CS 5523 Project2B:
Implementing the processes-as-threads (180 points)

1 Overall Description:

You will implement a runtime library that will turn threads into processes, but users do not have to change their multithreaded programs that are linked to this library.
If you are using the processes, you can actually isolate the executions of different threads', which provides a lot of benefits. For example, we have implemented a software system to automatically tolerate false sharing problems. Isolation can avoid two threads to access the same cache line at the same time. Dthreads also uses this framework to achieve the deterministic multithreading, which is the basis of many on-going research in this field.
For your convenience, we have provided a simple code base for you at `https://github.com/UTSASRG/CS5523Project` (under processes-as-threads directory). The current code is a memory allocator so that you will use it to build your processes-as-threads framework. We also provide a `selfmap.h` for you so that you can use this to analyze the memory mapping of the application. You don't have to use these provided programs but they may make your implementation a little bit easier.
You can also refer to our open-sourced repo at `https://github.com/plasma-umass/dthreads` for any references to implement this framework. But you can not simply copy the code from there.

2 Implementation tips

2.1 Replacing threads with Processes

You will have to intercept pthread_create() functions and replaced them with clone system calls. Example can be found at `https://github.com/plasma-umass/dthreads/blob/master/src/source/xthread.cpp`.
You can also pass the starting address for each process so that you can specify the address of stacks for these processes. For instance, you could make multiple processes share their stacks, in order to achieve the same semantics as multithreading. However, this step is not required for the project.

2.2 Shared Memory Semantics

You should make all processes share globals (library and threads), stacks, and heap. Thus, you will achieve the similar semantics as multithreaded programs. To achieve this, you can map a file-backed mapping before creating any child process so that different processes can share the globals, stacks, and heap.
However, you should handle this very carefully. For example, there is a segment for application and each library that used to hold initialized global variables. You can not simply call `mmap` to set this area to MMAP_SHARED, since this call will destroy initialized value of all variables. Thus, before calling `mmap`, you should backup those values at first. Then you should recover after `mmap` calls.

2.3 Synchronization

This is actually quite simple to do that. Now the pthreads library supports the synchronizations among multiple processes. Here is an example for you to do that. Other synchronizations can be handled in a similar way. You will require to intercept synchronizations, include locks, conditional variables, and barriers.

```
pthread_mutexattr_init(&mutex_attr);
pthread_mutexattr_setpshared (&mutex_attr, PTHREAD_PROCESS_SHARED);
pthread_mutex_init(realMutex, &mutex_attr);
```

3 Objectives:

- Learn the concept of processes, threads and virtual memory.
- Familiar with the performance difference and implementation difference between threads and processes.
- Familiar with memory mapped files.
- Familiar with different synchronizations of multithreading.
- Practice how to measure the performance of a software systems.

4 Submission Requirements

In this assignment, you will have to turn in two parts: source code and mythreads-design.pdf. But they will compress them into the same zip package using the following command: "tar -zcvf archive_name.tar.gz directory_to_compress".

1. **Source Code (source directory).** Your source code should the Makefile inside. Your program should be compiled successfully by using "make" command. The runtime library should be named as "libmythreads.so". If the program is not compiled as this name or the program cannot runnable, this part will be consider as 0. In the end, your library should be able to run most test cases under `https://github.com/UTSASRG/multithreadingtests/parsec`. Please put the status to the mythreads-design.txt: which applications are runnable or not. (80%)
2. **mythreads-design.txt.** more about this file are described below. (20%)

For mythreads-design.pdf, you will have to write a technical report on this. You can reference our dthreads paper at `https://github.com/plasma-umass/dthreads/blob/master/doc/dthreads-sosp11.pdf`.

1. **Design Description:** This part describes how you implement your library. What you have done to implement this project?
2. **Performance Comparison:** You will try to compare the performance of your library against the pthreads library (the default one). I suggest that you change the number of threads to see the performance difference. For example, you can try the same number of threads as the number of hardware cores at first. Then you can try to create a larger number of threads than that of cores. Then you will check the performance difference. To compare the orange to the orange, you should create the same number of threads for both `pthreads` and libmythreads.so.

I expecting that the writeup is about five pages.

NOTES: Helping to improve the sample program can get additional points, such as fixing bugs there.