13주차 2차시 TCP 기반 프로그래밍

【학습목표】

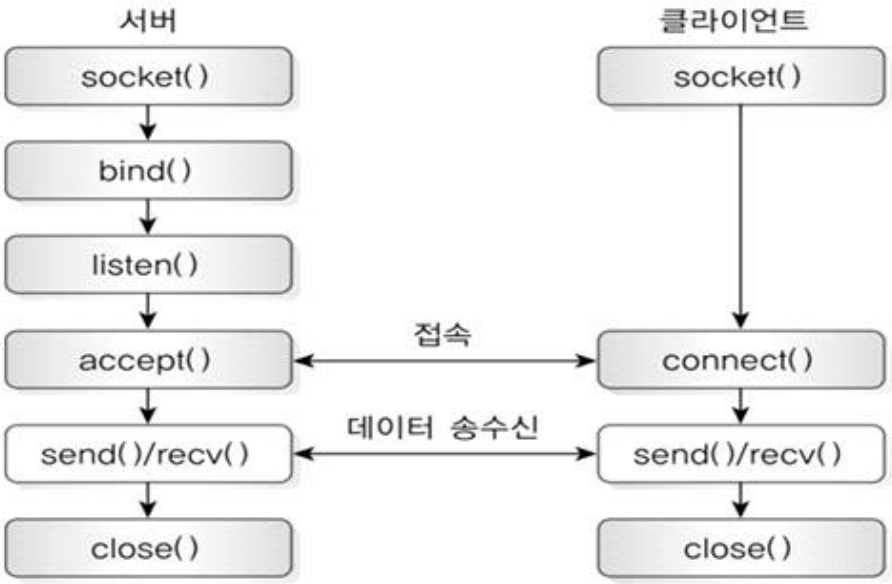
1. TCP 기반 프로그래밍의 원리를 설명할 수 있다.
2. TCP 기반 반복서버와 동시동작 서버를 설명할 수 있다.

학습내용1 : TCP 기반 프로그래밍

1. 동작 방식

* 반복서버 : 데몬 프로세스가 직접 모든 클라이언트의 요청을 차례로 처리
* 동시동작서버 : 데몬 프로세스가 직접 서비스를 제공하지 않고, 서비스와 관련있는 다른 프로세스를 fork 함수로 생성해 클라이언트와 연결시켜준다.

* 인터넷 소켓 활용 통신 절차



* 유형 - standalone 타입
- 시스템에서 독자적으로 프로세스가 구동되어 서비스를 제공하는 데몬
- 메모리에 항상 구동되어 있음.
- 웹서버(httpd), DB서버(mysqld) 등
- 데몬 실행 스크립트 파일 : /etc/init.d에 주로 있음

* 유형 - xinetd 타입
- inetd의 단점을 개선한 데몬
- 대부분의 리눅스에 사용
- 수퍼 데몬이라고 하며 다른 하위의 데몬을 관리하는 상위 데몬
- 외부에서 요청이 있을때만 하위 데몬을 실행시킨 후에 그 데몬이 서비스를 담당하도록 하고 서비스가 종료되면 데몬 종료
- xinetd 데몬 위치 : /etc/xinetd.d
- 데몬 설정 파일 : /etc/xinetd.conf

학습내용2 : 반복서버와 동시동작서버

1. 반복서버 - 서버

```
...
10  #define PORTNUM 9001
11
12  int main(void) {
13      char buf[256];
14      struct sockaddr_in sin, cli;
15      int sd, ns, clientlen = sizeof(cli);
16
17      memset((char *)&sin, '\0', sizeof(sin));
18      sin.sin_family = AF_INET;
19      sin.sin_port = htons(PORTNUM);
20      sin.sin_addr.s_addr = inet_addr("192.168.162.133");
21
22      if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
23          perror("socket");
24          exit(1);
25      }
26
27      if (bind(sd, (struct sockaddr *)&sin, sizeof(sin))) {
28          perror("bind");
29          exit(1);
30      }
31
32      if (listen(sd, 5)) {
33          perror("listen");
34          exit(1);
35      }
37      while (1) {
38          if ((ns = accept(sd, (struct sockaddr *)&cli, &clientlen)) == -1) {
39              perror("accept");
40              exit(1);
41          }
42          sprintf(buf, "%s", inet_ntoa(cli.sin_addr));
43          printf("*** Send a Message to Client(%s)\n", buf);
44
45          strcpy(buf, "Welcome to Network Server!!!");
46          if (send(ns, buf, strlen(buf) + 1, 0) == -1) {
47              perror("send");
48              exit(1);
49          }
50
51          if (recv(ns, buf, strlen(buf), 0) == -1) {
52              perror("recv");
53              exit(1);
54          }
55          printf("** From Client : %s\n", buf);
56          close(ns);
57      }
58      close(sd);
59
60      return 0;
61  }
```

소켓 주소구조체 생성

소켓 생성

클라이언트 접속 대기

클라이언트 접속

클라이언트에 정보전송

클라이언트의 데이터 수신

2. 반복서버 - 클라이언트

```
...
11  #define PORTNUM 9001
12
13  int main(void) {
14      int sd;
15      char buf[256];
16      struct sockaddr_in sin;
17
18      memset((char *)&sin, '\0', sizeof(sin));
19      sin.sin_family = AF_INET;
20      sin.sin_port = htons(PORTNUM);
21      sin.sin_addr.s_addr = inet_addr("192.168.162.133");
22
23      if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
24          perror("socket");
25          exit(1);
26      }
27
28      if (connect(sd, (struct sockaddr *)&sin, sizeof(sin))) {
29          perror("connect");
30          exit(1);
31      }
33      if (recv(sd, buf, sizeof(buf), 0) == -1) {
34            perror("recv");
35            exit(1);
36      }
37
38      printf("** From Server : %s\n", buf);
39
40      strcpy(buf, "I want a HTTP Service.");
41      if (send(sd, buf, sizeof(buf) + 1, 0) == -1) {
42            perror("send");
43            exit(1);
44      }
45
46      close(sd);
47
48      return 0;
49  }
```

소켓 주소구조체 생성

소켓 생성

서버에 연결 요청

서버의 데이터 수신

서버에 데이터 송신

서버

```
# ex12_1s.out
```

클라이언트

```
# ex12_1c.out
** From Server : Welcome to Network Server!!!
```

서버

```
# ex12_1s.out
*** Send a Message to Client(192.168.162.133)
** From Client : I want a HTTP Service.
```

클라이언트

```
# ex12_1s.out
*** Send a Message to Client(192.168.162.131)
** From Client : I want a FTP Service.
```

3. 동시 동작 서버 - 서버

```
...
10  #define PORTNUM 9002
11
12  int main(void) {
13      char buf[256];
14      struct sockaddr_in sin, cli;
15      int sd, ns, clientlen = sizeof(cli);
16
17      if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
18          perror("socket");
19          exit(1);
20      }
21
22      memset((char *)&sin, '\0', sizeof(sin));
23      sin.sin_family = AF_INET;
24       sin.sin_port = htons(PORTNUM);
25      sin.sin_addr.s_addr = inet_addr("192.168.162.133");
26
27      if (bind(sd, (struct sockaddr *)&sin, sizeof(sin))) {
28          perror("bind");
29          exit(1);
30      }
31
32      if (listen(sd, 5)) {
33          perror("listen");
34          exit(1);
35      }
37      while (1) {
38          if ((ns = accept(sd, (struct sockaddr *)&cli, &clientlen)) == -1) {
39              perror("accept");
40              exit(1);
41          }
42          switch (fork()) {
43              case 0:
44                  close(sd);
45                  strcpy(buf, "Welcome to Server");
46                  if (send(ns, buf, strlen(buf) + 1, 0) == -1) {
47                      perror("send");
48                      exit(1);
49                  }
50
51                  if (recv(ns, buf, strlen(buf), 0) == -1) {
52                      perror("recv");
53                      exit(1);
54                  }
55                  printf("** From Client: %s\n", buf);
56                  sleep(5);
57                  exit(0);
58          }
59          close(ns);
60      }
61
62      return 0;
63  }
```

fork로 자식 프로세스 생성

자식 프로세스가 클라이언트로 메시지 보내고 데이터 수신

```
# ex12_2s.out
*** Send a Message to Client(192.168.162.133)
** From Client : I want a HTTP Service.
```

- 클라이언트는 ex12_1c.c를 포트번호만 바꾸고 그대로 사용

* 클라이언트가 접속했을 때 서버의 실행상태

```
# ps -ef | grep pts/2
root  1571  1568   0   2월 08 pts/2      0:03 _ksh
root  7175  7172   0 09:55:32 pts/2       0:00 ex12_2s.out
root  7172  1571   0 09:55:18 pts/2       0:00 ex12_2s.out
```

- 서버 프로세스가 2개 임을 알 수 있다.
- 7172는 부모 프로세스, 7175는 자식 프로세스

4. 동시동작서버 - exec 함수 사용하기 - 서버

```
...
40      while (1) {
41          if ((ns = accept(sd, (struct sockaddr *)&cli,
                            &clientlen)) == -1) {
42              perror("accept");
43              exit(1);
44          }
45          printf("** Accept Client\n");
46
47          switch (fork()) {
48              case 0:
49                  printf("** Fork Client\n");
50                  close(sd);
51                  dup2(ns, STDIN_FILENO);
52                  dup2(ns, STDOUT_FILENO);
53                  close(ns);
54                  execl("./han", "han", (char *)0);
55          }
56          close(ns);
57      }
58
59      return 0;
60  }
```

클라이언트의 요청 처리를 위한 별도의 프로그램(han) 실행

5. 동시동작서버 - han 프로그램

```
01  #include <unistd.h>
02  #include <stdio.h>
03
04  int main(void) {
05      printf("Welcome to Server, from Han!");
06      sleep(5);
07
08      return 0;
09  }
```

간단한 환영메시지 출력

6. 동시동작서버 - exec 함수 사용 - 클라이언트

```
...
28      printf("==> Create Socket\n");
29      if (connect(sd, (struct sockaddr *)&sin, sizeof(sin))) {
30          perror("connect");
31          exit(1);
32      }
33
34      printf("==> Connect Server\n");
35      if ((len = recv(sd, buf, sizeof(buf), 0)) == -1) {
36            perror("recv");
37            exit(1);
38      }
39      buf[len] = '\0';
40
41      printf("==> From Server : %s\n", buf);
42
43      close(sd);
44
45      return 0;
46  }
```

연결요청

메시지 수신

클라이언트

```
# ex12_3c.out
==> Create Socket
==> Connect Server
==> From Server : Welcome to Server, from Han!
```

```
# ps
PID TTY          TIME CMD
676 pts/2        0:00 ksh
760 pts/2        0:00 ex12_3s.out
763 pts/2        0:00 han
```

han 실행

【학습정리】

1. 유형에 따른 타입
* standalone 타입
- 시스템에서 독자적으로 프로세스가 구동되어 서비스를 제공하는 데몬
- 메모리에 항상 구동되어 있음.
- 웹서버(httpd), DB서버(mysqld) 등

2. xinetd 타입
- inetd의 단점을 개선한 데몬- 대부분의 리눅스에 사용
- 수퍼 데몬이라고 하며 다른 하위의 데몬을 관리하는 하위 데몬
- 외부에서 요청이 있을때만 하위 데몬을 실행시킨 후에 그 데몬이 서비스를 담당하도록 하고 서비스가 종료되면 데몬 종료

3. 반복서버와 동시동작서버
* 반복서버(interative server)
- 데몬 프로세스가 직접 모든 클라이언트의 요청을 차례로 처리
* 동시동작서버(concurrent server)
- 데몬 프로세스가 직접 서비스를 제공하지 않고, 서비스와 관련있는 다른 프로세스를 fork() 함수로 생성해 클라이언트와 연결시켜준다.