HW 1: **Lists**

CS 162

Due: February 7, 2020

Contents

In this homework, you will gain familiarity with the list data structure used widely in Pintos. We hope that completing this assignment will prepare you to begin Project 1, where you will work with the the list data structure directly in the Pintos kernel. Our goal is to give you experience with how to use it in userspace, an environment where bugs are relatively easy to find and fix, before having to work with them in Pintos. It will also help you to see how you can do a lot of your development and testing of project code in a contained user setting, before you drop it into the Pintos kernel.
This assignment is due at 11:59 pm on 2/7/2020.

1 Getting Started

Log in to your Vagrant Virtual Machine and run:

```
$ cd ~/code/personal/
$ git pull staff master
$ cd hw1
```

Run `make` to build the code. Two binaries should be created: `words` and `lwords`.

1.1 Overview of Source Files

Below is an overview of the starter code:

`words.c`, `word_helpers.c`, `word_helpers.h`
This files contain a simple driver for parsing words, computing their aggregate count, and then writing the count to a stream, as in the previous homework.

`word_count.h`, `word_count.c`
These files provide the interface of and the implementation of the `word_count` abstraction using a conventional singly-linked data structure for the representation. They are, in effect, the preferred solution to your C refresher problem in Homework 0. You should study these files carefully and compare them to your previous work from Homework 0. You must **not** modify these files.

`list.c`, `list.h`
These files are the list library used in Pintos, which is based on the list library in Linux. You should be able to understand how to use this library based on the API given in `list.h`. You must **not** modify these files. If you're interested in learning about the internals of the list library, feel free to read `list.c` and the `list_entry` macro in `list.h`. You can find a good explanation of the `list_entry` macro here[1].

`word_count_l.c`
This file is the starter code for your implementation of the `word_count` interface specified in `word_count.h`, using the Pintos list data structure. We have already provided the type declarations for this in `word_count.h`. You must use those. Notice how the list element is embedded into the `struct`, rather than the `next` pointer. Also, the `Makefile` provides the `#define PINTOS_LIST` as a flag to `cc`. `words.c` is unchanged, as is its behavior. This exercise will cement your understanding of how to traverse and manipulate the kinds of lists that are used throughout Pintos. Your implementation of the `word_count` interface in `word_count_l.c`, when linked with the driver in `words.c`, should result in an application, `lwords` that behaves identically to `words`, but internally uses the Pintos list data structure to keep track of word counts.

[1] https://stackoverflow.com/questions/15832301/understanding-container-of-macro-in-the-linux-kernel

`debug.c`, `debug.h`
These files are taken directly from the PintOS kernel and provide helpful debugging routines and macros. You must **not** modify these files.

`gutenberg`
This directory contains test cases for the homework. Do **not** modify any files in this directory.

2 Check Yourself on Word Count

Read the basic `words` files and compare them with your Homework 0 solution. You will be building on this one, so make sure you understand it.

3 Warm Up (Optional)

Write a simple application that reads in the output produced by `words` and creates the corresponding `word_count` data structure. Print it out to check.

4 Using Pintos Lists to Count Words

The Pintos operating system makes heavy use of a particular linked list library taken directly from Linux. Familiarity with this library will make it easier to understand Pintos, and you will need to use it in your solution for the projects. The objective of this exercise is to build familiarity with the Pintos linked list library in userspace, where issues are easier to debug than in the Pintos kernel.

First, read `list.h` to understand the API to library.

Then, complete `word_count_l.c` so that it properly implements the new `word_count` data structure with the Pintos list representation. You **MUST** use the functions in `list.h` to manipulate the list.After you finish making this change, `lwords` should work properly.

The `wordcount_sort` function sorts the wordcount list according to the comparator passed as an argument. Although `words` and `lwords` sort the output using the `less_count` comparator defined in `word_helpers.c`, the `wordcount_sort` function that you write should work with any comparator passed to it as the argument `less`. For example, passing the `less_word` function in `word_helpers.c` as the comparator should also work. If you're having trouble with function pointer syntax when implementing this, here[2] is a good tutorial.

Hint #1: We provide a `Makefile` that will build `lwords` based on these source files. It compiles your program with the `-DPINTOS_LIST` flag, which is equivalent to putting a `#define PINTOS_LIST` at the top of the file. This selects a definition of the word count structure that uses Pintos lists. We recommend reading the `word_count.h` file to understand the new structure definition so you can see how the Pintos list structure is being used.

Hint #2: The provided `Makefile` uses the same `words.c` file to provide the `main()` function in both the `words` and `lwords` programs. So that the same `main()` function works for `lwords`, you should ensure that your implementation of each function in `word_count_l.c` is functionally equivalent to the corresponding function in `word_count.c`. In other words, both `word_count_l.c` and `word_count.c` must implement the same interface, namely the one in `word_count.h`.

[2]https://www.geeksforgeeks.org/function-pointer-in-c/

5 Autograder and Submission

To submit and push to autograder, first commit your changes, then do:

```
$ git push personal master
```

Within a few minutes you should receive an email from the autograder. (If you haven't received an email within half an hour, please notify the instructors via a private post on Piazza.) Please do not print anything extra for debugging, as this can interfere with the autograder.

Autograder Scores: As was the case with HW 0, your final score in the autograder will **not** be the maximum of your attempts, but rather your score for only your latest build. Any non-autograded components, like style and written portions, will be graded based on your last build for the assignment. Running a build after the deadline will consume slip days even if it doesn't change your score.

Weekly Survey: Remember to fill out your weekly survey! It will be due at the same time as this homework is due (February 7 at 11:59 PM). It will appear as a quiz on bCourses.