Name: Yukio Rivera
Date: 4/5/2022
Title: Lab 5 – Makefile
Description: Write up for Lab5

Explain what happens when you run this threadHello.c program; can you list how many threads are created and what values of i are passed?

When I run the threadHello.c program it prints a few different lines. I prints, “Hello from thread (int) with iteration (int)”, “Thread (int) returned”, and “Main thread done.” There are 10 threads created with the values of i being 0 - 9, provided by NTHREDS 10 in line 13, in my code. When you change NTHREADS it changes the number of threads that are created.

Do you get the same result if you run it multiple times? What if you are also running some other demanding processes (e.g., compiling a big program, playing a Flash game on a website, or watching streaming video) when you run this program?

The order of the threads changes but there are some common results, from the times that I ran it. I tried running the program while running youtube, a demanding video game, and compiling another program. I noticed that the more demand the other process, such as running an intense game, had on the computer the slower threadHello.c program would run. When I let the game fully load it took threadHello.c a while to print out the lines.

The function go() has the parameter arg passed to a local variable. Are these variables per-thread or shared state? Where does the compiler store these variables’ states?

The function go() variable seems to be shared state because they are running inside the same process/program that is determined once the program begins. While I was running the program multiple times I thought that the variable states would be stored in RAM since the thread would want to grab this information as quickly as possible.

The main() has local variable i. Is this variable per-thread or shared state? Where does the compiler store this variable?

I think that the local variable is stored per thread because its working in the same process but displaying different ints when creating different threads. I think the compiler stores this variable in RAM in order to access the int faster.

Write down your observations. You probably have seen that there is a bug in the program, where threads may print same values of i. Why?

I’m not sure if these statements change if you run it in extremely high numbers but from running it about x100 i noticed that “Hello from thread [int A] with iteration [int 2]” will usually print first with the value of [int 1] and [int 2] changing every time. The order that the iterations are printed also changes after every run. Usually after a few “hello from thread…” lines a “Thread [int] returned” would print out. More often than not the time all the “Hello from thread…” statements would print out first then the “thread [int] returned” would print out but every once and a while a few “Thread [int] returned” would print in between the longer print line. Then finally the last line “Main thread done.” would print out. I think this is a great example of "race condition", which occurs when a device or system attempts to perform two or more operations at the same time, but the operations must be done in the proper sequence to be done correctly. Learning about the race problem was interesting but watching it happen in real time and being able to fix it was a great experience.