CS39002: Operating Systems Lab

Spring 2015

Assignment 7 (Threads)

Multithreaded Web-crawler System

Due on: April 13

In this assignment, you have to develop a simple multithreaded web crawler. Each thread downloads an html page from a URL taken from a global “to-do” list, extracts URLs in HTML page, and puts the extracted URLs back to the “to-do” list.

The crawler does a breadth-first traversal of the web graph, i.e. starting from a given node, all nodes at distance N are fetched before any distance N+1 nodes. For a single threaded program, this property can be satisfied by maintaining the “to-do” list as queue. However, in case of multithreaded crawler, you have to maintain this property explicitly. Another requirement is that each URL should be crawled exactly once.

You should by start crawling from **cse.iitkgp.ac.in** and crawl till a specified depth. In the end, the crawler should print the following:

```
<depth>  <thread id>  <URL>
1        1            http://cse.iitkgp.ac.in
...
...
```

Please see below sample code for fetching URL using libcurl. For the HTML parsing and extracting links, see links at the end.

The system should work in the following way:

- At the beginning, there’s a **parent process**, which has ‘cse.iitkgp.ac.in’ as the **master URL.**
    - The system’s sequence of crawling pages can be considered to be a tree-like structure.
    - **Master URL** is at level 1 (root). The URLs obtained from an HTML page of a level N URL would be considered level N+1 URLs. e.g. all the URLs obtained after parsing ‘cse.iitkgp.ac.in’ would be at URL level 2.
    - You’d have to maintain a global status of the URL level at which the system is operating. At the start it’d be 1, and would be updated at a certain step, as described afterwards.
- The parent process initializes a global **to-do queue** and puts the master URL in it. It also initializes an empty **done list** and a **to-do-next queue**. The **URL level** is set to 1**.**
    - The **done list** contains (URL, *thread*_id) pairs, separated by blank records for separating URLs of different levels.

- Initalize all variables and mutexes. Initialize a condition variable / two barriers as needed for synchronization. All shared data should be accessed, controlled by mutexes.
- After the all initializations, the parent process spawns 5 threads. Then each thread operates in a loop, doing the following:
    - Access the to-do list to dequeue an URL.
    - **If** the queue is empty (this means all URLs at current level have been crawled), the threads wait for all other threads to reach this point and to-do-next list to be copied to to-do list. This is achieved using barriers or condition variables, described below in level-increase function. ***Exit loop*** if the level has reached a specified value, say 3; else skip to the beginning of the loop.
    - **If** the URL has already been crawled / being crawled by another thread (present in done list), skip to the beginning of the loop.
    - **Else** download the URL.
        - If the content type of the url is HTML, extract links, and populate them in the to-do-next list, using mutex lock.

- During the whole loop part, the threads should also print messages to the screen for the following events:
    - **Dequeueing of a url from to-do queue**: print thread number and and url
    - **When a thread finds to-do queue empty**: print “Thread <thread number> found to-do queue empty”
    - **Increment of level**: print the new level number

- After all the children threads have come out of the loop, they get terminated.
- Now, the parent thread would print the **done list** (both the URLs and corresponding thread numbers) with proper demarcation of levels, as marked by blank URLs.

- The URLs of the last level would still be in **to-do queue,** so print them to get the complete list. As no thread info is stored for these (and the HTML of these URLs wasn’t accessed by any thread) print ‘0’ for the thread number.

Required functions:

Level-increase function using condition variables:
Level_increase(condition, condition_mutex, nthreads)

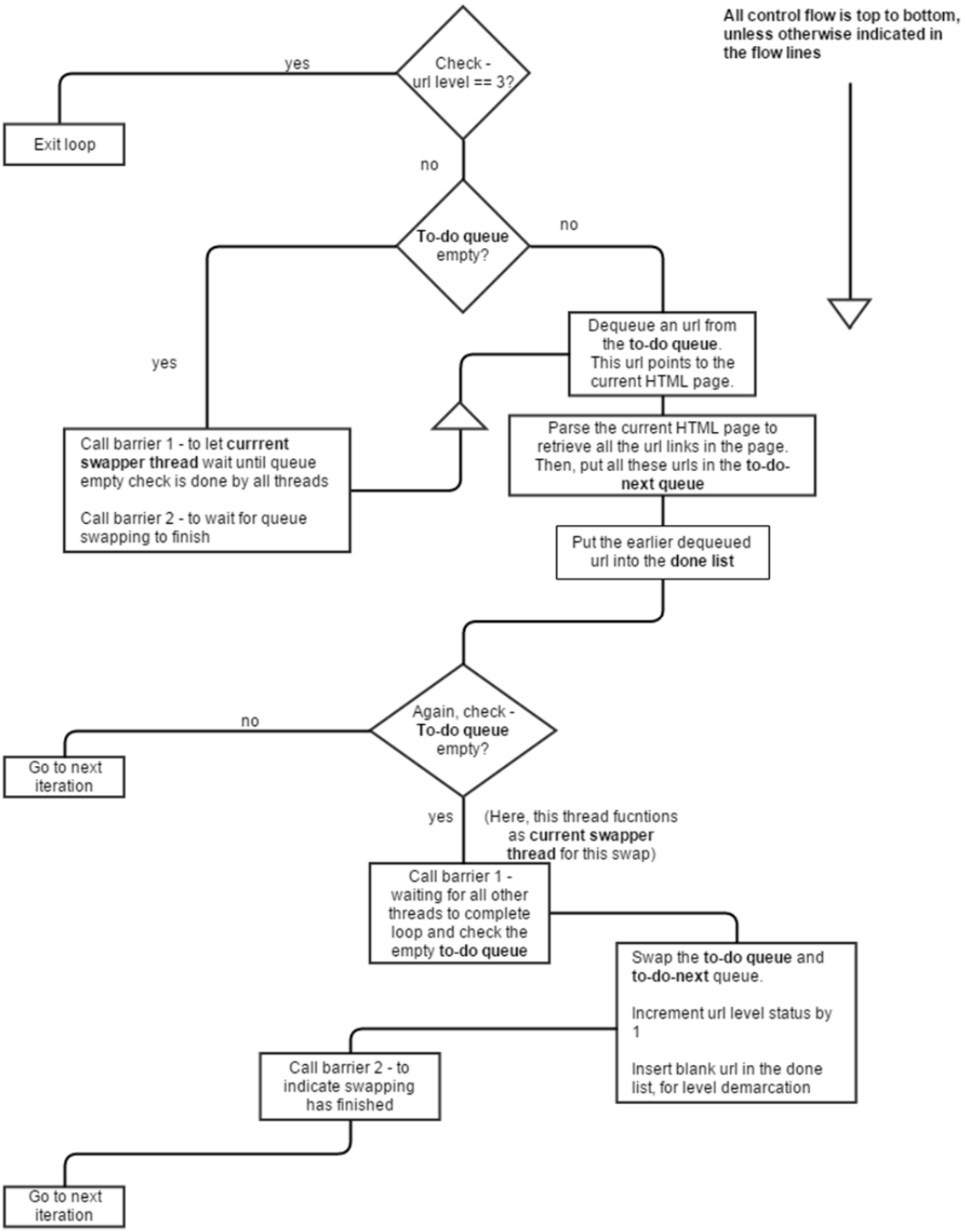
- Lock condition_mutex.
- Nthreads ++
- **If** nthreads == 5 , release mutex. Copy **to-do-next** to **to-do**. Increase level. Broadcast (condition)
- **Else** wait(condition). This automatically unlocks mutex.

Level-increase function using barriers:
Level_increase(barrier1, barrier2)

- Wait(barrier1)

- If *thread_id* == 1 copy **to-do-next** to **to-do**. Increase level.
- Wait(barrier2)

See the diagram below for further reference:



Sample code to download contents of a URL using CURL:

```
#include <stdio.h>

#include <curl/curl.h>

int main(void)

{

  CURL *curl;

  CURLcode res;

  curl_global_init(CURL_GLOBAL_DEFAULT);

  curl = curl_easy_init();

if(curl)

{

    curl_easy_setopt(curl, CURLOPT_URL, "https://example.com/");

   /* Perform the request, res will get the return code */

 res = curl_easy_perform(curl);

/* Check for errors */

if(res != CURLE_OK)

fprintf(stderr, "curl_easy_perform() failed: %s\n",

curl_easy_strerror(res));

    /* always cleanup */

curl_easy_cleanup(curl);

  }
```

```
  curl_global_cleanup();

return 0;

}
```

For extracting URLs from an HTML:
You might want to look at this sample code :
http://www.anotherchris.net/csharp/extracting-all-links-from-a-html-page/
Another reference:
http://stackoverflow.com/questions/499345/regular-expression-to-extract-url-from-an-html-link

References:

- **libcurl HTML parsing**
  - **libcurl homepage** http://curl.haxx.se/libcurl/
  - **libcurl code examples** http://curl.haxx.se/libcurl/c/example.html (including HTML parsing example)
  - **libcurl - checking file content type (HTML or otherwise)** http://curl.haxx.se/libcurl/c/curl_easy_getinfo.html
- **Thread constructs**
  - **Condition Variables** https://computing.llnl.gov/tutorials/pthreads/#ConditionVariables
  - **Barriers** http://pages.cs.wisc.edu/~travitch/pthreads_primer.html (Skip to Barriers part)