Lab Session 02

Debugging in Linux

- Add **-g** to **gcc** command
- **gdb *executable_name*** starts the debugger
    - **break *function_name*** sets a breakpoint at the start of a function
    - **break *file_name*:*line_number*** sets a braekpoint at a specific line
    - **run** starts execution
    - **continue**
    - **info threads** prints information on threads
    - **print *expression*** prints an expression, can even be a varriable
    - **thread *thread_number*** switches to a specific thread
    - **quit**

Home exercises

1. **[10p]** Let there be ***N*** elements, numbered from 0 to N-1. You want to split these elements into groups. You know there are ***P*** groups and each group has an id ***T_id*** (numbered from 0 to P-1). What are the general formulas, based on ***N***, ***P*** and ***T_id***, that represent the first and last element in each group? Try to make all groups similar in size and consider the case when the division of some variables leave a remainder.

Lab Exercises

1. **[10p]** Compile and run the program from the resources.
2. **[10p]** Compile the program with debug symbols and use gdb to see the threads.
   - Compile with -g flag
   - Start in gdb
   - Set breakpoint inside the thread function
   - Let the program run to the breakpoint
   - Display threads
3. **[10p]** Write a program that uses all the CPU cores at 100% capacity and runs for a few seconds. You need to show that you use all the cores at max capacity by using **top**, **htop** or task manager.
   - Hint: You might have to nest some loops and do some intense math for this one, like a lot of sqrt()
   - Watch out -O3 might simply remove or replace the code if it dems it to be useless. If so, compile without -O3
4. **[10p]** Parallelize the addition of elements in two vectors. The number of threads **P**, the number of elements **N**, and **NReps** (number of times to add the vectors, just so the program doesn't end too fast) must be given as parameters to the program.
   - Do **NOT** remove the NReps loop, move it inside the thread with the N loop
   - Make sure you start P threads, and you do it only once.
5. **[10p]** Run the previous program with different number of threads to prove it scales.
   - Hint: You will have to run the program with all I/O disabled.
   - Hint: The execution time with one thread should be around 10 seconds, short times are not relevant. Set N and NReps accordingly.
   - Test first with 1 or 2 threads.
   - Watch out for -O3 flag as it can optimize so much the code will no longer scale.
6. **[10p]** Write a multithreaded program that has a concurrency problem. Let's say both threads execute a = a + 2.
   - You can place the problematic code inside a loop to make the errors appear more often
   - Watch out for the -O3 flag, as it can just replace this code with a constant.
7. **[10p]** Use the script in the resources to run the program enough times to show the concurrency issue, even with one iteration.
8. **[10p]** Using a mutex, solve the concurrency issue. Use the script again to prove the solution works for numerous runs (1000+).
9. **[10p]** Write a program that starts two threads. The first one prints the number 1, the second one prints the number 2. Using a barrier, make sure the output is always a 1 followed by a 2. Prove the correctness by using your script to run the program numerous times.