ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC HÀNH HỆ ĐIỀU HÀNH

| CHỦ ĐỀ |

SOCKET

| GIẢNG VIÊN HƯỚNG DẪN |

Cô Vũ Thị Mỹ Hằng

Thầy Lê Quốc Hòa

| SINH VIÊN THỰC HIỆN |

21120302 – Huỳnh Trí Nhân

Thành phố Hồ Chí Minh – 2024

MỤC LỤC

1 ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

1.1 Đề tài

Viết chương trình Socket 1 server – n client thực hiện yêu cầu sau:

- Client gửi yêu cầu để Server tính toán số nguyên (chỉ cần các phép cơ bản +, -, *, /) và trả kết quả về cho client.
- Nếu client nhập sai cú pháp thì server gửi thông báo không hợp lệ về cho client.

Ví dụ:

- Client gửi 1 + 2. Server sẽ tính toán và trả về kết quả cho client là 3
- Client gửi (1 + 2)*3. Server sẽ tính toán và trả về kết quả cho client là 9

1.2 Đánh giá mức độ hoàn thành

Yêu cầu	Tự đánh giá
Khởi tạo server	100%
Khởi tạo client	100%
Muti-Threading: 1 server và nhiều client	100%
Giao tiếp client-server	100%
Xử lý phép toán với số nguyên	100%
Thực hiện chương trình trên GCC của hệ điều hành Linux	100%

2 HƯỚNG GIẢI QUYẾT PHẦN LẬP TRÌNH

2.1 Thuật toán đề xuất

2.1.1 Thiết kế concurrent server cho đa tiến trình khác sử dụng fork()

- Lệnh gọi Fork() tạo nhiều tiến trình con cho các máy khách đồng thời và chạy từng khối lệnh gọi trong khối điều khiển tiến trình (PCB) riêng của nó.

```
int cnt = 0;
// Tạo ra một tiến trình con để xử lý kết nối của trình khách
pid_t childpid;
while (1)
{
    printf("server waiting...\n");
    /* Chờ và chấp nhận kết nối */
    client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_address, &client_len);
    if (client_sockfd < 0)
    {
        printf("Error in accepting.\n");
        exit(1);
    }
    // In ra thông tin kết nối đã được thiết lập
    printf("\nConnection accepted from %s:%d\n",
           inet_ntoa(client_address.sin_addr),
           ntohs(client_address.sin_port));

    // In ra số lượng kết nối đã được thiết lập
    printf("Clients connected: %d\n\n",
           ++cnt);

    if ((childpid = fork()) == 0)
    {
        // Đóng socket chính của server
        close(server_sockfd);
        // Xử lý kết nối của trình khách
        handle_client(client_sockfd);
        exit(0);
    }
    /* Đóng kết nối */
    close(client_sockfd);
}
}
```

Hình 1 Concurrent Server với fork()

Ý tưởng của em là thiết kết đoạn mã có thể lắng nghe liên tục các kết nối từ nhiều client khác nhau. Một client có thể giao tiếp liên tục đến với server. Do đó em đã thiết kết đoạn code code trên với những chức năng sau:

1. Lắng Nghe Kết Nối Liên Tục: Server sử dụng một vòng lặp vô tận (while (1)) để liên tục lắng nghe các kết nối đến. Điều này đảm bảo rằng máy chủ luôn sẵn sàng chấp nhận kết nối mới từ các clients.
2. Chấp Nhận Kết Nối:

- Hàm accept() chờ đợi và chấp nhận một kết nối mới từ client. Nếu có lỗi, chương trình sẽ in ra thông báo lỗi và thoát.
- Khi một kết nối được chấp nhận, máy chủ in ra thông tin của kết nối đó, bao gồm địa chỉ IP và cổng của client.

3. Sử dụng fork() để tạo ra tiến trình con: Sau khi chấp nhận một kết nối, máy chủ gọi fork() để tạo một tiến trình con. Hàm fork() trả về 0 trong tiến trình con và trả về ID của tiến trình con trong tiến trình cha.
4. Xử Lý Kết Nối Trong Tiến Trình Con:

- Nếu fork() trả về 0, tức là đang ở trong tiến trình con, máy chủ đóng socket chính (server_sockfd) vì nó không cần thiết cho việc xử lý kết nối hiện tại.
- Tiến trình con sau đó gọi hàm handle_client() để xử lý kết nối đó, sau đó thoát.

5. Tiến Trình Cha Đóng Kết Nối:Trong tiến trình cha, sau khi fork(), socket của client được đóng (close(client_sockfd)). Điều này không ảnh hưởng đến tiến trình con vì mỗi tiến trình có bản sao riêng của các tài nguyên hệ thống.

2.1.2 Xử lý các phép tính toán

- Thuật toán em quyết định sử dụng trong bài này là thuật toán Shutting Yard. Biến biểu thức trung tố mà client nhập vào trờ thành một biểu thức hậu tố. Sau đó thực hiện tính toán trên biểu thức đó.

Here is the psedocode of the algorithm:

1. For all the input tokens:
    1. Read the next token
    2. If token is an operator (x)
        1. While there is an operator (y) at the top of the operators stack and either (x) is left-associative and its precedence is less or equal to that of (y), or (x) is right-associative and its precedence is less than (y)
            1. Pop (y) from the stack
            2. Add (y) output buffer
        2. Push (x) on the stack
    3. Else if token is left parenthesis, then push it on the stack
    4. Else if token is a right parenthesis
        1. Until the top token (from the stack) is left parenthesis, pop from the stack to the output buffer
        2. Also pop the left parenthesis but don't include it in the output buffer
    5. Else add token to output buffer
2. Pop any remaining operator tokens from the stack to the output

Hình 2 Thuật toán Shutting Yard

Let's take a small example and see how the pseudocode works. Here is the infix expression to convert:

4 + 4 * 2 / (1 - 5)

The following table describes the precedence and the associativity for each operator. The same values are used in the algorithm.

Operator	Precedence	Associativity
^	10	Right
*	5	Left
/	5	Left
+	0	Left
-	0	Left

Here we go:

Token	Action	Output	Operator stack
4	Add token to output	4	
+	Push token to stack	4	+
4	Add token to output	4 4	+
*	Push token to stack	4 4	* +
2	Add token to output	4 4 2	* +
/	Pop stack to output, Push token to stack	4 4 2 *	/ +
(	Push token to stack	4 4 2 *	(/ +
1	Add token to output	4 4 2 * 1	(/ +
-	Push token to stack	4 4 2 * 1	- (/ +
5	Add token to output	4 4 2 * 1 5	- (/ +
)	Pop stack to output, Pop stack	4 4 2 * 1 5 -	/ +
end	Pop entire stack to output	4 4 2 * 1 5 - / +	

Hình 3 Minh họa thuật toán

- Sau khi có được biểu thức hậu tố, em tiến hành tính toán trên biểu thức đó. Tiên hành push từng số vào stack cho đến khi gặp dâu thì pop 2 giá trị của stack để tính toán rồi lại push vào stack. Cứ tiếp tục cho đến khi stack còn lại con số cuối cùng, đó chính là kết quả.

- Từ đó em định nghĩa hàm xử lý yêu cầu của client với các bước như sau:

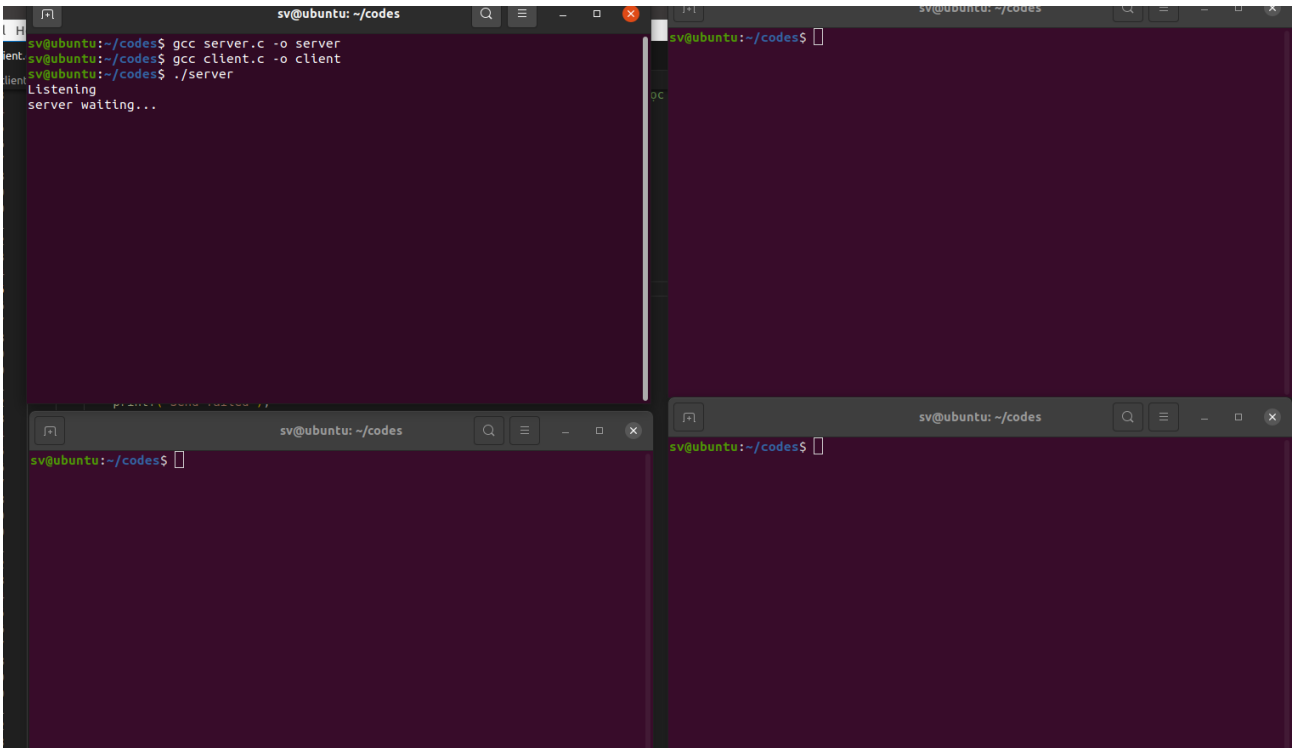
    - Tạo một recv với vòng (while()) để luôn nhận được thông điện từ client
    - Sau đó trả qua hàm checkformat xem có đúng định dạng không. Nếu đúng đưa qua bước kết tiếp. Nếu không đúng thì trả về thông báo cho client.
    - Chuyển đoạn infix thành postfix bằng thuật toán đã nêu trên.
    - Tính toán trên postfix và trả về kết quả cho client

- Cuối mỗi lần giao tiếp thì clear những buffer dữ

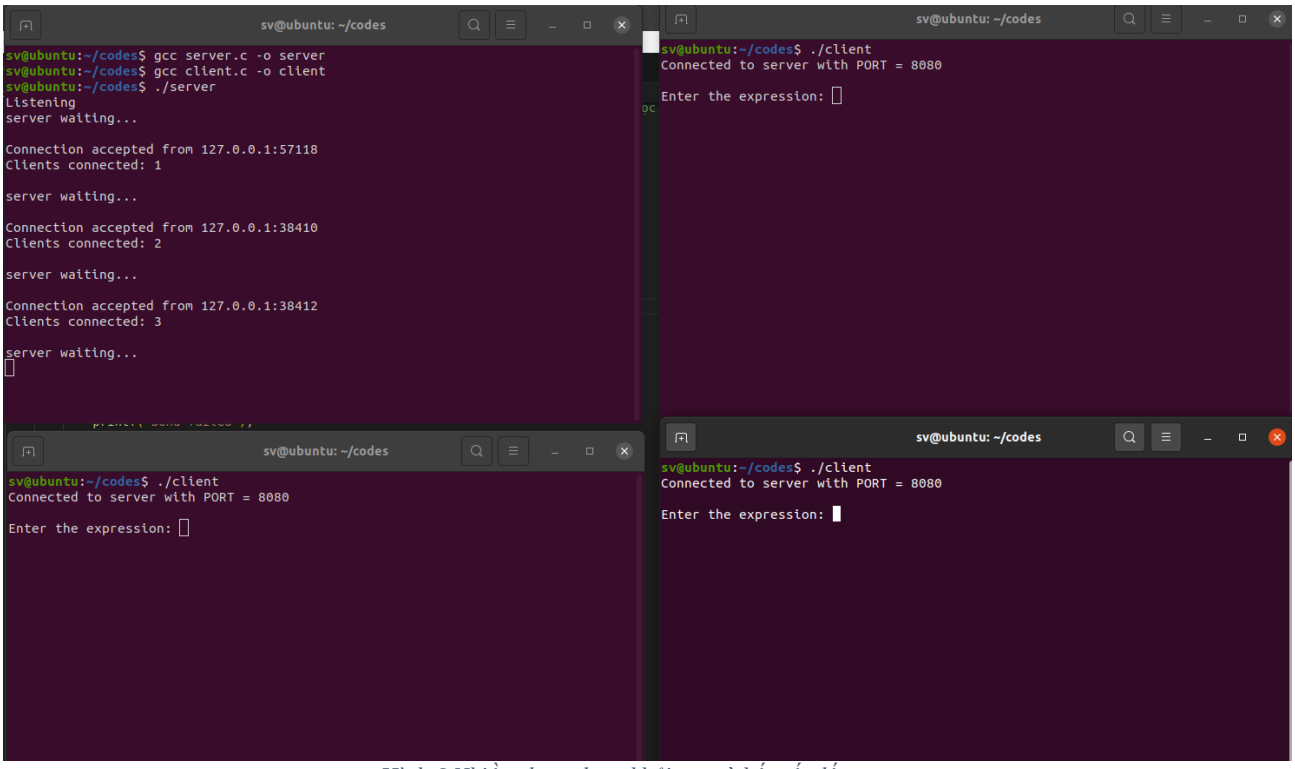
```c
// Xử lý yêu cầu của client trả về kết quả phép tính
void handle_client(int sock)
{
    int read_size;
    char client_message[2000];
    // Nhận tin nhắn từ client và gửi lại
    while ((read_size = recv(sock, client_message, 2000, 0)) > 0)
    {
        int result = 0;
        char postfix[2000];
        char infix[2000];
        char wrongformat[] = "Wrong format";
        // In ra tin nhắn từ client
        printf("\nClient: %s\n", client_message);

        strcpy(infix, client_message);

        // kiểm tra lỗi format
        if (!checkformat(infix))
        {
            sprintf(client_message, "%s", wrongformat);
        }
        else
        {
            // Xử lý phép tính
            ConvertInfixToPostfix(infix, postfix);
            printf("Postfix Converted: %s\n", postfix);
            result = calculatePostfix(postfix);
            sprintf(client_message, "%d", result);
        }
        // In ra tin nhắn đã được gửi lại cho client
        printf("Server: %s\n", client_message);
        write(sock, client_message, strlen(client_message));
        // Làm sạch buffer trước mỗi lần đọc
        memset(client_message, 0, sizeof(client_message));
    }
```

Hình 4 Xử lý thông điệp của client

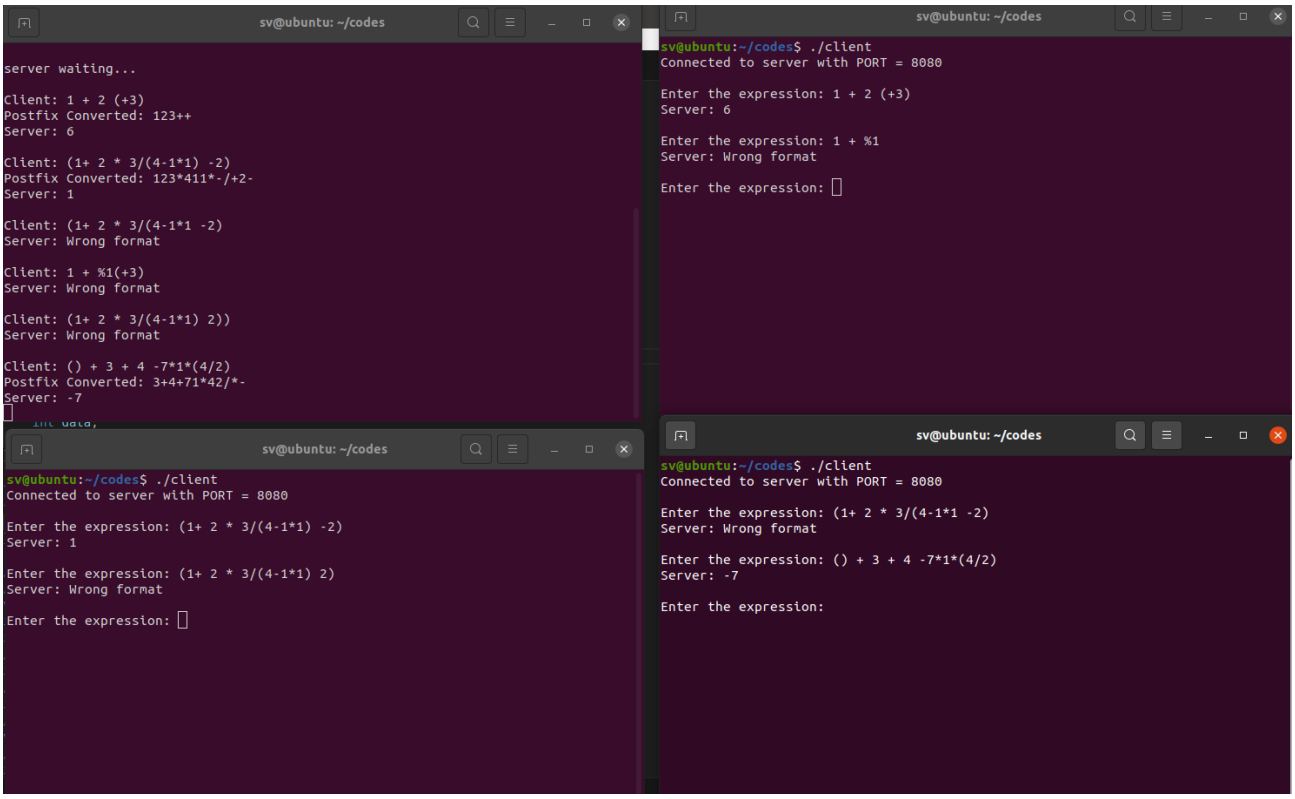
2.2 Kết quả chạy chương trình



Hình 5 Biên dịch và khởi tạo Server



Hình 6 Nhiều client được khởi tạo và kết nối đến server



Hình 7 Thực hiện nhiều yêu cầu từ client cùng lúc

- Em đã gửi phép tính lần lượt từ các client khác nhau để xem xét xem server có xử lý đồng thời được không
- Kết quả nhận được :
    - Server kết nối được nhiều client cùng lúc cho dù giao tiếp với tiến trình khác nhưng vẫn xử lý được tác vụ từ một tiến trình khác gửi đến tức là vẫn giữ kết nối với 3 client trong suốt quả trình hoạt động.

PHỤ LỤC

Thư mục 21120302

- File **21120302_client.c và 21120302_server.c** : chứa mã nguồn chương trình em đã cài đặt.
- File **21120302.pdf** chứa báo cáo bài thực hành.
- Kết quả chạy chương trình chạy trên **hệ điều hành Linux của Ubuntu bản 20.04.6.**
- Sử dụng editor VSCode có sẵn trên Ubuntu.

TÀI LIỆU THAM KHẢO

- Giáo trình Hệ Điều Hành HCMUS
- Tài liệu thực hành lớp Hệ Điều Hành 21_4
- Design a concurrent server for handling multiple clients using fork() - GeeksforGeeks
- Shunting Yard (aquarchitect.github.io)