TCP, 파이프 활용 채팅 프로그램

작업기간 :2024.09.11 ~ 2024.09.13

VEDA A반 김시후

프로젝트 개요

프로젝트 개요

▶ **TCP를 이용한 채팅 프로그램 작성**

- 서버와 클라이언트 모델 → 서버와 클라이언트 모두 프로그래밍

▶ **서버는 멀티 프로세스(fork()) 사용**

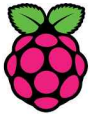
- 부모 프로세스와 자식 프로세스들 사이에 IPC(pipe만) 사용
- 채팅 서버는 데몬(백그라운드)으로 등록

▶ **구현할 기능**

- 채팅 서버(라즈베리 파이) / 클라이언트(우분투) 구현
- 서버와 클라이언트는 소켓으로 통신
- 채팅방 기능 : 로그인/로그아웃 기능
- 빌드 시스템은 make/cmake를 이용

▶ **제약 사항**

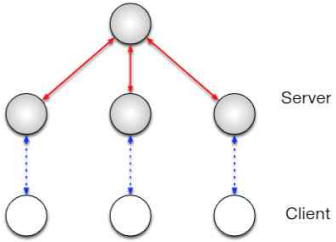
- select()/epoll() 함수 사용 불가
-메시지 큐나 공유 메모리 사용 불가



전체 파일 설명

파일명	설명
Makefile	전체 .c/.h 파일 빌드용 Makefile이다.
Macros.h	프로그램 전체에서 사용하는 매크로가 정의되어 있다. 서버 주소, 컴파일 시 서버/클라이언트 유무도 설정할 수 있다.
main.c	서버용, 클라이언트용 main 함수가 나뉘어 정의되어 있다. Macros.h 파일에서 서버용인지 클라이언트용인지 설정할 수 있다.
Server.h **Server.c**	서버에서 사용하는 구조체와 함수들의 정의/구현되어있다.
Client.h **Client.c**	클라이언트에서 사용하는 구조체와 함수들의 정의/구현되어있다.

구현 방법 개요



▶ 전체 구조

- 서버와 클라이언트는 TCP로 통신한다.
- 서버 및 클라이언트 내 프로세스간 통신은 pipe로 이뤄진다.

▶ 서버

- 서버는 **서버 소켓** 하나와 **클라이언트 소켓들**을 가진다.
- 서버는 **클라이언트가 연결될 때마다** 통신용 담당 프로세스를 할당하며(fork()), 종료되면 sigaction() 함수를 기반으로 프로세스를 회수한다.(좀비 프로세스 방지)
- 서버의 메인 프로세스는 **클라이언트 추가 연결** 및 **전체 메시지 취합/전송**을 담당한다.

▶ 클라이언트

- 클라이언트가 실행되면 서버에 접속을 시도한다.
- 접속에 성공하면 클라이언트는 2개의 프로세스를 진행한다. **부모 프로세스**는 **서버와의 통신**을 담당하고, **자식 프로세스**는 **사용자의 입력**을 담당한다.

클라이언트

클라이언트 main.c 구조

1. **C_Init()** : **클라이언트 초기화**
- 프로그램에 필요한 값들을 초기화한다.

2. **C_Connect()** : **서버 연결**
- 매크로에 정의된 주소로 서버 연결을 시도한다.
- 서버 연결에 실패할 경우 프로그램을 종료한다.

3. **C_ClientService()** : **채팅 서비스**
- 로그인, 로그아웃, 채팅 등의 기능을 제공한다.

4. **C_Close()** : **서버 연결 종료**
- 서버 연결을 종료하고 프로그램을 종료한다.

```c
#include "Macros.h"

#ifdef CLIENT_MODE
#include "Client.h"
int main()
{
    C_init();           // 클라이언트 초기화
    C_Connect();        // 서버 연결
    C_ClientService();  // 서비스
    C_Close();          // 서버 연결 종료
    return 0;
}
#endif
```

클라이언트 주요 함수(1) : C_ClientService()

- 부모/자식간 양방향 통신을 위해 파이프 2개 연결
- 버퍼 초기화 및 함수 인자 전달
- fork() 후 자식은 사용자 입력, 부모는 서버와의 통신을 담당

```
void C_ClientService() {
    // pipe1 : 자식->부모 / pipe2 : 부모->자식
    int pipe1[2], pipe2[2];
    c_ConnectPipe(pipe1);
    c_ConnectPipe(pipe2);

    char buffer[BUFFER_SIZE]; // 버퍼 초기화
    memset(buffer, 0, BUFFER_SIZE);

    pid_t pid = fork();
    if (pid < 0) { // fork 실패
        perror("fork failed");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0) { // 자식 프로세스: 사용자 입력 처리
        c_ChildProcess(pipe1, pipe2, buffer);
    }
    else {  // 부모 프로세스: 서버와의 통신 처리
        c_ParentProcess(pipe1, pipe2, buffer);
    }
}
```

클라이언트 주요 함수(2) : c_ChildProcess()

- 부모가 사용하는 파이프는 닫고 시작
- 로그인이 되지 않았다면 로그인부터 실행
- 로그인이 되어있다면 버퍼에 입력 후 부모로 송신
- 종료 시 자식용 파이프 닫기

```
void c_ChildProcess(int pipe1[2], int pipe2[2], char* buffer) {
    close(pipe1[0]);  // 부모용 읽기 파이프를 닫음
    close(pipe2[1]); // 부모용 쓰기 파이프 닫기

    while (1) {
        if(User.isLoginned == 0) {  // 아직 로그인 안됨
            c_LoginSystem(pipe1, pipe2, buffer);
            continue;
        }
        fgets(buffer, BUFFER_SIZE, stdin);
        buffer[strcspn(buffer, "\n")] = '\0'; // 줄바꿈 제거

        if (strcmp(buffer, "exit") == 0) {
            printf("종료\n");
            write(pipe1[1], "exit", strlen("exit"));
            break;
        }

        if (strcmp(buffer, "logout") == 0) {
            printf("로그아웃\n");
            write(pipe1[1], "logout", strlen("logout"));
            User.isLoginned = 0;
            continue;
        }

        c_AddId(buffer);     // 버퍼에 id 추가
        // 사용자 입력을 파이프로 부모에게 전달
        write(pipe1[1], buffer, strlen(buffer));
        memset(buffer, 0, BUFFER_SIZE);
    }
    close(pipe1[1]);
    close(pipe2[0]);
    exit(0);
}
```

클라이언트 주요 함수(3) : c_ParentProcess()

- 자식이 사용하는 파이프는 닫고 시작
- 채팅을 지속적으로 확인하고, 메시지 수신도 지속적으로 확인하기 위해 non-block 지정
- 로그인이 되지 않았다면 로그인용 통신부터 실행
- 로그인이 되어있다면 자식의 메시지 받고 서버에 송신
- 서버로부터 메시지 수신 후 출력
- 종료 시 부모용 파이프 닫기

```
void c_ParentProcess(int pipe1[2], int pipe2[2], char buffer[BUFFER_SIZE]) {
    close(pipe1[1]);  // 부모는 쓰기 파이프를 닫음
    close(pipe2[0]);    // 자식이 읽는 파이프 닫기
    c_MakeNonblock(pipe1[0]);        // 지속적인 입력 확인용
    c_MakeNonblock(User.sockfd);     // 지속적인 수신 확인용

    while (1) {
        if(User.isLoginned == 0) { // 로그인이 안되었다면
            // 로그인용 통신 진행
            c_LoginCommunication(pipe1, pipe2, buffer);
            continue;
        }

        // 파이프에서 사용자 입력 읽기
        int readResult = c_ReadUserInput(pipe1, pipe2, buffer);
        if(readResult == 1) break; // 프로세스 종료
        if(readResult == 2) continue; // 로그아웃

        // 서버로부터의 메시지 수신 처리
        c_ReceiveMessageFromServer(pipe1, pipe2, buffer);
    }
    close(pipe1[0]);
    close(pipe2[1]);
}
```

서버

서버 main.c 구조

1. **S_Init()** : **서버 초기화**

- 프로그램에 필요한 값들을 초기화한다.

2. **S_ClientService()** : **서버 활성화**

- 채팅 서버를 활성화한다. 클라이언트들이 접속할 수 있으며, 회원가입/로그인/로그아웃을 통해 채팅방에 접근하고, 채팅할 수 있다.
- 클라이언트 접속 시 해당 클라이언트의 **통신 담당 프로세스가 fork()** 된다.
- 클라이언트가 종료되면 sigaction 기반 함수로 자식 프로세스를 자동으로 회수한다.(좀비 프로세스 방지)

3. **S_Close()** : **서버 종료**

- 프로그램을 종료한다.

```c
#include "Macros.h"

#ifdef SERVER_MODE
#include "Server.h"
int main()
{
    S_Init();           // 서버 초기화
    S_ServerService();  // 서버 서비스
    S_Close();          // 서버 종료
    return 0;
}
#endif
```

서버 주요 함수(1) : S_Init()

▶ 서버의 데몬화, 멤버 변수 초기화, 소켓/파이프 초기화, 서버 주소 설정, 서버 주소-소켓 연결, 클라이언트 listen 등 서버에서 초기에 필요한 작업들을 모아둔 함수

```
void S_Init() {
    s_MakeDaemon();          // (1) 서버 프로그램을 데몬 프로세스로
    s_InitMembers();         // (2) 서버 멤버변수 초기화
    s_InitClientSocket();    // (3) 클라이언트용 소켓 초기화
    s_InitServerSocket();    // (4) 서버 소켓 초기화
    s_InitPipes();           // (5) 파이프들 초기화
    s_SetServerAddress();    // (6) 서버 주소 설정
    s_BindServerSocket();    // (7) 서버 주소 - 서버 소켓 연결
    s_ListenClients();       // (8) 클라이언트 Listen 등록
}
```

서버 주요 함수(2) : S_ServerService()

▶ 클라이언트가 접속하면 그 클라이언트를 담당하는 프로세스를 fork()

▶ fork() 후 다른 명령어들을 수행하므로 Sigaction을 활용해 자식 프로세스 종료 수집 (좀비 프로세스 발생 방지)

▶ 수신된 메시지가 있다면 메시지를 보낸 클라이언트 제외, 나머지에 송신

```
void S_ServerService() {
    s_SetSaHandler(); // 자식 프로세스 종료 시 자동 회수
    while(1) {
        int new_socket = s_AcceptNewSocket(); // 클라이언트 연결 -> 새 소켓 생성
        if(new_socket > 0) {    // 새 소켓 받아오기 성공 시
            int idx = s_UpdateNewSocket(new_socket);    // 등록
            s_ForkForClientMessage(idx);    // 해당 클라이언트용 추가 프로세스 실행
        }
        s_Broadcast();  // 전체 방송
    }
}
```

서버 주요 함수(2) : S_ServerService()

▶ 클라이언트가 접속하면 그 클라이언트를 담당하는 프로세스를 fork()

▶ fork() 후 다른 명령어들을 수행하므로 Sigaction을 활용해 자식 프로세스 종료 수집 (좀비 프로세스 발생 방지)

▶ 수신된 메시지가 있다면 메시지를 보낸 클라이언트 제외, 나머지에 송신

```
void S_ServerService() {
    s_SetSaHandler(); // 자식 프로세스 종료 시 자동 회수
    while(1) {
        int new_socket = s_AcceptNewSocket(); // 클라이언트 연결 -> 새 소켓 생성
        if(new_socket > 0) {    // 새 소켓 받아오기 성공 시
            int idx = s_UpdateNewSocket(new_socket);    // 등록
            s_ForkForClientMessage(idx);    // 해당 클라이언트용 추가 프로세스 실행
        }
        s_Broadcast();  // 전체 방송
    }
}
```

서버 주요 함수(3) : s_GetMessageFromClient()

- 자식 프로세스에 해당
- 클라이언트의 메시지를 받아 부모에 전달하는 역할
- 담당 클라이언트 소켓을 non-block으로 만들고 지속적으로 확인
- 수신된 정보는 해당 클라이언트 전용 파이프를 통해 부모 프로세스에 전달

```c
void s_GetMessageFromClient(int idx) {
    close(ChatServer.server_sock);
    char buffer[BUFFER_SIZE];

    s_MakeNonblock(ChatServer.client_socket[idx]);
    while (1) {
        // 클라이언트로부터 데이터 수신
        memset(buffer, 0, sizeof(buffer));
        int valread = read(ChatServer.client_socket[idx], buffer, BUFFER_SIZE);
        if (valread == 0) { // 클라이언트 연결 종료
            printf("클라이언트 연결 종료: %d\n", ChatServer.client_socket[idx]);
            break;
        }
        buffer[valread] = '\0';
        // 부모 프로세스에 메시지 전송
        write(ChatServer.pipes[idx][1], buffer, strlen(buffer));
    }
    close(ChatServer.client_socket[idx]);
    close(ChatServer.pipes[idx][1]);
    ChatServer.client_socket[idx] = 0;
    exit(0); // 프로세스 종료
}
```

추가 사항

기타 구현 방법

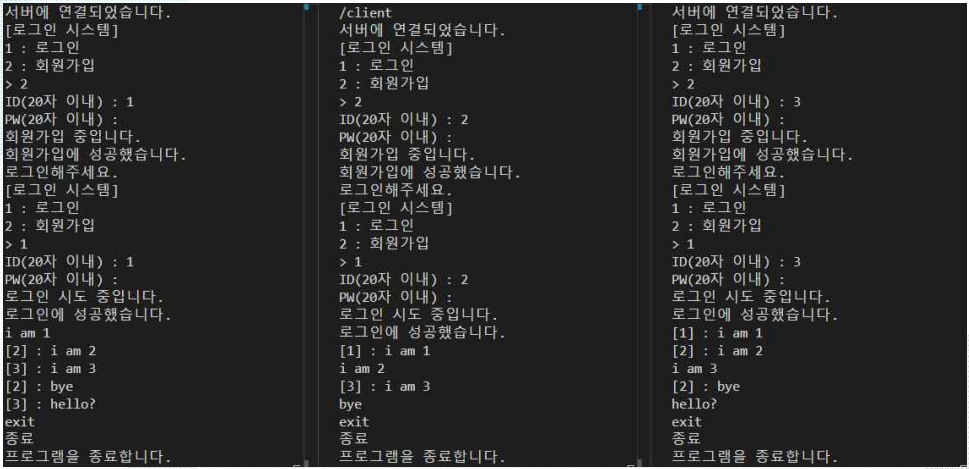
▶ **로그인/회원가입/로그아웃**

- 아이디/비밀번호 앞에 SI/SU/SO를 붙여 명령어처럼 사용(Sign In/Up/Out)
- 버퍼의 구성 : (명령어):(ID)|(PW) / 일반 메시지 : [(ID)] : (메시지)
- 변환은 클라이언트에서 이뤄짐.
- 서버 : 해당 버퍼들을 저장, 로그인 정보가 유무 확인(로그인) 및 추가(회원가입) 가능
- 로그인 성공 시 클라이언트의 양 프로세스에 로그인 되었음을 저장. 서버 역시 저장.
- c_SignIn(), c_SignUp(), c_LoginSystem(), c_LoginCommunication() 등 참고

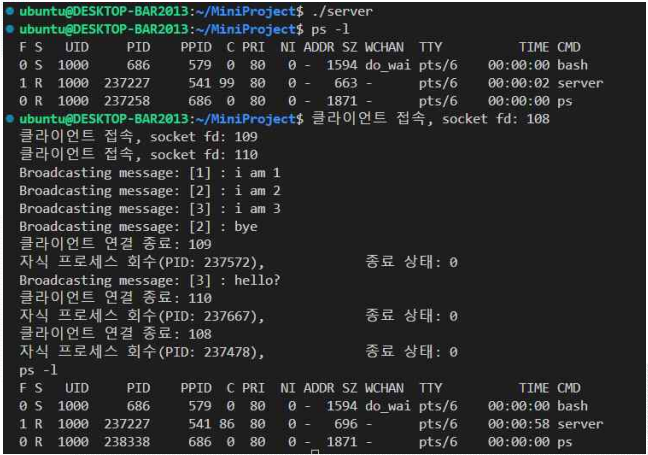
▶ **비밀번호 표시 제한**

- struct termios, tcgetattr() 활용
- c_ShowLetters(), c_HideLetters() 등 참고

구현 화면 - 클라이언트 3개 접속



구현 화면 - 서버



※ 라즈베리파이 ip 변경 문제로 로컬로 진행되었음

감사합니다