Project overview (1/4)

- 프로젝트의 목표
  - C언어를 이용하여 웹서버를 구현한다. (타 언어 허용 불가)
  - 서버와 웹브라우저의 동작 방식에 대해 학습한다.
- 프로젝트 제출
  - 제출 기한: 6월 7일 (일) 23:59
  - 제출 형식: 제출물을 "학번.zip"으로 압축
  - 제출 방법: Blackboard 과제 게시판에 제출
- 참고자료
  - 윤성우의 열혈 TCP/IP 소켓 프로그래밍, Orange media.

Project overview (2/4)

- 프로젝트 내용
  - Part I: Client의 request message를 화면에 출력하는 "Web server" 제작.
    - Safari, Chrome 등 Web browser를 Client로 사용하여 Server에 접속한다.
    - Server로 전달된 request message 각 filed에 해당하는 값이 무엇인지 RFC 1945에서 찾아 레포트를 작성한다. https://tools.ietf.org/html/rfc1945
  - Part II: Web server가 browser의 request에 response할 수 있도록 확장.
    - Browser로부터 받은 request message를 해석한다.
    - 요청 받은 파일에 대한 header를 작성한다.
    - header와 file을 붙여 HTTP response message를 완성하고 이를 client에게 전달한다.

Project overview (3/4)

- 프로젝트 주의사항
  - server는 ./server <port_number> 형태로 동작해야 한다.
    - 충돌 회피를 위해 port는 가급적 1024 이상의 번호를 사용하는 것이 좋다.
  - server와 client를 동일한 컴퓨터에서 동작시킨다면, browser에서 접속을 위한 주소로 localhost나 127.0.0.1을 사용한다.
  - "Content-Type"을 통해 최소한 html 파일을 인식할 수 있도록 만든다.
    - server 제작 후 http://<machine name>:<port number>/<html file name>을 입력하여 정상 작동하는지 확인한다.
    - 이 후 GIF, JPEG, MP3, PDF를 인식할 수 있도록 확장한다.

Project overview (3/4)

- 제출물
  1. 가능한 모든 라인에 주석이 달려있는 소스코드
  2. 소스코드를 컴파일 하기 위한 Makefile (동작해야 함)
  3. 프로젝트 레포트
     - 서버 디자인에 대한 대략적인 설명 및 도식
     - 구현 시 어려웠던 점과 해결 방법
     - 동작 예시: 본인이 작성한 샘플 입력 값에 대한 클라이언트와 서버의 출력 예
- 기타
  - 제출 양식 엄수
  - 채점은 Linux상에서 이루어지며 동작 불가 시 과제 0점, 카피 발견 시 F

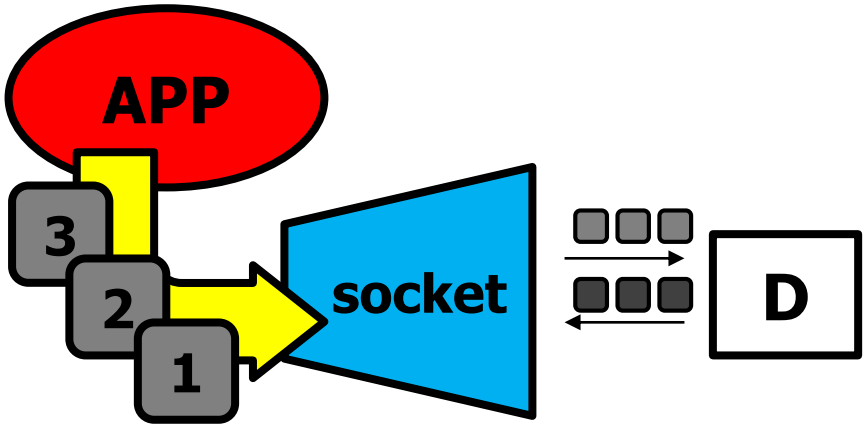
What is a socket

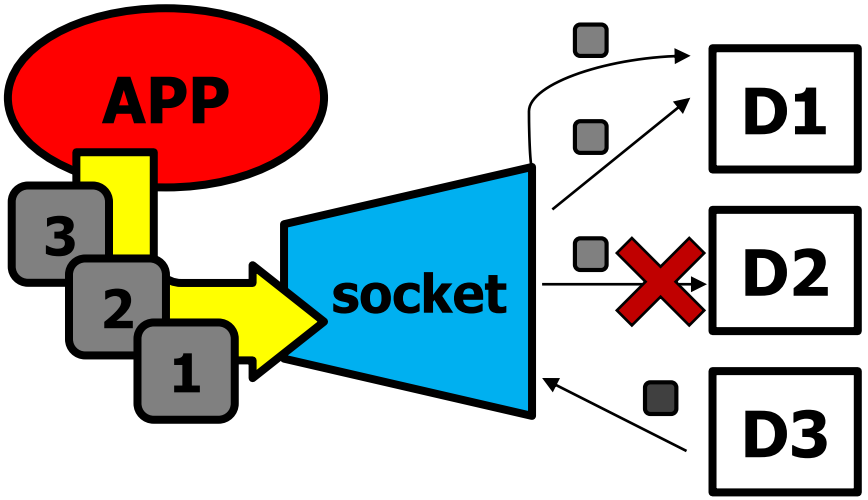
- 네트워크 프로그래밍이란?
  - 네트워크로 연결된 서로 다른 두 컴퓨터가 데이터를 주고 받을 수 있도록 만드는 것

- 소켓이란?
  - 네트워크 상에서 데이터를 주고받을 수 있도록 OS 수준에서 제공하는 소프트웨어적 장치
  - 송수신에 대한 원리를 몰라도 데이터 전달 가능

Two essential types of socket

- SOCK_STREAM
  - TCP를 의미함
  - 안정적으로 전송방식
  - 연결 지향적
  - 양방향 통신
- SOCK_DIAGRAM
  - UDP를 의미함
  - 비안정적 전송방식
  - 비연결 지향적
  - 단방향 통신

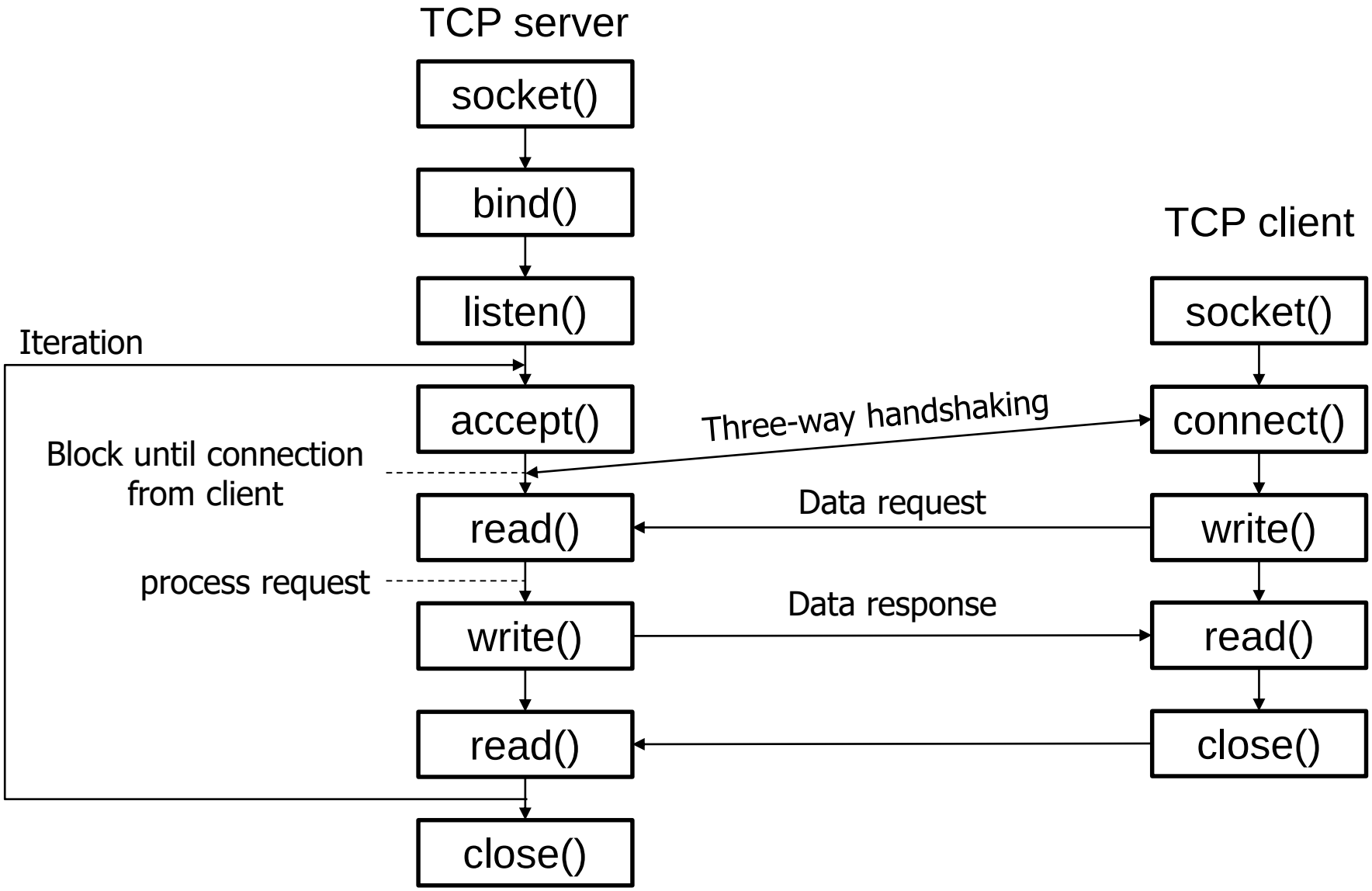




Socket APIs – server side

1. 소켓 생성 : socket()
2. IP 주소와 PORT 번호 할당 : bind()
3. 연결 요청 가능 상태로 변경 : listen()
4. 연결 요청에 대한 수락 : accept()

Big Picture: socket functions (TCP)



Socket creation and setup (1/2)

- `#include <sys/socket.h>` 필요
- socket 생성
  - (비유) 새로운 전화기를 장만하는 동작
  - `int socket(int domain, int type, int protocol);`
  - 반환 값: 성공 시 file descriptor, 실패 시 -1
- 주소 정보에 해당하는 IP, port 번호를 socket에 할당
  - (비유) 전화기에 전화번호를 부여하는 동작
  - `int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen);`
  - 반환 값: 성공 시 0, 실패 시 -1

Socket creation and setup (2/2)

- 연결요청 대기상태로 진입
  - (비유) 전화기에 케이블을 연결하는 동작
  - `int listen(int sockfd, int backlog);`
  - 반환 값: 성공 시 0, 실패 시 -1
- 연결을 허용
  - (비유) 전화벨이 울려 수화기를 드는 동작
  - `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`
  - 반환 값: 성공 시 file descriptor, 실패 시 -1

socket()

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- return: 성공 시 file descriptor, 실패 시 -1
- domain: 소켓이 사용할 프로토콜 체계 (protocol family)
  - PF_INET: IPv4 인터넷 프로토콜 체계
  - PF_INET6: IPv6 인터넷 프로토콜 체계
  - PF_LOCAL: 로컬 통신을 위한 UNIX 프로토콜 체계
  - PF_PACKET: Low level 소켓을 위한 프로토콜 체계
  - PF_IPX: IPX 노벨 프로토콜 체계
- type: 소켓의 데이터 전송방식
  - SOCK_STREAM: 연결지향형 소켓, TCP
  - SOCK_DGRAM: 비 연결지향형 소켓, UDP
- protocol: 두 컴퓨터간 통신에 사용되는 프로토콜
  - 여러 프로토콜이 있을 때 선택을 위해 사용함
  - IPPROTO_TCP: TCP 소켓 생성
  - IPPROTO_UDP: UDP 소켓 생성

bind()

```
#include <sys/types.h>           // (u)int[8|16|32]_t 자료형
#include <sys/socket.h>          // sa_family_t, socklen_t 자료형
#include <netinet/in.h>          // in_addr_t, in_port_t 자료형
int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen);
```

- return: 성공시 0, 실패 시 -1
- sockfd: 주소 정보(IP와 port)를 할당 할 소켓의 file descriptor
- myaddr: 할당하려는 주소 정보를 저장한 구조체 변수의 주소

```
struct sockaddr_in {
    sa_family_t     sin_family;     // 주소체계 (address family)
    uint16_t        sin_port;       // 16비트 TCP/UDP port 번호
    struct in_addr  sin_addr;       // 32비트 IP 주소
    char            sin_zero[8];    // 사용되지 않음
};
struct in_addr {
    in_addr_t       s_addr;         // 32비트 IPv4 인터넷 주소
};
```

- addrlen: 두 번째 인자로 전달한 구조체 변수의 크기

bind() – sockaddr_in

- sin_family: 프로토콜에 따라 주소 체계 선택
  - AF_INET: IPv4 인터넷 프로토콜에 적용하는 주소 체계
  - AF_INET6: IPv6 인터넷 프로토콜에 적용하는 주소 체계
  - AF_LOCAL: 로컬 통신을 위한 UNIX 프로토콜 주소 체계
- sin_port: **네트워크 바이트 순서**로 저장된 16비트 port 번호
- sin_addr: **네트워크 바이트 순서**로 저장된 32비트 IP주소
  - in_addr의 유일한 멤버인 s_addr의 자료형은 32비트 정수
- sin_zero: sockaddr_in의 크기를 sockaddr의 크기와 맞추기 위한 변수

```
struct sockaddr {
    sa_family_t     sin_family;  // 주소 체계 (Address family)
    char            sa_data[14]; // 주소 정보
}
```

  - sa_data에 IP 주소와 port 번호를 채우고 남는 공간에 0을 채움
  - 불편을 해소하기 위해 sockaddr_in이 등장
  - sockaddr_in에 값을 채운 후 sockaddr 타입으로 변환하여 사용

bind() – network byte order

- CPU가 데이터를 저장하고 해석하는 두 가지 방식
  - 데이터는 0x12345678이고, 0x20번지부터 저장된다고 가정
  - Big endian
    - 상위 바이트의 값을 작은 번지수에 저장하는 방식
    - `0x12 0x34 0x56 0x78`
  - Little endian
    - 상위 바이트의 값을 큰 번지수에 저장하는 방식
    - `0x78 0x56 0x34 0x12`
- 네트워크에서는 기본적으로 Big endian 방식을 사용
  - 바이트 순서 변환을 도와주는 함수가 존재
  - h는 호스트, n은 네트워크, s는 short형, l은 long 형
    - `unsigned short htons(unsigned short);`
    - `unsigned short ntohs(unsigned short);`
    - `unsigned long htonl(unsigned long);`
    - `unsigned long ntohl(unsigned long);`

bind() – extra functions (1/2)

```
#include <arpa/inet.h>
in_addr_t inet_addr (const char *string);
```

- 문자열로 표현된 IP주소를 32비트 정수형으로 변환
  - string: "166.104.0.1"과 같이 점으로 구분된 10진수 문자열
  - 성공 시 Big endian으로 변환된 32비트 정수 값
  - 실패 시 INADDR_NONE

```
#include <arpa/inet.h>
int inet_aton(const char *string, struct in_addr *addr);
```

- 문자열로 표현된 IP주소를 32비트 정수형으로 변환 후, in_addr 구조체로 반환
  - string: "166.104.0.1"과 같이 점으로 구분된 10진수 문자열
  - addr: 변환된 정보를 저장할 in_addr 구조체 변수의 주소
  - 성공 시 1, 실패 시 0 반환

bind() – extra functions (2/2)

```
#include <arpa/inet.h>
char * inet_ntoa(struct in_addr addr);
```

- inet_aton과 반대로 네트워크 바이트 순서로 정렬된 정수형 IP 주소를 문자열 형태로 반환
  - addr: 네트워크 바이트 순서로 정렬된 32비트 IP 주소 값
  - 성공 시 문자열 형태의 주소, 실패 시 -1 반환

- INADDR_ANY
  - addr.sin_addr.s_addr = htonl(INADDR_ANY);와 같이 사용할 경우, 서버의 IP 주소를 자동으로 할당

listen()

```
#include <sys/socket.h>
int listen(int sock, int backlog);
```

- return: 성공 시 0, 실패 시 -1
- sock: 연결 요청 대기상태에 두고자 하는 socket의 file descriptor. 이 함수의 인자로 전달된 descriptor의 socket이 server socket (listening socket)이 됨
- backlog: 연결 요청 대기 queue의 크기. 5로 설정하면 client의 연결 요청을 5개 까지 대기시킬 수 있음

accept()

```
#include <sys/socket.h>
int accept(int sock, struct sockaddr *addr, socklen_t *addrlen);
```

- return: 성공 시 생성된 socket의 file descriptor, 실패 시 -1
- sock: server socket의 file descriptor
- addr: 연결 요청한 client의 주소 정보를 담을 변수의 주소 값
  - 함수 호출이 완료되면 client의 주소 정보가 채워 짐.
- addrlen: addr에 전달 된 변수 크기를 바이트 단위로 계산한 값
  - 크기 정보를 변수에 저장한 다음 주소 값을 전달해야 함.
  - 함수 호출이 완료되면 client의 주소 정보 길이가 바이트 단위로 계산되어 채워 짐.

connect()

```
#include <sys/socket.h>
int connect(int sock, struct sockaddr *servaddr, socklen_t addrlen);
```

- return: 성공 시 0, 실패 시 -1
- sock: client socket의 file descriptor
- servaddr: 연결 요청할 server의 주소 정보를 담을 변수의 주소 값
- addrlen: servaddr에 전달 된 변수 크기를 바이트 단위로 계산한 값

Sample code – server (1/2)

```
int main(int argc, char *argv[]) {
  int sockfd, newsockfd, portno, n;
  socklen_t clilen;
  char buffer[256];
  struct sockaddr_in serv_addr, cli_addr;

  if (argc < 2) {
    fprintf(stderr,"ERROR, no port provided\n");
    exit(1);
  }

  sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

  if (sockfd < 0) error("ERROR opening socket");

  bzero((char *) &serv_addr, sizeof(serv_addr));
  portno = atoi(argv[1]);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  serv_addr.sin_port = htons(portno);
```

Sample code – server (2/2)

```
  if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
    error("ERROR on binding");

  listen(sockfd,5);

  clilen = sizeof(cli_addr);
  newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
  if (newsockfd < 0)
    error("ERROR on accept");

  bzero(buffer,256);
  n = read(newsockfd,buffer,255);

  if (n < 0) error("ERROR reading from socket");
  printf("Here is the message: %s\n", buffer);
  n = write(newsockfd,"I got your message",18);

  if (n < 0) error("ERROR writing to socket");
  close(sockfd);
  close(newsockfd);
  return 0;
}
```

Sample code – client (1/2)

```
int main(int argc, char *argv[]) {
  int sockfd, portno, n;
  struct sockaddr_in serv_addr;
  struct hostent *server;
  char buffer[256];

  if (argc < 3) {
    fprintf(stderr,"usage %s hostname port\n", argv[0]);
    exit(0);
  }

  portno = atoi(argv[2]);
  sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (sockfd < 0)
    error("ERROR opening socket");

  server = gethostbyname(argv[1]);
  if (server == NULL) {
    fprintf(stderr,"ERROR, no such host\n");
    exit(0);
  }
```

Sample code – client (2/2)

```
  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->h_length);
  serv_addr.sin_port = htons(portno);

  if (connect(sockfd,(struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    error("ERROR connecting");
  printf("Please enter the message: ");

  bzero(buffer, 256);
  fgets(buffer, 255, stdin);
  n = write(sockfd, buffer, strlen(buffer));
  if (n < 0)
    error("ERROR writing to socket");

  bzero(buffer, 256);
  n = read(sockfd, buffer, 255);
  if (n < 0)
    error("ERROR reading from socket");
  printf("%s\n", buffer);

  close(sockfd);
  return 0;
}
```