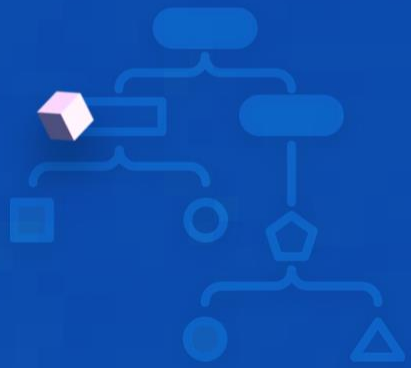
강원혁신플랫폼

리눅스프로그래밍

병행 처리 서버



Linux TCP 병행처리 서비스 방법 중, 단일 프로세스가 non-blocking I/O 모델을 사용, 여러 클라이언트 연결을 처리하여 여러 소켓을 동시에 모니터링하는 방법을 무엇이라고 하나요?

I/O 멀티플렉싱

학습목표

학습 내용

1. 병행 처리 서버
2. select( ) 함수

학습 목표

- 병행 처리 서버에 대해 설명할 수 있다.
- select( ) 함수에 대해 파악할 수 있다.

병행 처리 서버

병행 처리 서버

특징

- 동시 실행 메커니즘을 활용하여 여러 클라이언트 연결을 동시에 처리할 수 있는 서버 구현(forking or threading)
- 새 프로세스 생성 동시 실행(forking)
- 각 클라이언트 연결에 대한 자식 프로세스를 생성

TCP 반복횟수 송신, 메시지 수신

- tcp_client2.c: 반복 횟수를 전송하고, 반복 횟수만큼 메시지 수신

- tcp_server2.c: 반복 횟수를 수신하고, 반복 횟수만큼 메시지 송신

따라하기 tcp_server2.c


```
/* 상단 생략 */
    // Receive the value of n from the client
    recv(client_socket, &n, sizeof(n), 0);

    // Send a reply message back to the client n times
    for (i = 0; i < n; i++) {
        snprintf(buffer, BUFFER_SIZE, "Reply %d from server.", i + 1);
        send(client_socket, buffer, strlen(buffer), 0);
        sleep(1); // A small delay to demonstrate the server sending multiple replies
    }

/* 하단 생략 */
```

따라하기 tcp_client2.c


```
/* 상단 생략 */
    send(client_socket, &n, sizeof(n), 0);

    for (i = 0; i < n; i++) {
        int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            perror("Receive failed");  close(client_socket);  exit(EXIT_FAILURE);
        }
        buffer[bytes_received] = '\0';
        printf("Reply %d from server: %s\n", i + 1, buffer);
    }
/* 하단 생략 */
```

따라하기 tcp_server2.c, tcp_client2.c 실행결과

실행결과

```
$ gcc -o tcp_server2 tcp_server2.c
$ gcc -o tcp_client2 tcp_client2.c
$ ./tcp_server2
Server is listening on port 8080...
```

새로운 터미털 창1에서

```
$ ./tcp_client2
Enter the value of n: 100
Reply 1 from server: Reply 1 from server.
Reply 2 from server: Reply 2 from server.
```

새로운 터미털 창2에서

```
$ ./tcp_client2
Enter the value of n: 100
```

TCP concurrent server, fork()

- tcp_client2.c: 반복 횟수를 전송하고, 반복 횟수만큼 메시지 수신

- tcp_concurrent_server_fork.c: 반복 횟수를 수신하고, 반복 횟수만큼 메시지 송신, fork() 이용, 여러 클라이언트 동시 서비스

따라하기 tcp_concurrent_server_fork.c


```
/* 상단 생략 */
void handle_client(int client_socket, int n) {
    char buffer[BUFFER_SIZE];
    int i;

    for (i = 0; i < n; i++) {
        snprintf(buffer, BUFFER_SIZE, "Reply %d from server.", i + 1);
        send(client_socket, buffer, strlen(buffer), 0);
        sleep(1); // A small delay to demonstrate the server sending multiple replies
    }
    close(client_socket);
    exit(EXIT_SUCCESS);
}
```

따라하기 tcp_concurrent_server_fork.c


```
/* 상단 생략 */
    printf("Server is listening on port 8080...\n");

    while (1) {
        // Accept the client connection
        client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &addr_len);
        if (client_socket == -1) {
            perror("Accept failed");
            continue;
        }
```

따라하기 tcp_concurrent_server_fork.c


```
    // Fork a child process to handle the client connection
    pid_t pid = fork();
    if (pid == -1) {    perror("Fork failed"); close(client_socket); continue;
    } else if (pid == 0) { // Child process
        close(server_socket); // Child doesn't need the listening socket
        recv(client_socket, &n, sizeof(n), 0); // Receive the value of n from the client
        handle_client(client_socket, n); // Handle the client's request
        exit(EXIT_SUCCESS); // Child process exits after handling the client
    } else {  // Parent process
        close(client_socket); // Parent doesn't need the connected socket
    }
} /* 하단 생략 */
```

따라하기 tcp_concurrent_server_fork.c

실행결과

```
$ ./tcp_concurrent_server_fork
Server is listening on port 8080...
```

새로운 터미털 창1에서

```
$ ./tcp_client2
Enter the value of n: 100
Reply 1 from server: Reply 1 from server.
Reply 2 from server: Reply 2 from server.
```

새로운 터미털 창2에서

```
$ ./tcp_client2
Enter the value of n: 100
Reply 1 from server: Reply 1 from server.
Reply 2 from server: Reply 2 from server.
```

병행 처리 서버

스레딩(threading)

- 각 클라이언트 연결에 대한 새 스레드 생성
- 각 스레드는 동일한 프로세스 내에서 특정 클라이언트 연결을 동시에 처리

TCP concurrent server, pthread()

- tcp_client2.c: 반복 횟수를 전송하고, 반복 횟수만큼 메시지 수신

- tcp_concurrent_server_thread.c: 반복 횟수를 수신하고, 반복 횟수만큼 메시지 송신, pthread() 이용, 여러 클라이언트 동시 서비스

따라하기 tcp_concurrent_server_thread.c


```
/* 상단 생략 */
struct ClientData {
    int client_socket;
    int n;
};
```

따라하기 tcp_concurrent_server_thread.c


```
void *handle_client(void *arg) {
    struct ClientData *data = (struct ClientData *)arg;
    char buffer[BUFFER_SIZE];
    int i;

    for (i = 0; i < data->n; i++) {
        snprintf(buffer, BUFFER_SIZE, "Reply %d from server.", i + 1);
        send(data->client_socket, buffer, strlen(buffer), 0);
        sleep(1); // A small delay to demonstrate the server sending multiple replies
    }
    close(data->client_socket);    free(data);    return NULL;
}
```

따라하기 tcp_concurrent_server_thread.c


```
    pthread_t thread;
/* 중간 생략 */
        recv(client_socket, &n, sizeof(n), 0);
        struct ClientData *data = (struct ClientData *)malloc(sizeof(struct ClientData));
        data->client_socket = client_socket;
        data->n = n;

        if (pthread_create(&thread, NULL, handle_client, data) != 0) {
            perror("Thread creation failed");
            close(client_socket);
            free(data);
            continue;
        }
```

따라하기 tcp_concurrent_server_thread.c


```
        // Detach the thread so that it can clean up after itself
        pthread_detach(thread);
    }
/* 하단 생략 */
```

따라하기 tcp_concurrent_server_thread.c

실행결과

```
$ ./tcp_concurrent_server_thread
Server is listening on port 8080...
```

새로운 터미털 창1에서

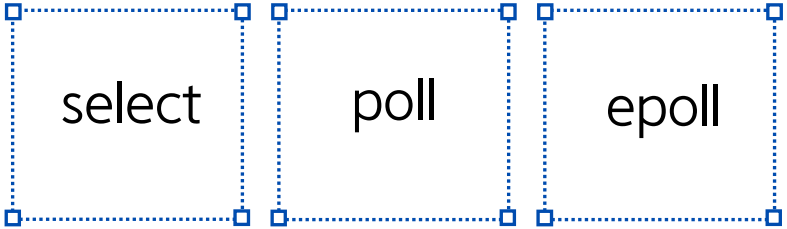
```
$ ./tcp_client2
Enter the value of n: 100
Reply 1 from server: Reply 1 from server.
Reply 2 from server: Reply 2 from server.
```

새로운 터미털 창2에서

```
$ ./tcp_client2
Enter the value of n: 100
Reply 1 from server: Reply 1 from server.
Reply 2 from server: Reply 2 from server.
```

병행 처리 서버

- I/O 멀티플렉싱



- select: 파일 설명자에서 이벤트를 대기하고 해당 이벤트가 발생할 때 응답. 파일 디스크립터는 읽기 세트, 쓰기 세트 및 예외 세트의 세트로 구성
- poll: I/O 다중화를 위한 간단한 메커니즘, 여러 파일 설명자에서 이벤트를 기다린 다음 해당 이벤트가 발생할 때 응답
- epoll: 확장 가능한 I/O 이벤트 알림 메커니즘, 많은 수의 파일 설명자를 처리할 때 더 효율적임

병행 처리 서버

I/O 멀티플렉싱

- 단일 프로세스가 non-blocking I/O 모델을 사용하여 여러 클라이언트 연결을 처리
- 여러 소켓을 동시에 모니터링

I/O 멀티플랙싱

구분	내용	비고
폴링 (Polling)	루프를 이용해서 처리해야 할 일이 있는지 주기적으로 지켜보다가 처리해야 할 작업들이 생기면 순차적으로 돌아가면서 처리하는 방법	poll( )
셀렉트 (Selecting)	지정된 파일 디스크립터의 변화를 확인해서 변화 (입력, 출력, 에러)가 감지되면 해당 작업을 처리하는 방법	select( )
인터럽트 (Interrupt)	프로세스가 어떤 작업을 처리하는 도중에 특정한 이벤트가 발생하면 시그널 등을 이용해서 해당 이벤트를 처리하는 방식	시그널

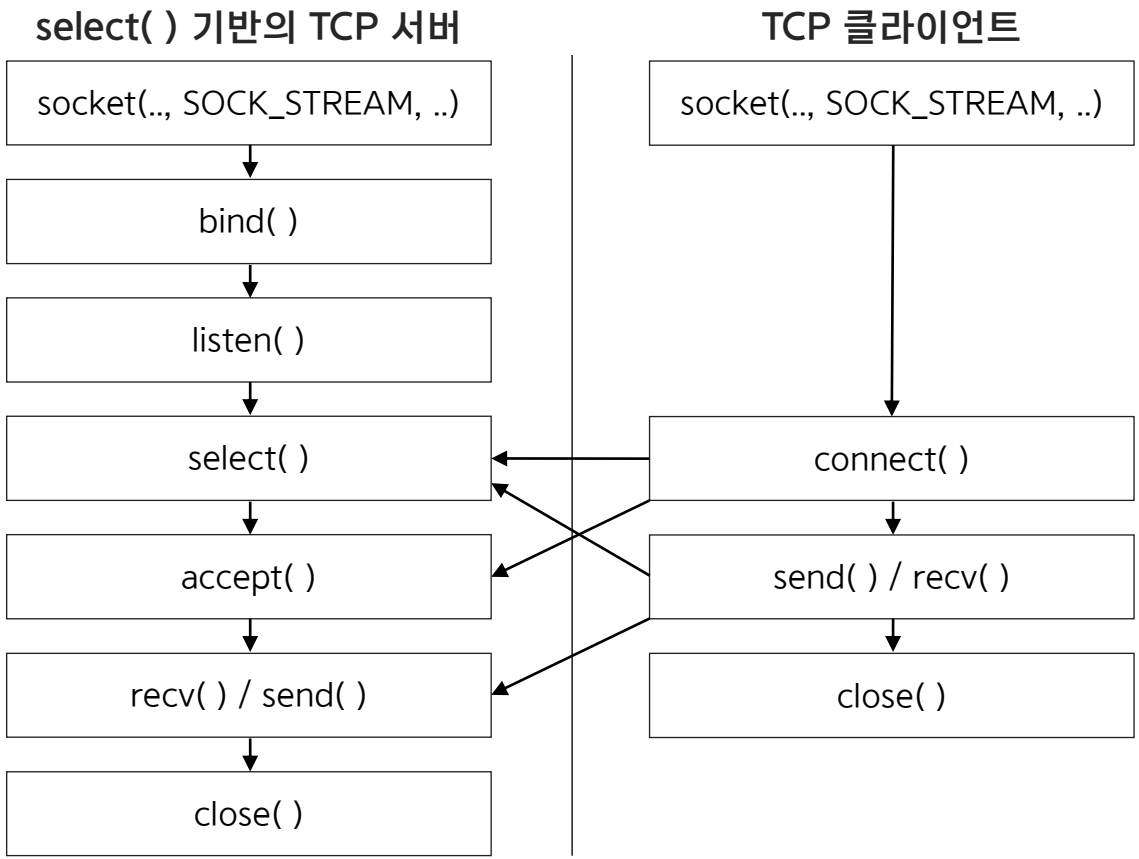
select( ) 함수

int select( ) 함수


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

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval
*timeout);
```

select( ) 함수를 이용한 통신 방식



select( ) 함수의 감지용 인자

인자	내용	비고
readfds	수신할 데이터가 있는지 확인함	입력 버퍼에 데이터 존재
writefds	데이터 전송이 가능한 상태인지 확인함	출력 버퍼에 충분한 여유 공간 존재
exceptfds	소켓에 예외 상황이 발생했는지 확인함	OOB 메시지 전송 등의 사항

fd_set 설정을 위한 매크로

매크로	내용
void FD_ZERO(fd_set *fdset);	파일 디스크립터 집합을 초기화(모두 해제)
void FD_CLR(fd, fd_set *fdset);	해당 파일 디스크립터 해제
void FD_SET(fd, fd_set *fdset);	해당 파일 디스크립터 설정
int FD_ISSET(fd, fd_set *fdset);	해당 파일 디스크립터가 설정되었는지 확인

따라하기 select_server.c


```
... 상부 헤더 생략
#define SERVER_PORT 5100                /* 서버의 포트 번호 */

int main(int argc, char **argv) {
    int ssock;    socklen_t clen;    int n;
    struct sockaddr_in servaddr, cliaddr;
    char mesg[BUFSIZ];
    fd_set readfd;                      /* select( ) 함수를 위한 자료형 */
    int maxfd, client_index, start_index;
    int client_fd[5] = {0};             /* 클라이언트의 소켓 FD 배열 */
```

따라하기 select_server.c


```
/* 서버 소켓 디스크립터 열기 */
if((ssock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket()");
    return -1;
}

memset(&servaddr, 0, sizeof(servaddr));     /* 운영체제에 서비스 등록 */
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERVER_PORT);
```

```
if(bind(ssock, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("bind()");       return -1;
}

if(listen(ssock, 8) < 0) {          /* 클라이언트의 소켓들을 위한 큐 생성 */
    perror("listen()");         return -1;
}

FD_ZERO(&readfd);                   /* fd_set 자료형을 모두 0으로 초기화 */
maxfd = ssock;          /* 현재 최대 파일 디스크립터의 번호는 서버 소켓의 디스크립터 */
client_index = 0;
```

따라하기 select_server.c


```
do {
    FD_SET(ssock, &readfd); /* 읽기 동작 감지 위한 fd_set 설정 */

    /* 클라이언트의 시작 주소부터 마지막 주소까지 fd_set에 설정 */
    for(start_index = 0; start_index < client_index; start_index++) {
        FD_SET(client_fd[start_index], &readfd);
        if(client_fd[start_index] > maxfd)
            maxfd = client_fd[start_index]; /* 가장 큰소켓 번호저장 */
    }
    maxfd = maxfd + 1; // 감시할 파일의 개수를 의미, 인덱스 형태이기 때문에 +1함
```

따라하기 select_server.c


```
/* select( ) 함수에서 읽기가 가능한 부분만 조사 */
select(maxfd, &readfd, NULL, NULL, NULL);   /* 읽기가 가능해질 때까지 블로킹 */
if(FD_ISSET(ssock, &readfd)) {              /* 읽기가 가능한 소켓이 서버 소켓인 경우 */
    clen = sizeof(struct sockaddr_in);      /* 클라이언트의 요청 받아들이기 */
    int csock = accept(ssock, (struct sockaddr*)&cliaddr, &clen);
    if(csock < 0) {
        perror("accept()");
        return -1;
    } else {
```

```
        /* 네트워크 주소를 문자열로 변경 */
        inet_ntop(AF_INET, &cliaddr.sin_addr, mesg, BUFSIZ);
        printf("Client is connected : %s\n", mesg);

        /* 새로 접속한 클라이언트의 소켓 번호를 fd_set에 추가 */
        FD_SET(csock, &readfd);
        client_fd[client_index] = csock;
        client_index++;
        continue;
    }
    if (client_index == 5) break;
}
```

```
/* 읽기 가능했던 소켓이 클라이언트였던 경우 */
for(start_index = 0; start_index < client_index; start_index++) {
    /* for 루프를 이용해서 클라이언트들을 모두 조사 */
    if(FD_ISSET(client_fd[start_index], &readfd)) {
        memset(mesg, 0, sizeof(mesg));

        /* 해당 클라이언트에서 메시지를 읽고 다시 전송(Echo) */
        if((n = read(client_fd[start_index], mesg, sizeof(mesg))) > 0) {
            printf("Received data : %s", mesg);
            write(client_fd[start_index], mesg, n);
            close(client_fd[start_index]);      /* 클라이언트 소켓을 닫는다. */
```

```
                /* 클라이언트 소켓을 지운다. */
                FD_CLR(client_fd[start_index], &readfd);
                client_index--;
            } // if
        } // if
    } // for
} while(strncmp(mesg, "q", 1));

close(ssock);                           /* 서버 소켓을 닫는다. */
return 0;
}
```

따라하기 select_server 실행결과

- 실행결과

```
$ ./select_server
Client is connected : 127.0.0.1
Received data : Hello World

여러 새로운 터미널 창에서 동시에 실행
$ ./tcp_client 127.0.0.1
Hello World
Received data : Hello World
```

요약정리

01 • TCP 네트워크 프로그래밍

02 • TCP 클라이언트·서버 함수