Lab 4

A Multi-threaded Web Crawler

4.1 Objectives

This lab is to design and implement a multi-threaded web crawler. In the previous lab, we practised memory sharing between processes as a means to communicate between processes. Sharing memory between threads are a lot easier since they live in the same address space. We do not need the operating system's involvement to have a shared memory region between threads. In addition, creating/destroying threads is less expensive than creating/terminating child processes.

However we still need to avoid race conditions in the memory region that threads are sharing. Aside from mutex and semaphore, the operating system also provides condition variable and atomic type facilities.

After this lab, students will be able to

- Design and implement a multi-threaded concurrent program using multiple synchronization patterns;
- Apply thread synchronization facilities in Linux to manage shared resources;
- Develop and evaluate a scalable solution for web crawling using multi-threading.

4.2 Starter Files

The starter files are on GitHub at url: https://git.uwaterloo.ca/ece252-s25/ece252_starter/-/tree/main/lab4/starter. It contains the following sub-directories where we have example code to help you get started:

- the curl_xml has example code to show how to use curl and libxml2 together to identify a possible png page and extract http(s) links from a html page.
- the tools has a shell script to compute statistics of timing data.

The lab4_eceubunt1.csv is the template file that you will need for submitting timing results (see Section 4.5.2).

4.3 Pre-lab Preparation

Read the entire lab4 manual to understand what the lab assignment is about. Build and run the starter code to see what they do. You should work through the provided starter code to understand how they work. The following activities will help you to understand the code.

1. Run the given starter code with the following URLs and examine responses from the server in the http header.
   - http://ece252-1.uwaterloo.ca/lab4
   - http://ece252-1.uwaterloo.ca/lab3/index.html
   - http://ece252-1.uwaterloo.ca/~yqhuang/lab4/Disguise.png
   - http://ece252-1.uwaterloo.ca:2530/image?img=1&part=1
2. Execute `man pthread_cond` to read the man page of condition variable.

Linux man pages are also available on line at https://linux.die.net/.

4.4 Lab Assignment

4.4.1 Problem Statement

In the previous labs, the URLs[1] to download the image segments are given. In this lab, you will need to search some HTTP lab servers to find these URLs. We have 50 different URLs, each of which links to a unique PNG image segment of a particular image. The mission is to search for these URLs on the lab servers[2].

To solve the problem, we will create a multi-threaded web crawler named `findpng2` to search the web given a seed URL and find all the URLs that link to PNG images.

[1] URL stands for Uniform Resource Locator (see https://en.wikipedia.org/wiki/URL). It is a web page address. For the purpose of this lab, it starts with the string "http://"

[2] This lab does not require one to concatenate these segments. However if you are interested in what these segments are, then you can use your catpng to restore the original image after downloading all the segments or directly concatenate the segments in memory using lab2/3 code. The simple PNG format, dimensions of each image segment and the http header that tells you which segment you are getting are the same as what we had in previous labs.

4.4.2 The findpng2 command

The expected behaviour of the `findpng2` is given in the following manual page of the command.

Man page of findpng2

NAME

findpng2 - search for PNG file URLs on the web

SYNOPSIS

findpng2 [OPTION]... SEED_URL

DESCRIPTION

Start from the SEED_URL and search for PNG file URLs on the world wide web and return the search results to a plain text file named `png_urls.txt` in the current working directory. Output the execution time in seconds to the standard output.

-t=NUM

create NUM threads simultaneously crawling the web. Each thread uses the curl blocking I/O to download the data and then process the downloaded data. The total number of pthread_create() invocations should equal to NUM specified by the -t option. When this option is not specified, assumes a single-threaded implementation.

-m=NUM

find up to NUM of unique PNG URLs on the web. It is possible that the search results is less than NUM of URLs. When this option is not specified, assumes NUM=50.

-v=LOGFILE

log all the visited URLs by the crawler, one URL per line in LOGFILE. When this option is not specified, do not log any visited URLs by the crawler and do not create any visited URLs log file.

OUTPUT FORMAT

The time to execute the program is output to the standard output. It will look like the following:

```
findpng2 execution time: S seconds
```

The search results is a list of PNG URLs, one URL per line saved in a file named `png_urls.txt` . The order of listing the search results is not specified. If the search result is empty, then create an empty search result file.

EXAMPLES

```
findpng2 -t 10 -m 20 -v log.txt http://ece252-1.uwaterloo.ca/lab4
```

Find up to 20 PNG URLs starting from http://ece252-1.uwaterloo.ca/lab4 using 10 threads. The output on the standard output will look like the following:

```
findpng2 execution time: 10.123456 seconds
```

The first two lines in the `png_urls.txt` file may look like the following:

```
http://ece252-2.uwaterloo.ca:2540/img?q=tyfoighidfyseoid==
http://ece252-1.uwaterloo.ca:2541/img?q=kjvjkjxsroutqpqkgh
```

An empty search result will generate `an empty png_urls.txt` file.

The first two lines in the `log.txt` file may look like the following:

```
http://ece252-1.uwaterloo.ca/lab4
http://ece252-1.uwaterloo.ca/~yqhuang/lab4/index.html
```

4.4.3 Web crawling

The `findpng2` is a tiny simplified web crawler. It searches the web by starting from a seed URL. The crawler visits the given URL page and finds two pieces of information.

The first piece is the URLs that link to valid PNG images[3]. The crawler adds PNG URLs found to a search result table. We want this table to contain unique URLs, hence if the found URL is already in the table, you should not add it to the table.

The second piece is a set of new URLs to further crawl. The crawler adds this set of new URLs to a URLs pool known as the `URLs frontier` . Since visiting web pages has costs, we do not want the crawler to visit the same page twice. Hence the crawler needs a mechanism to remember URLs that have been visited already. As the crawler visits the URLs in the URLs frontier, the process of finding the target PNG URLs and new URLs to further explore repeats until it finds no more new PNG URL or it reaches the maximum number of PNG URLs specified by the user input.

[3]A valid PNG image is a file whose first 8 bytes matches the PNG signature bytes

4.4.4 The HTTP

HTTP stands for "Hypertext Transfer Protocol". They carry important information about the client requests and the server responses. When the client sends an URL to the server, it makes an HTTP GET request to the server and the detailed information about the request is in the headers. The server will first respond with an HTTP response status code line. There are three categories we need to handle in this lab.

- HTTP/1.1 2XX. This is a success response. We need to process the data the link gives.
- HTTP/1.1 3XX. This is the case the link has been relocated. By feeding the `curl_easy_setopt` with the `CURLOPT_FOLLOWLOCATION`, curl will follow the relocated links. The `CURLOPT_MAXREDIRS` in the curl option setting allows one to specify maximum number of redirects to follow.
- HTTP/1.1 4XX. This is a broken link, usually caused by the client side. We do not process the link. But we need to remember this link has been visited.
- HTTP/1.1 5XX. This is also a broken link, usually caused by server internal error. We are not able to process the link. But again need to remember this link has been visited.

After the response status code line, the web server uses http response headers to send meta information in different fields about the web resource content it sends to the client. One of the fields is "Content-Type". For the purpose of the lab, we are only interested in two types of Content-Type. One is the `text/html`, which is a hyper text file where we find more URLs. The other one is the `image/png` which is a PNG image that we look for. You will process the following two cases of Content-Type:

- Content-Type: text/html
- Content-Type: image/png

The http header call back function of curl allows us to process all the header responses from the server. Another way is to use `curl_easy_getinfo` function to obtain a specific header information[4]. For example, with the second parameter of the function setting to `CURLINFO_CONTENT_TYPE`, we obtain the content type header information.

As you may recall from previous lab, the lab server uses an HTTP response header that has the format of "X-Ece252-Fragment: M" where $M \in [0, 49]$ to tell which image segment it sends to the client. If you are only interested in finding the PNG image segments that the lab web server has, then this piece of information is useful.

[4]Only standardized headers are supported. User defined headers such as those starting with X- are not supported.

After all the response headers are sent, the server sends out the actual contents of the web resource in the message body. The write call back function of curl allows us to process this piece of information.

4.4.5 Programming Tips

You will need a number of lists to keep track of different sets of URLs. One list is for the URLs frontier, which contains to-be-visited URLs. One list is for recording all the URLs that have been visited. Another list is for the PNG URLs that have been found. To crawl the web using multiple threads, these lists are shared between threads. Hence you need to synchronize them. Some lists can only be accessed by one thread both when reading and writing. Some lists may be accessed by multiple threads when reading, but only one thread when writing.

Another subtle difficulty is to know when to terminate the program. The program should terminate either when there are no more URLs in the URLs frontier or the user specified number of PNG URLs have been found. You may need some shared counters to keep track of information such as how many PNG URLs have been found and how many threads are waiting for a new URL.

If an URL has been visited already, then we do not want to visit it again. So a search of visited-URLs list is needed. Hashing will make the search very effective and you may consider using a hash table to represent this already-visited list. The glibc has hash table API (`man hsearch(3)`).

4.5 Deliverables

4.5.1 Pre-lab deliverables

None.

4.5.2 Post-lab Deliverables

Put the following items under the directory named lab4:

1. All the source code and a Makefile. The Makefile default target is `findpng2` executable file. That is command `make` should generate the `findpng2` executable file. We also expect that the command `make clean` will remove the object code and the default target. That is the `.o` files and the executable file should be removed.

2. A timing result .csv file named lab4_hostname.csv which contains the timing results by running the findpng2 on a server whose name is hostname. For example, `lab4_eceubuntu1.csv` means findpng2 was executed on the server

eceubuntu1 and the file contains the timing results. The first line of the file is the header of the timing result table. The rest of the rows are the timing result command line argument values and the timing results. The columns of the .csv file from left to right are values of T (the number of threads), M (the number of unique PNG links to search for) and $TIME$ (the corresponding findpng2 average execution time). We have an example .csv file in the starter code folder named `lab4_eceubuntu1.csv` for illustration purpose.

Run your findpng2 on eceubuntu1. Record the average timing measurement data for the (T, M) values shown in Table 4.1 for a particular host. Note that for each given (T, M) value in the table, you need to run the program n times and compute the average time. We recommend $n = 5$.

T	M	Time
1	1	
1	10	
1	20	
1	30	
1	40	
1	50	
1	100	
10	1	
10	10	
10	20	
10	30	
10	40	
10	50	
10	100	
20	1	
20	10	
20	20	
20	30	
20	40	
20	50	
20	100	

Table 4.1: Timing measurement data table for given (T, M) values.

Submit your lab project 4 on ECE Marmoset Submission and Testing Server.

4.6 Marking Rubric

Table 4.2 shows the rubric for marking the lab.

Points	Sub-points	Description
100		Post-lab
	10	clean project files organization
	5	Makefile correctly builds and cleans
	65	Implementation of multi-threaded `findpng2`
	20	Correct timing results in lab4_hostname.csv file

Table 4.2: Lab4 Marking Rubric