

README

multithread.c

Introduction

This program implements the process of multi-thread for readers and writers which connect all threads with synchronize and mutex. This source code represents the method how the reader and writer work by inputting random numbers.

Member

- | | | |
|------------|-------------|---------------|
| 1. 6488081 | Pittinunt | Sirigittikul |
| 2. 6488110 | Pattaraporn | Tongmahavised |

Source Code (function)

- **AccessDatabase (int type, int r, int i)**

This function takes the type to indicate whether the thread calling is reader (0) or writer (1). The writer thread, "r" will indicate index of shared resource to access and "i" identifies the calling thread.

- ***write_to_file (char* filename, int* str, int size)**

The function to write from buffer to the output file.

- **readFromFile (char* filename)**

The function to read value from input to assign in buffer.

- ***Reader (void *arg)**

The function for reading data. First, lock the shared resource to prevent access from other threads. Then loop check whether there is any writer accessing or waiting for the shared resource or not. If there is the reader must wait(sleep) until it receives a signal from a writer.

When the shared resource is available for reading, wake up a reader thread and wait until it is safe. The while loop is used to check whether there is an active reader (AR) or not. After that “lock.Release()” calls to unlock and allowing other threads to access. The function of “AccessDatabase (ReadOnly)” calls for performs the read-only to prevent the editing. If the “AR” counter is zero and there are waiting or writer, the reader will be woken up by the function called “okToWrite” which allow them to write to share resources.

- ***Writer (void *arg)**

The function for writing data. The writer thread performs 10 operations on the resource and waits for any other writers or readers currently accessing or waiting to access the resource to finish. The thread uses a mutex lock and two condition variables to coordinate access to the shared resource. When it's safe to write, the thread increases a “AW”, performs the write operation, and then decreases the “AW”. If there are any waiting writers, it signals one of them to proceed, and if there are waiting readers, it signals all of them to proceed. The writer thread repeats this process until it has completed all its writing operations.

- **removelist (List *root)**

The function to remove the value from list.

- **main (int argc, char* argv [])**

Starts by checking if the correct number of command-line arguments has been provided. If not, it displays an error message and exits the program. If the correct number of arguments has been provided, the function reads in the arguments and initializes the number of writer thread, number of reader thread, seed (a random number), and number of variables. Then initializes a mutex lock and two condition variables “okToRead” and “okToWrite”. Then creates an array of reader threads and an array of writer threads. Following, it reads in the data from a file specified in the command-line arguments and stores it in the shared resource called database. After that creates and starts the reader and writer threads and waits for all the writer threads to finish before waiting for all the reader threads to finish. Finally, the program exits.

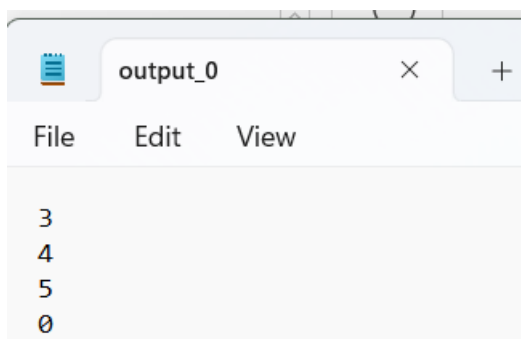
```

PS C:\Users\sinsin\OneDrive\Documents\C\6488110_multithread> ./multithread 1 1 input.txt output_0.txt 2 5
Write 0 DONE
Read 0 DONE
PS C:\Users\sinsin\OneDrive\Documents\C\6488110_multithread> ./multithread 2 2 input.txt output_1.txt 2 5
Write 1 DONE
Write 0 DONE
Read 0 DONE
Read 1 DONE

```

Problem

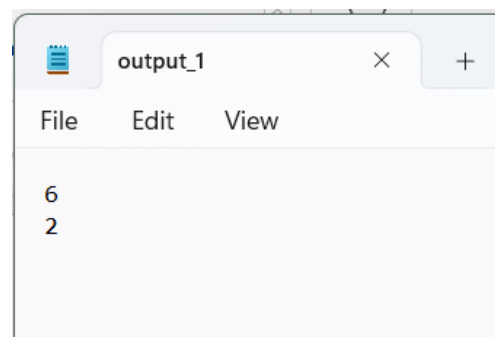
- The number of outputs and value in both output_0.txt and output_1.txt are wrong.



```

3
4
5
0

```



```

6
2

```

Overall

This program implements the process of multi-thread for readers and writers, connecting the threads with synchronize and mutex.

Initially reading the value from the input to storing it in the buffer. If a shared resource (input) is available, all reader threads that are waiting can access a shared resource. If not, the reader thread must wait until shared resource is unlocked. While the writer thread can access only one thread when the shared resource is available and then write into the output file. If not, the writer thread must wait until the shared resource is unlocked. The output will return to the output.txt file and the system will show the complete processes in order of who finishes first.