Exercise 1

Overview:

- Login to an LC cluster using your workshop username and OTP token
- Copy the exercise files to your home directory
- Familiarize yourself with LC's Pthreads environment
- Write a simple "Hello World" Pthreads program
- Successfully compile your program
- Successfully run your program - several different ways
- Review, compile, run and/or debug some related Pthreads programs (provided)

1. Login to the workshop machine

Workshops differ in how this is done. The instructor will go over this beforehand.

2. Copy the example files

In your home directory, create a pthreads subdirectory for the example codes, copy the example codes and then `cd` into your pthreads subdirectory:

```
mkdir pthreads
cp  /usr/global/docs/training/blaise/pthreads/*   ~/pthreads
cd  pthreads
```

3. List the contents of your pthreads subdirectory

You should notice a number of files as shown in the tables below.

File Name	Description
arrayloops.c arrayloops.f	Data decomposition by loop distribution. Fortran example only works under IBM AIX: see comments in source code for compilation instructions.
condvar.c	Condition variable example file. Similar to what was shown in the tutorial
detached.c	Demonstrates how to explicitly create pthreads in a detached state.
dotprod_mutex.c dotprod_serial.c	Mutex variable example using a dot product program. Both a serial and pthreads version of the code are available.
hello.c	Simple "Hello World" example
hello32.c	"Hello World" pthreads program demonstrating thread scheduling behavior.
hello_arg1.c	One correct way of passing the pthread_create() argument.
hello_arg2.c	Another correct method of passing the pthread_create() argument, this time using a structure to pass multiple arguments.

File Name	Description
join.c	Demonstrates how to explicitly create pthreads in a joinable state for portability purposes. Also shows how to use the pthread_exit status parameter.
mpithreads_serial.c mpithreads_threads.c mpithreads_mpi.c mpithreads_both.c mpithreads.makefile	A "series" of programs which demonstrate the progression for a serial dot product code to a hybrid MPI/pthreads implementation. Files include the serial version, pthreads version, MPI version, hybrid version and a makefile.

4. Compilers - What's Available?

- Visit the Compilers Currently Installed on LC Platforms webpage.
- Click on the name of the workshop Linux cluster (sierra) in the summary table near the top of the page. This will take you to a table of available compilers.
- You can also view the available compilers in the Compilers section of the Linux Clusters Overview tutorial.
- Now, in your cluster login window, try the `use -l compilers` command to display available compilers. You should see GNU, Intel and PGI compilers - several versions of each.
    - Question: Which version is the default version?
    - Answer: Use the `dpkg-defaults` command and look for the asterisk.

5. Create, compile and run a Pthreads "Hello world" program

- Using your favorite text editor (vi/vim, emacs, nedit, gedit, nano…) open a new file - call it whatever you'd like.
- Create a simple Pthreads program that does the following:
    - Includes the `pthread.h` header file
    - Main program creates several threads, each of which executes a "print hello" thread routine. The argument passed to that routine is their thread ID.
    - The thread's "print hello" routine accepts the thread ID argument and prints "hello world from thread #". Then it calls `pthread_exit` to finish.
    - Main program calls `pthread_exit` as the last thing it does If you need help, see the provided hello.c file.
- Using your choice of compiler (see above section 4), compile your hello world Pthreads program. This may take several attempts if there are any code errors. For example:

```
icc -pthread -o hello myhello.c
pgcc -lpthread -o hello myhello.c
gcc -pthread -o hello myhello.c
```

  When you get a clean compile, proceed.

- Run your `hello` executable and notice its output. Is it what you expected? As a comparison, you can compile and run the provided hello.c example program.
- **Notes**:
    - For the remainder of this exercise, you can use the compiler command of your choice unless indicated otherwise.
    - Compilers will differ in which warnings they issue, but all can be ignored for this exercise. Errors are different, of course.

6. Thread Scheduling

- Review the example code `hello32.c`. Note that it will create 32 threads. A `sleep();` statement has been introduced to help insure that all threads will be in existence at the same time. Also, each thread performs actual work to demonstrate how the OS scheduler behavior determines the order of thread completion.
- Compile and run the program. Notice the order in which thread output is displayed. Is it ever in the same order? How is this explained?

7. Argument Passing

- Review the `hello_arg1.c` and `hello_arg2.c` example codes. Notice how the single argument is passed and how to pass multiple arguments through a structure.
- Compile and run both programs, and observe output.
- Now review, compile and run the `bug3.c` program. What's wrong? How would you fix it? See the explanation in the bug programs table above.

8. Thread Exiting

- Review, compile (for gcc include the `-lm` flag) and run the `bug5.c` program.
- What happens? Why? How would you fix it?
- See the explanation in the bug programs table above.

9. Thread Joining

- Review, compile (for gcc include the `-lm` flag) and run the `join.c` program.
- Modify the program so that threads send back a different return code - you pick. Compile and run. Did it work?
- For comparison, review, compile (for gcc include the -lm flag) and run the `detached.c` example code.
- Observe the behavior and note there is no "join" in this example.

10. Stack Management

- Review, compile and run the bug2.c program.
- What happens? Why? How would you fix it?
- See the explanation in the bug programs table above. Also review and try out `bug2fix.c` program.