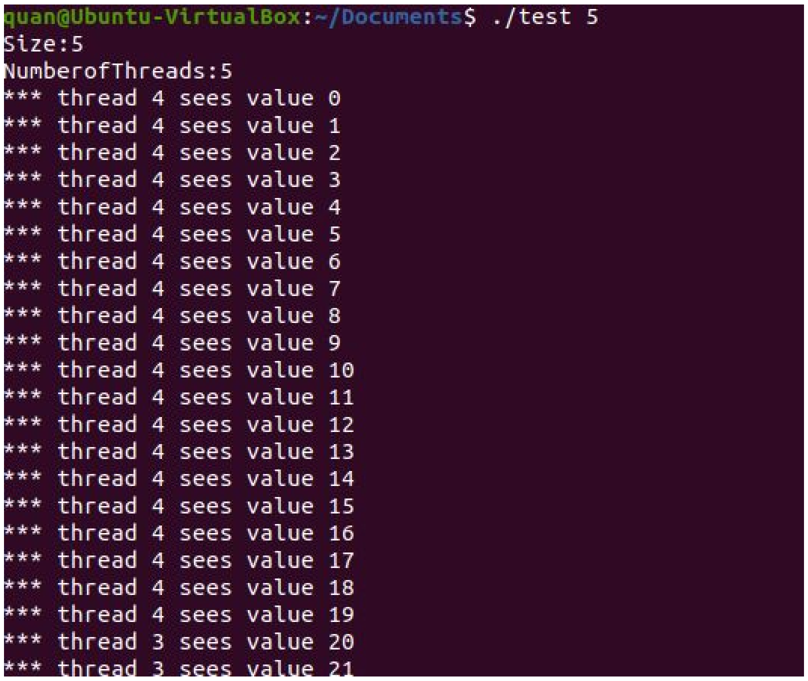


```
quan@Ubuntu-VirtualBox:~/Documents$ ./test 5
Size:5
NumberofThreads:5
*** thread 3 sees value 0
*** thread 0 sees value 1
*** thread 0 sees value 2
*** thread 4 sees value 3
*** thread 4 sees value 4
*** thread 3 sees value 5
*** thread 3 sees value 6
*** thread 2 sees value 7
*** thread 2 sees value 8
*** thread 1 sees value 9
*** thread 0 sees value 10
*** thread 4 sees value 11
*** thread 3 sees value 12
*** thread 2 sees value 13
*** thread 2 sees value 14
*** thread 1 sees value 15
*** thread 1 sees value 16
*** thread 1 sees value 17
*** thread 1 sees value 18
*** thread 0 sees value 19
*** thread 0 sees value 20
*** thread 0 sees value 21
*** thread 0 sees value 22
*** thread 0 sees value 23
```

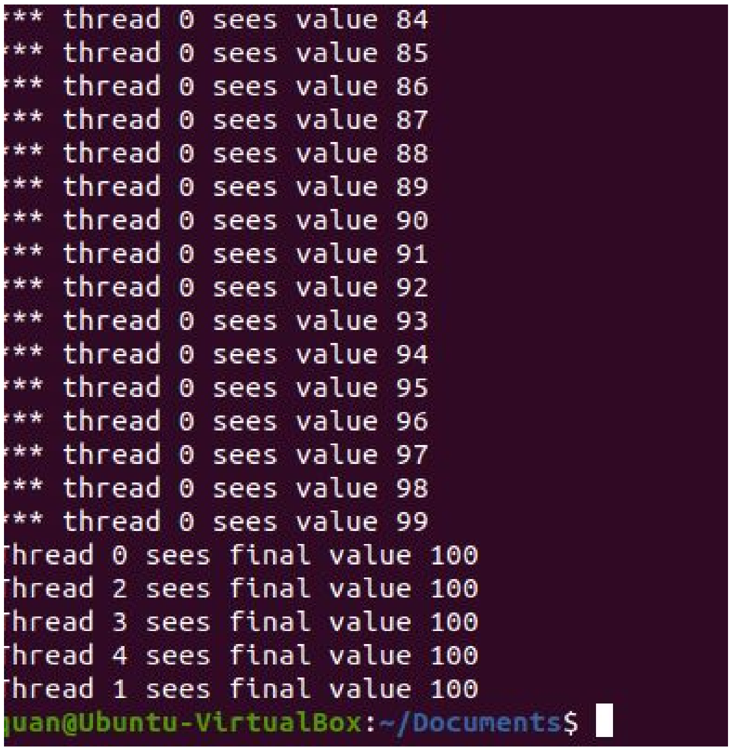


```
*** thread 1 sees value 80
*** thread 3 sees value 81
*** thread 3 sees value 82
*** thread 3 sees value 83
*** thread 2 sees value 84
*** thread 2 sees value 85
*** thread 1 sees value 85
*** thread 3 sees value 86
*** thread 2 sees value 87
*** thread 1 sees value 88
*** thread 3 sees value 89
*** thread 3 sees value 90
Thread 3 sees final value 91
*** thread 2 sees value 91
*** thread 1 sees value 92
*** thread 1 sees value 93
*** thread 2 sees value 94
Thread 2 sees final value 95
*** thread 1 sees value 95
*** thread 1 sees value 96
*** thread 1 sees value 97
Thread 1 sees final value 98
```

With 5 threads synchronization



```
quan@Ubuntu-VirtualBox:~/Documents$ ./test 5
Size:5
NumberofThreads:5
*** thread 4 sees value 0
*** thread 4 sees value 1
*** thread 4 sees value 2
*** thread 4 sees value 3
*** thread 4 sees value 4
*** thread 4 sees value 5
*** thread 4 sees value 6
*** thread 4 sees value 7
*** thread 4 sees value 8
*** thread 4 sees value 9
*** thread 4 sees value 10
*** thread 4 sees value 11
*** thread 4 sees value 12
*** thread 4 sees value 13
*** thread 4 sees value 14
*** thread 4 sees value 15
*** thread 4 sees value 16
*** thread 4 sees value 17
*** thread 4 sees value 18
*** thread 4 sees value 19
*** thread 3 sees value 20
*** thread 3 sees value 21
```



```
** thread 0 sees value 84
** thread 0 sees value 85
** thread 0 sees value 86
** thread 0 sees value 87
** thread 0 sees value 88
** thread 0 sees value 89
** thread 0 sees value 90
** thread 0 sees value 91
** thread 0 sees value 92
** thread 0 sees value 93
** thread 0 sees value 94
** thread 0 sees value 95
** thread 0 sees value 96
** thread 0 sees value 97
** thread 0 sees value 98
** thread 0 sees value 99
hread 0 sees final value 100
hread 2 sees final value 100
hread 3 sees final value 100
hread 4 sees final value 100
hread 1 sees final value 100
uan@Ubuntu-VirtualBox:~/Documents$
```

Analysis

As shown by the screenshots when no synchronization is involved between threads, the values for each thread becomes inconsistent. After inputting 1, 2, 3, 4, and 5 threads into the program we noticed that the use of an increasing number of threads meant that the inconsistency between threads also increased, and that the final value would not be the same for all threads with more than 1 thread involved. We solved this in the synchronized solution by adding mutexes and barriers, within #ifdef and #endif statements, to our original program. The barrier allows the threads to finish executing before starting a process on a new thread, and mutex lock makes sure that two or more threads execute at the same time. Synchronization of the program is only allowed if the “-D Flag”, which is called PTHREAD_SYNC, is written in the command line otherwise code that is written within #ifdef and #endif statements will be passed over without executing.

Contributions

Quan Do

- Wrote parts of main program
- Rewrote #ifdef and #endif statements to execute with -D Flag command
- Debugged Makefile
- Wrote parts of lab report

Rowland Sanders

- Wrote parts of main program
- Debugged part 1
- Wrote Makefile and ReadMe
- Wrote parts of lab report