1 Purpose

The purpose of this project is to introduce **threading** in a Linux environment using POSIX threads.
This project's requirements deliberately avoid solving concurrency problems. Instead, students are expected to *accept* and *observe* the race conditions that naturally arise when multiple threads execute without synchronization. These issues are intentionally left unresolved, are considered part of the learning objective, and will not interfere with correctness.

Correctness is defined by time-bounded program termination, correct computation, and adherence to specified execution behavior, **not** by deterministic ordering or race-free execution.

2 Task

Create a command-line program named `proj1` that demonstrates multi-threaded execution using POSIX threads.
The program consists of:

- one **main thread**,
- **n waiting threads** (created but not yet released),
- up to **k running threads**, where $1 \leq k \leq n$ is specified by the user at runtime.

2.1 Main Startup and Behavior

On initialize, the main thread

1. Uses `pthread_create` to creates a number of "paused" threads equal to the number of execution contexts, n, of the machine.
   - The threads must use a loop and a sleep-based wait function (provided in the Project 1 library; see `timings.h`) while paused.
2. After reading all input from STDIN (via I/O redirect from a file), prompts the user (via `/dev/tty`) for a number, k, of threads to use for this execution.
   - In C++, this can be accomplished by

     ```
     #include <fstream>
     std::ifstream tty_in("/dev/tty");
     if (tty_in) {
       int var;
       tty_in >> var;
     }
     ```
3. Using one of the three strategies below, allows the first k threads to begin their work.
   - `--all`: all k threads are released simultaneously.
   - `--rate`: the main thread releases exactly one thread every millisecond using a sleep or timed wait.
   - `--thread`: the main thread releases thread 1; each thread i releases thread $i + 1$ until k threads have been released.

   All release mechanisms are intentionally racy.
4. Waits until all threads have terminated and prints a final report of the results produced during execution.

2.2 Thread Work and Lifetime

Each thread is assigned a fixed index at creation time.

Assuming Thread indices are 1-based and row indices are 0-based, threads process input rows according to the rule:

$$\text{row index} = (\text{thread index}) \times m, \quad \text{where } 1 \leq m \leq \text{number-of-input-rows}$$

Each thread:

- logs when it begins work/is released,
- logs when it completes processing a row,
- exits when no additional rows remain for its index,
- EXITS IMMEDIATELY IF ITS INDEX EXCEEDS THE USER-REQUESTED THREAD COUNT,
- TERMINATES IF A SPECIFIED TIMEOUT EXPIRES.

Once a thread finishes its assigned work or terminates due to timeout, it exits cleanly.

2.3 Prohibited Techniques

The use of C++ threading libraries (`std::thread`, `std::mutex`, etc.) or POSIX synchronization primitives (mutexes, condition variables, barriers, atomics) is strictly prohibited.
Any attempt to synchronize thread execution will result in a score of zero and may be referred for academic integrity review.

3 Deliverables

```
proj1.zip
+-- Makefile                # Build configuration file for make program
+-- main.cc                 # Main entry point of program
+-- main.h                  # Header file for main entry
+-- README.md               # Details contents of source, header, and other files
```

During grading, source files will be relocated into a standard project structure (e.g., `src/` and `include/`) to match the build environment; directory layout is not graded.

3.1 Submission

You must submit a zip archive and only a zipped archive. Individual files and other archive types will not be accepted; 100% penalty.

All names are case-sensitive

Your project must adhere to basic C++ best practices. To that end, I am providing the following files and structure. Note these files are yours and, aside from Makefile, you should expect to create or modify them all.

- `proj1/README.md`
  A non-RTF text file, though you are free to use the standard markdown language of this file type if you choose. The file must describe the purpose of each included file (other than itself).

Cont.

- `proj1/include/main.h`
  If you elect to use a header for organizational purposes, I will look for this file in your submission. If you do not, you may ignore it.

- `proj1/src/main.cc`
  A C++ source file containing your application.

- `Makefile`
  I have provided you with a `Makefile` that will build all code, assuming it is placed in the correct locations. You **MUST NOT** alter this file. I will use my own version for testing and your `Makefile` will be ignored.

All source code must be styled. Use the Google style guidelines for your code; 15% penalty. Assuming you have access to Python 3 and Pip 3, install cpplint for ease of checking your code's style. Code so poorly styled as to be unreadable will not be considered.

4 Command-Line Interface

Exactly one execution mode must be specified:

- `--all`
- `--rate`
- `--thread`

An optional timeout may be provided:

`--timeout <milliseconds>`

Program input must be provided via standard input; example input files are provided in the `dat/` directory.

4.1 Example Interaction

```
$ ./proj1 --all --timeout 3000 < dat/small.txt
Enter max threads (1 - 8): 4
[thread 1] started        4  3
[thread 2] started
[thread 3] started
[thread 4] returned
[thread 1] completed row 0
[thread 1] returned
[thread 3] completed row 2
[thread 3] returned
[thread 2] completed row 1
[thread 2] returned
Thread      Start          Encryption
0 1         hello          d1b5f3c14ce32ce8...96d1b66f74575858
1 2         world          b2533efb15699f46...ec8fbdf556ccae8e
2 3         operating_sys...
```

5 Points

This section describes the points awarded for each correct project deliverable.

Any program that fails to terminate within the specified timeout, or requires manual intervention to stop, receives zero points for correctness.

- Correct `README.md`: 1 point.
  You must explain exactly what is contained in each file and your design decisions for its contents.
- Correct(build-able) entry file (`main.cc`): 1 point.
  Code that compiles, but contains warnings will only receive 0.85/1.0 points.
- Correctly executing program (`bin/proj1`): 2 points.
  The program must initialize, prompt the user for input, and exit no later than <timeout>s after input.
- Program correctly releases k paused threads at once, with correct output and report: 2 points.
- Program correctly releases 1 paused thread per ms for k threads, with correct report: 2 points.
- Program correctly releases 1 paused thread per thread (including main thread) for k threads, with correct report: 2 points.

The End.