Project 1: Multi-Process Web Browser

CSCi 4061: Introduction to Operating Systems - Weissman – Fall 2022

*To be completed by your group of **size 3**, see Piazza for details*

Posted	Sept. 21, 05:00 PM
Intermediate Submission Due	Sept. 28, 11:59 PM
Final Submission Due	Oct. 5, 11:59 PM

1 Introduction

Today most Web browsers rely on the use of multiple processes. This design provides two primary benefits:

1. Robustness:
   Webpage isolation via separate processes makes the web-browser immune to crashes by providing address-space isolation where each webpage is rendered by a separate process. An extension crash (one of the main reasons for browser crashes) in one webpage will have then have no effect on other webpages being managed by other processes.

2. Parallelism:
   Webpages are becoming ever more complex with contents ranging from JavaScript code to full- blown web-applications like Google docs. With the functionality of a web-browser transforming into a platform where web-applications can run independently and in parallel, a multi-process architecture for the browser is attractive on multi-node/multi-core systems. In this project, we will explore this architecture for a basic web-browser. Note: we will provide all browser graphics, event handling code (e.g., clicks and input retrieval), and functions that render browser tabs. Most of the gtk code is in wrapper.c.

2 Description

In this project, you will implement multi-process browser. Code has been given to you that draws tab windows, and an executable ***render*** that will fetch and render a URL in a tab window:

```
$ ./render <tab #: 0, 1, … > <URL>
```

1- Example of running the "render" executable

Code has also been given to you that acts as a ***controller*** which also runs as a separate process created by the browser main (i.e., parent). The ***controller*** allows you to create tabs with an associated URL.

- Tabs are numbered starting at 0 and only increase.
- You will increment the tab # on each new window.
- The browser is limited – e.g. the tab window will be unable to change the URL it is showing.
- You should set up the ***controller*** in the browser main.

Our code is based on a webkit which uses the gtk library to draw windows and provide user-provided input (i.e. URLs), the details of this library are not important at this point. One thing to note is that this library is programmed using *callbacks*, functions that are passed to the library that you write, that will be invoked from the library when certain events happen (e.g. creating a new tab). The only callback used at present is called: *uri_entered_cb()*. This callback is invoked when a URL is entered in the URL bar in the ***controller*** window and a return is hit – meaning a new tab window should be created. This is the opposite of how we normally use libraries. So, you will learn a little bit about this cool concept. The *controller* enters this library (via *show_browser*()) and is blocked until the user takes an action on the controller window. That poses a few challenges described in the next section. You will also learn how to use fork/exec/wait. The reason wait should be used is that **ALL** child processes must be waited on by the parent, included the ***controller***. To see how the multi-process browser should run we have also provided an executable solution you may run.

3 Termination

As stated above, the termination of all child processes must be waited for. A tab window process is terminated when the X button is hit. Unfortunately, the ***controller*** is still blocked and the tab process becomes a zombie since the ***controller*** is unable call wait until it exits *show_browser*() . The ***controller*** will exit this function when a X is hit on the controller window. At this point, you can wait for zombies. But there also may be other active tab window processes that must be killed since a controller X shuts everything down. The simplest thing to do is to kill **all** processes and wait on all of them. After this, the controller can terminate. Don't forget to have the ***browser*** *main*() wait on the ***controller***.

4 Constraints

You must enforce several constraints. The controller keeps running even if any constraint is violated.

Constraints:

1. The maximal # of tabs created cannot exceed MAX_TABS. If it does, you will issue an alert and not create the tab. Even if a tab is deleted (X on the tab window), the number of tabs is not reduced (think about why this is?).

2. The URL format must begin with {http|https}:// If it does not, you will issue an alert and not create the tab.

3. Your browser must use a "blacklist" that contains URLs that cannot be used.

    (a) The blacklist contains one URL per line.

    (b) The one tricky bit is that there are two equivalent forms of a URL (e.g. espn.com and www.espn.com) you must catch both even if the blacklist only lists one of these formats.

The constraint processing will give you some experience with file I/O (e.g. fopen, fgets), and string processing (e.g. strcpy, strlen, strcmp, strtok). Look them up! You may also find sprintf and sscanf useful for concatenating and parsing strings.

A couple of gotchas: fgets puts a '\n' at the end of the string that is read,and be careful about pointer sharing within strings.

5 Intermediate Submission

You to you started, you must submit an intermediate submission due Sept. 28 11:59 pm. You will create the controller process and its window and allow a URL to be entered. When URL is entered, you will print just it out.

6 Possible split

There are several discrete tasks that can be used a basis to decompose the work, but I recommend you ALL gain a shared understanding of the code workflow. Tasks include dealing with termination, constraints, and bringing up tab processes and their windows.

7 Grading Criteria

10% - Intermediate submission

20% - Documentation of code, coding style. (Indentations, readability of code, use of defined constants, good names, modularity, etc.), following submit instructions (see below)

70% - Functionality. Constraints, and Error Handling.

1. Tab creation (20%): creates new tabs and URL rendering
2. Termination (20%): X terminates a child, controller X causes everything to shut down, with all children waited on
3. Too many tabs:(5%) must catch this
4. Bad URL format (5%) must catch this
5. Blacklist (10%): must catch this
6. Error checking for all system calls (10%)

8 Documentation

Within your code you should use one or two sentences to describe each function that you write. You do not need to comment every line of your code. However, you might want to comment portions of your code to increase readability.

9 Deliverables:

Tar file containing your code and a Makefile that can be used to build and execute your code. At the top of **browser.c** please include the following comment:

```
/* CSCI-4061 Fall 2022 - Project #1
 * Group Member #1: <name> <x500>
 * Group Member #2: <name> <x500>
 * Group Member #3: <name> <x500>
*/
```

You submit ONLY ONE tar file per group. You will be docked points if you do not follow these instructions.