Each thread within a process has its own:

1. Stack
2. Program Counter
3. Set of Registers
4. State (running, blocked, ready)

Need to implement:

1. int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine) (\*void), void \*arg);
   * Create new thread within a process
   * Store the ID of created thread in the location referenced by thread
   * attr shall always be NULL.
2. int pthread\_exit(void \*value\_ptr)
   * ignore value\_ptr
   * clean up all information relating to the terminating thread
   * Process will exit with status 0 after last thread has been terminated
3. Pthread\_t pthread\_self(void);
   * Return the thread ID of calling thread

Code should be compiled into an object file: threads.o. This will be my thread library. No main function, so no standalone executable can be made. Hence the object file.

Implementation:

* Need a data structure to hold information about a single thread: set of registers, a pointer to its stack area, status (running, ready to run, has exited). This is known as the Thread Control Block (TCB). **Struct probably?**
* Store each TCB as an element of an array. The array will contain all “active”(?) threads.

Assumptions

* The third party program will not create more than 128 threads

Concept of thread-safe: An example of something that isn’t thread-safe is the use of a static variable in a function. After the function has been called, the static does not lose its value. Instead, it remains the same on the next call of the function.