DUE DATE: **By the end of of your lab session**

CSE-381: Systems 2

Exercise #7

Max Points: 20

Name: Noah Dunn

Objective: The objective of this exercise is to:
1. Understand programs that use C++ threads
2. Observe and control threads externally via Linux tools.

Submission: Save this MS-Word document using the naming convention `MUid.docx` (example `raodm.docx`) before proceeding with this exercise.

Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from the terminal/output window into this document. You may discuss the questions with your instructor.

Part #0: Introduction to threads

Estimated time to complete: 25 minutes

Now wait for your instructor to introduce threads by covering the first 15 slides in the lecture slides titled `09_Threads.pdf` on Canvas.

The concepts and code are involved. So, ensure you pay attention to it so that you can complete this exercise successfully.

Part #1: Understand a simple multi-threaded C++ program [10 points]

Estimated time to complete: 15 minutes

Background: Using threads in C++ is very simple -- any function/method can be called/run as a thread. This makes running threads in C++ very straightforward using the `pthreads` library. However, the `pthreads` library should be explicitly linked into the program. Luckily, the `NetBean's` Miami University C++ project already links in `pthreads` library for us.

Exercise: The objective of this exercise is to understand the operation of a straightforward multithreaded C++ program.

1. Create a `NetBeans` project called `Timer` (name main file also `Timer`) -- ensure you use `Miami University C++ Project`. Download the supplied `Timer.cpp` program and copy it to your `NetBeans` project. Compile the program and ensure it compiles successfully.

2. Study the program – the program creates a few threads and all they do is sleep -- that is they don't do any computations. One of the threads counts down a timer and the program ends when the timer expires.

3. From the program answer the following questions (**in general, these are the level of detail you should be thinking about in every starter code**):
   a. Which method in the program is used to start threads? How (describe in English) did you figure it out?

      The std::thread() method inside the main method is used to start threads. I read the comments, and they indicated that the thread method was used to create, and then start a thread.

   b. Where/how are arguments passed to the thread method (copy-paste line(s) of code below)?

      ```
              for (int i = 0; (i < thrCount); i++) {
                      // Create a thread and add it to list of threads.
                      thrList.push_back(std::thread(timer, maxTime, i));
        }
      ```

   c. If the timer method was modified to accept references as in, `void timer(int& maxTime, int& threadID)`, illustrate the line of code to be used to pass parameters:

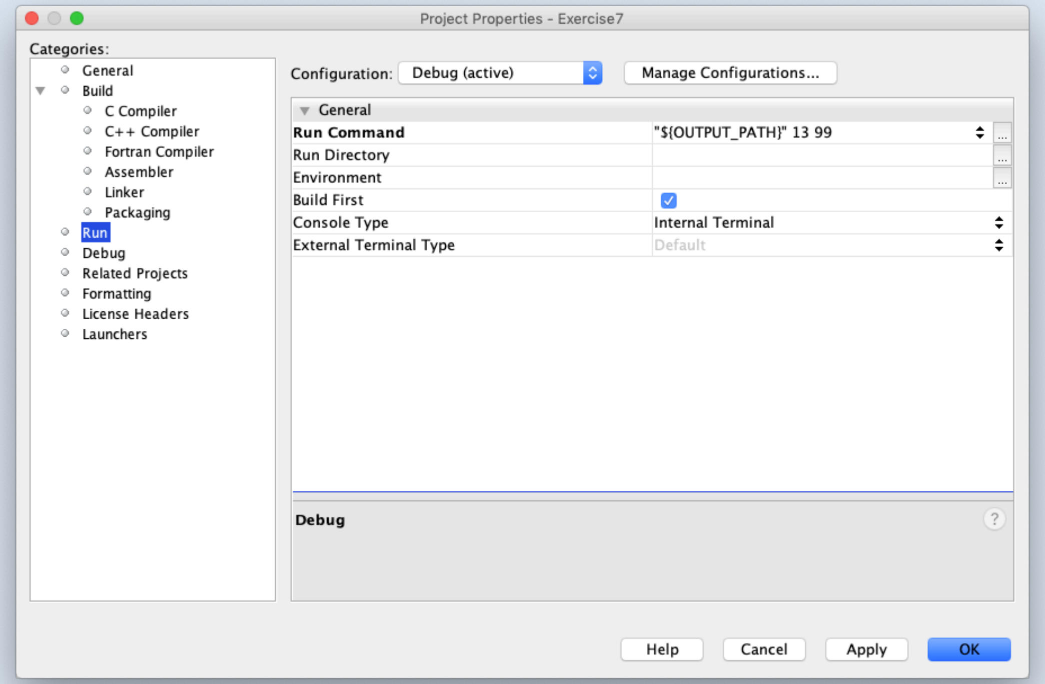
      We need to use the std::ref() method to pass references correctly in a thread.
      thrList.push_back(std::thread(timer, std::ref(maxTime), std::ref(i)));

   d. Show the command to use to run the program with 13 threads for 99 seconds. What are these values 13 and 99 to be passed to the program called?

      ./Timer 13 99
      13 and 99 are referred to as Command line arguments

   e. How should the above arguments (i.e., 13 and 99) be set in NetBeans in order to pass it onto this program? Show a screenshot below illustrating how to set it in NetBeans -- if you are not sure how to do it see video on `Canvas→Video Demonstrations` page

      g


Project Properties - Exercise7

Categories: General, Build (C Compiler, C++ Compiler, Fortran Compiler, Assembler, Linker, Packaging), Run, Debug, Related Projects, Formatting, License Headers, Launchers

Configuration: Debug (active)

General	
Run Command	"${OUTPUT_PATH}" 13 99
Run Directory	
Environment	
Build First	✓
Console Type	Internal Terminal
External Terminal Type	Default

f. From `NetBean's` compile output, copy-paste the command printed by `NetBeans` that links in `pthread` library. Note this line of command has `-l` (lower-case ell) towards the end, where `-l` stands for "link-in the library".

g++ -fsanitize=address -DGNUCXX_DEBUG -o Exercise7 build/Debug/GNU-Linux/Timer.o -lboost_system -lpthread -lmysqlpp

g. From the above command what are the other two libraries that are being linked into the program?

Boost_System library and MySQLPP

Part #2: Observe behaviors of multi-threaded programs [10 points]

Estimated time to complete: 30 minutes

Background: In most modern operating systems, including Linux, threads are implemented natively in the operating system. The operating system is aware of multiple threads being used by a process. Furthermore, the OS manages multiple threads by suitably scheduling them and handling resources used by various threads. Most operating systems also provide the ability to run foreground or joinable threads and background or detached threads.

In Linux, there is very little distinction between processes and threads. In fact, the OS uses a common `clone` (read manual page on `clone` for details) system call to create processes and threads using slightly different parameters to the `clone` system call. Consequently, in Linux threads are called Light Weight Processes or LWP and similar to a regular process, each LWP is assigned a `tid` (thread ID, similar to `pid`) gets an entry in the virtual `/proc/` file system. The `/proc` file system in Linux is a direct memory map to the kernel's internal data structures. The entries `/proc/` file system can be used to obtain additional information about the operation of various threads.

Exercise: The objective of this exercise is to explore the support provided by Linux for monitoring and controlling threads.

1. This part of the exercise is to be conducted using the C++ `Timer.cpp` program.

2. Open two separate terminal windows to the Linux server used for this course -- you will run the `Timer` (C++ executable) in one terminal and observe its behaviors in another terminal.

3. Compile the program via NetBeans. Run the program from a terminal window. The program will print the process ID (pid) for the process and then start counting down a simple timer.

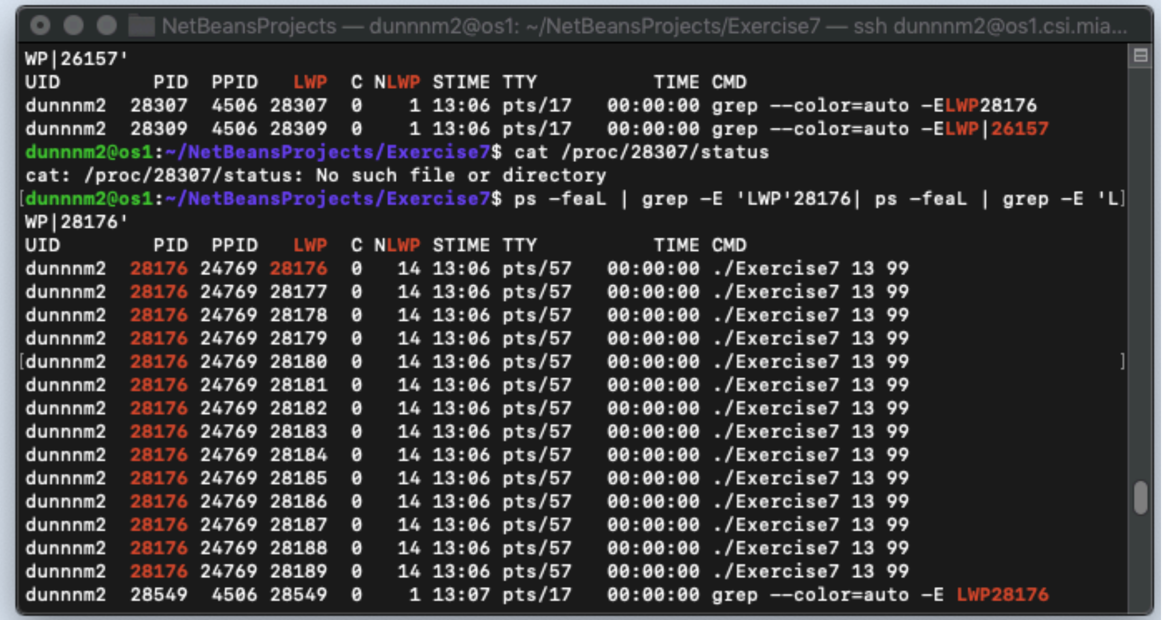
4. Using the process ID (**`pid`**) printed by the program, view the status information about the overall process by inspecting the `/proc/`***`pid`***`/status` file -- e.g. `/proc/`**`2018`**`/status` (if you still haven't figured out how to view files in Linux, remember to use the `less` command: `less /proc/`**`pid`**`/status`). From the output copy-paste the following information about the main process:

What is the process ID (`pid`)	25153
What is the parent process ID (`ppid`):	24769
Number of threads in process (`Threads`):	14 (13 generated plus the starter one)

Notice the discrepancy in the number of threads reported by the operating system versus the number of threads the program claims to have created. Why is there a discrepancy?
We had an original thread used to create every other thread.

5. Now, use the `ps` command to list the `pid` and `tid` (thread ID) of the process and its LWP (aka threads) using the following command (where <pid> is the number reported by the timer program) and copy-paste it into the space further below (the `grep` command searches & prints lines with word `LWP` or `<pid>`):

```
$ ps -feaL | grep -E 'LWP|<pid>'
```



```
NetBeansProjects — dunnnm2@os1: ~/NetBeansProjects/Exercise7 — ssh dunnnm2@os1.csi.mia...
WP|26157'
UID         PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
dunnnm2  28307  4506 28307  0    1 13:06 pts/17   00:00:00 grep --color=auto -ELWP28176
dunnnm2  28309  4506 28309  0    1 13:06 pts/17   00:00:00 grep --color=auto -ELWP|26157
dunnnm2@os1:~/NetBeansProjects/Exercise7$ cat /proc/28307/status
cat: /proc/28307/status: No such file or directory
dunnnm2@os1:~/NetBeansProjects/Exercise7$ ps -feaL | grep -E 'LWP'28176| ps -feaL | grep -E 'L
WP|28176'
UID         PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
dunnnm2  28176 24769 28176  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28177  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28178  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28179  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28180  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28181  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28182  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28183  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28184  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28185  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28186  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28187  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28188  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28176 24769 28189  0   14 13:06 pts/57   00:00:00 ./Exercise7 13 99
dunnnm2  28549  4506 28549  0    1 13:07 pts/17   00:00:00 grep --color=auto -E LWP28176
```

6. Choose one LWP ID (fourth column in the above output) and view the status information about the LWP by inspecting the `/proc/<LWP>/status` file. From the output copy-paste the following information about the LWP-process:

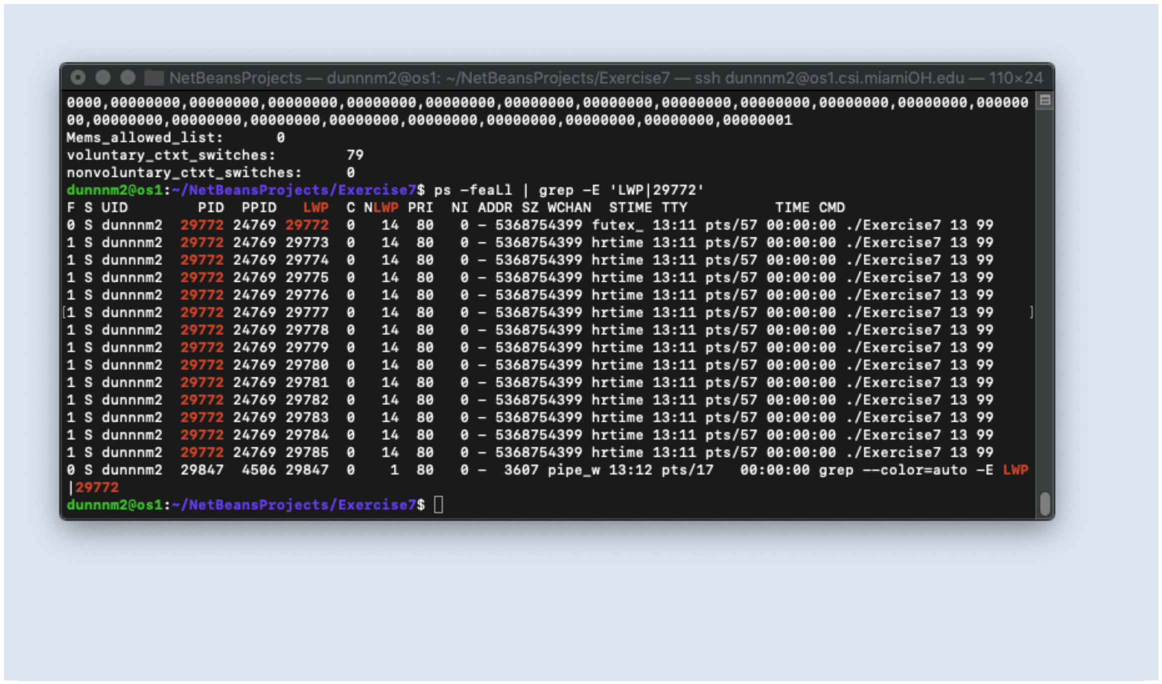
What is the process ID (`pid`)	28178
What is the parent process ID (`ppid`):	24796

7. **Priority experiment:** Each process in Linux has a priority in the range 0 to 99, with zero being highest priority and 99 being lowest priority. By default processes and their threads are assigned priority depending on user settings indicated by the `PRI` column. This priority value can be changed at a process level (and consequently for each thread) using a "niceness" value for each process. The niceness value generally ranges from -20 to +19, with -20 being the most favorable or highest priority for scheduling and 19 being the least favorable or lowest priority. The current scheduling priority for processes and associated niceness values can be viewed either using `ps` command or via the `top` command.

In this step we will record the priority and niceness values for the process and try to change it.

a. First record the current priority and priority value for the process and its threads using the following command (arguments include uppercase and lowercase 'L' letter):

```
$ ps -feaLl | grep -E 'LWP|<pid>'
```



```
NetBeansProjects — dunnnm2@os1: ~/NetBeansProjects/Exercise7 — ssh dunnnm2@os1.csi.miamiOH.edu — 110×24
0000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,000000
00,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        79
nonvoluntary_ctxt_switches:     0
dunnnm2@os1:~/NetBeansProjects/Exercise7$ ps -feaLl | grep -E 'LWP|29772'
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S dunnnm2  29772 24769 29772  0   14  80   0 - 5368754399 futex_ 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29773  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29774  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29775  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29776  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29777  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29778  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29779  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29780  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29781  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29782  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29783  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29784  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  29772 24769 29785  0   14  80   0 - 5368754399 hrtime 13:11 pts/57 00:00:00 ./Exercise7 13 99
0 S dunnnm2  29847  4506 29847  0    1  80   0 -  3607 pipe_w 13:12 pts/17   00:00:00 grep --color=auto -E LWP
|29772
dunnnm2@os1:~/NetBeansProjects/Exercise7$ 
```

b. Indicate the current priority and niceness value for all the threads in the space below.

The priority values for each of the threads in the process:	80

(priority value is the 9th column in the output and is typically 60 to 80)

The niceness value for the threads is :	0

(It is the 10th column of output and typically tends to be zero by default)

c. Now <u>decrease</u> (to increase priority you need to supply negative values) the priority for two of threads by increasing their niceness values to +5 and +10 respectively, using the command below (**two separate times, once for each thread**):

```
$ renice <niceValue>  <tid>
Example:
$ renice +5 6483
```

d. Next, record the new priority and niceness value for the process and its threads using the `ps` command (arguments include uppercase and lowercase ‘L’ letter):

```
$ ps -feaLl | grep -E 'LWP|<pid>'
```

```
0 S dunnnm2  30471 24769 30471  0   14  85   5 - 5368754399 futex_
13:14 pts/57 00:00:00 ./Exercise7 13 99
1 S dunnnm2  30471 24769 30472  0   14  90   10 - 5368754399 hrtime
13:14 pts/57 00:00:00 ./Exercise7 13 99
```

8. Finally, observe the life-cycle action of threads by trying to stop just one thread in the program. From the output above, identify the pid/tid for a thread. **Do not use the `pid` of the main process as reported by the `timer` program!** Kill the thread using the `kill` command (`kill <tid>`) and note your observation below:

Program was terminated and all threads were killed.

Part #3: Submit files

Once you have successfully completed the lab exercise, upload the following at the end of the lab exercise:

- This MS-Word document saved as a PDF file with the convention *`MUid`*`.docx`.

Upload each file individually to Canvas. Do not upload archive files such as zip/7zip/rar/tar/gz etc. Ensure you click the `Submit` button on Canvas once you have uploaded all the necessary files.