Due date：Fri，Oct 25，23:59 (Adelaide time).

Task: Multi-Threaded Network Server for Pattern Analysis

Objective:

Concurrency is among the most challenging parts of Operating Systems to understand. Developing a full appreciation for the problem space is essential for your later professional career. The ability to think in concurrent design architectures and to program multi-threaded with proper synchron r this reason, this assignment **eaded** **network server** rocessing text data, and a e will provide hands-on exper andling, and understanding t cking I/O.

1. Setup

You will require some large text files for this assignment. Consider using resources like the Gutenberg Project (https://www.gutenberg.org) to obtain such files. Download plain text format books (UTF-8) and save them locally for later use.

To send these text files to your program, consider utilising the netcat tool (nc). For instance, to transmit a text file to your server, you may use the following command:

```
nc localhost 1234 -i <delay> < file.txt
```

Ensure that the first line of each text file contains the title of the respective book. This makes your program later easier, as you can grasp a book identifier easily from the incoming data stream.

2. Multi-Thread

Write a multi-th uage of your choice (C, Pytho ng connections on socket in any language, for ex ks to an external site., or take a reference from for socket implementation.

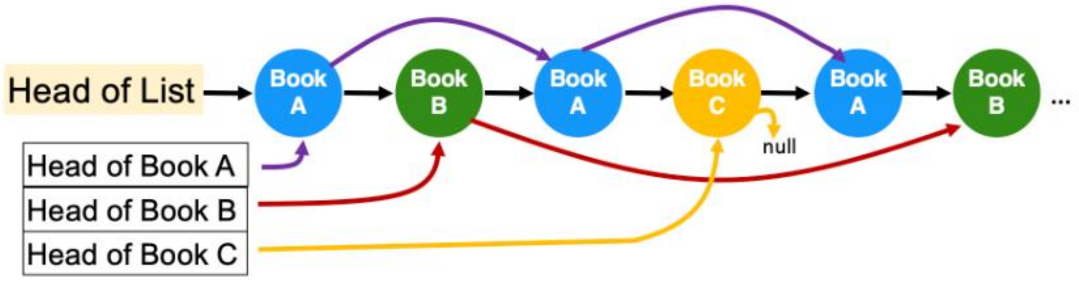
For more info: https://www.ibm.com/docs/en/zos/2.4.0?topic=programming-c-socket-call-guidanceLinks to an external site.

The server should listen to a networking port (> 1024). Ensure that the server efficiently manages multiple simultaneous connections. The program should create **a new thread for each incoming connection** to handle client communication. This approach has to allow multiple clients to connect simultaneously. In each thread, implement non–blocking readsLinks to an external site. from the sockets to efficiently receive and store data in one shared data structure – a list. Every line read is linked into that shared list that is the same across all threads. Links to an external site.

Part 1 – manage that shared list, which involves multiple tasks:

- Manag or line you will cre The purpose is to b a has arrived and be t process we discus g the node at the en
- Additi y embedding a seco shared list and having a head pointer for each book. That means each thread needs to be able to update (additional) links to nodes that contain book lines in the correct order.
- Print a book: output each received book correctly: traversing the list from the book's header by "book_next" reproduces the complete book in the correct order.

A diagram of the shared multi–link list is shown below



Note that the shared list has multiple links per node as described below:

node–>next – links to the next element in the shared list.

node–>book_ne ok.

node–>next_fre that had th

e search terms

Program output

As you add each ting the

addition of that

When a connection closes, you should write the received book; the filename is book_xx.txt where "xx" is the number (order) in which the connection was accepted. For example, if you have three connections your program should write three files: book_01.txt, book_02.txt and book_03.txt.

Part 2 – Multithreaded Analysis

Implement two or more_analysis threads that read from that shared data structure in a similar fashion that you have learned from the consumer/producer problem and are able to compute the frequency of an specific search pattern within the received data (e.g., maintain a linked list of notes that contain a particular search string). The pattern would be given by the command line.

In periodic confi onds), one of
the analysis thre he most
frequent occurre one thread
that initiated the d to write to
the screen and tput of the
analysis should

3. Testing

Develop tests th afely to avoid data race conditions. You may initially test your program with a small number of input streams but make sure you also test your program with at least 10 simultaneous input streams to ensure robustness and scalability.

Submission:

Deliverables

- Multi–threaded network server code in either C, Python or Java programming language.
- Makefile to produce *assignment3*

The server should be started with:

```
./assignment3 -
```

where 12345 is y" is the

seach pattern: it ppears in the

book and create

- A REA to run your server ng netcat) to send c

Grading Criteria

Part 1 is worth 80% and will be marked using an automatic script

- The network server can accept incoming connections from a listen socket and is non–blocking 10%

- Efficient handling of multiple connections using threads, ensuring the server remains responsive 10%
- Correct log printed 15%
- Books are received and printed into files correctly 30%
- Scales to 10 threads 15%

Part 2 is worth 20% and will be assessed both automatically and manually.

- Patter
- Result
- Book t s of the select