실습1

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<string.h>  int main() {  FILE\* fp = fopen("c:\\tmp.txt", "r");  char buf[7];  fgets(buf, sizeof(buf), fp);  printf("%s\n", buf);  fclose(fp);  return 0;  } |

텍스트이(가) 표시된 사진

자동 생성된 설명

**친구관리 프로그램**

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  #include <io.h>  typedef struct \_friend {  char\* name;  char\* message;  int status;  } friend;  friend\* list[100];  int list\_num = 0;  void Add() {  int size = 0;  int mes = 0;  friend\* frd = 0;  frd = (friend\*)malloc(sizeof(friend));  printf("name size: ");  scanf("%d", &size);  frd->name = (char\*)malloc(size);  printf("name: ");  int len;  len = \_read(0, frd->name, size);  if (frd->name[len - 1] == '\n')  frd->name[len - 1] = '\0';  printf("message size: ");  scanf("%d", &mes);  frd->message = (char\*)malloc(mes);  printf("message: ");  int lenn;  lenn = \_read(0, frd->message, mes);  if (frd->message[lenn - 1] == '\n')  frd->message[lenn - 1] = '\0';  printf("status (0 / 1/ 2) : ");  scanf("%d", &frd->status);  list[list\_num++] = frd;  }  void Delete() {  int del\_num;  printf("몇번째 학생을 삭제하실 건가요? : ");  scanf("%d", &del\_num);  for (int i = del\_num - 1; i < list\_num - 1; i++) {  memcpy(&list[i], &list[i + 1], sizeof(friend));  }  }  void Print\_one() {  char find[10];  printf("검색할 이름을 입력해주세요 : ");  scanf("%s", find);  for (int i = 0; i < list\_num; i++) {  if (strcmp(list[i]->name, find) == 0) {  printf("name: %s\n", list[i]->name);  printf("message: %s\n", list[i]->message);  printf("status: %d\n", list[i]->status);  }  }  }  void View() {  printf("======== List =========\n");  for (int i = 0; i < list\_num; i++) {  printf("name: %s\n", list[i]->name);  printf("message: %s\n", list[i]->message);  printf("status: %d\n", list[i]->status);  printf("----------------------------\n");  }  }  void Save() {  FILE\* fp;  fp = fopen("data.txt", "wt");  for (int i = 0; i < list\_num; i++) {  fprintf(fp, "%s %s %d\n", list[i]->name, list[i]->message, list[i]->status);  }  fclose(fp);  }  void Load() {  FILE\* fp = fopen("data.txt", "rt");  int line = 0;  char c;  if (fp == NULL) {  printf("가져올 데이터가 없습니다. \n");  }  else {  while ((c = fgetc(fp)) != EOF) {  if (c == '\n')  line++;  }  list\_num = line;  fseek(fp, 0, SEEK\_SET);  for (int i = 0; i < line; i++) {  list[i] = (friend\*)malloc(sizeof(friend));  list[i]->name = (friend\*)malloc(10);  list[i]->message = (friend\*)malloc(10);  fscanf(fp, "%s %s %d\n", list[i]->name, list[i]->message, &list[i]->status);  }  fclose(fp);  printf("불러오기가 완료되었습니다. \n");  }  }  void Bye() {  printf("프로그램을 종료하겠습니다.\n");  exit(0);  return 0;  }  int main() {  printf("======== CHOOSE NUMBER =========\n");  printf(" 1. Add \n");  printf(" 2. Delete \n");  printf(" 3. Print\_one \n");  printf(" 4. View \n");  printf(" 5. Save \n");  printf(" 6. Load \n");  printf(" 7. Bye \n");  int num = 0;  while (1) {  printf("번호를 입력하세요 : ");  scanf("%d", &num);  switch (num) {  case 1:  Add();  break;  case 2:  Delete();  break;  case 3:  Print\_one();  break;  case 4:  View();  break;  case 5:  Save();  break;  case 6:  Load();  break;  case 7:  Bye();  break;  }  }  return 0;  } |

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**#1 파일 디스크립터와 파일 포인터 개념과 차이점**

**파일 :** 스트림으로 취급되어, 연속된 바이트라고 볼 수 있음

파일 포인터를 이용해 입출력 동작이 발생하는 위치를 알 수 있음

파일을 처음 열면 파일 포인터는 파일의 첫 번째 바이트를 가리키게 되고, 입출력 연산이 실행되며 파일 포인터가 자동적으로 이동됨.

**파일 포인터:** file\* 타입의 포인터 변수

C라는 언어에서 제정한 표준 입출력 객체, 구조체 형식으로 더 많은 정보를 지님

Ex) **FILE \*fopen(const char \*name, const char \*mode)**

파일 포인터로 해당 파일에 대한 정보를 돌려줌

**파일 디스크립터:** int 타입으로 선언되는 고유 식별 번호

각 OS별로 제공되는 IO시스템 콜에서 각 IO자원을 구별하기 위해 사용

OS에서 제공하는 입출력 식별자

+) OS= Operating System, 컴퓨터 시스템의 각종 하드웨어 적인 자원과 소프트웨어적인 자원을 효율적으로 운영 관리함으로써, 사용자가 시스템을 이용하는데 편리함을 제공하는 시스템 소프트웨어를 일컫는 말

IO = 입력(Input)/출력(Output)의 약자로, 컴퓨터 및 주변장치에 대하여 데이터를 전송하는 프로그램, 운영 혹은 장치를 일컫는 말

|  |  |
| --- | --- |
| **파일 포인터** | **파일 디스크립터** |
| 어떤 플랫폼에서든 동일한 동작 | 특정한 플랫폼에서만 동작할 가능성 높음 |
| 고수준 파일 입출력 | 저수준 파일 입출력 (open, write) |

**#2 네트워크 포트 개념**

**포트 :** **애플리케이션 상호 구분(프로세스 구분)을 위해 사용하는 번호**

**포트 숫자**는 **IP 주소가 가리키는 PC에 접속할 수 있는 통로(채널)**을 의미

사용중인 포트 중복 사용 X

0~65,535까지 사용 O

0번 ~ 1023번: [잘 알려진 포트](https://ko.wikipedia.org/wiki/%EC%9E%98_%EC%95%8C%EB%A0%A4%EC%A7%84_%ED%8F%AC%ED%8A%B8) well (-known port)

IANA(최상위 도메인 등을 관리하는 단체)의 권고안

보안 등 여러 이유로 다른 용도로 사용하거나 잘 알려진 포트에 할당된 기능을 등록된 포트 또는 동적 포트에 할당해 사용하기도 함

1024번 ~ 49151번: 등록된 포트 (registered port), 비특권 포트

시스템에서 슈퍼 유저 권한이 없어도 사용 가능

클라이언트가 서버쪽에 서비스를 요청하고 서비스를 받기 위해, 서버쪽에 등록되는 클라이언트쪽 포트

49152번 ~ 65535번: 동적 포트 (dynamic port)

각종 어플리케이션이 통신 작업을 수행할 필요할 때 사용자가 미리 지정하지 않아도 프로그램 등이 임의로 할당하여 사용되는 포트를 의미

**<자주 사용되는 포트>**

22 : SSH

53 : DNS (**웹사이트의 IP 주소와 도메인 주소를 이어주는 환경/시스템)**

80 : HTTP

443: TLS/SSL 방식의 HTTP

**+) TLS/SSL이란?**

SSL(Secure Sockey Layer) - 보안 소켓 계층

데이터를 안전하게 전송하기 위한 인터넷 암호화 통신 프로토콜

전자상거래 등의 보안을 위해 넷스케이프(netscape)에서 처음 만듦

SSL이후에 표준화하여 만들어진 것 => TLS

**#3 TCP vs UDP**

**TCP** - 인터넷상에서 데이터를 메세지의 형태로 보내기 위해 IP와 함께 사용하는 프로토콜

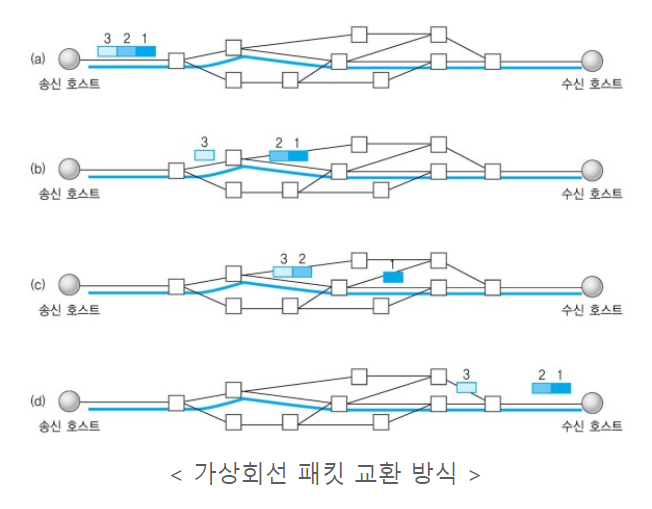
IP – 데이터의 배달을 처리

가상 회선 방식 – 발신지와 수신지를 연결하여 패킷을 전송하기 위한 논리적 경로를 배정함

→ TCP서버는 서버와 클라이언트를 1대 1로 연결

연결형 서비스로 신뢰성을 보장하고 데이터의 흐름이나 혼잡 제어와 같은 기능도 함

→ CPU를 사용하기 때문에 속도에 영향을 줌



**UDP** – 데이터를 데이터그램 단위로 처리하는 프로토콜

+ 데이터 그램 - 패킷교환에서 각각 독립적으로 취급되는 각각의 패킷

정보를 주고 받을 때 정보를 보내거나 받는다는 신호절차를 거치지 않음

연결을 설정하고 해제하는 과정 존재 X

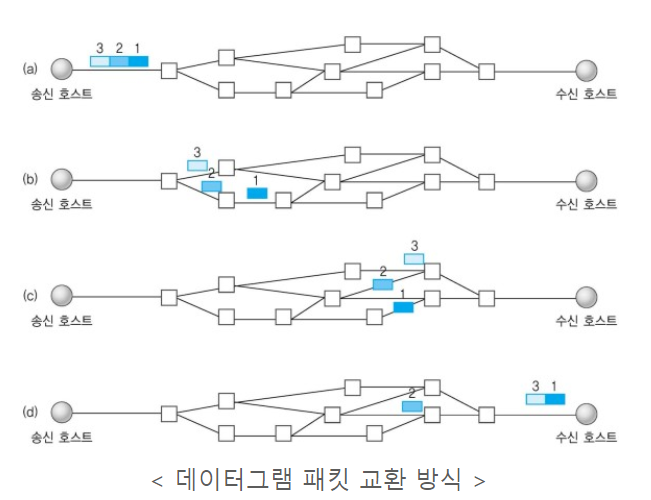
→ UDP 서버는 연결 자체가 없어서 서버 소켓과 클라이언트 소켓의 구분이 없음

서버와 클라이언트가 1대1, 1대N, N대M 등으로 연결 O

서로 다른 경로로 처리해도 패킷에 순서 부여나 재조립을 하지 않음

→ TCP보다 속도는 빠르지만 신뢰성 낮음 (데이터의 전송을 보장하지 않음)

신뢰성 < 연속성 / 실시간 서비스에 주로 사용됨



|  |  |
| --- | --- |
| TCP | UDP |
| 네트워크의 계층들 중 전송 계층에서 데이터를 보내기 위해 사용하는 프로토콜 | |
| 연결형 서비스 | 비연결형 서비스 |
| 가상 회선 방식 | 데이터 그램 |
| 속도 느림 | TCP 보다 속도가 빠름 |
| 신뢰성 높음 | 신뢰성 낮음 |

#4 소켓

Server socket

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<string.h>  #include<unistd.h>  #include<arpa/inet.h>  #include<sys/types.h>  #include<sys/socket.h>  #define BUFF\_SIZE 1024  int main() {  int server\_socket;  int client\_socket;  int client\_addr\_size;  struct sockaddr\_in server\_addr;  struct sockaddr\_in client\_addr;  char buff\_rcv[BUFF\_SIZE + 5];  char buff\_snd[BUFF\_SIZE + 5];  // internet 기반의 소켓 생성 (INET)  server\_socket = socket(PF\_INET, SOCK\_STREAM, 0);  if (server\_socket == -1) {  printf("server socket fail\n");  exit(1);  }  //서버의 IP주소와 PORT 번호를 sockaddr\_in 구조체에 저장  memset(&server\_addr, 0, sizeof(server\_addr));  server\_addr.sin\_family = AF\_INET;  server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  //구조체에 저장한 주소를 socket에 연결  if (bind(server\_socket, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) == -1) {  printf("bind() error\n");  exit(1);  }  if (listen(server\_socket, 5) == -1) {  printf("listen() fail\n");  exit(1);  }  while (1) {  client\_addr\_size = sizeof(client\_addr);  //클라이언트의 연결을 수락  client\_socket = accept(server\_socket, (struct sockaddr\*)&client\_addr, &client\_addr\_size);  if (client\_socket == -1) {  printf("accept() fail\n");  exit(1);  }  //클라이언트로부터 전달받은 데이터를 읽음  read(client\_socket, buff\_rcv, BUFF\_SIZE);  printf("receive: %s\n", buff\_rcv);  printf(buff\_snd, "%d : %s", strlen(buff\_rcv), buff\_rcv);  //전달받은 데이터를 다시 전송  write(client\_socket, buff\_snd, strlen(buff\_snd) + 1);  //클라이언트 소켓 종료  close(client\_socket);  }  } |

Client socket

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<string.h>  #include<unistd.h>  #include<arpa/inet.h>  #include<sys/types.h>  #include<sys/socket.h>  #define BUF\_SIZE 1024  int main(int argc, char\*\* argv) {  int client\_socket; //client socket 정보를 구하기 위해 변수 선언  struct sockaddr\_in server\_addr);  char buff[BUFF\_SIZE + 5];  client\_socket = socket(PF\_INET, SOCK\_STREAM, 0);  if (client\_socket == -1) {  printf("socket create fail\n");  exit(1);  }  memset(&server\_addr, 0, sizeof(server\_addr));  server\_addr.sin\_family = AF\_INET; // IPv4 인터넷 프로토콜  server\_addr.sin\_port = htons(49445); //사용할 port 번호는 49445  server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //서버의 주소  if (connect(client\_socket, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) == -1) {  printf("connect fail\n");  exit(1);  }  //클라이언트로 자료를 송신, NULL 까지 포함해서 전송  write(client\_socket, argv[1], strlen(argv[1]) + 1);  read(client\_socket, buff, BUFF\_SIZE); //클라이언트로부터 전송된 자료를 읽어들임  printf("%s\n", buff);  close(client\_socket); //클라이언트 소켓 종료  return 0;  } |

텍스트이(가) 표시된 사진

자동 생성된 설명

Netstat 명령어

기본 명령어 문법: **netstat [-a] [-l] [-r] [-n] [-c] [-p프로토콜] [-t] [-u]**

|  |  |
| --- | --- |
| 옵션 | 기능 |
| -a | 모든 연결 및 수신 대기 포트를 표시 |
| -c | 현재 실행 명령을 매 초마다 실행 |
| -l | LISTEN 하고 있는 포트를 보여줌 |
| -t | TCP 로 연결된 포트를 보여 줌 |
| -u | UDP 로 연결된 포트를 보여 줌 |
| -n | 주소나 포트 형식을 숫자로 표현함 |
| -p 프로토콜 | 해당 프로세스를 사용하고 있는 프로그램 이름을 보여줌. |
| -r | 라우팅 테이블을 보여 줌 |

명령어 실행시 -> 프로토콜, 로컬 주소, 외부 주소, 상태

**프로토콜 :** 사용한 프로토콜

  로컬 주소 : 열려 있는 사용자 컴퓨터의 IP/호스트네임과 포트 번호

  외부 주소**:** 사용자의 컴퓨터에 접속되어 있는 IP/호스트네임과 포트 번호

  상태 : 연결 상태

예시)

텍스트이(가) 표시된 사진

자동 생성된 설명