7.7.1

Khai báo các biến cần thiết, bao gồm i, shmsize, shmid để lưu trữ thông tin về shared memory, ptr và head để trỏ đến shared memory, value để lưu trữ giá trị, msqid để lưu trữ ID của message queue, và keyx để lưu trữ key được tạo từ đường dẫn file và ID.

Tạo message queue thông qua hàm msgget() với các tham số keyx (key được tạo từ đường dẫn và ID) và 0666 | IPC\_CREAT (quyền truy cập và tạo message queue).

Tạo shared memory thông qua hàm shmget() với các tham số keyx (key được tạo từ đường dẫn và ID), shmsize \* sizeof(int) (kích thước của shared memory), và IPC\_CREAT | 0660 (tạo shared memory và cấp quyền truy cập).

Khai báo các biến cần thiết, bao gồm msqid để lưu trữ ID của message queue, keyx để lưu trữ key được tạo từ đường dẫn file và ID, pid để lưu trữ process ID, pipe\_c2p để lưu trữ pipe từ quá trình con đến quá trình cha, num để lưu trữ số nguyên và readSize để lưu trữ kích thước đọc từ pipe.

Tạo pipe để truyền dữ liệu giữa quá trình con và quá trình cha thông qua hàm pipe().

7.7.2

Kiểm tra nếu pid khác 1, tức là không phải process gốc.

Nếu danh sách liên kết q rỗng, chương trình sẽ gửi tín hiệu SIGUSR1 đến process có pid.

Ngược lại, chương trình sẽ kiểm tra process ID đầu tiên trong danh sách liên kết. Nếu nó trùng với pid, chương trình sẽ gửi tín hiệu SIGUSR1 đến process đó và xóa node tương ứng khỏi danh sách liên kết. Nếu không, chương trình sẽ thêm pid vào danh sách liên kết.

Nếu pid là 1, tức là process gốc:

Nếu danh sách liên kết q không rỗng, chương trình sẽ lấy process ID đầu tiên từ danh sách và gửi tín hiệu SIGUSR1 đến process đó, sau đó xóa node tương ứng khỏi danh sách liên kết.

Khai báo các biến cần thiết, bao gồm pid để lưu trữ process ID, wpid để lưu trữ process ID của tiến trình con, status để lưu trữ trạng thái kết thúc của tiến trình con, readSize để lưu trữ kích thước đọc, msqid để lưu trữ ID của message queue, semid để lưu trữ ID của semaphore, shmid để lưu trữ ID của shared memory, và ptr là con trỏ để trỏ tới vùng nhớ được chia sẻ.

Tạo shared memory thông qua hàm shmget() với các tham số keyx (key được tạo từ đường dẫn và ID), SHIMSIZE \* sizeof(int) (kích thước vùng nhớ chia sẻ), và IPC\_CREAT | 0660 (quyền truy cập và tạo shared memory).

Gắn shared memory vào vùng nhớ của tiến trình thông qua hàm shmat().

Thiết lập giá trị ban đầu cho các biến trong vùng nhớ chia sẻ.

Thiết lập seed cho hàm rand() dựa trên thời gian hiện tại.

Sử dụng vòng lặp để fork các tiến trình con. Số lần lặp được chỉ định là 3. Trong mỗi tiến trình con, gọi hàm doprocessing() để xử lý dữ liệu và chờ một khoảng thời gian ngẫu nhiên trước khi gửi một tin nhắn thông qua message queue.

Hàm doprocessing() được gọi bởi mỗi tiến trình con để xử lý dữ liệu và thực hiện các thao tác trên shared memory và semaphore.

Trước khi thực hiện xử lý, tiến trình con gửi một tin nhắn chứa PID của nó thông qua message queue.

Tiến trình con đăng ký một signal handler cho tín hiệu SIGUSR1. Khi nhận được tín hiệu này, tiến trình sẽ dừng vòng lặp.

Tiến trình con thực hiện các phép toán trên semaphore để khóa và giải phóng semaphore.

Tiến trình con truy cập vào vùng nhớ chia sẻ để thay đổi dữ liệu dựa trên một số điều kiện giả định.

Tiến trình con thông báo qua message queue rằng nhiệm vụ của nó đã hoàn thành.

Hàm sigHandleSigUsr1() được gọi khi tiến trình con nhận được tín hiệu SIGUSR1. Trong trường hợp này, nó chỉ in ra một thông báo và dừng vòng lặp.

9

+ client

Giao tiếp client = server bằng tcp socket

Khai báo các biến cần thiết, bao gồm biến iSock để lưu trữ socket, serverData để lưu trữ thông tin về server, và buffer để lưu trữ dữ liệu gửi và nhận qua socket.

Tạo socket bằng hàm socket()

Kết nối server = hàm connect(), gửi tới server thông qua hàm write(), Sau khi gửi tin nhắn, chương trình đọc phản hồi từ server thông qua hàm read(), và in nội dung phản hồi ra màn hình.

+ server

Khai báo các biến cần thiết, bao gồm biến iSock để lưu trữ socket của server, iClientSock để lưu trữ socket của client, serverData và clientData để lưu trữ thông tin về server và client, iClientSize để lưu trữ kích thước của clientData, và buffer để lưu trữ dữ liệu gửi và nhận qua socket.

Gán thông tin địa chỉ IP và cổng của server cho socket thông qua hàm bind()

6

Định nghĩa cấu trúc TestData gồm một chuỗi ký tự str có kích thước 128, một số nguyên lval và một số thực dval.

Trong hàm main(), khai báo các biến cần thiết.

Mở một file có tên là "MapFile" với các quyền đọc và ghi (O\_RDWR) và tạo file nếu nó chưa tồn tại (O\_CREAT). Nếu việc mở file gặp lỗi, thông báo lỗi và thoát khỏi chương trình.

Xác định kích thước của trang bộ nhớ (page size) bằng cách sử dụng hàm sysconf(\_SC\_PAGE\_SIZE).

Tính toán kích thước cần thiết cho memory map dựa trên số lượng instance của TestData. Kích thước được làm tròn lên đến bội số của trang bộ nhớ.

Sử dụng hàm write() để ghi dữ liệu của mỗi instance TestData vào file. Trước đó, khởi tạo một instance TestData với các giá trị được cung cấp và ghi dữ liệu vào file. Sau đó, di chuyển vị trí con trỏ file sử dụng hàm lseek() để chuẩn bị ghi instance TestData tiếp theo.

In ra thông tin của mỗi instance TestData đã ghi vào file.

Sử dụng hàm mmap() để tạo một memory map và gán con trỏ ptr trỏ đến vùng nhớ được map. Nếu việc tạo memory map gặp lỗi, thông báo lỗi và thoát khỏi chương trình.

Tạo một con trỏ ptr1 trỏ đến đầu của memory map.

Sử dụng vòng lặp để duyệt qua từng instance TestData trong memory map và in ra thông tin của từng instance.

Sử dụng hàm msync() để đồng bộ dữ liệu từ memory map với file tương ứng. Điều này đảm bảo rằng dữ liệu được lưu trữ trong memory map được cập nhật trong file.

Sử dụng hàm munmap() để hủy bỏ memory map. Nếu việc hủy bỏ memory map gặp lỗi, thông báo lỗi.

Đóng file descriptor bằng hàm close().