System Programming
3rd Laboratory (week of 12th March 2019)

Debugging Threads

I

Compile the **prog1.c** program (with the name **prog1**) and run it.

- This program receives as an argument the number of threads to be created.
- Each thread enters into an infinite loop.
- Kill the running program with Ctrl-C

DO NOT CORRECT THIS PROGRAM

To debug and modify the execution of each thread we will use the debugger (**ddd** or **gdb**).

- Recompile the program using the **-g** option:
  - **gcc prog1.c -pthread -g -o prog1**
- run gdb/ddd with the program as argument:
  - **gdb prog1**
- inside gdb execute the program with a numerical argument:
  - **run 3**
- All the threads will enter into an infinite loop….
- Stop the program pressing **Crt-C**
- List the available thread issuing the command:
  - **info threads**
- Observe this list and select one thread to debug
- Use the command **thread** to change to one of the threads. For instance:
  - **thread 2**
- From now on, the thread 2 will be controlled by the debugger:
  - up, down, step, next, set commands
- Issue the **step** command until the thread gets out of the sleep
- When the PC is on the while (n<10) line, change the value of n:
  - **set var n=20**
- Issue the command **continue** to resume execution.
  - Now a thread will terminate
- do the same procedures until all threads are terminated

Simple Threads

II

Correct the program form the previous exercise

III

Change the previous program so that all the created threads read characters from the keyboard and exit when the read character is **‘\n’**.
The main thread should execute the same code as all the other threads.
Each thread will count how many characters has read and will print this value before terminating.

What happens if the main thread reads a ‘k’ before all others?

Arguments into threads

IV

Change exercise II so that the main thread sends into each thread a different random integer (between 0 and 20). The thread will use the received value as the counter of the loop. Each thread will use the received value as counter for the loop.
Since the thread routine/function receives a pointer as argument, experiment three different ways to send an integer value into a thread:

- pthread_create(&thread_id, NULL, thread_func, value);
- pthread_create(&thread_id, NULL, thread_func, &value);
- pthread_create(&thread_id, NULL, thread_func, ptr);

Waiting for threads to dye

V

Change the exercise III so that the main thread does not read from the keyboard, but waits for the dead of every other thread, using the **pthread_join** system call.

Getting thread return values

VI

Change the previous exercise so that each thread, when terminating will return the character count to the main thread.
The main thread will receive this value and print it in the screen.
Experiment the following ways to get a value out of the thread:

- pthread_exit(count)
- pthread_exit(&count)
- pthread_exit(ptr)

System calls

- int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
  void *(*start_routine) (void *), void *arg);
- pthread_join
- void pthread_exit(void *retval);

Documentation

- Compiling for debugging
  - ftp://ftp.gnu.org/pub/old-gnu/Manuals/gdb/html_node/gdb_16.html
- Your program's arguments
  - ftp://ftp.gnu.org/pub/old-gnu/Manuals/gdb/html_node/gdb_18.html
- Debugging programs with multiple threads
  - ftp://ftp.gnu.org/pub/old-gnu/Manuals/gdb/html_node/gdb_24.html
- Assignment to variables
  - ftp://ftp.gnu.org/pub/old-gnu/Manuals/gdb/html_node/gdb_111.html
- Chapter 2 Basic Threads Programming
  - docs.oracle.com/cd/E19120-01/open.solaris/816-5137/tlib-12926/index.html
- Chapter 8 Compiling and Debugging
  - docs.oracle.com/cd/E19120-01/open.solaris/816-5137/compile-74765/index.html