Linux Programming
13장. 소켓 (Socket)

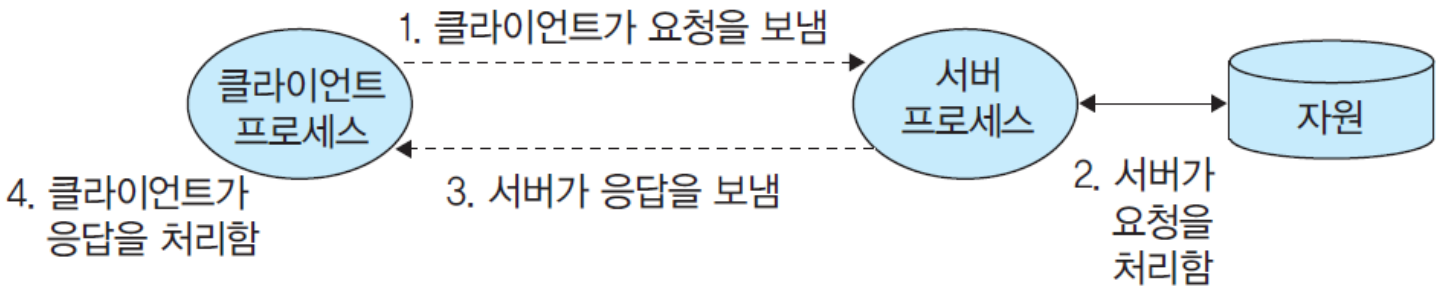
컴퓨터공학과 송석일

sisong@ut.ac.kr

클라이언트-서버 모델

- 네트워크 응용 프로그램
  - 클리이언트-서버 모델을 기반으로 동작

- 클라이언트-서버 모델
  - 하나의 서버 프로세스와 여러 개의 클라이언트로 구성
  - 서버는 어떤 자원을 관리하고 클라이언트를 위해 자원 관련 서비스 제공



Socket 개요 (1/6)

- Socket 이란 ?
  - 같은 기계뿐만 아니라 다른 기계 사이의 프로세스간에 사용하는 양방향 통신기법
  - 소켓을 경유한 프로세스 통신은 클라이언트-서버 모델(client-server model)에 기초하고 있음.

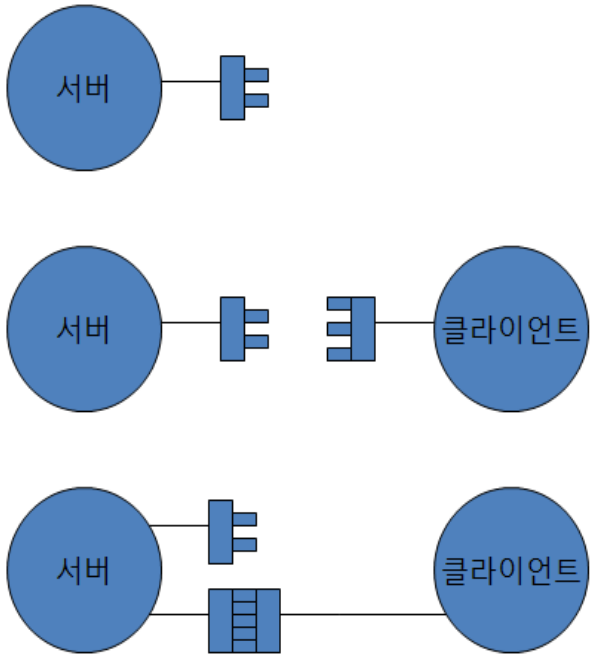
- 소켓 사용 예
  - 어느 한 기계에 존재하는 파일을 다른 기계에서 프린트하기
  - 어느 한 기계에서 다른 기계로 파일을 전송하기
  - 메신저
  - 웹서버, 웹브라우저 등

Socket 개요 (2/6)

- Socket 의 종류
  - 도메인(domain), 유형(type), 프로토콜(protocol) 에 따라 분류
- 도메인
  - 서버와 클라이언트 소켓이 존재하는 장소를 가리킴
  - PF_UNIX : 클라이언트와 서버는 동일한 기계에 존재해야 함
  - PF_INET : 클라이언트와 서버는 인터넷 어느 곳에 존재 가능
- 유형 :
  - 클라이언트와 서버 사이에 존재할 수 있는 통신의 유형
  - SOCK_STREAM : 일련 번호가 붙은, 신뢰적, 양방향 연결에 기초한 바이트의 가변 길이의 스트림 (TCP)
  - SOCL_DGRAM : 전보와 비슷한, 무연결(connectionless), 비신뢰적 고정 길이의 메시지 (UDP)

Socket 개요 (3/6)

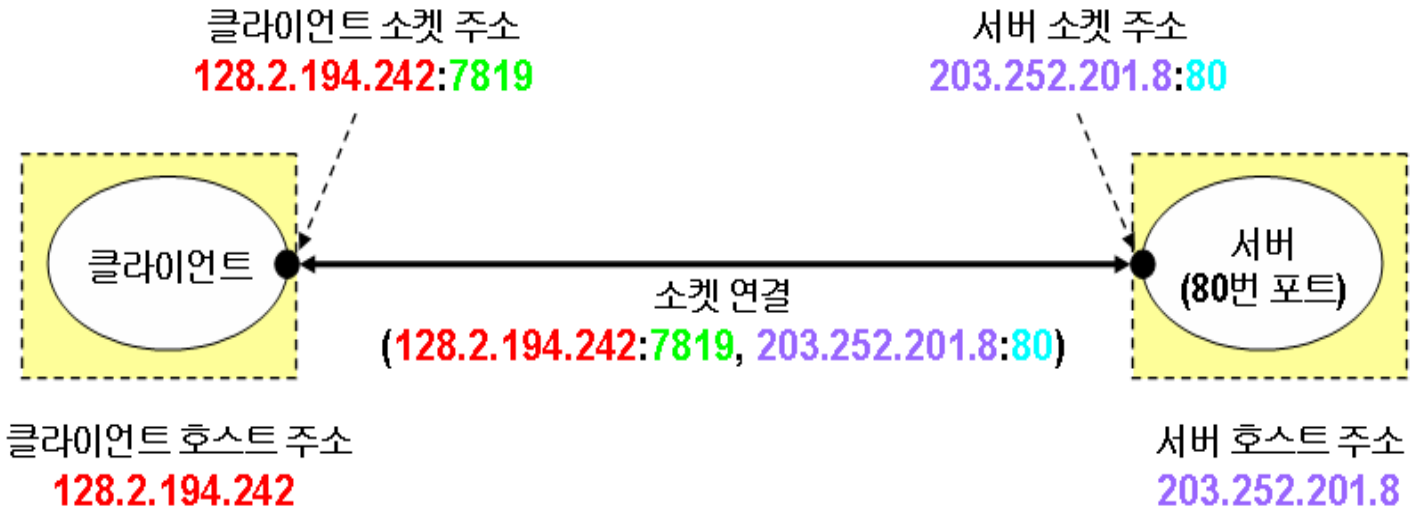
- 서버가 소켓 생성
- 클라이언트가 소켓을 만든 후 서버에 연결 요청
- 서버가 클라이언트의 연결 요청을 수락하여 소켓 연결 완성



Socket 개요 (3/6)

- 인터넷 소켓 연결
  - 서로 다른 호스트에서 실행되는 클라이언트-서버 사이의 통신
  - 소켓을 식별하기 위해 (호스트의 IP 주소, 포트 번호)를 사용

- 인터넷 소켓 연결 예

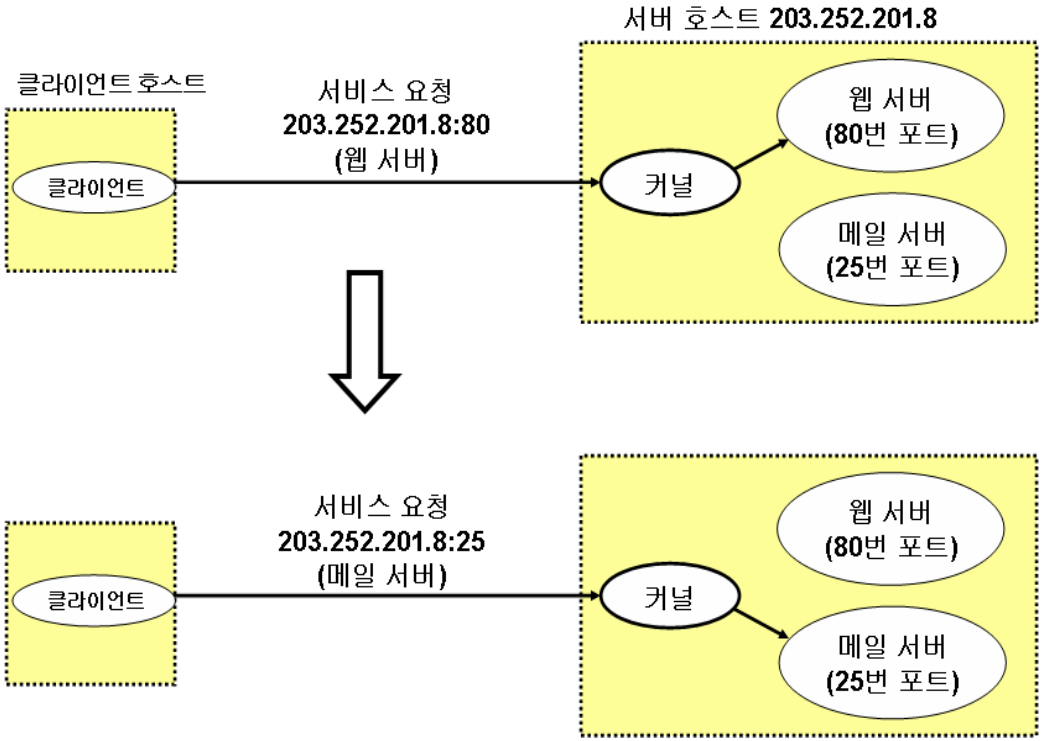


Socket 개요 (3/6)

- 인터넷 소켓 통신을 사용하는 SW
  - 웹 브라우저, ftp, telnet, ssh

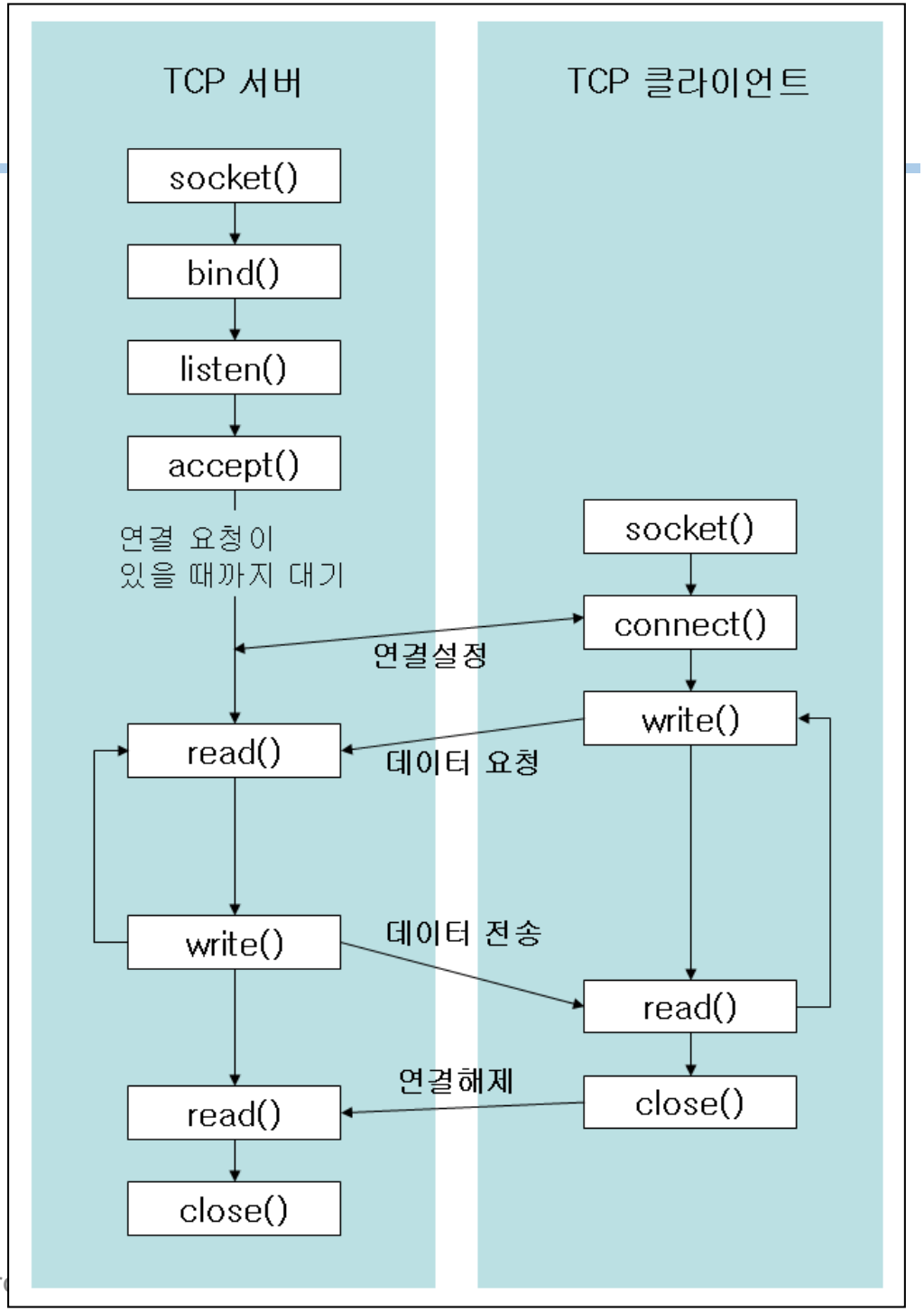
- 주요 서버와 서비스

서버	포트	자원
웹 서버	80	파일과 CGI 프로그램
시간 서버	13	시계
메일 서버	25	이메일 "spool" 파일
ftp 서버	20,21	파일
텔넷 서버	23	가상 터미널



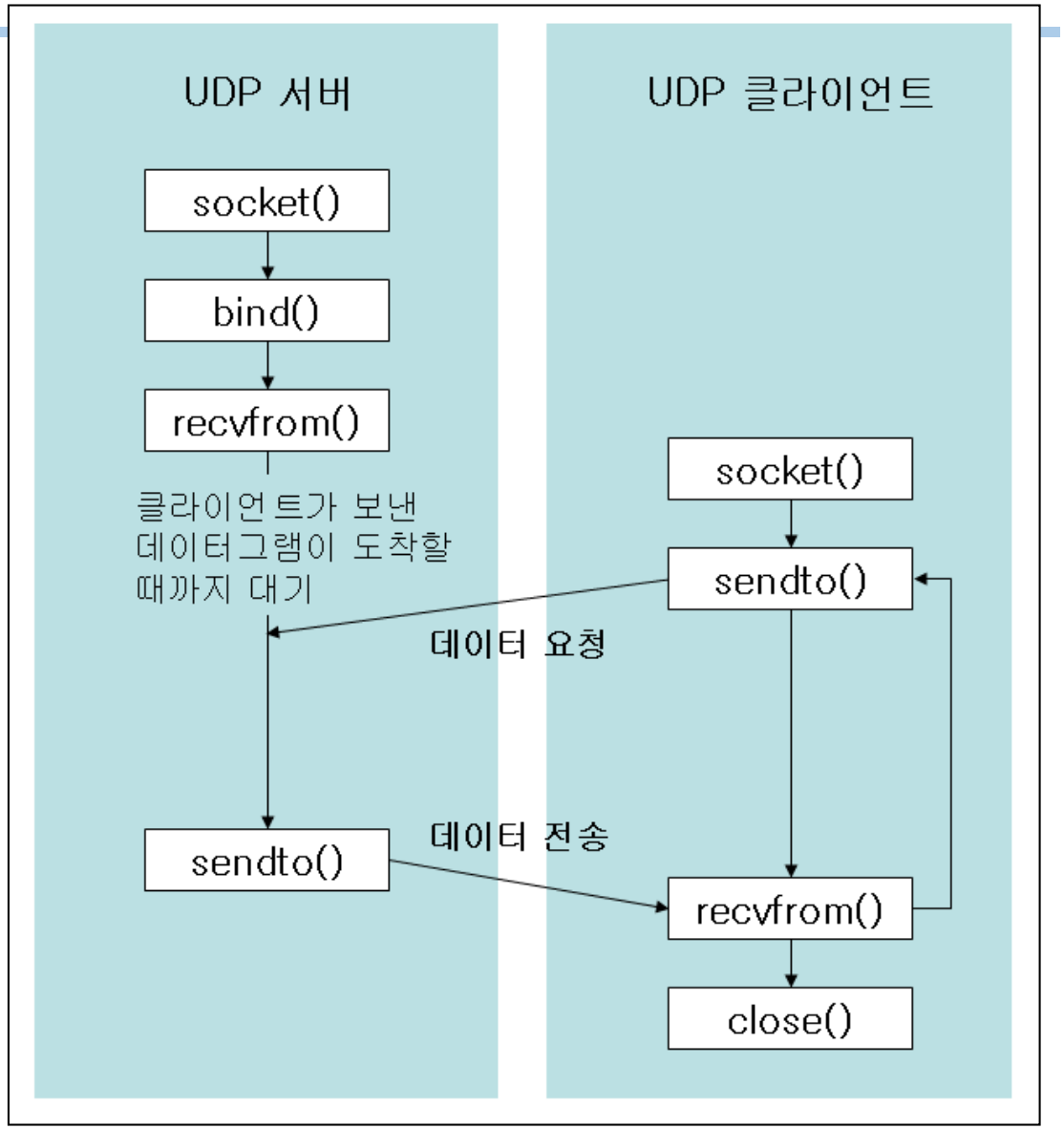
Socket 개요 (4/6)

- TCP Socket
  - Connection Oriented(연결형)
  - 신뢰성 보장



Socket 개요 (5/6)

- UDP Socket
  - Connectionless(비 연결형)
  - 비 신뢰적
  - 전송 속도가 빠름



Socket 개요 (6/6)

- Socket은 파일을 사용하는 것과 유사 함

```
// 파일 생성
int fd = open("myfile", ...);
...
read(fd, ...) // 읽기
write(fd, ...) // 쓰기
```

```
// 소켓 생성
int sock = socket(...);
...
recv(sock, ...) // 받기
send(sock, ...) // 보내기
```

소켓 생성 및 제거

- int socket(int domain, int type, int protocol)
  - 명명되지 않은 소켓 생성
  - 매개변수
    - domain

이름 (매크로 상수)	의미
PF_UNIX, PF_LOCAL	로컬 통신 (동일한 시스템에 있는 프로세스 간의 통신이다.)
PF_INET	IP version 4의 인터넷 통신 프로토콜
PF_INET6	IP Version 6의 인터넷 통신 프로토콜
PF_IPX	노벨 네트워크의 통신 프로토콜

    - type
      - SOCK_STREAM, SOCK_DGRAM
    - protocol : 보통 0 사용
  - 성공한다면, 새로 생성된 소켓에 연관된 파일 기술자를 반환 하고, 그렇지 않으면 –1을 반환
- int close(int sockfd)

소켓에 이름 부여

- int bind(int fd, struct sockaddr* address, int addressLen)
  - 명명되지 않은 소켓에 이름을 부여
  - 매개변수
    - address에 저장된 소켓 주소를 파일 기술자 fd 소켓에 연관 시킴
    - addressLen은 주소 구조체의 길이를 포함해야 함
    - 입력될 주소의 유형과 값은 소켓 도메인에 의함
    - address
      - PF_UNIX 소켓 : sockaddr_un 구조체를 struct sockaddr* 으로 전달
      - PF_INET 소켓 : sockaddr_in 구조체를 struct sockaddr* 으로 전달
  - 성공한다면 0을 반환 함, 실패한다면 –1을 반환

```
struct sockaddr_in {
        sa_family_t     sin_family;     // 주소 체계(address family)
        uint16_t        sin_port;       // 16비트 TCP / UDP Port
        struct in_addr  sin_addr;       // 32비트 IPv4 주소
        char            sin_zero[8];    // struct sockaddr 과 크기를 맞춰줌
};

struct in_addr {
        uint32_t        s_addr;     // 32비트 IPv4 인터넷 주소
};

Local Unix 프로토콜
struct sockaddr_un {
        sa_family_t     sun_family;     // 주소 체계
        char            sun_path[108];  // 경로 이름
};
```

마지막!!! 포괄하는것

```
struct sockaddr {
        sa_family_t     sin_family;
        char            sa_data[14];
};
```

소켓 큐 생성 및 연결 수용

- Int listen(int fd, int queueLength)
  - 대기중인 소켓 연결의 최대 수를 명시
  - 매개변수
    - fd : 파일 기술자
    - queueLength : 연결 최대수

소켓 큐 생성 및 연결 수용

- int accept(int fd, struct sockaddr* address, int* addressLen)
  - 클라이언트 연결 요구를 수용
  - 동작
    - 클라이언트의 연결 요구가 발생하면 원래의 명명된 서버 소켓과 동일한 속성을 갖는 명명되지 않은 소켓을 생성하고, 그것을 클라이언트의 소켓에 연결한 다음, 클라이언트와 통신을 위해서 사용될 수 있는 새로운 파일 기술자를 반환 함.
    - 원래의 명명된 서버 소켓은 다른 연결을 받아들이기 위해서 사용됨
  - 매개변수
    - address : 클라이언트의 주소
    - addressLen : 클라이언트의 주소 크기
  - 반환
    - 성공한다면 클라이언트와 대화하기 위해 사용될 수 있는 새로운 파일 기술자를 반환 하고 그렇지 않으면 –1을 반환 함

연결하기

- int connect(int fd, struct sockaddr* address, int addressLen)
  - 동작
    - address 구조체 내에 주소가 저장 되어 있는 서버 소켓에 연결을 시도
    - 성공한다면, fd는 서버와 통신에 사용됨
    - address 구조체의 유형은 bind()를 설명할 때에 언급한 것과 동일한 규칙들을 따라야 함.
  - 매개변수
    - address
      - PF_UNIX socket : sockaddr_un 사용
      - PF_INET socket : sockaddr_in 사용
    - addressLen : 주소 구조체의 크기와 동일해야 함.
  - 반환
    - 만일 연결되면 0을 반환 함.
    - 만일 서버 소켓이 존재하지 않거나 대기 큐가 가득 차 있다면 –1을 반환 함.

```
/*  Server 1  */
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;

    server_sockfd = socket(PF_INET, SOCK_STREAM, 0);

    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(9734);
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
    listen(server_sockfd, 5);
    while(1) {
        char ch;
        printf("server waiting\n");
        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd,
    (struct sockaddr *)&client_address, &client_len);

        read(client_sockfd, &ch, 1);
        ch++;
        write(client_sockfd, &ch, 1);
        close(client_sockfd);
      }
    }
```

```
/*  Server 2  */

#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>

int main()
{
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;

    server_sockfd = socket(PF_INET, SOCK_STREAM, 0);

    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(9734);
    server_len = sizeof(server_address);
    bind(server_sockfd,
        (struct sockaddr *)&server_address, server_len);
```

```
    listen(server_sockfd, 5);
    while(1) {
        char ch;

        printf("server waiting\n");

        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd,
            (struct sockaddr *)&client_address, &client_len);

        if(fork() == 0) {
            read(client_sockfd, &ch, 1);
            sleep(5);
            ch++;
            write(client_sockfd, &ch, 1);
            close(client_sockfd);
            exit(0);
        }

        else {
            close(client_sockfd);
        }
    }
}
```

```
/* client */
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
    int sockfd;
    int len;
    struct sockaddr_in address;
    int result;
    char ch = 'A';

    sockfd = socket(PF_INET, SOCK_STREAM, 0);

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("127.0.0.1");
    address.sin_port = htons(9734);
    len = sizeof(address);

    result = connect(sockfd, (struct sockaddr *)&address, len);

    if(result == -1) {
        perror("oops: client");
        exit(1);
    }

    write(sockfd, &ch, 1);
    read(sockfd, &ch, 1);
    printf("char from server = %c\n", ch);
    close(sockfd);
    exit(0);
}
```

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>

int main()
{
    int sockfd;
    int len;
    struct sockaddr_un address;
    int result;
    char ch = 'A';

    sockfd = socket(PF_UNIX, SOCK_STREAM, 0);

    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "server_socket");
    len = sizeof(address);

    result = connect(sockfd, (struct sockaddr *)&address, len);

    if(result == -1) {
        perror("oops: client1");
        exit(1);
    }

    write(sockfd, &ch, 1);
    read(sockfd, &ch, 1);
    printf("char from server = %c\n", ch);
    close(sockfd);
    exit(0);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>

int main()
{
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_un server_address;
    struct sockaddr_un client_address;

    unlink("server_socket");
    server_sockfd = socket(PF_UNIX, SOCK_STREAM, 0);

    server_address.sun_family = AF_UNIX;
    strcpy(server_address.sun_path, "server_socket");
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
    listen(server_sockfd, 5);
    while(1) {
        char ch;

        printf("server waiting\n");

        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd,
                               (struct sockaddr *)&client_address,
                               &client_len);

        read(client_sockfd, &ch, 1);
        ch++;
        write(client_sockfd, &ch, 1);
        close(client_sockfd);
    }
}
```