Zhiwei Liu

**Architecture:**
The code is split into two different C files, ***simple_threading_timer*** and ***signal_interrupt_timer***. Both programs handle printing the timer information by calling another function from within the thread. The difference is that ***simple_threading_timer*** has the thread die and passes the information from the thread to a new one every second while ***signal_interrupt_timer*** simply uses an interrupt function on the thread every second.

**Memory Leaks:**
There are a total of three memory leaks according to valgrind. ***simple_threading_timer*** has two memory leaks, and ***signal_interrupt_timer*** has the last one of the three. For both programs, including ***timer_delete(timerId)*** rectifies one of ***simple_threading_timer***'s memory leaks and eliminates them completely from ***signal_interrupt_timer***. The last memory leak in ***simple_threading_timer*** appears to be a memory leak from within one of the libraries being used. As such, without access to the library's variables, the memory leak cannot be feasibly fixed.

**Time:**
Using the time utility ***simple_threading_timer*** appears to have a consistent system-time of around 0.004 to 0.005 seconds. The real-time and user-time all vary depending on how long the program is left running. For ***signal_interrupt_timer*** there doesn't seem to be any consistency in the system-time. Real-time is still just how long the program is left running. User-time however appears to be a consistent, 0 to 0.001 second time.

**Kbhit_sample:**
The code for ***simple_threading_timer*** is almost the exact same except the while loop now ends when the function ***kbhit()*** returns 1 and if you hit the enter/return key. ***kbhit()*** appears to be a key input handler, checking if the key inputted is not an End of File. The program seems to have the same memory leaks which can be fixed in the same way. The time utility reports that the system-time, user-time, and real-time are all at varying amounts.

**Spin Loop:**
The spin loop seems to make the user-time and the real-time align more closely.

**Change this code: while(getchar()!='\n'):**
When this code is changed to **while(getchar()=='\n')** in the regular timer files, the program starts to act strange. From testing it appears that the program requires the user to input a character, press enter/return, then press enter/return again to exit the program. When this code is changed in the **kbhit_sample** files, the program appears to run as usual.