# VEDA
# Mini Project

Chatting Program

A반 김효은

# Contents

# About Project

**멀티 프로세싱으로 구현한 TCP 소켓 통신 채팅 프로그램**

**: 여러 클라이언트(사용자)가 서버에 연결 되었을 때, 입력한 메시지를 주고 받기 가능**

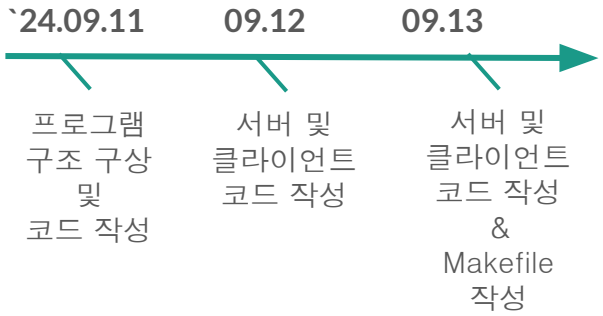
**[구현 기능]**

- 부모 프로세스와 자식 프로세스 사이 파이프 사용.
- 자식 프로세스와 클라이언트 사이 소켓 통신 사용.

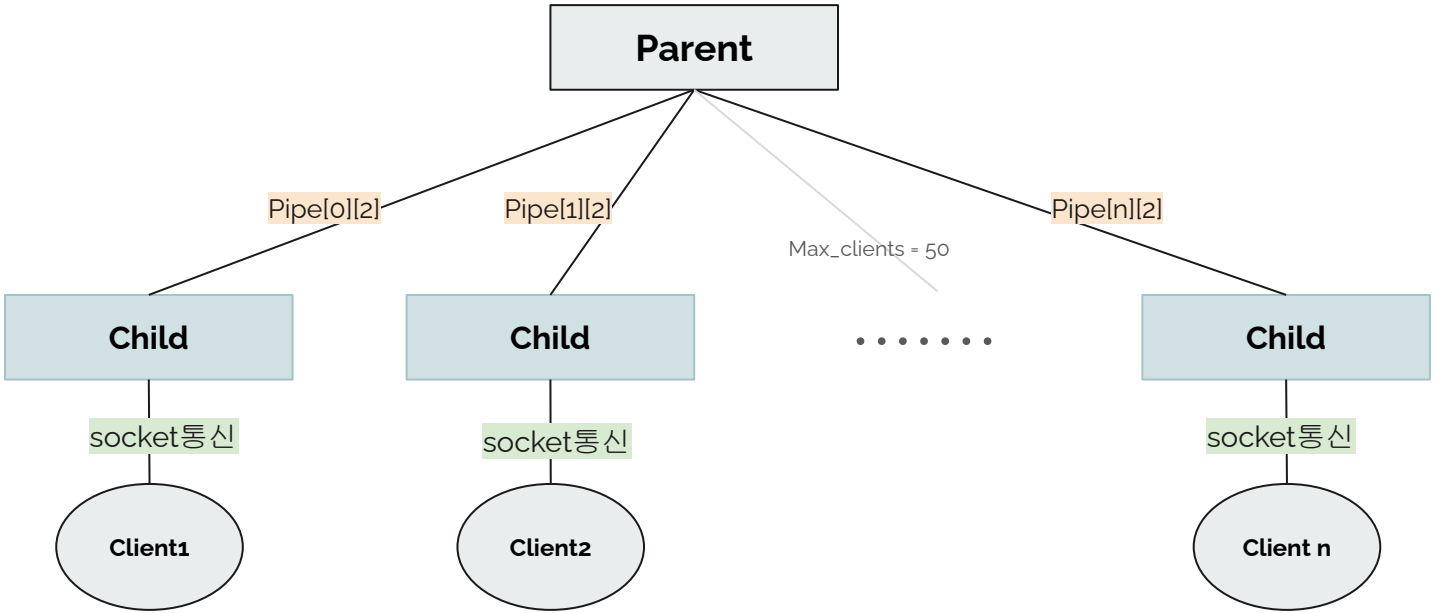
**[제약 사항]**

- epoll함수, select함수 사용 불가
- 메시지 큐나 공유 메모리 사용 불가

**[진행 일정] 24.09.11 ~ 24.09.13**



# Detail - 프로그램 구조 설계



# Detail - Code Review

## [Server]

```
//자식 프로세스
if (pid == 0) {     // Child process
    close(ssock);                           // 자식은 서버 소켓을 닫음
    close(pipefds[num_clients][0]);         // 파이프의 읽기 끝 닫기

    //int getname = read(csock, name,strlen(name));

    while (1) {
        memset(mesg, 0, BUF_SIZE);
        int n = read(csock, mesg, BUF_SIZE);    //클라이언트 소켓 읽고
        if (n > 0) {
            printf("Received from client: %s\n", mesg);
            write(pipefds[num_clients][1], mesg, n);        // 파이프에 쓰기 -> 부모에게 전달
        } else if (n == 0) {
            printf("Client disconnected.\n");
            close(csock);
            break;
        } else if (errno != EAGAIN && errno != EWOULDBLOCK) {
            perror("read()");
            break;
        }
    }
    close(pipefds[num_clients][1]);  // 파이프의 쓰기 끝 닫기
    exit(0);
```

## [Child]

- 소켓이 계속 연결 요청시 accept()
- 파이프 읽기 pipe[num_clients][0] close
- 클라이언트가 보낸 데이터는 소켓 통신을 통해 read
- pipe[num_clients][1]을 통해 클라이언트로부터 받은 데이터 부모에게 write
- 파이프 쓰기 pipe[num_clients][1]] close

# Detail - Code Review

## [Server]

```
clen = sizeof(cliaddr);
int csock = accept(ssock, (struct sockaddr *)&cliaddr, &clen);    // 클라이언트 접속 accept

if (csock < 0) {
    // non-blocking mode이기 때문에, 에러가 accept 실패가 아닐 수 있음
    if (errno == EWOULDBLOCK || errno == EAGAIN) {
        // 더 이상 연결 대기 중인 클라이언트가 없는 경우
        // 부모 프로세스는 계속해서 클라이언트와 통신을 수행
                    // 다른 클라이언트에게 전달
        for (int i = 0; i < num_clients; i++) {
            memset(mesg, 0, BUF_SIZE);
            int n = read(pipefds[i][0], mesg, BUF_SIZE);  //파이프 통해 자식으로부터 읽기
            if (n > 0) {
                printf("Broadcasting message from client %d: %s\n", i, mesg);
                for (int j = 0; j < num_clients; j++) {
                    if (i != j) {                    //자신이 쓴것은 다시 받지 않도록 자신 제외
                        write(clients[j], mesg, n);  // 다른 클라이언트들에게 전달
                    }
                }
            } else if (n < 0 && errno != EAGAIN && errno != EWOULDBLOCK) {
                perror("read() from pipe");
            }
        }
        continue;                                                // 계속해서 다음 작업 대기
    } else {
        perror("accept() failed");
        continue;
    }
```

**[Parent]**

- 소켓이 계속 연결 요청시 accept()
- pipe[num_clients][1]은 close
- pipe[num_clients][0]를 non blocking모드로 자식 프로세스가 보낸 데이터 read
- 데이터가 자식으로부터 넘어온 경우, 지금 데이터를 보낸 자식을 제외한 나머지 클라이언트에 write

# Detail - Code Review

## [Client]

```
if ((pid = fork()) == 0) {

    while (1) {
        memset(message, 0, BUF_SIZE);
        str_len = read(sock, message, BUF_SIZE - 1);
        if (str_len > 0) {
            message[str_len] = 0;
            printf("\n");
            printf("[Message from server]: %s", message);
        } else if (str_len == -1 && errno == EAGAIN) {
            continue;
        } else if (str_len == 0) {
            printf("Server disconnected.\n");
            break;
        }
    }
} else {
    while (1) {
        printf("%s: ", name);
        fgets(message, BUF_SIZE, stdin);
        if (!strcmp(message, "q\n") || !strcmp(message, "Q\n")) {
            close(sock);
            exit(0);
        }

        // Send message to server
        if (write(sock, message, strlen(message)) == -1) {
            perror("write()");
            break;
        }
    }
```

**[Parent]**

- 데이터 입력 받기
- 해당 데이터를 소켓 통신을 통해 서버의 자식 프로세스에 write
- q 나 Q를 입력하면 클라이언트 접속 종료

**[Child]**

- 서버가 전달한(부모가 전달)한 데이터를 read해 다른 클라이언트의 메시지 확인

# Result

[배운 점]

pipe / 양방향 pipe 사용

- 단방향 특성을 가진 pipe를 사용할때 한번에 데이터를 주고 받고 싶은 경우 양방향 사용

fork() 함수와 non_blocking mode

- 부모와 자식 관계 특성 파악
- 계속 기다려서 다음으로 진행이 안되는 것을 방지하고자 nonblocking 모드 사용 필요

TCP socket 통신

- 소켓 통신 순서를 정확하게 파악

multiprocessing

- 해당 프로그램 구현을 통해 멀티프로세싱의 단점(메모리 낭비 등)을 정확하게 파악

[실행방법]

command 입력 순서:

```
make (makefile을 사용해 miniServer.o와
        miniClient.o 생성)
./miniServer
./miniClient 여러 개 실행
```

[개선할 점]

사용자 UI 수정 필요

- 클라이언트에서 이름까지 소켓 통신으로 넘겨줘야 클라이언트에게 보낼 때 “이름: 메시지 내용” 가능
- 다른 클라이언트에서 메시지를 받은 후 enter입력 수정 필요