Báo cáo Week 11 trên lớp - Thực hành Lập trình mạng - Kỳ 2025.1

Nguyễn Khánh Nam - 20225749

1. So sánh poll()/ select() với fork()/pthread(). Khi nào nên sử dụng cái gì? Bài tập của nhóm em nên sử dụng cái gì

- Mô hình Đa tiến trình/Đa luồng (fork()/pthread()):
  - Cơ chế: Với mỗi kết nối (connection) từ client, server sẽ tạo ra một tiến trình (fork) hoặc một luồng (pthread) riêng biệt để xử lý.
  - Ưu điểm: Lập trình tuần tự đơn giản, tận dụng được kiến trúc đa nhân (multi-core) nếu xử lý tính toán nặng.
  - Nhược điểm: Tiêu tốn tài nguyên hệ thống (RAM cho stack, CPU cho context switching) khi số lượng kết nối tăng cao. Khả năng mở rộng (scalability) kém.
- Mô hình I/O Multiplexing (select()/poll()):
  - Cơ chế: Sử dụng một tiến trình duy nhất để giám sát trạng thái của nhiều socket cùng lúc. Tiến trình sẽ bị chặn (block) tại hàm select hoặc poll cho đến khi một trong các socket có sự kiện (sẵn sàng đọc/ghi) hoặc hết thời gian chờ.
  - Ưu điểm: Tiết kiệm tài nguyên, không cần tạo nhiều tiến trình/luồng, giảm thiểu chi phí chuyển ngữ cảnh (context switching).
  - Nhược điểm: Phức tạp hơn trong việc quản lý trạng thái, không tận dụng được đa nhân CPU cho một tác vụ đơn lẻ.

Bài tập lớn: Game Server hoặc Chat Server với logic xử lý không quá nặng về CPU nhưng cần duy trì nhiều kết nối), sử dụng select() hoặc poll().

2. Theo bạn poll() hay select() được coi là hiệu quả hơn trong các hệ thống xử lý I/O nhiều kết nối đồng thời? Hãy giải thích chi tiết cơ chế nội bộ của từng hàm và ví dụ minh họa trong trường hợp có 5000 socket đang hoạt động.

Nhận định: Trong trường hợp có 5000 socket, poll() hiệu quả hơn về mặt khả thi, nhưng cả hai đều có hạn chế về hiệu năng so với các cơ chế hiện đại hơn (như epoll). Tuy nhiên, xét trong phạm vi so sánh giữa hai hàm này: poll() vượt trội hơn select().

Giải thích cơ chế và Ví dụ minh họa:

- Cơ chế của select():
  - Sử dụng cấu trúc dữ liệu fd_set là một mảng các bit (bitmap) để đại diện cho các file descriptor (FD).
  - Kích thước của fd_set bị giới hạn bởi hằng số FD_SETSIZE, mặc định thường là 1024.
  - Với 5000 socket: select() mặc định không thể hoạt động vì vượt quá giới hạn 1024 của FD_SETSIZE. Để chạy được, lập trình viên phải biên

dịch lại nhân hoặc thư viện với FD_SETSIZE lớn hơn, điều này không chuẩn hóa và rủi ro. Ngoài ra, độ phức tạp thuật toán là O(N) vì kernel phải quét qua toàn bộ bitmap để tìm socket kích hoạt.

Cơ chế của poll():

- Sử dụng một mảng các cấu trúc struct pollfd.
- Không có giới hạn cứng về số lượng file descriptor giám sát, ngoại trừ giới hạn bộ nhớ hệ thống và giới hạn số lượng file mở của tiến trình (ulimit).
- Với 5000 socket: poll() hoàn toàn có thể hoạt động bằng cách khởi tạo mảng pollfd có kích thước 5000.
- Hiệu năng: Tuy nhiên, mỗi lần gọi poll(), hệ thống phải sao chép toàn bộ mảng 5000 cấu trúc này từ không gian người dùng (User Space) sang không gian nhân (Kernel Space) và ngược lại, gây lãng phí băng thông bộ nhớ.

3. Khi đặt giá trị timeout trong select() và poll(), hệ thống xử lý như thế nào để đảm bảo đúng thời gian chờ? Sinh viên hãy giải thích chi tiết cơ chế xử lý timeout ở cấp độ kernel và ảnh hưởng của việc đặt giá trị timeout không hợp lý (ví dụ: quá lớn hoặc bằng 0).

Cơ chế tại cấp độ Kernel: Khi ứng dụng gọi select() hoặc poll() với tham số timeout, một System Call (lời gọi hệ thống) được thực hiện:

1. Chuyển trạng thái: Tiến trình chuyển từ trạng thái RUNNING sang trạng thái WAITING (hoặc INTERRUPTIBLE SLEEP). Tiến trình được đưa ra khỏi hàng đợi lập lịch của CPU (CPU Runqueue).
2. Đăng ký Timer: Kernel thiết lập một bộ định thời (software timer) tương ứng với giá trị timeout được truyền vào.
3. Đánh thức (Wake-up): Tiến trình sẽ được đánh thức và chuyển về trạng thái READY khi một trong hai điều kiện sau xảy ra:
    - Có sự kiện I/O trên ít nhất một socket được giám sát (ngắt phần cứng từ Network Interface Card).
    - Bộ định thời hết hạn (Timer expiration).

Ảnh hưởng của giá trị Timeout:

- Timeout = NULL (hoặc số âm trong poll):
    - Tiến trình sẽ đợi vô hạn (Wait forever). Nếu không có sự kiện I/O nào, tiến trình sẽ bị treo vĩnh viễn, có thể dẫn đến tình trạng "Deadlock" nếu logic chương trình phụ thuộc vào phản hồi mạng mà mạng bị ngắt.
- Timeout = 0:
    - Hàm trả về ngay lập tức (Return immediately). Đây là kỹ thuật Polling (thăm dò).
    - Tác động: Nếu đặt trong vòng lặp while, CPU sẽ phải hoạt động liên tục (Busy-waiting) để kiểm tra trạng thái, gây tiêu tốn 100% CPU vô ích.

- Timeout > 0 (Hợp lý):
    - Cho phép chương trình có độ trễ cho phép để đợi dữ liệu, nhưng vẫn định kỳ lấy lại quyền điều khiển CPU để thực hiện các tác vụ bảo trì khác (như kiểm tra kết nối, gửi heartbeat).

4. fd_set sử dụng một tập hợp các macro (FD_SET, FD_CLR, FD_ISSET, FD_ZERO) để quản lý file descriptor. Sinh viên hãy giải thích cơ chế hoạt động chi tiết của fd_set, bao gồm cách kernel kiểm tra trạng thái của các descriptor trong tập hợp này. Hãy viết code minh họa và phân tích bộ nhớ khi FD_SETSIZE = 1024.

Cơ chế hoạt động của fd_set:

fd_set là một cấu trúc dữ liệu trừu tượng, nhưng về bản chất kỹ thuật, nó là một mảng các bit (bitmask). Các macro thao tác trên bitmask này như sau:

1. FD_ZERO: Xóa toàn bộ các bit về 0.
2. FD_SET: Bật bit tại vị trí tương ứng với file descriptor fd lên 1.
3. FD_CLR: Tắt bit tại vị trí fd về 0.
4. FD_ISSET: Kiểm tra xem bit tại vị trí fd đang là 0 hay 1.

Cách Kernel kiểm tra trạng thái: Khi hàm select(maxfd, ...) được gọi:

1. Kernel sao chép fd_set từ User Space vào Kernel Space.
2. Kernel quét các bit từ 0 đến maxfd - 1. Nếu bit thứ i được bật, Kernel sẽ kiểm tra bộ đệm của socket tương ứng.
3. Quan trọng: Trước khi trả về, Kernel sẽ ghi đè lại fd_set. Chỉ những bit ứng với socket *thực sự có sự kiện* mới được giữ là 1, các bit khác sẽ bị reset về 0. Do đó, lập trình viên phải khởi tạo lại fd_set trước mỗi lần gọi select.

Phân tích bộ nhớ khi FD_SETSIZE = 1024:

Mỗi file descriptor được đại diện bởi 1 bit. -> Tổng số bit cần thiết: 1024 bits = 128 bytes

Code minh họa:

```
#include <sys/select.h>
#include <stdio.h>

int main() {
    fd_set readfds;

    // 1. Phân tích bộ nhớ
    // Kích thước của fd_set phụ thuộc vào FD_SETSIZE (thường là 1024)
    // 1024 bit / 8 = 128 bytes
    printf("Kich thuoc cua fd_set: %lu bytes\n", sizeof(fd_set));

    // 2. Minh họa thao tác bit
    FD_ZERO(&readfds); // Reset toàn bộ 1024 bit về 0

    // Giả sử muốn giám sát socket có FD = 5
    FD_SET(5, &readfds);
    // Lúc này, bit thứ 5 trong mảng nhớ (tính từ 0) được bật lên 1.
    // Trạng thái bitmask (đơn giản hóa): 00000100...000

    // Kiểm tra
    if (FD_ISSET(5, &readfds)) {
        printf("Socket 5 da duoc them vao tap hop giam sat.\n");
    }
    return 0;
}
```