FreeDay 사용일수 0일

2019F 시스템 프로그래밍

과제2 warmup : pthread를 이용한 client-side socket programming

Group 25

2014210042 한 윤
2014210055 김 민섭

제출일자 2019년 11월 11일

목차

1. 작성한 소스코드에 대한 설명

- **작성한 소스코드의 main함수는 개략적으로 다음의 로직들로 구성되어 있다.**

1) 서버의 몇 번 포트들과 연결할 것인지 프로그램 사용자로부터 입력을 받는다.

```c
int main(int argc, char* argv[]) {
    pthread_t thread[NUM_OF_PTHRDS];
    int ports[NUM_OF_PTHRDS];
    int i;

    // store port values into array
    for (i = 0; i < NUM_OF_PTHRDS; i++) {
        printf("enter socket #%d port number: ", i + 1);
        scanf("%d", &ports[i]);
    }
```

2) 매크로 상수로 정의해둔 만큼의 pthread 를 반복문으로 생성한다.

```c
#include <math.h>

#define NUM_OF_PTHRDS   5
```

```c
    // create pthreads
    for (i = 0; i < NUM_OF_PTHRDS; i++) {
        if (pthread_create(&thread[i], NULL, packet_receive, (void *)&ports[i]) == 0) {
            printf("pthread %d is created\n", i + 1);
        }
    }
```

3) 생성된 각각의 pthread 의 처리 로직(소켓 생성, 서버와 연결, 수신 데이터 로그 기록)을 기다리는 코드를 작성한다.

```c
// create pthreads
for (i = 0; i < NUM_OF_PTHRDS; i++) {
    if (pthread_create(&thread[i], NULL, packet_receive, (void *)&ports[i]) == 0) {
        printf("pthread %d is created\n", i + 1);
    }
}
```

4) 모든 스레드가 종료되었고 프로그램이 정상적으로 수행되었음을 알리는 문구를 콘솔상에 출력한다.

```
// wait pthreads
printf("Waiting for all the pthreads done\n");
print_divider();
for (i = 0; i < NUM_OF_PTHRDS; i++) {
    pthread_join(thread[i], NULL);
}

print_divider();
printf("%s", "All the packets were received completely\n");
printf("AND all logs were stored in each of .txt files individually.\n");

return 0;
```

- **pthread 를 생성할 때 인자로 전달해주는 함수(packet_receive)의 로직은 다음과 같다.**

1) 파일 입출력 관련 변수, 시스템 시간 저장 변수, 수신 메시지 기록용 변수등을 선언한다.

```
void* packet_receive(void* data) {
    pthread_t t_id = pthread_self();                // point to opened pthread id
    struct sockaddr_in server_addr;
    char buff[MAX_RECV_SIZE + 1];                   // received message buffer
    unsigned int recv_cnt = 0;                      // the # of receiving count from server
    unsigned int port = *(unsigned int *)data;      // connected port number
    unsigned int msg_lngth;                         // received message length
    char msg_lngth_str[10];
    FILE *fp;                                       // .txt file pointer
    char filename[10];                              // .txt file name
    char time_str[15];                              // h:m:s.ms string
    char log_msg[MAX_RECV_SIZE + 100];              // log message buffer
```

2) 소켓을 생성하고 예외처리를 한다. 연결이 정상적으로 이뤄지지 않았다면 -1을 반환하므로 이 경우 콘솔에 문구를 띄우고 프로그램을 비정상적으로 종료시킨다. 연결이 정상적으로 이뤄졌다면 정상적으로 연결되었음을 알려주는 문구를 보여준다.

```c
// create socket
int socket_fd;
if ((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
    printf("(TID: %ld) socket creation failed\n", t_id);
    exit(1);
}

// connect socket
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
server_addr.sin_addr.s_addr = inet_addr("10.37.129.4");

if (connect(socket_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
    printf("(TID: %ld) socket connection failed. (port #%u)\n", t_id, port);
    exit(1);
} else {
    printf("(TID: %ld) socket connection success. (port #%u)\n", t_id, port);
}
```

3) 서버에서 수신하는 메시지를 txt 파일에 기록하기 위한 file pointer 변수를 미리 생성하여 fopen 처리해둔다. 소켓 생성과 마찬가지로 예외처리를 해준다.

```c
// open file with append mode
sprintf(filename, "%d", port);
strcat(filename, ".txt");
if ((fp = fopen(filename, "wa")) == NULL) {
    printf("(port #%u) file cannot be opened.\n", port);
    exit(1);
}
```

4) 연결된 소켓을 통해 서버로부터 메시지를 수신한다. 서버는 무한정 메시지를 송신하므로 while() 을 통해 일정 횟수만큼 수신을 했다면 루프를 빠져나와 함수가 종료되도록 한다. 반복문의 1회 반복마다 다음의 작업들을 수행한다.

- i. 수신한 메시지를 저장할 버퍼(배열)와 txt 파일에 기록할 문자열 변수(배열)를 초기화한다
- ii. recv()를 이용하여 서버로부터 메세지 수신한다. 이 때 매세지 길이는 다른 변수에 저장해둔다
- iii. get_time_str() 이라는 미리 만들어둔 함수로 현재 시간을 형식에 맞춰 문자열로 얻어온다. int 형 이었던 메시지 길이를 문자열로 변환하여 저장한다.
- iv. 현재 시간 문자열, 메시지 길이, 메시지 내용을 strcat()을 이용하여 concatenation 한 뒤 3번에서 열어놓은 file pointer를 이용해 txt파일에 기록을 한다.

```c
// receive packets from server
// AND log them into .txt files.
while (recv_cnt < 50) {
    // flush array at every iteration
    memset(buff, 0, sizeof(buff));
    memset(log_msg, 0, sizeof(log_msg));

    msg_lngth = recv(socket_fd, buff, MAX_RECV_SIZE, 0);
    get_time_str(time_str, sizeof(time_str));
    sprintf(msg_lngth_str, "%u", msg_lngth);

    strcat(log_msg, time_str);                  // h:m:s.ms
    strcat(log_msg, " ");
    strcat(log_msg, msg_lngth_str);             // h:m:s.ms <msg_length>
    strcat(log_msg, " ");
    strcat(log_msg, buff);                      // h:m:s.ms <msg_length> <msg>

    printf("%s\n",log_msg);
    fputs(log_msg, fp);
    fputs("\n", fp);
    recv_cnt++;
}
```

```c
void get_time_str(char *time_str, int len) {
    time_t t;
    struct tm *tm;
    struct timeval tv;
    int ms_int;
    char ms_str[3];

    gettimeofday(&tv, NULL);
    ms_int = lrint(tv.tv_usec / 1000.0);
    tm = localtime(&tv.tv_sec);

    strftime(time_str, len, "%H:%M:%S", tm);
    sprintf(ms_str, "%d", ms_int);
    strcat(time_str, ".");
    strcat(time_str, ms_str);

    return;
}
```

2. pthread를 사용하는 이유

fork는 동일한 프로세스를 데이터 영역, 스택 영역까지 복사하여 생성을 하는데 이 과정은 많은 오버헤드와 메모리 낭비가 발생한다. 쓰레드는 데이터영역은 쓰레드간에 공유가 가능하기 때문에 context switching 비용이 fork를 했을 때보다 줄어든다. 따라서 pthread api를 이용해 멀티 스레드로 작업을 하면 좀더 빠르게, 적은비용으로 처리가 가능하기 때문에 pthread를 사용하였다.

3. 과제 수행시의 trouble 과 trouble shooting

- 서버를 구동하기 위한 1개의 VM 과 클라이언트를 구동하기 위한 1개의 VM을 각각 만들었는데, 서버의 ifconfig 명령 결과에서 확인되는 ip 주소로 ping 을 보내보아도 응답하지 않음.
  >> 원인은 Mac 에서 가상환경을 구성해주는 Parallels Desktop 이라는 application 이 자동적으로 서버의 ethernet adapter 에 dhcp ip를 할당해 주지 않는 것이었다. Dhcp 주소를 할당해주는 linux 명령어인 dhclient 를 실행하니 adapter 에 ip 주소가 부여되면서 문제가 해결되었다.

- 서버 코드를 보니 sending 하는 부분의 while 문에 break 가 걸려있지 않았다. 따라서 적당한 횟수만큼 메시지를 수신하면 서버와 통신하는 스레드를 종료하도록 코드를 작성했다.

- C 언어에서 현재의 시간, 분, 초, 밀리초를 얻는 방법을 잊어버려서 구글 검색을 통해 해결하였다. 시간, 분, 초까지를 얻는데에는 strftime() 함수를 사용했고, 밀리초를 얻는데에는 gettimeofday() 함수를 사용했다. 밀리초를 구하기 위해 gettimeofday 함수로 구한 마이크로초를 1000으로 나눈 값을 floor해야 했는데, 이를 위해 math.h 를 소스에 포함시키고 컴파일시에 -lm 옵션을 주어 정상적으로 컴파일 될 수 있도록 했다.