시스템 프로그래밍 2 차 과제 보고서

제출자 2019320069 정채운

2019320122 이권은

제출일자 2021 년 11 월 8 일

Freeday 사용일수 0 일

소스 코드 설명

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h> //localtime(), tm structure
#include <sys/timeb.h> //timeb structure, ftime()
#include <sys/types.h> //time_t structure in timeb
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <pthread.h>
#define MAX_MSG 65535
```

필요한 라이브러리들을 추가하고, IP 패킷의 최대 크기를 고려하여 서버가 클라이언트에게 보낼 메시지의 최대 길이를 65535 로 설정했다.

int main()

```c
int main(){
    pthread_t thread[5];
    unsigned int server_port[5] = {4444, 5555, 6666, 7777, 8888}; //사용할 port번호 다섯 개. 임의 지정함


    for (int i = 0; i < 5; i++) {
        if (pthread_create(&thread[i], NULL, start_routine, &server_port[i]) != 0) {
            printf("pthread %d create failed\n", i);
            exit(1); //EXIT_FAILURE
        }
    }
    for (int i = 0; i < 5; i++) {
        if (pthread_join(thread[i], NULL) != 0) {
            printf("pthread %d join failed\n", i);
            exit(1); // EXIT_FAILURE
        }
    }
    return 0;
}
```

main 함수에서는 통신에 사용할 포트번호 5 개를 임의로 지정했다. 해당 번호들은 통신 시 서버 쪽에 입력해야 한다. pthread_create 을 이용하여 thread 5 개를 차례로 만드는데, 이 때 start_routine 함수에 포트번호를 매개변수로 넘겨준다. 그리고 pthread_join 을 이용하여 thread 의 작업이 끝나길 기다린다.

void* start_routine(void* arg)

```
//pthread_create에 인자로 넘겨줄 함수. 소켓 생성해서 서버와 연결하고 메세지 수신, 로그 저장. 매개변수:포트번호
void * start_routine(void * arg) {
    unsigned int port = *(unsigned int *)arg;

    //소켓 생성
    int client_socket = socket(AF_INET, SOCK_STREAM, 0); //IPv4 Internet protocols, TCP
    //마지막 인자가 0인 이유: 'IPv4 인터넷 프로토콜 체계'에서 동작하는 '연결지향형 데이터 소켓'은 IPPROTO_TCP하나밖에 없기 때문.
    //그래서 그냥 TCP 소켓이라고도 부르는 것
    if(client_socket < 0){//socket함수는 실패 시 -1을 리턴하므로
        printf("Client: Can't open stream socket\n");
        exit(1);
    }
```

pthread 에 분기시켜서 실행할 함수로 전달한 start_routine 에 대한 설명이다. 먼저 매개변수로 받아온 포트번호를 unsigned int 형식으로 저장한다. socket 을 사용하여 클라이언트의 socket 을 만들어주는데, 이 때 AF_INET 을 address family 로 지정하고, type 에 SOCK_STREAM 을 줘서 IPv4 Internet Protocol, TCP 로 통신하도록 하였다. socket 은 실패 시 -1 을 반환하므로 if 문으로 예외처리를 해주었다.

```
//서버 주소 설정
struct sockaddr_in server_addr; //서버 주소 저장 예정
memset(&server_addr, 0, sizeof(server_addr)); //initialize
server_addr.sin_family = AF_INET; //IPv4 Internet protocols
server_addr.sin_port = htons(port); // host byte order to network byte order. big endian방식으로 정렬
server_addr.sin_addr.s_addr = inet_addr("192.168.56.102"); //ifconfig로 확인한 서버의 네트워크 주소. 문자열로 표현한 주소를 network byte로
```

서버 주소를 sockaddr_in 에 저장한다. 이때 포트번호는 htons 를 이용하여 network byte order (big endian 형식)으로 바꾼다. IP 주소 역시 inet_addr 를 이용하여 network byte 로 바꾼다.

```
//connection 요청
if(connect(client_socket, (struct sockaddr*)&server_addr, sizeof(server_addr))<0){ //connect()는 실패 시 -1을 리턴하므로
    printf("port %u connection failed\n", port);
    exit(1);
}
```

connect 를 이용하여 서버에 connection 을 요청한다. -1 이 리턴될 경우 실패한 것이므로 예외처리 해준다.

```
//로그 저장할 파일 오픈
char filename[10];
sprintf(filename, "%u.txt", port); //<포트번호>.txt 파일명 지정
FILE * fp = fopen(filename, "wa"); //write(쓰기), append(추가쓰기) 옵션
if (fp == NULL) {
    printf("File %s open failed\n", filename);
    exit(1); // EXIT_FAILURE
}
```

통신 시 로그를 저장할 파일을 오픈하는 부분이다. 파일 이름은 <포트번호>.txt 로 지정하고, fopen 에 wa 옵션을 줘서 write 와 append 가 가능하도록 하였다.

```
//패킷 수신할 때마다 파일에 기록
unsigned int msg_cnt = 0; //수신 횟수 제한
char msg[MAX_MSG+1]; //받은 메시지 저장할 버퍼
unsigned int msg_len; //받은 메시지 길이
char time[15]; //h:m:s.ms
while (msg_cnt < 100) { //수신 횟수를 100회로 제한
    memset(msg, 0, sizeof(msg));
    msg_len = recv(client_socket, msg, MAX_MSG, 0); //socket으로부터 data를 받아오는 함수. 성공 시 실제 데이터 길이, 실패 시 -1 리턴
    if(msg_len < 0){
        printf("receive failed\n");
    }
    get_time(time); //현재 시간 h:m:s.ms 형식으로 저장
    fprintf(fp, "%s %d %s\n", time, msg_len, msg); //파일에 시간, data 길이, data 쓰기
    msg_cnt++;
}
fclose(fp);
```

서버로부터 받아온 data 를 저장한다. data 를 수신할 횟수는 msg_cnt 와 while 문을 이용하여 임의로 설정하였다(100 회). recv 를 이용하여 msg 에 data 를 받아온다. recv 함수는 성공 시 데이터의 길이, 실패 시 -1 을 리턴하므로 반환값이 0 보다 작을 때 경고를 출력하도록 했다. 따로 만든 get_time 함수를 호출하여 time 문자열에 시간을 받아오고, 파일에 시간, data 의 길이, data 를 저장한다. 모든 data 를 받은 뒤에는 fclose 로 파일을 닫는다.

```
close(client_socket); //참조 카운터가 0이면 socket 닫고 통신 종료
pthread_exit(NULL); //pthread_join에 NULL값을 넘겨주고 thread 종료
//exit은 program을 종료, pthread_exit은 해당 thread만 종료
//return은 pthread_cleanup_push로 등록된 handler들을 호출하지 않지만, pthread_exit은 handler들을 호출하여 pthread_clenaup_pop을 수행
```

해당 thread 의 통신을 끝내기 위해 close 로 클라이언트의 socket 을 닫아주고, pthread_exit 으로 thread 를 종료시킨다. 이때 pthread_exit 에 넣은 매개변수(위의 경우 NULL)은 pthread_join 에 넘기게 된다. pthread_exit 은 program 을 종료하는 exit 과 달리 thread 만 종료한다는 차이점이 있다. 또한 return 과 비교하면 return 은 pthread_cleanup_push 로 등록된 handler 들을 호출하지 않지만, pthread_exit 은 handler 들을 호출하여 pthread_cleanup_pop 을 수행한다.

void get_time(char* time)

```
//현재 시간 h:m:s.ms 형식으로 저장
void get_time(char * time) {
    struct timeb tb;
    struct tm *tm;
    ftime(&tb); // sets the time and millitm members of the timeb structure
    tm = localtime(&tb.time); //달력시간(년,월,일,시,분,초)로 변환 (returns a pointer to borken-down time structure)
    sprintf(time, "%02d:%02d:%02d.%03d", tm->tm_hour, tm->tm_min, tm->tm_sec, tb.millitm);
    //0은 채워질 문자, 2는 총 자리수. time에 형식에 맞춰 저장한다
}
```

get_time 은 서버와 통신 시 원하는 형식(h:m:s.ms)에 맞춰 시간을 가져오기 위해 별도로 만든 함수이다. 위에서 설명한 start_routine 함수에서 이 함수를 문자열 포인터 time 을 매개변수로 주며 호출한다.

timeb 구조체는 다음과 같은 멤버 변수들로 구성된다. 밀리초를 가져오기 위해 사용하였다.

```
time_t          time     The seconds portion of the current time.
unsigned short  millitm  The milliseconds portion of the current time.
short           timezone The local timezone in minutes west of Greenwich.
short           dstflag  TRUE if Daylight Savings Time is in effect.
```

tm 구조체는 시간을 달력시간(년, 월, 일, 시, 분, 초. broken-down time 이라고도 함)의 형식으로 저장한다. 다음과 같은 멤버 변수들로 구성된다.

```
int    tm_sec   seconds [0,61]
int    tm_min   minutes [0,59]
int    tm_hour  hour [0,23]
int    tm_mday  day of month [1,31]
int    tm_mon   month of year [0,11]
int    tm_year  years since 1900
int    tm_wday  day of week [0,6] (Sunday = 0)
int    tm_yday  day of year [0,365]
int    tm_isdst daylight savings flag
```

먼저 ftime 으로 tb 변수의 time(초)과 millitm(밀리초)를 설정하고, localtime 을 이용하여 tb 의 time 을 broken-down time 으로 바꿔준다. 그 후 sprintf 를 사용하여 매개변수로 받은 문자열 포인터에 원하는 형식으로 시간을 저장한다.

pthread 를 사용하는 이유

각 포트마다 fork로 프로세스를 만들지 않고 pthread로 스레드를 만들어 사용하는 이유는 공유 메모리의 존재이다. 스레드는 code, data, file등 resource를 공유하는 공유 메모리를 갖는데, 이는 프로세스 간 통신(IPC)와 비교했을 때 정보 교환이 빠르기 때문에 통신 속도가 높아진다. 또한 프로세스는 생성할 때마다 system call로 자원을 할당해주어야 하지만 스레드는 이런 과정이 필요치 않아 자원의 효율성을 높일 수 있다.

과제 수행 시 어려웠던 부분과 해결 방법

처음에는 시간을 가져오기 위해 clock_gettime을 사용했는데, clock_gettime에서 받는 매개변수인 timespec 구조체는 초(tv_sec)와 나노초(tv_nsec)로 이루어져 있었다. 즉, 시간을 h:m:s.ms 형식으로 표기하기 위해 밀리초를 가져오려면 나노초를 변환하는 추가 과정이 필요했다. 그래서 이런 과정이 필요 없도록 애초에 밀리초를 사용하는 시간과 관련된 구조체는 없는지 찾아보았고, timeb 구조체를 알게 되어 이번 과제에서는 timeb와 ftime함수를 사용하였다.