운영체제 과제2

201921166 정의철

1. 설계 내용

다이어그램은 이미지 크기가 큰 관계로 문서 하단에 게시하였습니다.

가. 소켓 생성

서버와 클라이언트 모두 IPv4, TCP 소켓을 사용한다. 또한 Buffer size는 1024byte로 설정하였다. 서버는 0x00000000(모든 IP로부터 연결 허용), 5555 포트를 개방하였다.

나. 에러 처리

서버와 클라이언트 모두 소켓을 생성하고, 연결을 수행하는 과정에서 에러가 발생할 경우 error()함수를 통해 사용자에게 문제 발생을 알리고 종료하도록 하였다. 따라서 개발과정 중에 어느 부분에 문제가 발생하였는지를 쉽게 파악하여 디버깅할 수 있었다.

다. 프로세스 분기

서버에서 accept() 단계부터 무한루프를 돌도록 설계하였다.

클라이언트로부터 연결 요청이 올 경우, 서버에서 accept를 수행하고 클라이언트 소켓의 디스크립터를 client_socket 변수에 저장한다. 이후 fork() 함수를 통해 분기가 실행되고, 부모 프로세스는 다시 accept() 라인으로 돌아가서 클라이언트의 요청이 올 때 까지 블로킹 상태에 놓이게 된다.

```
for (;;) {
  client_socket = accept();
  pid = fork();
  if (pid ≠ 0) {
    continue;
  }

  send();
  recv();
  .
  .
  .
  break;
}
```

2. 주요 코드 설명

가. 답변 재요청

```
for (;;) {
    read(client_socket, buffer, sizeof(buffer));
    client_ans_val = atoi(buffer);  // string → 정수형
    if (client_ans_val ⩾ 1 && client_ans_val ⩽ 3) {
        // 사용자 응답이 1~3 사이일 경우
        write(client_socket, "0", 2);   // 정상 '0' 전송
        break;
    } else {
        write(client_socket, ERPY, sizeof(ERPY));
    }
}
```

설문조사에 있는 답 문항은 1~3번 까지 총 3개로서, 클라이언트로부터 1에서 3 사이의 답변을 받지 못한 경우 ERPY=("1에서 3사이로 입력해 주세요.") 문장을 전송을 전송하여 다시 답변을 받도록 한 코드이다. 클라이언트의 입력은 atoi()함수를 통해 char* -> int로 변환한다. 무한 반복문 내에 위치한 코드로서, 사용자가 제대로 된 입력을 하기 까지 계속해서 반복된다.

나. 클라이언트 구분

```
char client_address_string[INET_ADDRSTRLEN] = { 0 };
inet_ntop(AF_INET, &client_address.sin_addr.s_addr, client_address_string, INET_ADDRSTRLEN);
printf("클라이언트(IP: %s)의 응답: %zd, %s\n",             \
client_address_string, client_ans_val, ANSW[client_ans_val -1]);
```

inet_ntop() 함수를 통해서 클라이언트의 ip 주소를 취득하여 다수 클라이언트에서 응답이 올 때 구분하기 쉽도록 하였다.

다. 프로세스 분기

```
pid = fork();
if (pid ≠ 0) {
    continue;
}
```

accept() 함수는 반복문 내에 위치하고 있어서, fork() 함수 실행 이후에 부모 프로세스는 다시 반복문 처음으로 돌아가, accept()를 호출하도록 continue를 실행시킨다.

3. Makefile 설명

```
CC = gcc

TARGET = server client

all: $(TARGET)

server:
	$(CC) -o $@ $@.c

client:
	$(CC) -o $@ $@.c

clean:
	rm -f $(TARGET)
```

CC 매크로를 gcc로 정의하여, gcc를 통해 컴파일 할 것임을 명시했습니다. 이는 하단에서 $(CC)를 통해 활용됩니다.

TARGET 매크로를 server와 client로 정의하였습니다.

all 타겟의 dependency를 $(TARGET)로 설정하였습니다. 이는 곳, all을 실행하기 위해서는 TARGET 매크로에 정의된 server타겟과, client타겟이 필요하다는 것을 의미합니다. 따라서 all을 실행하면, 자동적으로 server타겟과 client타겟이 실행됩니다.

server타겟은 gcc컴파일러를 통해 server.c를 빌드하여 server라는 executable file을 생성하라는 명령을 수행합니다.

마찬가지로 client타겟은 gcc컴파일러를 통해 client.c를 빌드하여 client라는 executable file을 생성하라는 명령을 수행합니다.

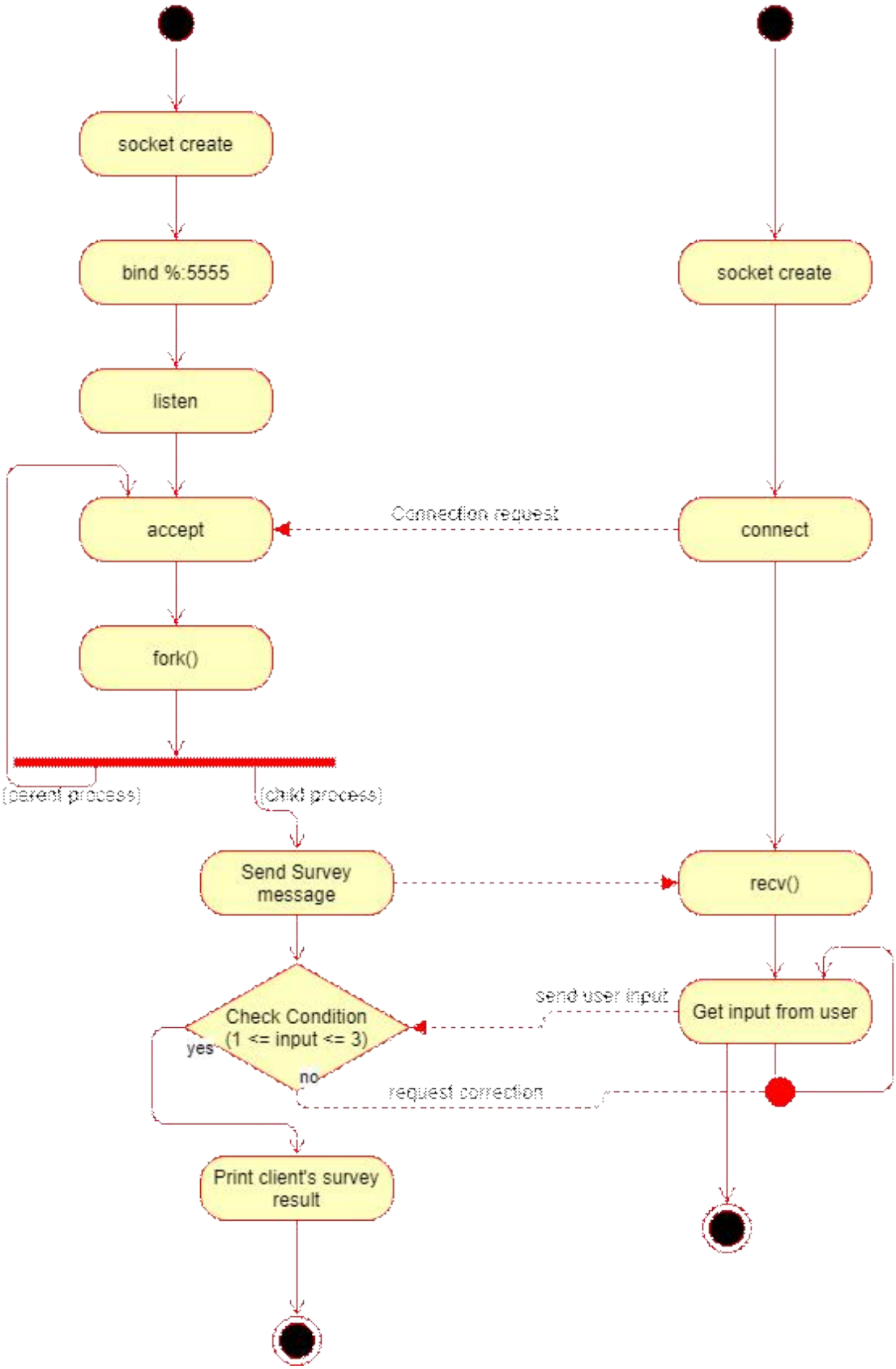
$@매크로는 현재 타겟의 이름을 반환하는 내부 매크로입니다. 따라서 client타켓 내에서 $@는 'client'를 리턴합니다.

4. 개발 history 및 소감

이번 과제는 프로세스 분기, 즉 fork()를 사용하는 과제였습니다. 스레드 분기의 경우는 Heap 메모리 공간을 공유하기 때문에, 서로 다른 스레드 간에 전역변수와 인스턴스를 쉽게 주고받을 수 있었습니다. 하지만 프로세스를 분기시키게 되면 Heap 등 모든 메모리 공간을 복사 후, 다른 공간에 할당되기 때문에 부모 프로세스와 자식 프로세스 간 데이터를 주고받는데 어려움이 있습니다. 이는 Pipe나 Shared Memory 등으로 해결할 수 있지만 프로그램을 간단하게 짜고자 하였습니다. 서로 데이터를 주고받지 않아도 메모리 내용이 복제된다는 점을 이용하여, 문제없이 실행되도록 프로그래밍 하여 해결했습니다.

소켓 통신 같은 내용은 자바에서만 다루어 보았지, C에서는 구현해본 적이 없었습

니다. 또한 프로세스 분기 같은 내용은 자바에서 스레드를 주제로 다루어 보았기 때문에 이번 과제에서는 큰 도움이 되지 못하였습니다. 하지만 GCC에서 사용하는 GNU C Library는 오픈소스 라는 점을 이용하여 Documentation을 참조하여 개발하였습니다. 오픈소스 Documentation을 보는 방법을 배워 볼 수 있어서 큰 도움이 되었습니다.


socket create
bind %:5555
listen
accept
fork()
(parent process)
(child process)
Send Survey message
Check Condition (1 <= input <= 3)
yes
no
request correction
Print client's survey result
socket create
connect
Connection request
recv()
Get input from user
send user input