HW 1: List

In this homework, you will gain familiarity with threads and processes from the perspective of a user program, which will help you understand how to support these concepts in the operating system.

Along the way, you will gain experience with the Pintos list data structure, but in the context of a user program running on Linux. We hope that completing this assignment will prepare you to begin Project Userprog, where you will work with the implementations of these constructs in the Pintos kernel. In particular, we hope that you gain experience with how to use them for both development and testing purposes in a contained environment where bugs are relatively easy to catch, before having to work with them in Pintos.

Getting started

To get started, log in to your development environment and get the starter code.

```
cd ~/code/personal/
git pull staff main
cd hw-list
```

To build the code, run `make`, which should create four binaries: `pthread`, `words`, `pwords`, and `lwords`. Make sure not to commit and push any binaries when submititng to the autograder. You can get rid of unnecessary binaries using `make clean`.

Note: You do not have to add the `-f` flag to run the executable in this assignment.

Skeleton

You'll notice that for some files, only the object files without the source are provided. Part of being able to program means being able to work with the abstractions you're provided, so we've left out implementations which are not necessary for you to complete this assignment.s

`list.h` provides the Pintos list abstraction which is taken directly from the Pintos source code. `list.c` provides the implementations, but you should be able to use this library solely based on the API

given in `list.h`. **You must not modify these files**.

`word_count.h` defines the API you will implement. We have already provided necessary data structures `word_count_t` and `word_count_list_t` which **you must use**.

`words.o` and `word_count.o` provide compiled implementations of methods necessary to run the `words` program from Homework Intro. You can use the outputs of these programs as sanity checks on what your other programs that you'll build should output.

`word_count_l.c` will house your implementation of the the API in `word_count.h` using Pintos lists. The `Makefile` will provide the macro definition of `PINTOS_LIST` when compiling. When `word_count_l.c` is linked with the driver in `lwords.o` and compiled, it should result in an application `lwords` that behaves identically to the frequency mode of words but internally using Pintos lists instead of traditional linked lists as seen in Homework Intro.

Similarly, `word_count_p.c` will house your implementation of the API in word count.h using Pintos lists and proper synchronization of the word count data structure for a multithreaded program. Unlike `lwords`, you'll need to write the driver program in `pwords.c`. When `word_count_p.c` and `pwords.c` are put together and compiled, it should create an application pwords that behaves identically as `lwords` and frequency mode of `words` but internally uses multiple threads.

`word_helpers.h` provides an API for parsing and counting words. `word_helpers.o` provides compiled implementations for these methods.

`pthread.c` implements an example application that creates multiple threads andprints out certain memory addresses and values. You may find it helpful to base your `pwords.c` implementation off of `pthread.c`. While you won't be writing any code in `pthread.c`, you will be reading and analyzing it.
