Lab4: Introduction to Threads

Grading Form

Goal In this lab you will be introduced to system calls, thread programming and to the need for mutual-exclusion when more than one thread is used.

Part 0: System calls

There is a procedure called syscall() that is used to generate system calls (or software interrupts) directly in the OS to request system services.

int syscall(int syscallNumber, arguments);

The system-call numbers for the different system-services are found in ***/usr/include/sys/syscall.h.***

1. Write the following program ***hello-syscall.c*** compile it and run it.

```
#include<sys/syscall.h>
#include<stdio.h>
#include<string.h>

void
main()
{

     char * hello_with_syscall = "Hello World with syscall\n";
     char * hello_without_syscall = "Hello World without syscall\n";
     char * hello_with_printf = "Hello World with printf\n";

     write( 1, hello_without_syscall, strlen( hello_without_syscall ));

     syscall( SYS_write, 1, hello_with_syscall, strlen( hello_with_syscall ));

     printf( "%s", hello_with_printf );

}
```

2. Using strace see the different system calls that ***hello-syscall*** generates.

To output all the system calls:

```
strace hello-syscall
```

To send that to an output file ***t.out***

```
strace -o t.out hello-syscall
```

To count how many system calls are used to compile this program:

```
strace -o t2.out -c cc -o hello-syscall hello-syscall.c
```

Copy the last 10 lines of t.out into a file strace.out and write

```
^^^^ write ^^^^^
^^^^ syscall ^^^^
^^^printf ^^^
```

under the lines of strace that correspond to each of the calls. Copy the file into the lab4-src/ directory.

Part 1: Thread Creation

1. Copy the file lab4-src.tar.gz to your directory, uncompress it and untar it.

```
gunzip lab4-src.tar.gz
tar -xvf lab4-src.tar
```

2. Examine the contents of the files thr1.cc.

3. Change directoy to lab4-src and build the executables:

```
cd lab4-src
make
```

Run the program thr1 several times to verify that the output is compatible with your expectations.

4. Now modify the thr1.cc program to have two more threads: one printing "D" and the other one printing "E".

5. Examine the program thr2.cc and the run it. Explain why the thr2 program is having this output even though two threads are created.

Add a README file with a textual description of the output you obtained in step 3 and the explanation of step 5, also you will turnin your modified thr1.cc from step 4,

Part 2. Mutual Exclusion

1. Examine the file **count.cc** that you have already built in part 1.

2. Run **count** several times and see that the final count sometimes (or always) is wrong.

3. Add a mutex lock to count.cc to make sure that the final count will be always correct. See the manual pages for pthread_mutex_init(), pthread_mutex_lock(), and pthread_mutex_unlock(). Run your modified program several times to make sure that you always get the correct final count.

Turn in your modified count.cc.

Part 3. Spin locks

1. Examine the contents of the file count_spin.cc

2. Run count_spin several times and see that the final count is wrong as in part 2.

3. In the body of the procedures my_spin_lock() and my_spin_unlock() implement the spin locks as covered in class. Use the procedure pthread_yield() to yield the execution of the CPU. Add to the procedure increment() the calls to my_spin_lock() and my_spin_unlock(). Rebuild and try count_spin again. Make sure that the final count is now correct.

4. Measure the time of count, and count_spin with and without pthread_yield() and verify the the total count is still correct. Fill in the following table. Run each program 5 times and write in the table the run where the real time has been the minimum. Use the time command.

	System (Kernel) Time	User Time	Real Time
pthread_mutex (count)			
spin lock (count_spin with thr_yield)			
spin_lock (count_spin without thr_yield)			

Write this table in the README file and turn it in with the other files. Also add to the README file the answers to the following questions:

1. Explain the differences in user time between **count_spin** with and without thr_yield.

2. Explain the difference in system time between **count** and **count_spin** with thr_yield.

Part 4. Deadlocks

See the code int deadlock.c and run it:

```
./deadlock
```

You will see that the program will hang due to a deadlock. See the code and fix it so it does not hang. Turnin your fixed deadlock.c code.

Turning in

Place the README file in the lab4-src directory and turn it in with the command:

```
turnin -c cs252 -p lab4 lab4-src

Run this command from data.
```

This lab is due Monday October 24th, at 11:59pm.