데이터 통신 UDP echo

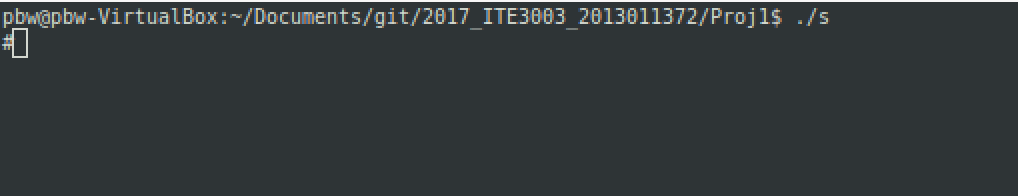
2013011372 박병욱

hconnect.hanyang.ac.kr

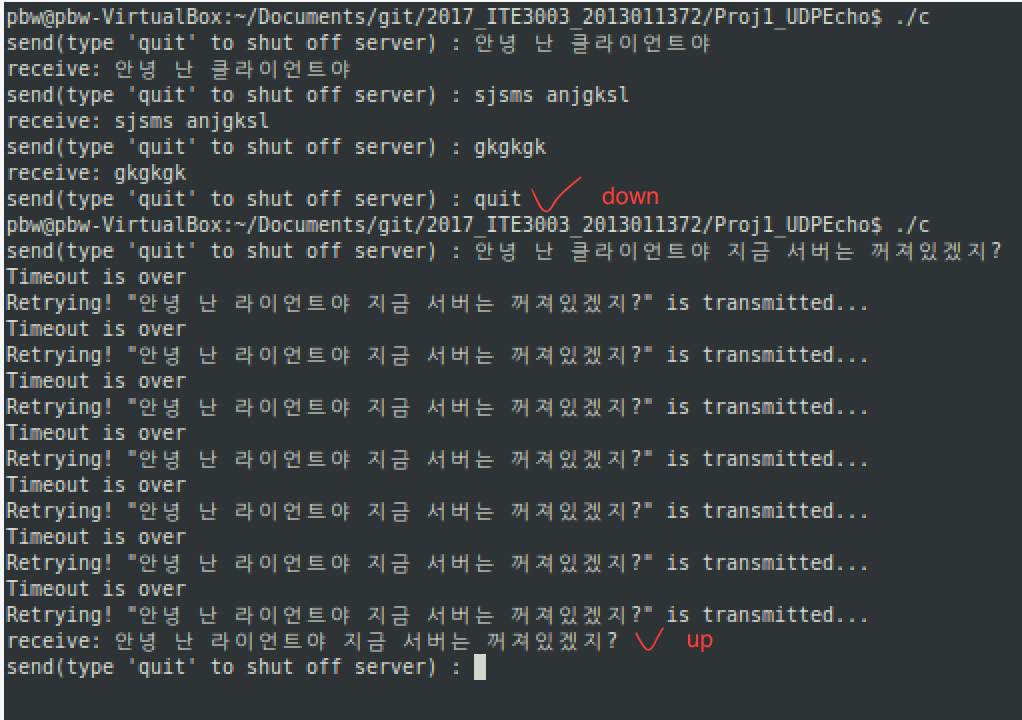
* [Proj1\_UDPEcho](https://hconnect.hanyang.ac.kr/2017_ITE3003_10057/2017_ITE3003_2013011372/tree/master/Proj1_UDPEcho)

디렉토리에 첨부하였습니다

1. 서버실행



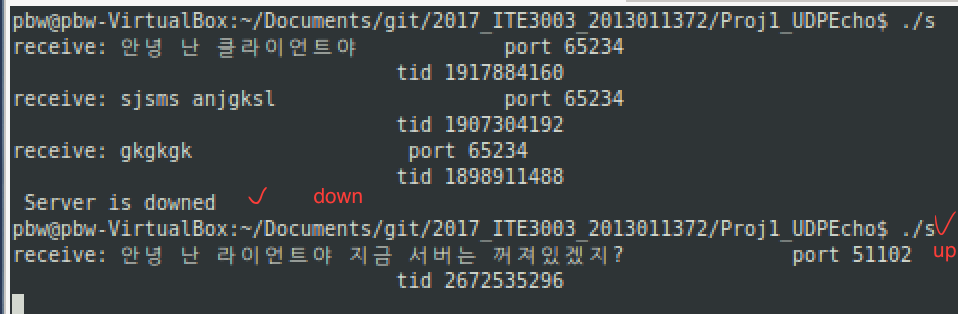
2. 클라이언트 실행/ 패킷전송/ 서버 종료/ 클라이언트 재실행/ 타임아웃 테스트/ 서버 재실행



down : 서버가 꺼진 상태

up : 서버가 켜진상태

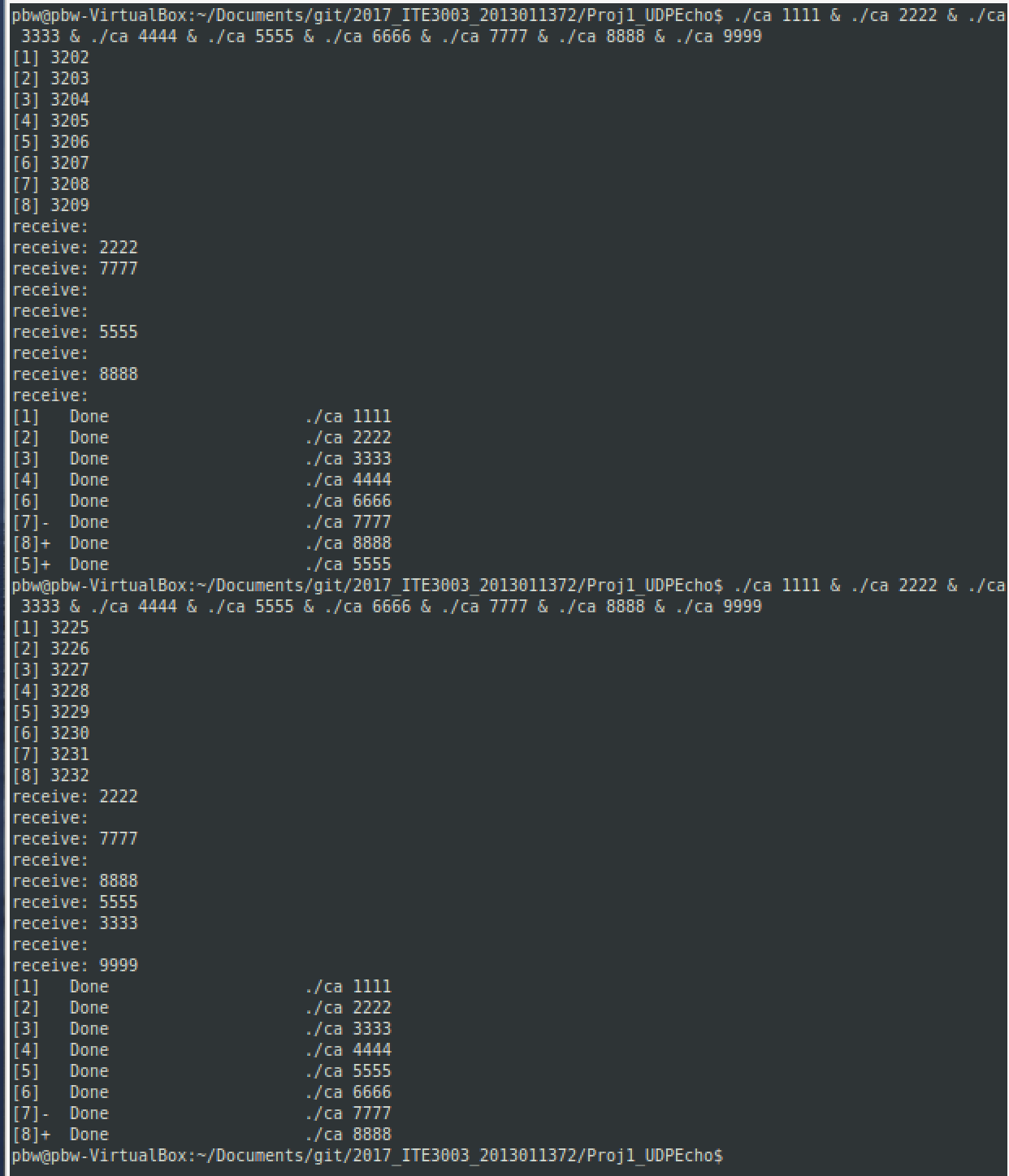
2.0 현 상황에서 서버모습



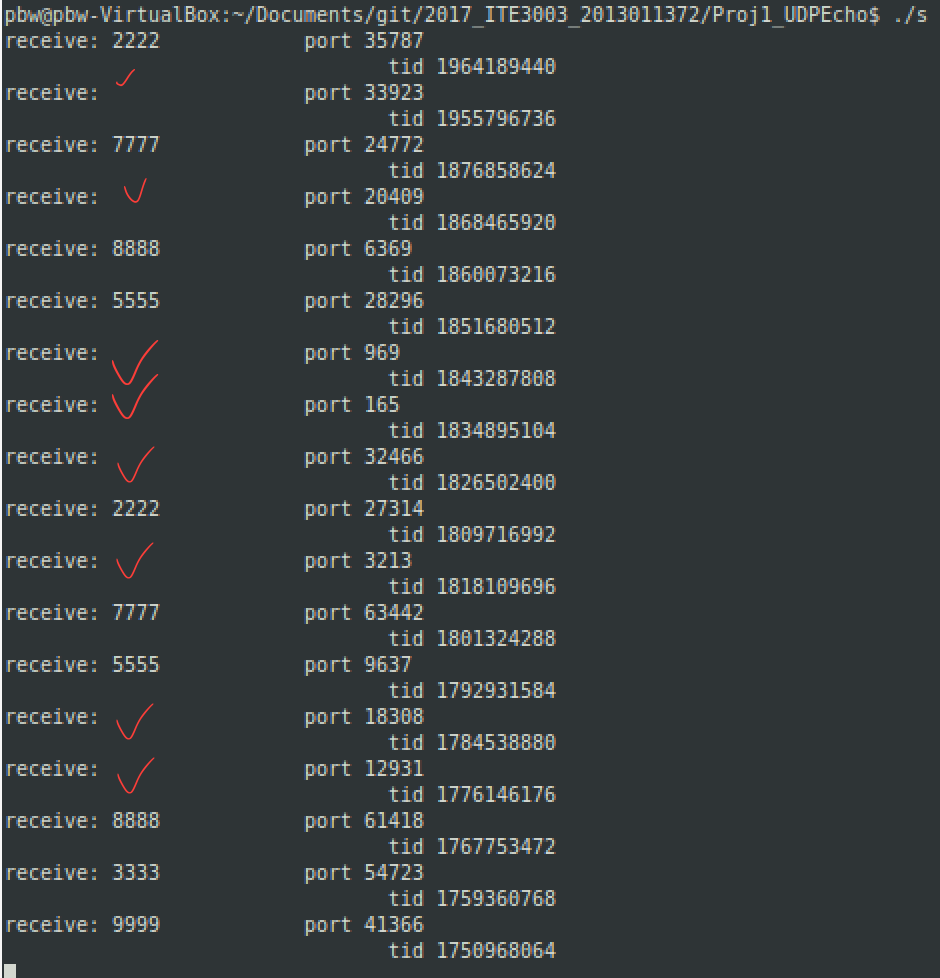
down : 서버가 꺼진 상태

up : 서버가 켜진상태

3. 타임아웃을 설정하지 않은 다수의 클라이언트 실행



3.0.현상황에서 서버의 모습



체크된 부분은 패킷이 손실되었다.

4. 출력 결과 설명

* tid는 스레드의 id를 의미한다.
* port 는 들어온 클라이언트의 port number를 의미한다.
* $./c 실행시

- 클라이언트는 stdin으로 부터 입력을 받고, 이 패킷 sendto()를 통해 서버에게 전송하고, 이에 대한 결과를 recvfrom()을 통해 받고 출력한다. 이를 반복한다.

- 이 때, “quit”이 입력되었을 경우, 서버와 클라이언트 모두를 종료한다.

* $./s 실행시

- 서버는 socket(),bind()를 통해 소켓을 생성한 후 server\_addr구조체와 소켓을 연결하고, 이 소켓으로 패킷을 받는다.

- recvfrom()함수를 통해 패킷을 받고, 패킷을 받을 때마다, pthread\_create()를 통해, 새로운 스레드를 만들고, 데이터를 전송한다.

-이 때, 스레드는 RcvThread()함수를 실행하며, 인자로 넘어온 threadArgP에서 sockfd와 client\_addr, buff\_rcv를 카피해서 각각 스레드 스택에 저장한다.

-이를 이용하여 sendto()함수로 클라이언트에게 패킷을 보낸다.

* $./ca arg 실행시

-argv[1]를 패킷으로 보낸다.

-타임아웃이 설정되어 있지 않았다.

5. 소스코드

5.0 server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <pthread.h>

#define BUFF\_SIZE 4096

#define MAX\_THREAD 100

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

char g\_buff\_rcv[BUFF\_SIZE+5];

struct threadArg{

int sockfd;

struct sockaddr\_in client\_addr;

char\* buff\_rcv;

};

// – UDP 패킷이 오면 오는 즉시 패킷을 받고 화면에 표시해주는 기능

void\* RcvThread(void\* threadArgP){

int sockfd;

int client\_addr\_size;

struct sockaddr\_in client\_addr;

char buff\_rcv[BUFF\_SIZE+5];

char buff\_snd[BUFF\_SIZE+5];

pthread\_mutex\_lock(&mutex); //protect thread race

//copy thread arguments to each thread stack

struct threadArg arg = \*((struct threadArg\*)threadArgP); //protect thread race

free(threadArgP);

sockfd = arg.sockfd;

client\_addr = arg.client\_addr;

strcpy(buff\_rcv, arg.buff\_rcv);

memset(&g\_buff\_rcv, 0 ,sizeof(g\_buff\_rcv)); //prevent using the past data

pthread\_mutex\_unlock(&mutex);

printf( "receive: %s \t\t port %d\n", buff\_rcv, client\_addr.sin\_port);

printf(" tid %u\n",(unsigned int)pthread\_self());

sprintf( buff\_snd, "%s", buff\_rcv);

sendto( sockfd, buff\_snd, strlen( buff\_snd)+1, 0,

( struct sockaddr\*)&client\_addr, sizeof( client\_addr));

//flush buff receive & buff send

memset( &buff\_rcv, 0, sizeof( buff\_rcv));

memset( &buff\_snd, 0, sizeof( buff\_snd));

pthread\_exit(0);

return 0;

}

int main( void)

{

int sock;

int client\_addr\_size;

struct sockaddr\_in server\_addr;

struct sockaddr\_in client\_addr;

pthread\_t tid[MAX\_THREAD];

int th\_id;

int receiveByte;

static int threadcnt=0;

struct threadArg \*threadArgP;

sock = socket( PF\_INET, SOCK\_DGRAM, 0);

if( -1 == sock)

{

printf( "socket() error\n");

exit( 1);

}

memset( &server\_addr, 0, sizeof( server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(7788);

server\_addr.sin\_addr.s\_addr= htonl( INADDR\_ANY);

if( -1 == bind( sock, (struct sockaddr\*)&server\_addr, sizeof( server\_addr) ) )

{

printf( "bind() error\n");

exit( 1);

}

while( 1)

{

client\_addr\_size = sizeof( client\_addr);

receiveByte = recvfrom( sock, g\_buff\_rcv, BUFF\_SIZE, 0 ,

( struct sockaddr\*)&client\_addr, &client\_addr\_size);

if(receiveByte > 0){

pthread\_mutex\_lock(&mutex); //protect thread race

//If "quit" was received then shutdown server program

if(strcmp(g\_buff\_rcv,"quit") == 0){

printf(" Server is downed\n");

break;

}

//prepare for new thread argumnets

threadArgP = malloc(sizeof(struct threadArg));

threadArgP->sockfd = sock;

threadArgP->client\_addr = client\_addr;

threadArgP->buff\_rcv = g\_buff\_rcv;

pthread\_mutex\_unlock(&mutex);

th\_id = pthread\_create(&(tid[threadcnt]), NULL, RcvThread,(void \*)threadArgP);

if(th\_id != 0)

{

perror("Thread Create Error");

return 1;

}

threadcnt++;

}

}

close(sock);

}

5.1 client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <signal.h>

#include <errno.h>

static void sig\_handler(int signo)

{

return;

}

#define chop(str) str[strlen(str)-1] = 0x00;

#define BUFF\_SIZE 1024

int main( int argc, char \*\*argv)

{

int sock;

int server\_addr\_size;

int timeoutflag=0;

struct sockaddr\_in server\_addr;

char buff\_rcv[BUFF\_SIZE+5];

char buff\_snd[BUFF\_SIZE+5];

// signal 설정

struct sigaction sigact, oldact;

sigact.sa\_handler = sig\_handler;

sigemptyset(&sigact.sa\_mask);

sigact.sa\_flags = 0;

sigact.sa\_flags |= SA\_INTERRUPT;

if (sigaction(SIGALRM, &sigact, &oldact) < 0)

{

perror("sigaction error : ");

exit(0);

}

sock = socket( AF\_INET, SOCK\_DGRAM, 0);

if( -1 == sock)

{

printf( "socket() error\n");

exit( 1);

}

memset( &server\_addr, 0, sizeof( server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(7788);

server\_addr.sin\_addr.s\_addr= inet\_addr( "127.10.234.1");

while(!feof(stdin)){

//if timeout was over, then do not get input again

if(timeoutflag == 0){

printf("send(type 'quit' to shut off server) : ");

if(NULL != fgets(buff\_snd,BUFF\_SIZE,stdin)){

chop(buff\_snd);

}

}

//send data

if(-1 == sendto( sock, buff\_snd, strlen( buff\_snd)+1, 0,

( struct sockaddr\*)&server\_addr, sizeof( server\_addr)))

{

printf("Data send error\n");

}

//if "quit" was typed shutdown client and server

if(strcmp(buff\_snd,"quit") == 0 ) break;

server\_addr\_size = sizeof( server\_addr);

//if data was send then, set alarm(Timeout) that limits the data receiving time

if(strlen(buff\_snd) != 0) alarm(2);

if (recvfrom( sock, buff\_rcv, BUFF\_SIZE, 0 ,

( struct sockaddr\*)&server\_addr, &server\_addr\_size) <= 0){

//if interrupt is occured in this block, it means timeout is over

if (errno == EINTR)

{

printf ("Timeout is over\n");

printf("Retrying! \"%s\" is transmitted...\n",buff\_snd);

timeoutflag = 1;

continue;

}

}

timeoutflag = 0;

printf( "receive: %s\n", buff\_rcv);

memset( &buff\_rcv, 0, sizeof( buff\_rcv));

memset( &buff\_snd, '\0', sizeof( buff\_snd));

alarm(0);

}

close( sock);

return 0;

}