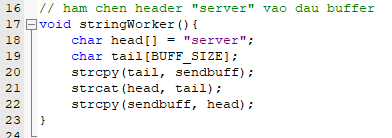
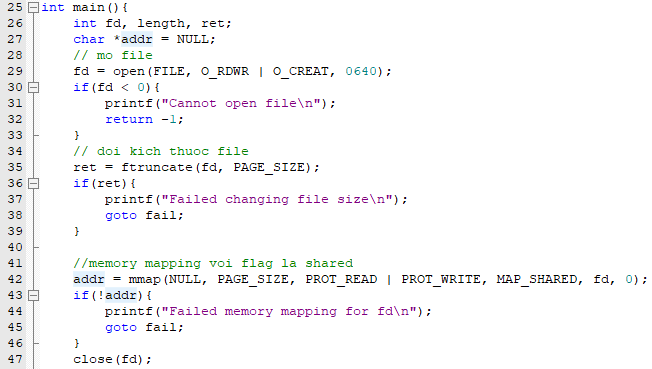
Mã nguồn cả 4 bài tại: <https://drive.google.com/drive/folders/1i-CbYOh4RTVfVpIY-MFIGbesuWqZKrCD?usp=sharing>

Bài 1: TH3.lab1

\*Cả 2 bên server và client đều có hàm chèn header vào đầu buffer và hàm main bao gồm các chức năng mở file, đổi kích thước file, map memory:

header có tác dụng phân biệt ai là người gửi (“client” hay “server”)

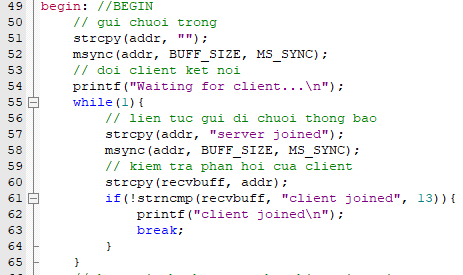




- Tạo vùng mmap với flag MAP\_SHARED cho phép các tiến trình khác cùng truy cập được vùng nhớ (file) này.

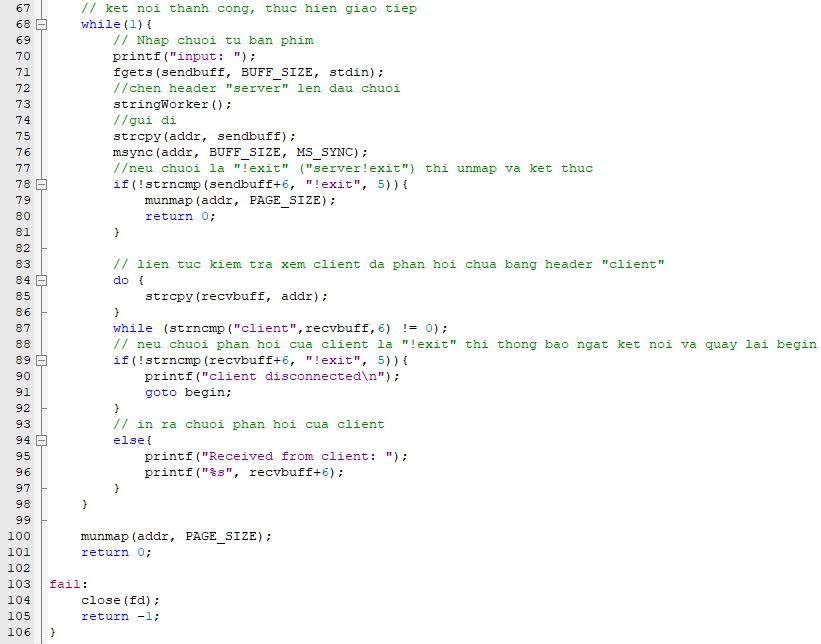
\*Với bên server:

- Việc kết nối bao gồm:



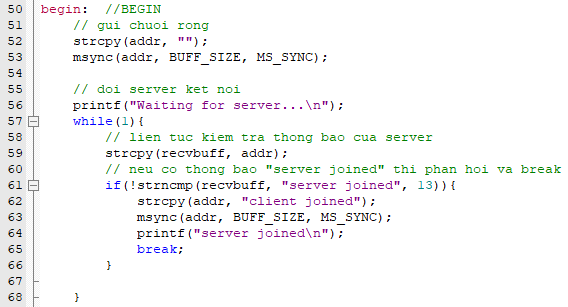
- Mỗi khi gửi chuỗi bằng strcpy() thì phải sử dụng hàm msync để ép tất cả các dữ liệu có thể có trong bộ đệm nhưung chưa được gửi phải gửi ngay lập tức vào con trỏ addr.

- Hàm while:

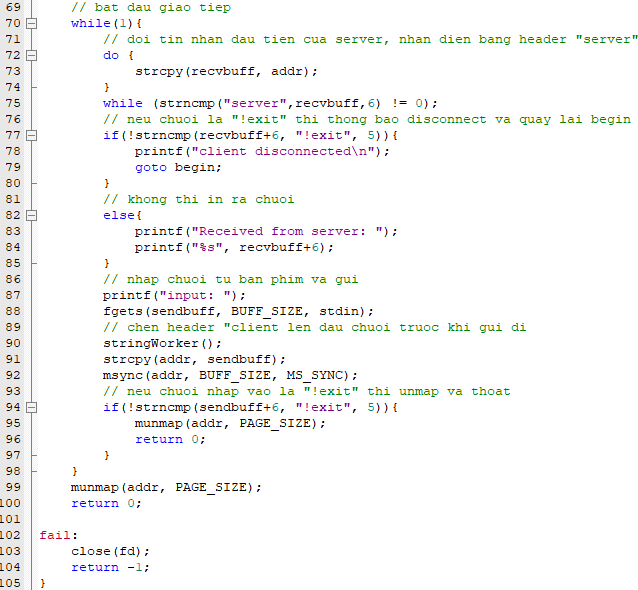


\*Với bên client:

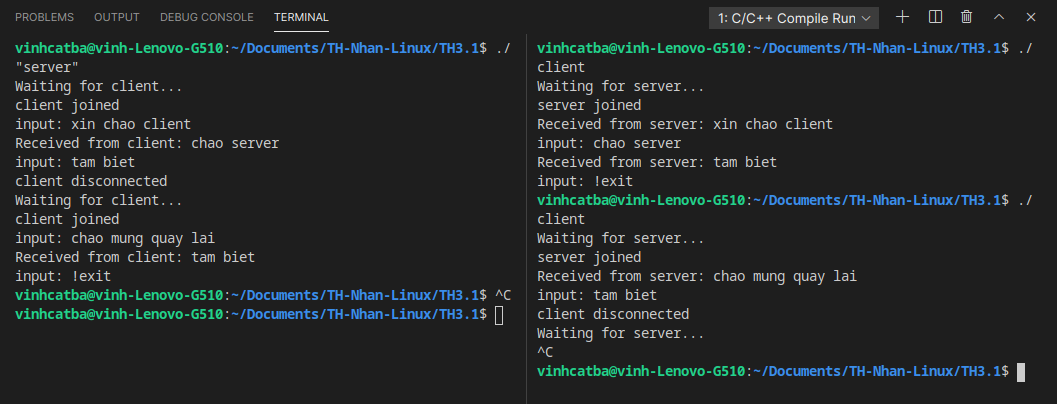
- Việc kết nối bao gồm:



- Hàm while:



\*Kết quả thực hiện:

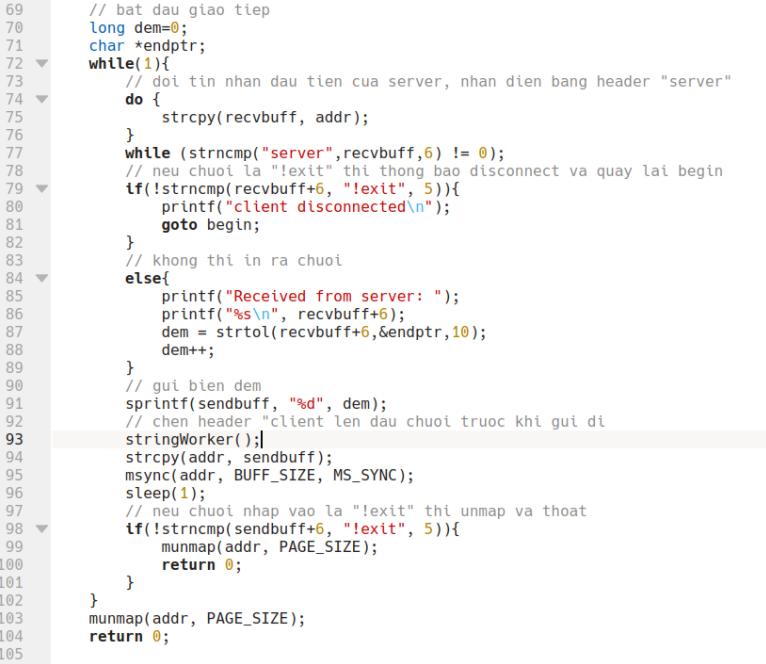


Bài 2: TH3.lab2

\*Về cơ bản bài TH3.lab2 tương tự bài TH3.lab1 bọn em chỉ chỉnh sửa hàm while. Thay vì gửi chuỗi nhập vào từ bàn phím, sử dụng 1 biến đếm. Qua hàm strtol để chuyển đổi chuỗi nhận được từ buffer lưu vào biến đếm. Sau đó tăng biến đếm lên 1 và sử dụng hàm sprintf() để gửi lại biến đếm vào buffer và gửi đi.

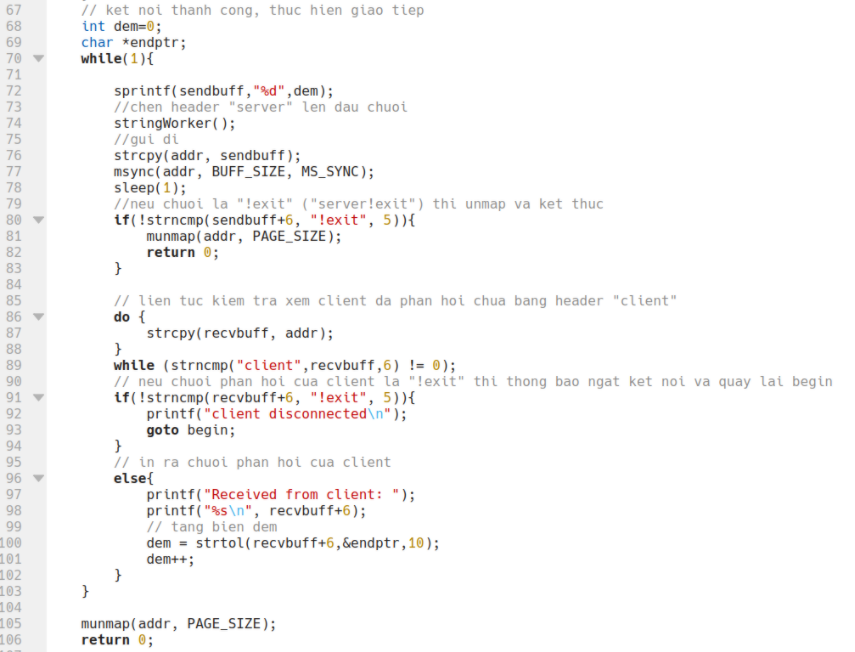
\*Bên phía client:

- Hàm while:



\*Bên server:

- Hàm while:



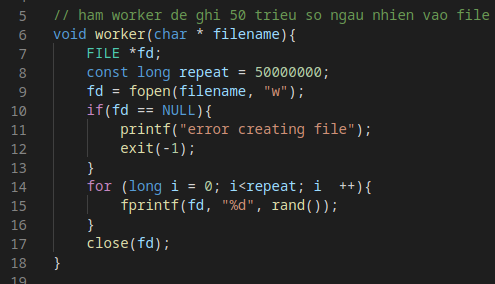
\*Kết quả thực hiện: Biến đếm sẽ được gửi qua lại giữa client và server, mỗi lần gửi lại được tăng lên 1 đơn vị.



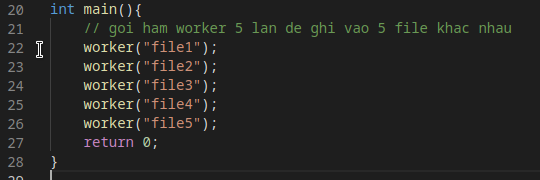
Bài 3: TH4.lab1

\*Không sử dụng thread.

- Hàm woker để mở file và ghi 50 triệu số ngẫu nhiên vào file:

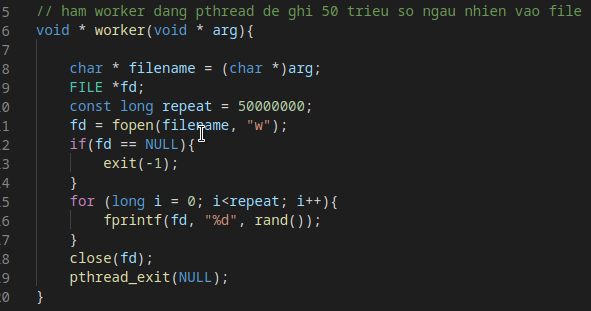


- Hàm main: gọi hàm worker 5 lần để ghi 5 file khác nhau

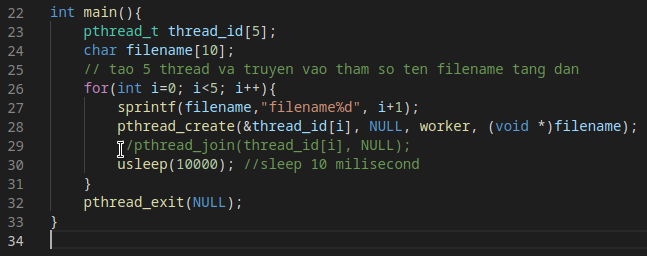


\*Sử dụng thread.

- Hàm woker: nội dung tương tự như bên không dùng thread, nhưng là dưới dạng 1 thread. Ép kiểu tham số arg về (char \*) để lấy tên file khi tạo thread.



-Hàm main: tạo mảng thread\_id 5 phần tử, sau đó dùng vòng lặp for để lặp 5 lần, mỗi lần lặp thì đổi tên file bằng hàm sprintf(), rồi tạo thread và truyền filename vào. Sử dụng usleep(10000) để ngừng 10 milisecond, việc này đảm bảo tất cả các thread đều được tạo. 5 thread được tạo ra sẽ chạy song song với nhau. pthread\_exit(NULL) ở main sẽ đợi đến khi cả 5 thread hoàn tất rồi mới kết thúc.

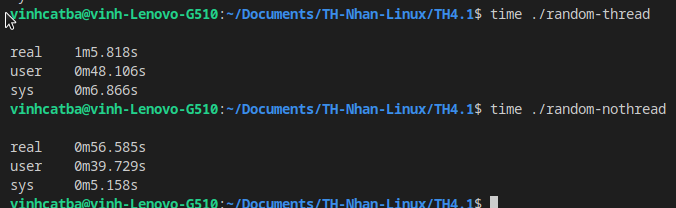


ở đây nếu sau khi tạo thread ta dùng hàm pthread\_join() thì chương trình sẽ đợi đến khi thread vừa tạo thực hiện xong rồi mới chạy vòng lặp tiếp, tức là các thread sẽ được chạy lần lượt thay vì song song.

\*Kết quả giữa việc sử dụng và không sử dụng thread. Ta sử dụng lệnh

time ./tenfile

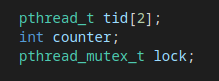
để chạy file với kết quả thời gian thực thi. Kết quả thời gian chạy thực tế (real) cho thấy dùng thread (ở trên) chậm hơn so với không dùng thread (ở dưới)

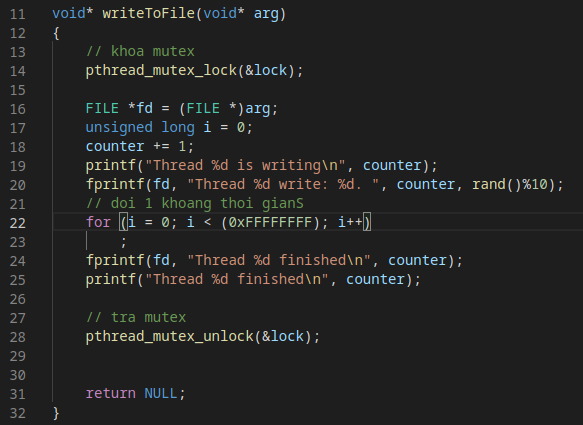


- Như vậy việc sử dụng đa luồng sẽ tốn nhiều thời gian hơn đơn luồng. Lí do là vì đối với bài này, hệ thống thiên về xử lý số liệu nên việc sử dụng đơn luồng sẽ tối ưu hơn. Ổ cứng cũng không thể tối ưu cho việc ghi file đa nguồn, trừ khi sử dụng raid.

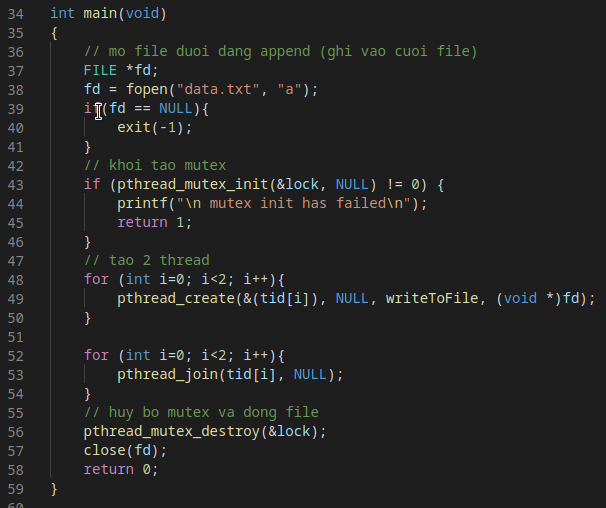
Bài 4: TH4.lab2

đầu tiên tạo biến mutex, biến id thread và 1 biến counter toàn cục

ta có hàm writeToFile dưới dạng thread: Hàm này sẽ khóa mutex rồi ghi vào file một số ngẫu nhiên qua con trỏ fd (được truyền vào từ tham số hàm). Hàm for có tác dụng tăng thời gian chạy (tăng thời gian khóa mutex). Cuối cùng dùng hàm pthread\_mutex\_unlock() để mở khóa mutex.

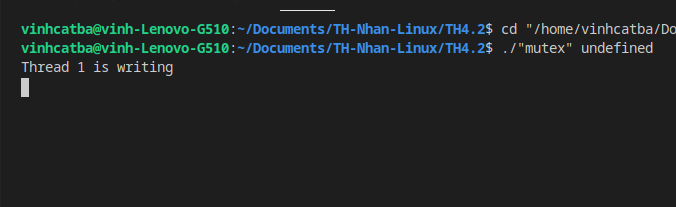


Trong hàm main: tạo con trỏ file fd và mở file. Khởi tạo mutex bằng hàm pthread\_mutex\_init(). Sau đó dùng vòng lặp for để tạo 2 thread cùng dùng chung hàm writeToFile, cùng dùng chung con trỏ file fd. Sau đó đợi cả 2 thread chạy xong thì hủy mutex bằng hàm pthread\_mutex\_destroy().

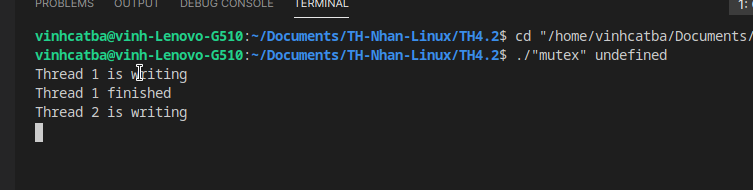
Ở đây mặc dù 2 thread được tạo gần như cùng lúc (thread 2 chỉ sau thread 1 một chút) nhưng do thread 1 đã khóa mutex nên thread 2 bắt buộc phải đợi thread 1 thực hiện hết và mở khóa mutex thì mới chạy được.

Kết quả:

thread 1 đang chạy và khóa mutex nên thread 2 không thể chạy:



thread 1 chạy xong, mở khóa mutex, lúc này thread 2 mới có thể chạy:



Cuối cùng cả 2 chạy xong, nội dung file được ghi:

