COSE322-00
시스템프로그래밍
유혁 교수님

2차 과제

Pthread를 이용한 Client-Socket Programming

제출일: 2019.11.14 (프리데이 사용일수: 0)
개발환경:
가상머신: Oracle VM VirtualBox
운영체제: Ubuntu 16.04 LTS

조장: 2015131102 강수진
조원: 2015131116 배은초

Pthread를 이용한 Client-Socket Programming

2015131102 강수진, 2015131116 배은초

목차

1. pthread를 사용하는 이유 (Without pthread)

server input port : 1111, 2222, 3333, 4444, 5555

```
#include "stdio.h"
#include "stdlib.h"
#include "pthread.h"
#include "string.h"
#include "sys/time.h"
#include "sys/types.h"
#include "sys/socket.h"
#include "sys/stat.h"
#include "unistd.h"
#include "netinet/in.h"
#include "arpa/inet.h"

#define SERVER_IP_ADDR "192.168.56.102"
#define BUFF_SIZE 1024
#define SOCKET_NUM 5

void* mysub(){
    int clientSocket[SOCKET_NUM]; //client socket 만들어서 fd 반환된 값,
    struct sockaddr_in serverAddr[SOCKET_NUM];
    char buff[SOCKET_NUM][BUFF_SIZE];

    //소켓 생성. AF_INET 은 IPv4 Internet protocols
    for (int i = 1; i<SOCKET_NUM+1; i++) {
        clientSocket[i] = socket(AF_INET, SOCK_STREAM, 0);

        //에러 체크
        if (clientSocket[i] != -1) {
            printf("Client : Open stream socket\n");
        } else {
            printf("Client : Cannot open stream socket\n");
            exit(0);
        }
        memset(&serverAddr, 0, sizeof(serverAddr));
        serverAddr[i].sin_family = AF_INET;
        serverAddr[i].sin_port = htons(i*1111); //포트 번호
        serverAddr[i].sin_addr.s_addr = inet_addr(SERVER_IP_ADDR);

        //connect and error check
```

```
        if(connect(clientSocket[i], (struct sockaddr *) &serverAddr[i], sizeof(serverAddr[i])) != -1) {
            printf("Client : Connect to server\n");
        } else {
            printf("Client : Cannot connect to server\n");
            exit(0);
        }
    }

    //write contents and arrival times of packets to the file for 3 seconds
    for (int i =1; i<SOCKET_NUM+1; i++) {
        for(int j =0; j<50; j++) {
            read(clientSocket[i], buff, BUFF_SIZE);
            printf("[PORT %d] : %s\n", i*1111, buff);
        }
    }
    return 0;
}

int main() {
    mysub();
    return 0;
}
```

위의 코드는 하나하나 소켓을 생성한 후에, for 문을 통해 순서대로 read 해서 프린트 한 것이다.

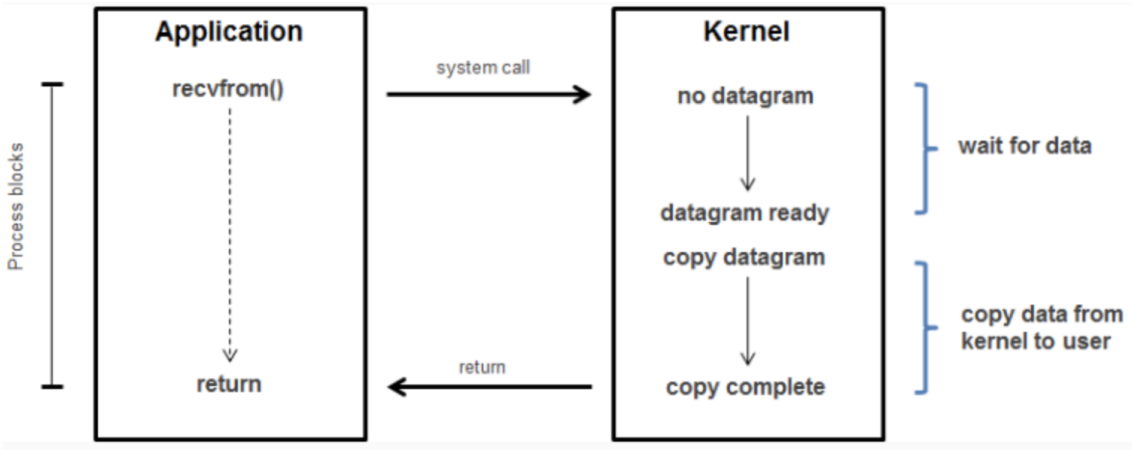
어떤 포트를 통해 data가 전송될지 모르기 때문에 연결된 모든 포트에서 지속적으로 read를 수행해야 하지만 해당 코드에서는 for문에서 자신의 차례가 오기전엔 read를 수행 할 수 없다.

또한 이렇게 하는 경우 아래와 같이 중간에 멈춰버릴 때도 있었다.

```
    YJTPZSREAFBVKLRIBLWROYAKYVONYOBWAXMZPFFRMINWTGHWRDQHDQRELIRLWSHYRVZIAGCOQPLMXUIOZYVD
[PORT 3333] :
    QOJDWAOVUYVMVVWYDYMUNXGKRQARPXWHOFLMHBHBBFNXCKVFIJZYHFKAYNTPKRWAYJNFLWJOBYLDIIJTSKTBSFDQSZFFSB
    HRNUYARHOSGCYQKHLETGHLMLBGMINEMVXZRWZIFNDNRBGCKTIDASPODSWRDMVPHTOYRNJYCMOUPUYZPICPATFFOCWROSGX
    NWXEJGENUSHJOFKEONTOIBWYDSRTKZSZVPDGYKUUCBERJOVZDQPORLOWEGPQFHQDZVJXFDTIHXBQOYRTOGJGSYCWGUONBE
    QCCABHFVROUSEIQVEHENPWLTURNKFRPXTTZVAFSUVMOAVFXBODODZCWVVMGCFXAAQZVSGNMCCBEZIDAWHOBISZGONOQSNS
    TDUQXAGKEINKHVOHTXXWHQVNGJBYBQRWTLNTNTFSBSCIPSRIPQEXIBMOKQPOGGKCTXVISAATSFDHZWRPNVOVYAMJSBZZJJ
    BCJWKBWNXQUCZTZQKONALLDXWVAVUJHXNSTYVRNSJJXICYYPMNRZZUXXSZTMJCKWUFWPZRZHRQVFHUGCTDUTYICAUAWCXM
    BKYJFHAEOUWLBDFHFZMBULMYNIYKKXYLIYVPFXVVTRGUULDCMSFIFRGSCFEMECAMAVBHUYCQPIKKWQOKITTNNBHPGMBNQB
    ZRYDAVBDLRNXDLPRYXKTMXUUODGQQWTRPUUSPYVCRKBUWRNWQXPDXJXLMFDCDZUTTQNKOIMFUNBSGOOXODALNZYBEEGHDA
    CWSPGJZUOUJSOQGDNWIPHXOIZUMFDPHGNBXVKXPBTYTJQBMFYXUHUKPVEDCKULQTYUVYBCJREZJCTXAIVFYJKNSMKEYVTG
    ORDLREPBXVCGXXEXFZEEIOTCCFGDAZJQSMDMSTNROPAMMELUFQAPEVTJAZMDBXVTMZFGUUYILAWAGIWLAWCGRVPTWDYXBW
    TNXYTRVTBGTYIZIEMKAQQTLIPKLPHMLABIBXZYSDEODPNLTCVWSNRGVGQJWZXJAZSDYTBQWHEZWUMSYJOQXH
[PORT 3333] :
    YUOQDMQDXQEPVCLWUHFBJCVXWTHKLGTMCHCIVSLVKPKFTXBNHHOQLLPHGWTSEMEHWGPFLWJOBYLDIIJTSKTBSFDQSZFFSB
    HRNUYARHOSGCYQKHLETGHLMLBGMINEMVXZRWZIFNDNRBGCKTIDASPODSWRDMVPHTOYRNJYCMOUPUYZPICPATFFOCWROSGX
    NWXEJGENUSHJOFKEONTOIBWYDSRTKZSZVPDGYKUUCBERJOVZDQPORLOWEGPQFHQDZVJXFDTIHXBQOYRTOGJGSYCWGUONBE
    QCCABHFVROUSEIQVEHENPWLTURNKFRPXTTZVAFSUVMOAVFXBODODZCWVVMGCFXAAQZVSGNMCCBEZIDAWHOBISZGONOQSNS
    TDUQXAGKEINKHVOHTXXWHQVNGJBYBQRWTLNTNTFSBSCIPSRIPQEXIBMOKQPOGGKCTXVISAATSFDHZWRPNVOVYAMJSBZZJJ
    BCJWKBWNXQUCZTZQKONALLDXWVAVUJHXNSTYVRNSJJXICYYPMNRZZUXXSZTMJCKWUFWPZRZHRQVFHUGCTDUTYICAUAWCXM
    BKYJFHAEOUWLBDFHFZMBULMYNIYKKXYLIYVPFXVVTRGUULDCMSFIFRGSCFEMECAMAVBHUYCQPIKKWQOKITTNNBHPGMBNQB
    ZRYDAVBDLRNXDLPRYXKTMXUUODGQQWTRPUUSPYVCRKBUWRNWQXPDXJXLMFDCDZUTTQNKOIMFUNBSGOOXODALNZYBEEGHDA
    CWSPGJZUOUJSOQGDNWIPHXOIZUMFDPHGNBXVKXPBTYTJQBMFYXUHUKPVEDCKULQTYUVYBCJREZJCTXAIVFYJKNSMKEYVTG
    ORDLREPBXVCGXXEXFZEEIOTCCFGDAZJQSMDMSTNROPAMMELUFQAPEVTJAZMDBXVTMZFGUUYILAWAGIWLAWCGRVPTWDYXBW
    TNXYTRVTBGTYIZIEMKAQQTLIPKLPHMLABIBXZYSDEODPNLTCVWSNRGVGQJWZXJAZSDYTBQWHEZWUMSYJOQXH
```

예상대로라면 port 5555까지 수행되어야 할 것 같다.

이렇게 되는 이유는 linux에서 모든 소켓 통신이 기본 blocking으로 동작하기 때문이다. I/O 작업이 진행되는 동안 유저 프로세스는 자신의 작업을 중단한채 대기하는 방식이다.



위의 그림처럼

1. 유저는 커널에게(유저->커널) read작업을 요청하고
2. 데이터가 입력될 때까지 대기하다가
3. 데이터가 입력되면 유저에게(커널->유저) 결과가 전달되어야만 유저 자신의 작업에 비로소 복귀할 수 있다.

따라서 해당 그림에서는 3333 포트의 소켓이 read 함수를 실행하는 순간 block이 되어서 프로세스가 잠시 중단되는데, server 쪽에서 해당 포트로 데이터를 보내지 않아서 client는 아무런 그대로 block 상태로 빠지는 것이다.

이에 더해 여러개의 소켓을 병렬적으로 처리하지 못하기 때문에, 제대로된 데이터 리시브 타임을 읽어낼 수 없다.

이러한 한계점을 해결하기 위해 pthread를 사용해야한다.

소스코드 (With pthread)

overview

해당 파일은 main 함수에서 포트번호(1111, 2222, 3333, 4444, 5555)와 mysub 함수를 인자로 넘겨 pthread를 생성한 후 각각을 join 시킨다.

pthread 생성

```
//create thread
for(int i=0;i<SOCKET_NUM;i++){
    if ((pthread_create(&p_thread[i], NULL, mysub, &portNumber[i])) != 0) {
        printf("create thread error\n");
        return 0;
    }
}
```

`pthread_create()`는 새로운 쓰레드를 생성한다. 첫번째 매개변수인 `thread` 는 쓰레드가 성공적으로 생성되었을때 생성된 쓰레드를 식별하기 위해서 사용되는 쓰레드 식별자이다. 두번째 매개변수인 `attr` 은 쓰레드 특성을 지정하기 위해서 사용하며, 기본 쓰레드 특성을 이용하고자 할경우에 NULL 을 사용한다. 세번째 매개변수인 `start_routine`는 분기시켜서 실행할 쓰레드 함수이며, 네번째 매개변수인 `arg`는 위 start_routine 쓰레드 함수의 매개변수로 넘겨진다.

해당 함수가 성공할경우 0을 리턴하고 실패했을경우 0 이 아닌 에러코드 값을 리턴하기에 위와 같이 에러 처리를 할 수 있다.

pthread join

```
//join thread
for(int i=0;i<SOCKET_NUM;i++) {
    if (pthread_join(p_thread[i], NULL) != 0) {
        printf("join thread error\n");
        return 0;
    }
}
```

`pthread_join()` 은 특정 쓰레드가 종료하기를 기다린다. 첫번째 매개변수 `th` 는 기다릴 쓰레드의 식별자이고 두번째 매개변수 `thread_return` 은 쓰레드의 리턴값이다. thread_return 이 NULL이 아닌 경우 해당 포인터로 쓰레드의 리턴 값을 받아올수 있다.

리턴값은 성공하면 0, 실패하면 에러코드를 리턴하기 때문에 위와 같이 에러 처리를 할 수 있다.

소켓 생성

```
//소켓 생성. AF_INET 은 IPv4 Internet protocols
clientSocket = socket(AF_INET, SOCK_STREAM, 0);

//에러 체크
if (clientSocket != -1) {
    printf("Client : Open stream socket\n");
} else {
    printf("Client : Cannot open stream socket\n");
    exit(0);
}
```

소켓 생성을 위해 `socket()` 함수를 사용했다. 해당 함수는 return 값으로 file descriptor을 반환한다. 에러 상황에서는 -1을 반환하기 때문에 위와 같이 에러 체크를 할 수 있다.

`socket()`의 첫번째 인자는 `domain`으로 어떤 영역에서 통신할 것인지에 대해, 즉 protocol family를 지정해주면 된다. 인자로 넣은 `AF_INET`은 IPv4 인터넷 프로토콜을 의미한다. 두번째 인자는 `type`으로 어떤 타입의 프로토콜을 사용할 것인지에 대해 설정할 수 있다. 인자로 넣은 `SOCK_STREAM`은 TCP에 해당한다. 마지막 인자인 `protocol`에 넣은 `0` 값은 caller가 protocol을 명시하기 보다는, 그 일을 서비스 provider에게 위임하겠다는 것을 의미한다.

서버 설정

```
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(myPortNum); //포트 번호
serverAddr.sin_addr.s_addr = inet_addr(SERVER_IP_ADDR);
```

서버의 ip주소와 port번호를 sockaddr_in 구조체에 저장한다.

연결

```
    //connect and error check
    if(connect(clientSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr)) != -1) {
        printf("Client : Connect to server\n");
    } else {
        printf("Client : Cannot connect to server\n");
        exit(0);
     }
```

`connect()` 함수는 생성한 소켓을 통해 서버로 접속을 요청한다. 실패 시 -1을 return 하기 때문에 위와 같이 에러 체크를 할 수 있다.

파일 생성

```
    char outputFileName[10];
    sprintf(outputFileName, "%d.txt", myPortNum);
    FILE *outputFile;
    outputFile = fopen(outputFileName,"w");
```

우선 파일의 이름을 지정하기 위해 `sprintf()` 함수를 사용했다. 쉽게 설명하자면, printf 에서 화면에 출력하는 대신에 화면에 출력할 문자열을 인자로 지정한 문자열에 쓸 수 있다.

이후 log를 쓸 파일을 만들기 위해 `fopen( )` 함수를 사용했다. 해당 call은 starem을 control 하기위한 필수적인 정보를 담고 있는 FILE 타입의 object를 초기화한다. 이때 뒤에 "w"는 모드의 종류로, 해당 모드는 쓰기만 가능하다. 파일이 존재하면 그 파일을 지우고 생성한다.

log 쓰기

```
fprintf(outputFile, "%ld %s\n", strlen(buff), buff;
```

`fprintf()` 함수는 데이터를 형식에 맞추어 스트림에 쓴다.

특정한 스트림에 일련의 데이터를 특정한 형식에 맞추어 쓰게 된다. 이 때, 그 형식은 형식 문자열에 지정되어 있으며, 출력할 데이터는 형식 문자열 다음에 오는 인자들에 써주면 된다.

시간

```
// time 저장위한 include
#include <sys/time.h>
// 생략

// time 저장 위한 struct선언
  struct timeval tv
  struct tm *tm;
//생략

//시간 받기
gettimeofday(&tv, NULL);
tm = localtime(&tv.tv_sec);
```

```
//시간 파일에 프린트
fprintf(outputFile, "%02d:%02d:%d %03ld \n", tm->tm_hour, tm->tm_min, tm->tm_sec, (tv.tv_usec)/1000);
```

우선 패킷을 수신하는 시간을 저장하기 위해 gettimeofday 함수를 사용하였다. gettimeofday 함수는 주어진 timeval 구조체 포인터를 이용해 timeval 구조체 tv에 시간을 저장한다. 이후 timeval 구조체의 tv_sec 변수를 준 localtime함수로 그 시간의 wallclock time을 tm구조체에 저장한다. 이후 서식에 맞추어 파일에 프린트해주었다. tm에 millisecond단위 변수가 없으므로 millisecond는 tv에서 참조해 주었다.

참고) tm 구조체

```
struct tm {
    int tm_sec;         /* seconds range 0-59*/
    int tm_min;         /* minutes range 0-59*/
    int tm_hour;        /* hours range 0-23*/
// 생략 }
```

길이

```
strlen(buff)
```

문자열 길이를 구할 수 있는 `strlen()` 함수를 이용한다.

내용

```
read(clientSocket, buff, BUFF_SIZE);
```

해당 함수를 통해 `clientSocket`에서 읽어온 내용을 `BUFF_SIZE`만큼 `buff`에 담는다.

Trouble Shooting

```
for(int i=1;i<SOCKET_NUM+1;i++){
        int portNum = i*1111;
        int* p_portNum = &portNum;
        thr_id = pthread_create(&p_thread[i], NULL, mysub, p_portNum);
    }
```

스레드 크리에이트 할때 처음에 위와 같이 포트넘버를 인자로 넘기려고 했다. 그런데 자꾸 에러가 나서 mysub 안에서 portNum을 프린트해보니 5555로 모든 같은 값이 찍혔다. 모두 같은 주소를 가르키고 있기 때문에 나는 에러인 듯 했다. C언어에 대한 이해의 부족에서 온 에러임을 깨닫고 따로 포트 넘버를 모아 놓는 배열을 만들기로 했다.