

네트워크 프로그래밍 및 실습

2007. 12. 1.

정유진

한국외국어대학교 컴퓨터공학전공

본 교재는 2007년도 정보통신부 NEXT 사업의
지원을 받아서 네트워크 프로그래밍 및 실습 과목의
실습 교안으로 작성되었음

<차례>

Chapter 1. 리눅스 설치

1. 리눅스 개괄

| | |
|----------|---|
| 1. 리눅스란? | 9 |
|----------|---|

2. 리눅스 설치

| | |
|---------------|----|
| 1. 설치 준비 | 10 |
| 2. 파티션 준비 | 11 |
| 3. CD-ROM 부팅 | 11 |
| 4. 설치 모드 선택 | 12 |
| 5. Startup | 12 |
| 6. 설치 언어 선택 | 12 |
| 7. 사용자 동의 | 13 |
| 8. 키보드 선택 | 13 |
| 9. 파티션 설정 | 14 |
| 10. 부트로더 설정 | 17 |
| 11. 네트워크 설정 | 18 |
| 12. 지역 시간대 설정 | 18 |
| 13. 관리자 암호 설정 | 18 |
| 14. 설치 패키지 선택 | 19 |
| 15. 설치 시작 | 20 |
| 16. 로그인 유형 선택 | 21 |
| 17. 재부팅 | 21 |

chapter 2. 기본 명령어

1. 명령어 매뉴얼과 도움말 얻기

| | |
|------------|----|
| 1. man | 24 |
| 2. apropos | 25 |
| 3. info | 26 |

2. 파일 관련 명령어

| | |
|----------|----|
| 1. ls | 27 |
| 2. rm | 28 |
| 3. cd | 29 |
| 4. cp | 29 |
| 5. mv | 30 |
| 6. ln | 31 |
| 7. mkdir | 32 |
| 8. rmdir | 32 |

| | |
|--|----|
| 9. chmod | 33 |
| 10. chown | 35 |
| 11. 파일 내용 확인 명령어 - cat, head, tail, more, less | 36 |
| 12. df | 38 |
| 13. wc | 39 |
| 14. du | 39 |
| 15. find | 39 |
| 16. touch | 40 |
| 17. 명령어 찾기 - which, whatis, whereis | 41 |

3. 프로세스 관련 명령어

| | |
|----------------|----|
| 1. ps | 42 |
| 2. kill | 43 |
| 3. top | 44 |
| 4. nohup | 45 |

4. 압축 명령어

| | |
|---------------|----|
| 1. tar | 46 |
| 2. gzip | 48 |

Chapter 3. 에디팅

1. vi 개괄

| | |
|-----------------|----|
| 1. vi의 역사 | 51 |
| 2. vi의 장점 | 51 |

2. 기본적인 vim 사용법

| | |
|------------------------|----|
| 1. vim의 모드 이해 | 52 |
| 2. 파일 열기, 저장, 종료 | 54 |
| 3. 입력 | 56 |
| 4. 이동 | 57 |
| 5. 편집 | 59 |
| 6. 되돌리기와 되살리기 | 63 |
| 7. 문자열 탐색 | 63 |
| 8. 문자열 치환 | 64 |

3. vim을 강력하게 하는 고급 테크닉

| | |
|----------------------------|----|
| 1. 여러 파일을 편집하는 방법 | 69 |
| 2. 반복되는 문자열을 저장해서 쓰기 | 72 |
| 3. 매크로의 사용 | 74 |
| 4. 다중 창 사용하기 | 76 |

| | |
|--------------------|----|
| 5. 마킹으로 이동하기 | 79 |
| 6. 셸 명령어 사용 | 82 |

Chapter 4. 리눅스 프로그래밍 환경

1. 네트워크 환경

| | |
|----------------------|----|
| 1. 네트워크 서비스 시작 | 85 |
|----------------------|----|

2. 네트워크 관련 명령어

| | |
|---------------------|----|
| 1. ping | 87 |
| 2. traceroute | 88 |
| 3. route | 88 |
| 4. netstat | 89 |

3. gcc를 사용해 컴파일 하기

| | |
|------------------------|----|
| 1. 가장 단순한 컴파일 명령 | 92 |
| 2. gcc 옵션 | 93 |

Chapter 5. 소켓 프로그래밍 기초

1. 소켓 개요

| | |
|---|-----|
| 1. 소켓의 기본 동작 방식 | 96 |
| 2. 서버/클라이언트 개념 | 99 |
| 3. 소켓으로 작성하는 서버/클라이언트 프로그램의 동작 방법 | 101 |

2. 바이트 순서

| | |
|-----------------------------|-----|
| 1. 호스트 바이트 순서 | 104 |
| 2. 네트워크 바이트 순서 | 104 |
| 3. 바이트 순서 바꾸기 프로그램 | 105 |
| 4. 호스트 바이트 순서 확인 프로그램 | 109 |

3. 도메인이름과 IP주소 상호 변환하기

| | |
|--------------------------|-----|
| 1. IP 주소 변환 | 111 |
| 2. 도메인 주소 변환 | 113 |
| 3. gethostbyname() | 113 |
| 4. gethostbyaddr() | 115 |

4. TCP 클라이언트 프로그램

| | |
|-------------------------------|-----|
| 1. TCP 클라이언트 프로그램 작성 절차 | 117 |
| 2. TCP 클라이언트 예제 프로그램 | 120 |

5. TCP 서버 프로그램

| | |
|----------------------------|-----|
| 1. TCP 서버 프로그램 작성 절차 | 128 |
| 2. TCP 에코 서버 프로그램 | 131 |
| 3. 연습문제 풀이 | 132 |

6. UDP 프로그램

| | |
|-------------------------|-----|
| 1. UDP 프로그램 작성 절차 | 138 |
| 2. UDP 에코 프로그램 | 139 |
| 3. UDP 에코 서버 | 140 |

Chapter 6. 고급 소켓 프로그래밍

1. 소켓의 동작 모드

| | |
|-----------------|-----|
| 1. 블록 모드 | 144 |
| 2. 논블록 모드 | 144 |
| 3. 비동기 모드 | 144 |

2. 다중처리 기술

| | |
|----------------|-----|
| 1. 멀티태스킹 | 145 |
| 2. 다중화 | 145 |

3. 비동기형 채팅 프로그램

| | |
|------------------------|-----|
| 1. 채팅 서버 프로그램 구조 | 147 |
| 2. select() | 148 |
| 3. 채팅 서버 프로그램 | 149 |
| 4. 통신용 소켓 구분 | 152 |
| 5. 채팅 클라이언트 프로그램 | 153 |

4. 폴링형 채팅 프로그램

| | |
|-------------------------|-----|
| 1. fcntl() | 156 |
| 2. 폴링형 채팅 서버 | 157 |
| 3. 폴링형 채팅 서버 프로그램 | 157 |
| 4. 연습문제 풀이 | 161 |

Chapter 7. 소켓 옵션

1. 소켓 옵션 종류

| | |
|-----------------------|-----|
| 1. SO_KEEPALIVE | 171 |
| 2. SO_LINGER | 171 |
| 3. shutdown() | 173 |

| | |
|-------------------------------|-----|
| 4. SO_RCVBUF와 SO_SNDBUF | 173 |
| 5. SO_REUSEADDR | 174 |
| 6. 기타 옵션 | 175 |

2. 소켓 옵션 변경

| | |
|---------------------------|-----|
| 1. 소켓 옵션 변경 함수 | 177 |
| 2. 소켓 옵션 변경 예제 프로그램 | 178 |
| 3. 연습문제 풀이 | 179 |

Chapter 8. 프로세스

1. 프로세스의 이해

| | |
|----------------------|-----|
| 1. 프로세스 | 191 |
| 2. 프로세스 식별자 | 191 |
| 3. 프로세스 식별자 확인 | 191 |
| 4. 프로세스 그룹 | 191 |

2. 프로세스의 생성과 종료

| | |
|----------------------------|-----|
| 1. 프로세스의 생성 - fork() | 192 |
| 2. 프로세스의 종료 | 192 |

3. 데몬 서버 구축 방법

| | |
|-------------------|-----|
| 1. 데몬 프로세스 | 193 |
| 2. 데몬 서버 종류 | 194 |
| 3. 연습문제 풀이 | 196 |

Chapter 9. 시그널

1. 시그널 종류

| | |
|------------------|-----|
| 1. 시그널의 종류 | 202 |
|------------------|-----|

2. 시그널 처리

| | |
|-----------------------|-----|
| 1. 시그널 처리 기본 동작 | 204 |
| 2. 시그널 핸들러 | 204 |
| 3. 시그널 처리 예 | 206 |

3. SIGCHLD와 프로세스 종료

| | |
|----------------------|-----|
| 1. 프로세스의 종료 | 208 |
| 2. 프로세스의 종료 처리 | 209 |
| 3. 연습문제 풀이 | 213 |

Chapter 9. 프로세스간 통신

1. 파이프

| | |
|------------------------------|-----|
| 1. 파이프의 정의 | 226 |
| 2. pipe() | 226 |
| 3. fork() 호출 | 226 |
| 4. 불필요한 파일 디스크립터 해제 | 227 |
| 5. 파이프를 이용한 에코 서버 프로그램 | 227 |

2. FIFO

| | |
|-------------------------------|-----|
| 1. FIFO의 정의 | 232 |
| 2. mkfifo() | 232 |
| 3. FIFO를 이용한 프로세스간 통신 | 232 |
| 4. FIFO를 이용한 에코 서버 프로그램 | 232 |

3. 메시지큐

| | |
|----------------------|-----|
| 1. 메시지큐의 정의 | 236 |
| 2. 타입이 있는 메시지큐 | 236 |
| 3. 메시지큐 생성 | 237 |
| 4. 메시지 송수신 | 238 |
| 5. 메시지큐 제어 | 239 |
| 6. 메시지큐 이용 예 | 239 |

4. 공유메모리

| | |
|-----------------------|-----|
| 1. 공유메모리의 정의 | 244 |
| 2. 공유메모리 생성 | 244 |
| 3. 공유메모리 첨부 | 245 |
| 4. 공유메모리의 분리 | 245 |
| 5. 공유메모리 제어 | 246 |
| 6. 공유메모리 정보 얻기 | 246 |
| 7. 공유메모리 삭제 | 247 |
| 8. 공유메모리의 동기화문제 | 247 |

5. 세마포어

| | |
|--------------------|-----|
| 1. 세마포어의 정의 | 250 |
| 2. 세마포어 생성 | 250 |
| 3. 플래그 사용 예 | 250 |
| 4. 세마포어 객체 | 251 |
| 5. 세마포어 연산 | 251 |
| 6. 세미포어 제어 | 252 |
| 7. 세마포어 이용 예 | 254 |

| | |
|--------------------------|-----|
| 8. 공유메모리의 동기화문제 처리 | 256 |
| 9. 연습문제 풀이 | 259 |

Chapter 10. 멀티스레드 프로그래밍

1. 스레드의 생성과 종료

| | |
|---------------------|-----|
| 1. 스레드의 정의 | 275 |
| 2. 스레드 생성과 종료 | 275 |
| 3. 스레드의 사용 예 | 276 |
| 4. 스레드의 상태 | 278 |
| 5. 스레드의 분류 | 278 |

2. 스레드 동기화

| | |
|-------------------|-----|
| 1. 동기화 문제 | 279 |
| 2. 동기화 문제 예 | 279 |
| 3. 플래그 사용 예 | 280 |
| 4. 뮤텝스 | 281 |
| 5. 뮤텝스 사용 예 | 284 |

3. 스레드간 통신

| | |
|-----------------------|-----|
| 1. 조건변수 사용 방법 | 286 |
| 2. 조건변수의 동작 | 286 |
| 3. 조건변수의 생성과 삭제 | 287 |
| 4. 조건변수 사용 예 | 287 |

4. 멀티스레드 에코 서버 프로그램

| | |
|---------------------------|-----|
| 1. 멀티스레드 에코 서버 프로그램 | 291 |
| 2. 연습문제 풀이 | 293 |

Chapter 1. 리눅스 설치

SECTION

01 리눅스 개괄

1. 리눅스란?

리눅스는 1989년 핀란드 헬싱키대학에 재학 중이던 리누스 토르발스(Linus Torvalds)가 유닉스를 기반으로 개발한 공개용 오퍼레이팅시스템(OS)으로, 1991년 11월 버전 0.02이 일반에 공개되면서 확대 보급되기 시작하였다. 유닉스(Unix)가 중대형 컴퓨터에서 주로 사용되는 것과는 달리, 리눅스는 워크스테이션이나 개인용 컴퓨터에서 주로 활용된다.

리눅스는 소스 코드를 완전 무료로 공개하여 전 세계적으로 약 5백만 명이 넘는 프로그램 개발자 그룹을 형성하게 되었으며, 이들에 의해 단일 운영체제의 독점이 아닌 다수를 위한 공개라는 원칙하에 지속적인 업그레이드가 이루어지고 있다.

파일구성이나 시스템 기능의 일부는 유닉스를 기반으로 하면서, 핵심 커널 부분은 유닉스와 다르게 작성되어 있다. 인터넷 프로토콜인 TCP/IP를 강력하게 지원하는 등 네트워킹에 특히 강점을 지니고 있으며, 유닉스와 거의 유사한 환경을 제공하면서 무료라는 장점 때문에 프로그램 개발자 및 학교 등을 중심으로 급속히 사용이 확대되고 있다.

리눅스는 각종 주변기기에 따라 혹은 사용하는 시스템의 특성에 맞게 소스를 변경할 수 있으므로 다양한 변종이 출현하고 있다.

SECTION

02 리눅스 설치

리눅스를 설치하는 방법에는 여러 가지 방법이 있다. CD 또는 DVD를 사용한 그래픽모드, 텍스트 모드의 설치가 있으며, 하드디스크 및 네트워크를 이용한 방식도 사용된다. 여기서는 CD를 이용한 그래픽 모드 설치방법에 대해 알아보도록 하겠다.

1. 설치 준비

가. BIOS 설정

시스템이 CD-ROM으로 부팅되게 하려면 우선 시스템 부팅 시 BIOS 셋업에서 부팅 순서를 "CD-ROM 우선"으로 변경해주어야 한다. 컴퓨터 전원을 켜고 초기 화면의 지시에 따라 [Delete]키 또는 [F2]키를 누른다. BIOS 셋업으로 들어가는 방법은 메인보드에 사용된 롬바이오스의 종류에 따라 다르므로, 각자 시스템에 맞는 방식으로 BIOS 셋업을 실행한다.



그림 1 AMI BIOS Setup



그림 2 AWARD BIOS Setup

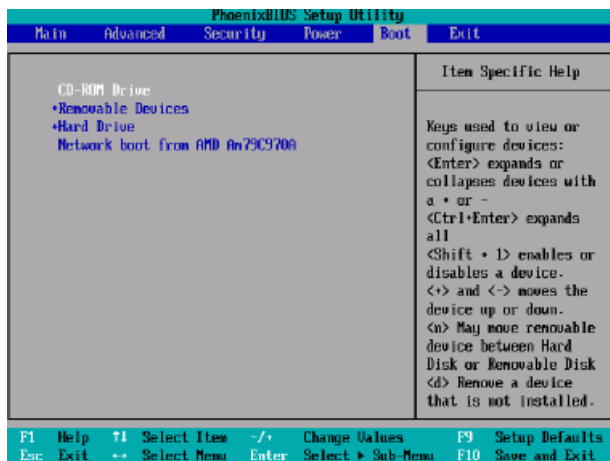


그림 3 Phoenix BIOS Setup

2. 파티션 준비

파티션이란 분할 영역이라고도 하는데, 하나의 하드디스크를 몇 개의 논리적 공간으로 분리하는 것을 말한다. 파티션 분할을 하면 하드디스크의 자료가 모두 지워지기 때문에 포맷을 하기 전에 먼저 파티션을 나누어 주어야 한다.

이렇게 나누어진 파티션에 운영체제를 설치하여 사용할 수 있으며, 하드디스크가 2개 이상의 파티션으로 나누어져 있다면, 각 파티션 별로 운영체제를 설치하여 여러 운영체제를 멀티부팅 방식으로 사용할 수도 있다.

현재의 시스템에 리눅스만을 설치하여 사용하려는 경우 또는 리눅스 설치를 위한 여분의 파티션이 있는 상태라면, 바로 다음 단계인 "설치 단계"로 넘어가면 된다. 그러나 현재의 시스템에 하나의 파티션만이 존재하고 여기에 이미 MS Windows와 같은 다른 운영체제가 설치되어 있다면, 리눅스 설치 시에 리눅스를 설치할 파티션이 없기에, 기존 파티션에 설치를 하기 된다.

이 경우 기존의 운영체제는 삭제되기 때문에 기존 데이터를 보존해야 할 필요가 있거나, 2개 이상의 운영체제를 멀티부팅으로 사용해야 한다면, 리눅스 설치 이전에 별도의 파티션 프로그램(파티션매직 등)을 이용하여 파티션을 분리해 두어야 한다. 리눅스를 설치하기 위해서는 별도의 여유 파티션 공간이 필요하다는 것을 기억하기 바란다.

3. CD-ROM 부팅

이제 CD-ROM 드라이브에 리눅스 설치 CD를 넣고 부팅을 한다. 그러면 아래와 같은 화면이 나타나며, [Enter]키를 누르면 그래픽 환경의 리눅스 설치 프로그램인 "아나콘다"가 구동된다.



그림 4 CD 부팅 화면

4. 설치 모드 선택

리눅스를 설치하는 방식 중 가장 많이 사용되는 방식이 그래픽 모드의 설치방식이다. 그래픽 모드 방식은 앞서 언급한대로 위의 화면에서 [Enter]키를 입력하여 진행할 수 있다.

그러나 그래픽 카드 및 마우스와 같은 GUI도구가 인식되지 않는 경우 그래픽 모드로의 설치가 수행되지 않을 수 있다. 이 경우에는 텍스트 모드의 설치방식을 통해 리눅스를 설치해야 한다. 텍스트 모드로 설치하기 위해서는 위의 화면에서 "linux text"라고 입력하고 [Enter]키를 누른다. 이외에도 HDD, HTTP, FTP, NFS 등을 이용한 설치방법이 있으나, 여기서는 가장 일반적인 방법인 "그래픽 모드"의 설치에 대해 알아본다.

5. Startup

그래픽 모드로 설치를 진행하게 되면, 잠시 후 아래 그림과 같이 로디화면과 설치 첫 페이지인 "Startup" 페이지를 만날 수 있다.



그림 5 리눅스 설치 프로그램 로딩 화면



그림 6 설치 첫 단계(Startup) 화면

6. 설치 언어 선택

앞 화면에서 Next 버튼을 눌러 진행하면 아래 그림과 같이 설치 시 사용할 언어를 선택하는 화면이 나타난다. "Korean(한국어)"를 선택하고 Next 버튼을 누르면 이후에 보이는 모든 페이지들은 한글 환경으로 나오게 된다.

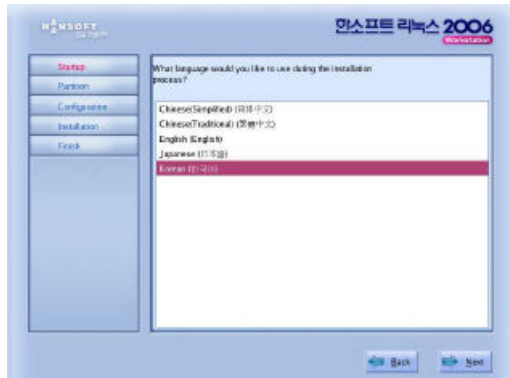


그림 7 언어 선택 화면

7. 사용자 동의

사용자 동의서의 내용을 확인한 다음 해당 사항에 동의하는 경우에만 설치를 계속 진행할 수 있다.

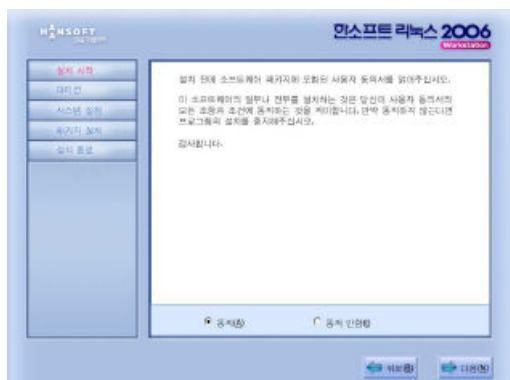


그림 8 사용자 동의 확인 화면

8. 키보드 선택

현재 시스템의 키보드를 설정한다. 대부분 자동으로 키보드를 인식하므로 기본 선택된 값으로 설치를 계속 진행하면 된다. (예: U.S. 영어)

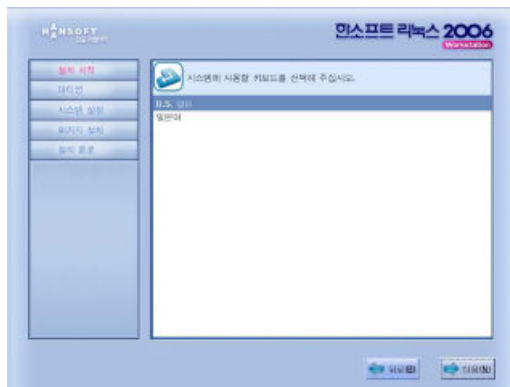


그림 9 키보드 선택 화면

9. 파티션 설정

리눅스 설치 중 가장 중요한 파티션 설정 부분이다. 다른 설치 사항들은 대부분 윈도우즈 설치와 비슷하기에 큰 어려움 없이 진행할 수 있지만, 파티션 설정 부분은 윈도우즈와 조금 달라서 처음 리눅스를 접하는 이들이 가장 어려워하는 부분이기도 하다.

아래 그림을 보면 [자동 파티션 분할]과 [Disk Druid를 통한 수동 파티션 설정]의 2가지 항목이 존재한다.

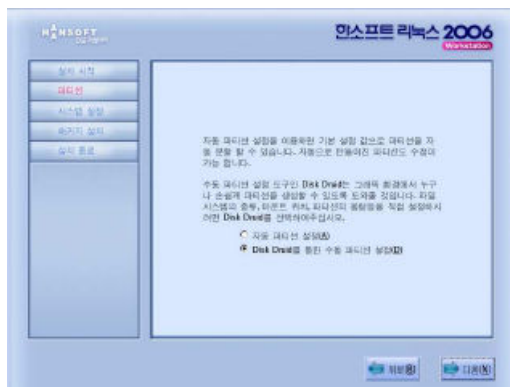


그림 10 파티션 설정 방법 선택 화면

[자동 파티션 분할]은 현재 시스템에 있는 하드디스크의 파티션 분할 상태를 무시하고, 리눅스에서 제공하는 가장 최적화된 구조화 파티션을 재조정하는 것을 말한다. 즉, 최적화된 환경으로 리눅스만을 사용하는 경우라면 이 옵션으로 쉽게 설치할 수 있는 장점이 있으나, 기존에 사용하던 파티션은 모두 초기화되므로, 윈도우즈가 이미 설치되어 있어 같이 사용하기를 원하거나 기존 데이터의 보존이 필요한 경우에는 이 옵션을 사용하지 않도록 해야 한다.

반면 [Disk Druid를 통한 수동 파티션 설정]은 사용자가 직접 파티션을 조정할 수 있으며, 기존의 데이터를 보존하기 위해서는 이 옵션으로 설치를 진행해야 한다. 이후 설명되는 내용은 [Disk Druid를 통한 수동 파티션 설정]을 기준으로 설명한다.

가. 현재 파티션 현황

시스템에 장착되어 있는 하드디스크가 파티션 설정이 되어있지 않은 빈 하드디스크라면 아래와 같이 하드디스크 드라이브를 초기화 한다는 메시지가 나타난다. [예])를 선택하고 계속 진행한다.

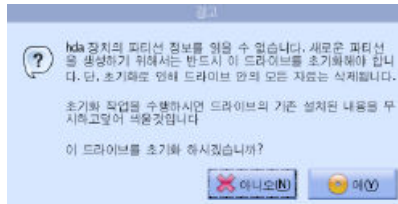


그림 11 하드디스크 초기화

다음으로 파티션을 설정하는 메인화면이 나타난다.



그림 12 파티션 설정 메인화면

현재 하드디스크의 파티션 현황과 용량을 한 눈에 볼 수 있도록 도표화되어 있다. 리눅스에서의 파티션 명칭은 윈도우즈화면 조금 다르다. 윈도우즈에서는 앞 쪽에 있는 파티션부터 순서대로 C, D, E... 등의 명칭을 사용하지만, 리눅스에서는 hda, hdb 등과 같이 하드웨어적인 세부사항을 파티션 명칭으로 나타내므로 더 명확한 파티션 구분이 가능하다.

위의 그림은 10GB의 IDC 하드디스크(hda)를 인식한 모습이며, 아직 아무런 파티션이 설정되어 있지 않은 초기 상태이다. 윈도우즈에서는 하나의 파티션에 OS가 설치되지만, 리눅스는 하나의 OS를 여러 개의 파티션으로 분할하여 설치할 수 있다. 가장 일반적인 리눅스 설치 방법은 아래와 같이 2개의 파티션으로 구성하는 방법이다.

- 루트(/) 파티션
- 스왑(swap) 파티션

루트(/) 파티션은 윈도우즈에서와 같이 운영체제(OS)가 설치되는 최상위 파티션으로서, 그 하단에 각각의 디렉토리 및 파일들이 생성된다. 윈도우즈에서는 Documents and Settings, Program Files, Windows 등의 디렉토리들이 생성되며, 리눅스에서는 /boot, /usr, /root, /home 등의 디렉토리들이

생성된다.

스왑(swap) 파티션은 윈도우즈의 가상메모리와 같은 역할을 하는 파티션으로, 윈도우즈에서는 하드디스크의 여유 공간을 가상메모리 공간으로 할당하여 사용하지만, 리눅스에서는 별도의 파티션으로 나누어서 사용하게 된다.

리눅스에서는 루트(/) 아래에 있는 각각의 폴더들을 별도의 파티션으로 설정하여 설치가 가능하다. 서버시스템에는 보안성 향상과 관리의 편의를 위해 주요 디렉토리를 별도의 파티션으로 구현하기도 한다. 그리고 사상메모리(swap)를 별도의 고정된 파티션으로 사용함으로써 윈도우즈 보다 안정적이고 효과적인 메모리 관리가 가능하다.

나. 파티션 추가 / 편집 / 삭제

앞서 말한바와 같이 루트(/) 파티션과 스왑(swap) 파티션을 생성해 보도록 하겠다. 화면 하단에 있는 버튼들 중에서 [새로 생성] 버튼을 클릭하면 아래 그림과 같이 파티션을 추가할 수 있는 창이 나타난다.

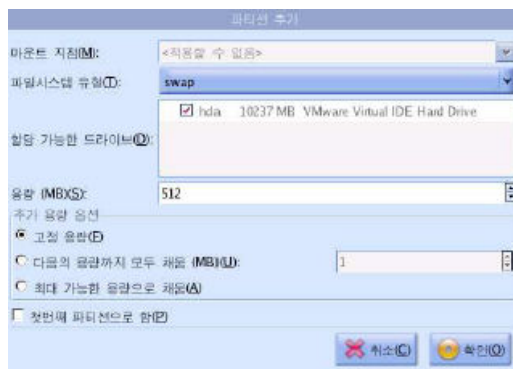


그림 13 스왑(swap) 파티션 추가 화면

먼저 스왑(swap) 파티션 생성을 위해 [파일시스템 유형]에서 "swap"을 선택한다. [용량] 부분에는 현재 시스템 메모리의 2배 정도를 입력한다. (예: 현재 시스템의 메모리가 256MB이면, "512"라고 입력한다.) 그런 다음 확인을 누르면 화면에 스왑(swap) 파티션이 생성된 것을 확인할 수 있다.

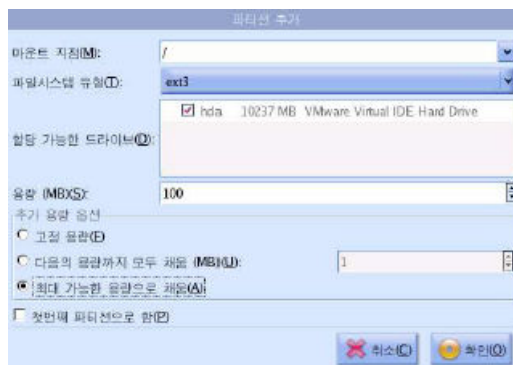


그림 14 루트(/) 파티션 추가 화면

루트(/) 파티션 생성을 위해 다시 한 번 [새로 생성] 버튼을 클릭한다. [마운트 지점] 부분을 "/"로 선택하고 [파일시스템 유형]에서 "ext3"를 선택한다. 그 다음 [용량] 부분에서 사용될 적절한 용량을 지정하는데, 남은 공간 모두를 루트(/)로 지정하여 사용할 경우에는 하단에 있는 [최대 가능한 용량으로 채움]에 체크를 하면 자동으로 최대용량을 지정해 준다. [확인] 버튼을 눌러 현재 상태를 확인한다.



그림 15 생성된 파티션 확인 화면

위 그림과 같이 루트(/) 파티션과 스왑(swap) 파티션이 모두 만들어지면, [다음] 버튼을 눌러 다음 단계로 계속 진행한다.

파티션 작업 도중 실수를 하여 파티션을 잘못 생성하였다면 [편집] 및 [삭제] 버튼을 이용하여 적정히 수정한다. 파티션 설정 부분에서 파티션을 구성하더라도 실제 파티션 수정 작업은 추후 본격적인 설치 과정에 가서야 진행되므로 현 단계에서는 마음껏 수정하여 작업할 수 있다.

10. 부트로더 설정

하나의 시스템에 여러 개의 운영체제를 설치하여 사용 경우, 부팅시 원하는 운영체제를 선택할 수 있게 해주는 도구가 부트로더이다. 예를 들면 현재 시스템에 윈도우즈가 같이 설치되어 있다면 이 단계에서 윈도우즈와 리눅스 둘 중 하나를 선택해서 먼저 부팅시킬 수 있다.

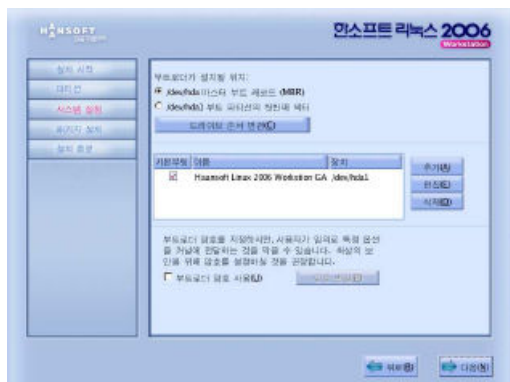


그림 16 부트로더 설정 화면

위의 그림은 현재 시스템에 다른 운영체제가 설치되어 있지 않으므로 리눅스의 항목만이 나타난 경

18 네트워크 프로그래밍 및 실습

우다. 우측의 [편집] 버튼을 눌러 각 항목의 이름을 변경하거나, 기본적으로 부팅되도록 선택되어질 항목을 수정할 수 있다.

11. 네트워크 설정

네트워크를 설정하는 단계다. 일반적으로 DHCP(자동 IP할당) 방식의 네트워크를 많이 사용하므로 기본 설정값 그대로 [다음] 버튼을 눌러 진행한다. 만약 인터넷 서비스 회사로부터 별도의 고정IP를 할당받아 사용하는 경우라면, 상단의 [편집] 버튼을 눌러 IP 관련 설정을 진행한다.

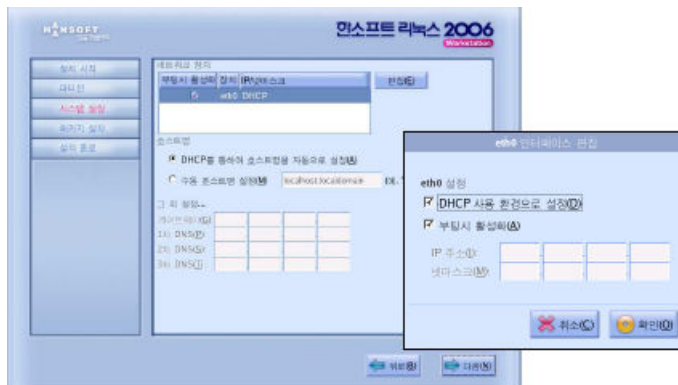


그림 17 네트워크 설정 화면

12. 지역 시간대 설정

지역 시간대를 설정하는 화면이다. 자신이 속한 시간대에 가장 근접한 도시를 지도에서 찾아 클릭하거나 하단의 리스트에서 찾아 설정할 수 있다. 한글 환경으로 설치를 진행하고 있다면 자동으로 대한민국의 시간대인 "아시아/서울(Seoul)"이 설정되어 있다.



그림 18 지역 시간대 설정 화면

13. 관리자 암호 설정

관리자(root 계정) 암호를 설정한다. root는 리눅스 시스템을 관리하는 최고 권한을 가진 계정이다.

그러므로 암호 설정에 신중을 기하고 외부에 알려지지 않도록 주의해야 한다. 일반 사용자 계정은 설치가 완료된 이후에 관리자로 로그인하여 생성해 줄 수 있다.

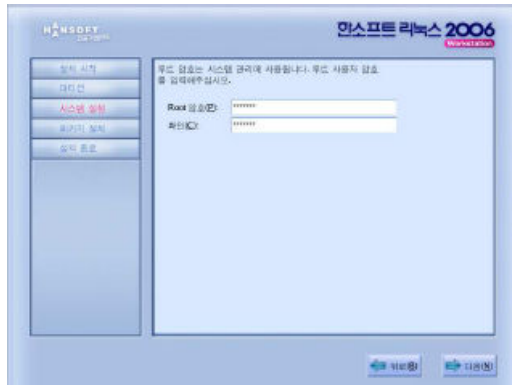


그림 19 관리자 암호 설정 화면

14. 설치 패키지 선택

패키지를 설치하는 범위를 지정한다. 크게 “최소설치”와 “전부설치”로 나누어지며 사용자정의 형식으로 패키지를 하나하나 선택하여 설치하고 싶은 경우에는 마지막에 있는 “설치할 패키지 리스트를 직접 선택합니다.”에 체크를 하고 다음으로 넘어간다.

일반적으로는 리눅스 관련 패키지들을 모두 설치하여 안정적으로 리눅스를 운영할 수 있는 “전부설치” 방식을 권장한다.



그림 20 전부설치를 선택한 화면

사용자정의 형태로 진행하면 아래와 같이 패키지들을 직접 선택하여 설치할 수 있는 화면이 나타난다. 그룹으로 나누어 선택할 수 있으며, “자세한 정보”를 클릭하면 해당 그룹에 포함되어 있는 패키지들도 상세하게 선택할 수 있다.

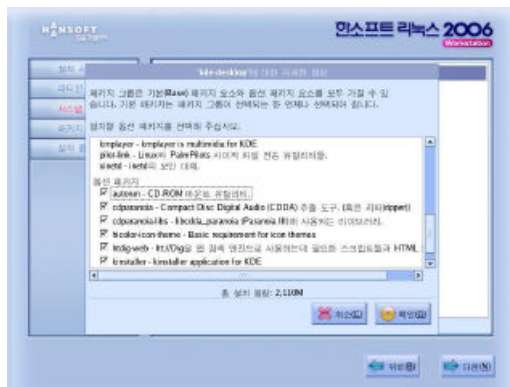


그림 21 패키지 정보 화면

15. 설치 시작

지금까지 설정한 내용을 바탕으로 본격적인 설치를 진행한다.



그림 22 설치 초기 화면

설치 초기에는 아래 그림들과 같이 파일시스템 포맷, 설치 이미지 전송, RPM 트랜잭션 준비 등을 진행한 후 설치과정이 이어진다.



그림 23 설치 진행 화면

설치 중간 CD를 교체하라는 메시지가 나오면 CD를 바꾼 후 계속 설치를 진행한다.

16. 로그인 유형 선택

설치가 완료되면 시스템에 “부트로더”를 적재하고 아래 그림과 같이 로그인 유형을 선택하는 화면이 나타난다. 기본 로그인 유형은 텍스트 환경이나, 그래픽 환경의 로그인을 원하는 경우 [그래픽 환경]을 체크하고 적절한 화면해상도와 색상 수를 지정한다.



그림 24 로그인 환경 지정 화면

17. 재부팅

설치가 완료되면 재부팅을 하여 바로 리눅스를 사용할 수 있다.

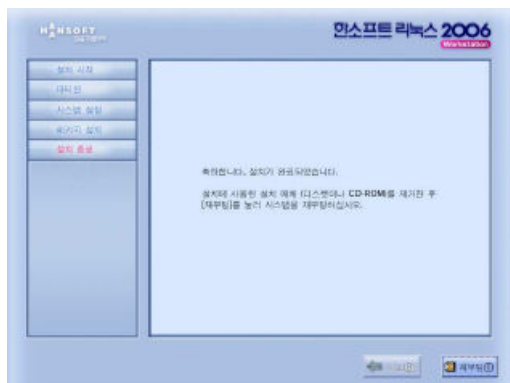


그림 25 설치 완료 화면



그림 26 부트로더 화면

시스템이 재부팅되면 위의 그림과 같이 부트로더 화면이 나타나고, 곧이어 리눅스 부팅과정이 진행된다.

로그인 유형을 그래픽 모드로 하였다면 아래와 같이 로그인 화면이 나타난다. (텍스트 모드인 경우 콘솔 상태에서의 로그인 화면이 나타난다.)



그림 27 로그인 화면

설치 시에 입력했던 root 관리자 암호를 이용하여 로그인을 진행하면 아래 그림과 같이 바탕화면이 나타난다. (그래픽 모드의 경우)

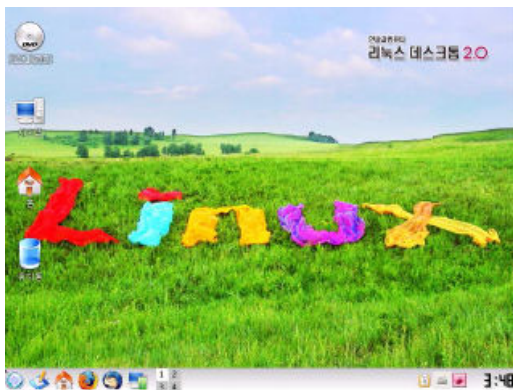


그림 28 리눅스 데스크톱 기본 화면

Chapter 2. 기본 명령어

SECTION

01 명령어 매뉴얼과 도움말 얻기

리눅스에는 외우기 힘들 정도로 많은 명령어가 있다. 이들 모두를 알고 있으면 좋겠지만 그것은 사실상 불가능 하다. 사실 어떤 명령어가 있다는 것과 어떤 상황에서 무엇을 확인하고자 할 때는 어떤 명령어를 사용해야 한다는 것 정도만 알아도 충분하다.

1. man

영한사전에서는 어떤 단어를 비슷한 단어와 함께 소개하여 쉽게 외울 수 있도록 한다. 또한 단어 하나에 많게는 수십 가지 뜻이 있기도 한다. 우리가 일반적으로 사용하는 간단한 명령어 역시 다른 명령어와 연계하여 쉽게 외울 수 있는 경우가 많으며 수십 가지의 기능으로 확장하기 위해 옵션을 추가할 수 있다.

이런 모든 기능을 잘 사용하기 위해서는 영한사전과 같은 존재, 다시 말해 도움말 기능이 있어야 할 것이다. 물론 MS 윈도우에 도움말 기능이 있듯이 리눅스에도 맨 페이지라는 도움말 기능이 있다. 어떤 명령어의 설명을 보고 싶다면 다음과 같이 입력한다.

```
# man [명령어]
```

그러면 cron이라는 명령어에 대해 설명을 보는 경우를 살펴보자.

```
# man cron
```

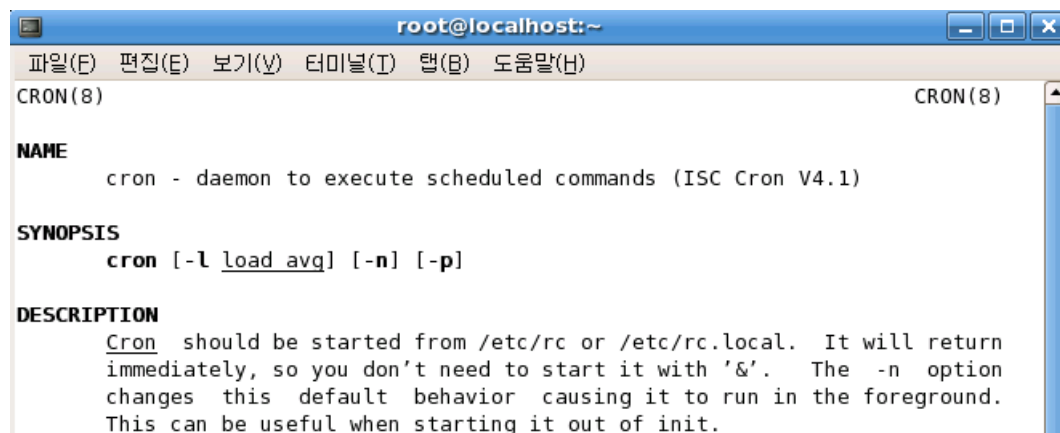


그림 29 man 명령어

이런 맨 페이지는 /usr/share/man 디렉토리에 섹션별로 저장되어 있으며, 이 온라인 도움말 시스템은 각 명령과 용어에 대해 다음과 같은 내용을 포함한다.

- 명령어 이름(name)
- 개요(synopsis)와 설명
- 환경(environment)와 매개변수(parameter) 등

또한 각 섹션은 다음과 같이 분류한다. 물론 같은 단어로 된 명령어가 여러 섹션에 걸쳐 있을 수 있는데, 이 경우는 특별히 옵션을 주지 않으면 섹션의 숫자가 적은 쪽을 출력한다. crontab은 1번과 5번 섹션에서 소개된다. 명령어에 대한 맨 페이지를 보고 싶다면 `man 1 crontab`, 파일의 포맷에 대해 알고 싶다면 `man 5 crontab`이라 입력하면 된다. 그냥 `man crontab`이라고 입력하면 1번 섹션의 내용을 볼 수 있다.

| 섹션 | 설명 |
|----|---|
| 1 | User Command ls, grep, finder와 같은 일반 명령 |
| 2 | System calls 리눅스 프로그래밍을 위한 시스템 호출과 관련된 내용 |
| 3 | C library functions 리눅스 라이브러리 함수와 관련된 내용 |
| 4 | File formats(syntax) 특수 파일(FIFO, 소켓 등)에 대한 문서 |
| 5 | File formats(syntax) 중요한 설정 파일에 대한 정보 |
| 6 | Game descriptions 시스템 테스트 프로그램에 대한 설명 |
| 7 | Cover text, text format, etc. 표준과 규칙에 대한 정보, 프로토콜, 문자세트, 시그널 목록에 대한 정보 |
| 8 | System administration mount, fsck 등 시스템 관리자가 사용하는 명령에 대한 설명 |
| 9 | Linux kernel routines 커널 프로그래밍을 위한 정보 |
| n | "New" or commands that didn't fit elsewhere 새로운 명령에 대비한 공간. 조로 Tcl/Tk 프로그래밍에 대한 내용 |

표 1 man 섹션별 설명

2. apropos

물론 man을 이용하여 명령어를 사용하는 방법을 확인할 수 있겠지만, 명령어 이름을 정확히 알지 못할 경우에는 다음과 같이 키워드를 이용하여 해당 키워드가 포함된 맨 페이지를 찾아낼 수 있다. 이 명령은 `whatis` DB라고 하는 맨 페이지의 내용의 데이터베이스를 검색하여 빠르게 해당 명령을 찾아준다.

```
# apropos [키워드]
```

```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# apropos login
ScrollKeeper [scrollkeeper] (7) - An open document cataloging and metadata management system
audit_getloginuid      (3) - Get a program's loginuid value
audit_setloginuid      (3) - Set a program's loginuid value
callback               (8) - call a user back, presenting a login prompt
chsh                   (1) - change your login shell
cuserid [getlogin]     (3) - get user name
faillog                (5) - Login failure logging file
faillog                (8) - examine faillog and set login failure limits
getlogin               (3) - get user name
getlogin_r [getlogin] (3) - get user name

```

그림 30 apropos 명령어

3. info

info는 최신 버전의 GNU 맨 페이지 유틸리티이다. 이것은 맨 페이지의 발전된 형태로 기획되었지만 아직 맨 페이지 만큼 널리 사용되지는 못하고 있다. 최신의 정보가 많으며, 하이퍼텍스트 링크와 같은 강력한 기능을 사용할 수도 있다.

```
# info [명령어]
```

```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
file: libc.info, Node: Logging In and Out, Prev: XPG Functions, Up: User Accounting Database

29.12.3 Logging In and Out
-----

These functions, derived from BSD, are available in the separate
`libutil' library, and declared in `utmp.h'.

Note that the `ut_user' member of `struct utmp' is called `ut_name'
in BSD. Therefore, `ut_name' is defined as an alias for `ut_user' in
`utmp.h'.

--zz-Info: (libc.info.gz)Logging In and Out, 50 lines --Top-- Subfile: libc.info
Welcome to Info version 4.7. Type ? for help, m for menu item.

```

그림 31 info 명령어

SECTION

02 파일 관련 명령어

어떤 환경에서라도 컴퓨터를 사용하는 작업의 기본은 파일을 읽고 쓰는 것에서 출발한다. 가장 기본적인 작업을 할 수 있는 파일 관련 명령어에 대해 알아보자.

1. ls

ls 명령은 디렉토리 내 파일의 목록을 보기 위해 사용한다.

ls [-옵션]

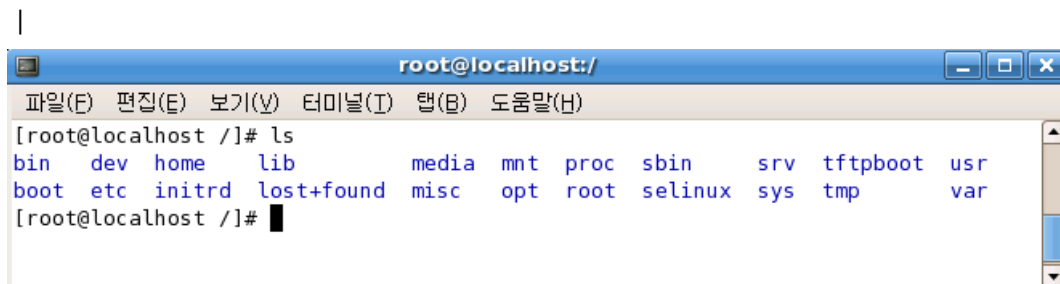


그림 32 ls 명령어

| | |
|------------|-------------------------------------|
| ls -a | 모든 파일 보기 |
| ls -l | 자세히 보기(파일의 유형, 허가권, 링크카운트, 파일소유권 등) |
| ls -c | 시간 순서대로 정렬해서 보기 |
| ls -d | 디렉토리만 보기 |
| ls -f | 디스크에 저장된 순서로 보기(정렬 안하기) |
| ls -i | 고유번호(inode)와 함께 보기 |
| ls -k | kb 단위로 보기 |
| ls -m | 파일이름을 가로로 나열하기 |
| ls -r | 내림차순 정렬로 보기 |
| ls -t | 시간 순서대로 정렬해서 보기 |
| ls -R | 하위 디렉토리 안의 파일까지 보여주기 |
| ls --color | 파일의 형태에 따라 색깔을 다르게 보여주기 |
| ls -F | 파일과 디렉토리를 심볼로 구분하여 보여주기 |

표 2 ls 기본 옵션

점(.)으로 시작하는 파일과 디렉토리는 ls 명령만으로는 볼 수 없으며 -a 옵션을 함께 사용해야 한다. 이와 같은 파일을 숨김 파일이라 한다.

```

root@localhost:/
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost /]# ls -al
합계 193
drwxr-xr-x 26 root root 4096 2월 14 19:25 .
drwxr-xr-x 26 root root 4096 2월 14 19:25 ..
-rw-r--r-- 1 root root 0 2월 14 19:25 .autofsck
drwxr-xr-x 2 root root 4096 3월 15 2006 .automount
-rw-r--r-- 1 root root 0 1월 27 20:52 .autorelabel
drwxr-xr-x 3 root root 4096 1월 27 20:10 .texmf-var
drwxr-xr-x 2 root root 4096 1월 28 04:25 bin
drwxr-xr-x 4 root root 1024 1월 27 20:04 boot
drwxr-xr-x 11 root root 4080 2월 14 19:26 dev
drwxr-xr-x 94 root root 12288 2월 14 20:32 etc
drwxr-xr-x 2 root root 4096 3월 17 2006 home
drwxr-xr-x 2 root root 4096 3월 17 2006 initrd
drwxr-xr-x 11 root root 4096 1월 28 04:17 lib

```

그림 33 ls -al

숨김 파일은 주로 다른 사용자들이 함부로 수정해선 안 되거나 일반적으로는 변경할 일이 거의 없는 환경 파일로 많이 사용된다. 가령, 넷스케이프 웹 브라우저를 처음 실행하면 사용자의 홈 디렉토리에 .netscape라는 디렉토리가 생성되고 사용자의 브라우저 설정과 관련된 파일들이 복사된다. 또한 처음 사용자를 생성할 때 복사되는 설정 파일 역시 파일 이름 앞에 점(.)이 붙는다.

2. rm

불필요한 파일을 삭제하기 위해 rm이라는 명령어를 사용한다. DOS의 del명령어를 사용하는 것과 같은 것으로 remove의 약어이다.

```
# rm [파일이름]
```

다음은 rm 명령과 함께 사용될 수 있는 옵션이다. 이중 rm -i는 사용자가 리눅스를 안전하게 사용할 수 있도록 앨리어스(alias)로 설정되어 있는 경우가 대부분이다.

| | |
|-------|------------------------------|
| rm -r | 디렉토리까지 지워버리는 옵션 |
| rm -f | 삭제하기 전에 확인과정 없이 강제로 지워버리는 옵션 |
| rm -i | 사용자에게 정말로 지울 것인지 확인 |
| rm -v | 파일 지우는 정보를 자세히 보여줌 |

표 3 ls 기본 옵션

```

root@localhost:/
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost /]# rm hello.c
rm: remove 일반 빈 파일 `hello.c'? y
[root@localhost /]#

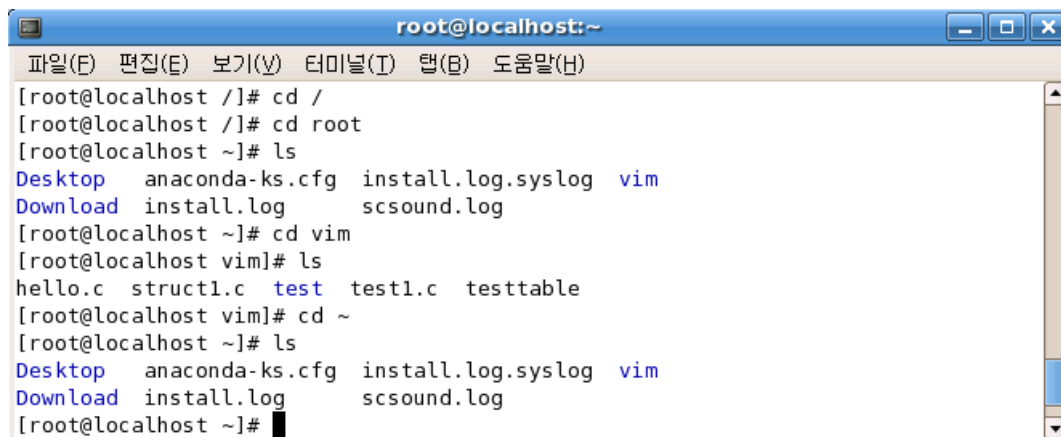
```

그림 34 rm 명령어

-rf 옵션을 사용할 때에는 디렉토리와 함께 그 안에 들어있는 파일 및 서브디렉토리까지 한마디 말도 없이 모두 삭제한다. 편리한 명령이기는 하지만 주의해야한다.

3. cd

이 명령은 디렉토리 이동 명령이다.



```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost /]# cd /
[root@localhost /]# cd root
[root@localhost ~]# ls
Desktop  anaconda-ks.cfg  install.log.syslog  vim
Download install.log      scsound.log
[root@localhost ~]# cd vim
[root@localhost vim]# ls
hello.c  struct1.c  test  test1.c  testtable
[root@localhost vim]# cd ~
[root@localhost ~]# ls
Desktop  anaconda-ks.cfg  install.log.syslog  vim
Download install.log      scsound.log
[root@localhost ~]#
  
```

그림 35 cd 명령어

디렉토리를 이동할 때 경로 맨 앞에 슬래시(/)가 있는 경우가 있고, 없는 경우도 있다. 슬래시(/)가 있는 경로를 절대 경로라 하며, 없으면 상대 경로라 한다.

다음은 cd 명령과 함께 사용될 수 있는 옵션에 대한 설명이다.

| | |
|-------|-------------------------|
| cd | 사용자의 홈 디렉토리로 이동 |
| cd ~/ | 사용자의 홈 디렉토리의 서브디렉토리로 이동 |
| cd - | 방금 전에 있던 디렉토리로 이동 |

표 4 cd 기본 옵션

| | |
|-------------------------------|------------------------------|
| [root@localhost /]# cd aaa | 현재 디렉토리에 존재하는 aaa라는 디렉토리로 이동 |
| [root@localhost /]# cd ./aaa | 현재 디렉토리에 존재하는 aaa라는 디렉토리로 이동 |
| [root@localhost /]# cd ../bbb | 상위 디렉토리에 존재하는 bbb라는 디렉토리로 이동 |
| [root@localhost /]# cd ~ | 홈 디렉토리로 이동 |

4. cp

cp는 파일을 복사하는 명령어로, DOS의 copy 명령어와 같다. 복사할 파일의 경로와 복사할 위치 혹은 새 이름을 입력하면 복사된다. 이 경우 두 파일은 별개의 파일이 되어 다른 inode를 사용하게 된다.


```
# cp [복사할 이름] [복사되어 생성될 이름]
```

```
root@localhost: ~/vim/file
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost file]# ls
hello.c
[root@localhost file]# cp hello.c test.c
[root@localhost file]# ls
hello.c test.c
[root@localhost file]#
```

그림 36 cp 명령어

다음은 cp 명령과 함께 사용될 수 있는 옵션이다.

| | |
|-------|-----------------------------------|
| cp -a | 파일의 속성, 링크 정보 등을 유지하면서 복사 |
| cp -b | 이미 파일이 존재할 경우 백업 파일 생성 |
| cp -f | 기존의 파일을 강제로 지우고 복사 |
| cp -i | 복사할 때 물어봄 |
| cp -l | 하드링크 형식으로 복사 |
| cp -P | 원본파일에 경로와 함께 지정했을 경우 그 경로를 그대로 복사 |
| cp -p | 원본파일의 소유주, 그룹, 권한, 시간정보 복사 |
| cp -r | 경로와 함께 경로 안의 파일들을 모두 복사 |

표 5 cp 기본 옵션

5. mv

mv 명령은 파일명을 변경하거나 현재 위치한 디렉토리에서 다른 디렉토리로 파일을 이동시킬 때 사용되는 명령으로, move에서 자음만 따온 것이다.

```
# mv [옮길 파일명] [이동할 디렉토리나 파일명]
```

```
root@localhost: ~/vim/file
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost file]# ls
hello.c
[root@localhost file]# mv hello.c test.c
[root@localhost file]# ls
test.c
[root@localhost file]#
```

그림 37 mv 명령어

다음은 mv 명령과 함께 사용될 수 있는 옵션에 대한 설명이다.

| | |
|-------|--------------|
| mv -b | 백업파일을 만든다 |
| mv -f | 강제로 이동 |
| mv -i | 사용자에게 확인시킨다 |
| mv -u | 업데이트 한다 |
| mv -v | 과정을 자세히 보여준다 |

표 6 mv 기본 옵션

6. ln

ln은 파일 사이에 링크를 만드는 명령이다. 이런 링크는 원본 파일을 보호하며 환경 변수를 변경하지 않고도 새로 설치한 프로그램을 쉽게 사용할 수 있는 등 다양한 방법으로 유용하게 사용할 수 있다.

```
# ln [옵션] [링크시킬 파일이나 폴더] [링크로 만들어질 파일]
```

다음은 ln 명령과 함께 사용될 수 있는 옵션에 대한 설명이다.

| | |
|-------|-----------------------|
| ln -s | 심볼릭 링크 |
| ln -n | 파일이 존재하면 겹쳐쓰기를 하지 않는다 |
| ln -f | 파일이 존재해도 겹쳐쓴다 |

표 7 mv 기본 옵션

가. 하드 링크(Hard links)

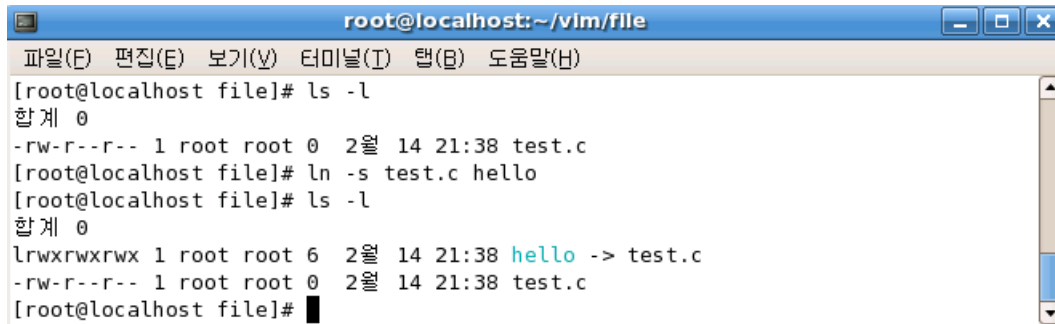
하드 링크는 하나의 파일을 여러 곳으로 연결하는데 사용한다. 예를 들어 test.c와 hello.c는 항상 같은 내용을 지닌다. 다른 이름을 갖는 두 개의 파일이지만 하나의 inode로 생성되므로 결국은 같은 파일이다. 주의할 점은 같은 inode를 사용하기 때문에 서로 다른 파일시스템 사이에는 링크를 만들 수 없다는 점이다.

```
root@localhost: ~/vim/file
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost file]# ls
test.c
[root@localhost file]# ln test.c hello.c
[root@localhost file]# ls
hello.c test.c
[root@localhost file]#
```

그림 38 ln 명령어

나. 심볼릭 링크(Symbolic links)

겉보기에는 보통 파일로 보이지만 사실 원본 파일이 어디를 가리키고 있는지 알려주는 역할을 할 뿐인 파일을 심볼릭 링크라고 말한다. 심볼릭 링크의 경우 링크를 다른 곳으로 이동시키면 링크가 깨져서 사용할 수 없다.



```

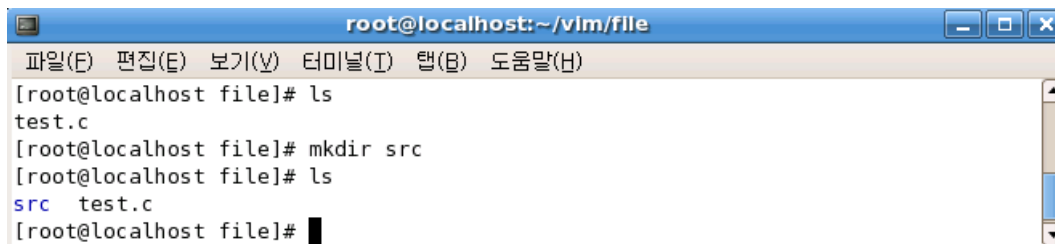
root@localhost:~/vim/file
파일(F) 편집(E) 보기(V) 터미널(T) 랩(B) 도움말(H)
[root@localhost file]# ls -l
합계 0
-rw-r--r-- 1 root root 0  2월 14 21:38 test.c
[root@localhost file]# ln -s test.c hello
[root@localhost file]# ls -l
합계 0
lrwxrwxrwx 1 root root 6  2월 14 21:38 hello -> test.c
-rw-r--r-- 1 root root 0  2월 14 21:38 test.c
[root@localhost file]#

```

그림 39 링크의 확인

7. mkdir

디렉토리를 만들 때 mkdir을 사용한다.



```

root@localhost:~/vim/file
파일(F) 편집(E) 보기(V) 터미널(T) 랩(B) 도움말(H)
[root@localhost file]# ls
test.c
[root@localhost file]# mkdir src
[root@localhost file]# ls
src  test.c
[root@localhost file]#

```

그림 40 mkdir 명령어

```
# mkdir [디렉토리명]
# mkdir -p aaa/bbb
```

-p 옵션은 여러 단계의 하부 디렉토리까지 만들어 준다.

8. rmdir

rmdir은 디렉토리를 삭제하는 명령이다. rmdir로 지울 때는 반드시 해당 디렉토리가 비어 있어야 한다. 파일이나 다른 디렉토리가 있는 디렉토리를 지울 때는 에러메시지가 나타난다. 일반적으로 rmdir은 다음과 같은 형식으로 사용한다.

```
# rmdir [디렉토리명]
```

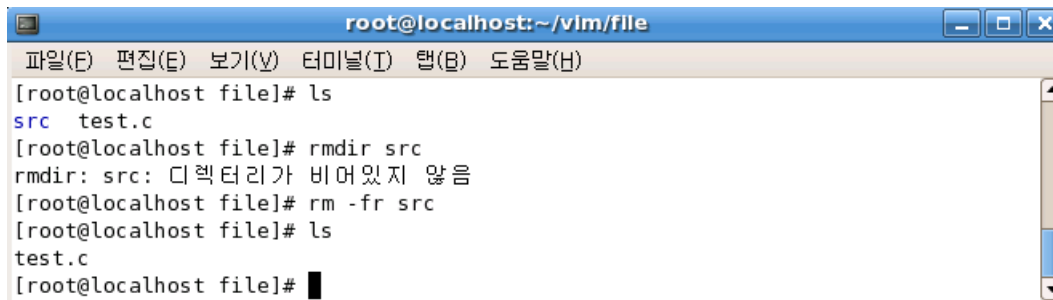


그림 41 rmdir 명령어

mkdir과 마찬가지로 디렉토리 내부의 디렉토리까지 한번에 삭제하기 위해 -p 옵션을 사용할 수 있지만, 디렉토리는 그대로 있고 내부에 파일이 없는 상태에서만 사용할 수 있기 때문에 많이 쓰이는 것은 아니다.

9. chmod

chmod는 파일의 사용 권한을 조정하는 명령어이다.

```
# chmod [option] [파일]
```

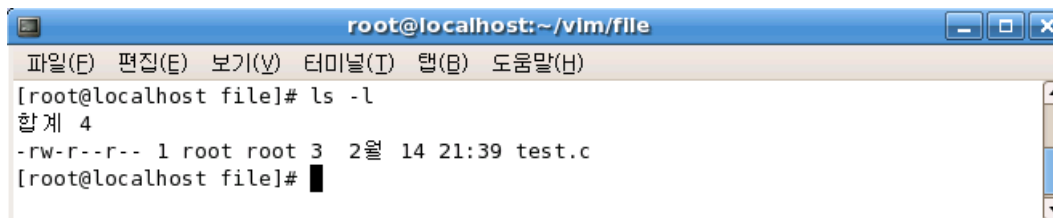


그림 42 파일 권한 확인

예문에서 보면 -rw-r--r-- 이라는 부분이 있다. 이 부분은 이 파일(test.c)의 권한을 설정한 것으로, 총 10칸으로 되어 있다.

| 구간 | 의미 |
|-----------------------|--------------------------------------|
| 앞 1칸 | 디렉토리(d), 링크파일(l), 스티키 비트(s) 등 파일의 특성 |
| 그 다음 3칸(owver = 주인) | 파일 소유자의 읽고 쓰고 실행하기(rwx) |
| 그 다음 3칸(group = 그룹) | 파일 소유 그룹의 읽고 쓰고 실행하기(rwx) |
| 그 다음 3칸(others = 다른이) | 그 외 사람들의 읽고 쓰고 실행하기(rwx) |

표 8 파일 권한 보기

그림에서 나타난 대시(-)가 의미하는 것은 허가권이 없음을 나타내는 것으로, 여기서는 주인은 읽고 쓰기만 할 수 있고, 그 외의 사람들은 읽을 수는 있지만 쓰거나 실행할 수는 없다.

파일의 권한은 8진수의 숫자 더하기 방식으로 나타낼 수 있다. 읽기는 4(read), 쓰기는 2(write), 실행

은 1(execute)이다. 이들 숫자의 조합으로 생길 수 있는 조합은 다음과 같다.

- 1 = 실행만 (1)
- 2 = 쓰기만 (2)
- 3 = 쓰기와 실행만 (2+1)
- 4 = 읽기만 (4)
- 5 = 읽기와 실행 (4+1)
- 6 = 읽기와 쓰기 (4+2)
- 7 = 읽기와 쓰기와 실행 (4+2+1)

즉, 8진수를 이용하는 방법은 각각 4, 2, 1의 숫자를 더한 값을 100단위는 소유자, 10단위는 그룹, 1단위는 타인으로 지정하여 사용한다. 편의를 위하여 다음에 표를 하나 정리해 보겠다.

| 사용자 범위 | 허가 | 상징적인 방법 | 절대적인 방법 |
|--------|----|---------|---------|
| 주인 | 읽기 | r | 400 |
| | 쓰기 | w | 200 |
| | 실행 | x | 100 |
| 그룹 | 읽기 | r | 40 |
| | 쓰기 | w | 20 |
| | 실행 | x | 10 |
| 다른이 | 읽기 | r | 4 |
| | 쓰기 | w | 2 |
| | 실행 | x | 1 |

표 9 chmod 옵션(1)

실제의 경우에는 다음과 같이 사용된다.

| 명령 | 주인 | 그룹 | 다른이 |
|-----------------|-----|-----|-----|
| chmod 640 file1 | rw- | r-- | --- |
| chmod 754 file1 | rwX | r-x | r-- |
| chmod 664 file1 | rw- | rw- | r-- |

표 10 chmod 옵션(2)

절대적인 방법으로 표시하는 법은 해당 부분의 숫자들의 합으로 허가를 변경하는 것이다. 예를 들어 소유자(주인) 범주에서 읽기와 쓰기 그리고 실행을 허용하면 $400+200+100=700$ 이고, 그룹모드에서 읽기와 실행만 허용하면 $40+10=50$, 마지막으로 타인모드에서도 역시 읽기와 실행을 허용하면 $4+1=5$ 가 된다. 이를 모두 더하게 되면 $700+50+5=755$ 가 된다. 이와 같은 방식으로 예문의 권한(644)을 모든 사람에게 실행 권한만 허용하도록 하려면 다음과 같이 실행한다.

```
# chmod 111 file1
```

숫자가 나오는 것을 보기만 해도 머리가 아픈 사람도 있을 것이다. 이와 같이 8진수로 나타내는 방법이 널리 사용되기는 하지만, 다음과 같은 방법으로도 사용할 수 있다.

| 기호 | 의미 | 기호 | 의미 |
|----|-------------|----|-----------------|
| + | 허가 권한 부여 | u | 주인(소유자) |
| - | 허가 권한 제거 | g | 그룹 권한 |
| = | 허가 권한 유지 | o | 다른이(타인) |
| S | 소유자와 그룹만 실행 | a | 수유자, 그룹, 다른이 모두 |

표 11 chmod 옵션(3)

위의 예에서 그룹과 다른 이에게 쓰기 권한을 주려면 다음과 같이 추가로 입력한다.

```
# chmod g+w,o+w test.c
```

hello.c 파일을 모든 사용자가 읽도록 할 때는 다음과 같이 입력한다.

```
# chmod a+r hello.c
```

10. chown

chown은 파일의 소유권을 바꾸어 주는 명령이다.

```
# chown [owner] [파일]
# chown [owner:group] [파일]
# chown [owner.group] [파일]
# chown [.owner] [파일]
```

다른 리눅스 시스템에서 복사한 파일의 소유권이 자신의 것과 일치하지 않을 때 그 파일의 소유권을 자신의 소유권으로 바꾸거나 혹은 보안을 위해 특정 사용자의 소유권을 사용하도록 해야 할 때가 있다. 이렇게 소유권을 변경할 때 사용하는 명령어가 chown이다.

chown 명령은 셸 명령행에서 지정하는 순서에 따라서 소유권이 변경된다. 앞의 예처럼 하면 된다. 그럴 경우 그룹은 변하지 않고 소유권과 바뀌게 된다. 밑의 그림과 같이 chown 명령 뒤에 변경하고자 하는 소유자와 그룹 사이에 점(.) 또는 콜론(:)을 사용하면 소유자와 그룹 모두를 변경할 수 있다.

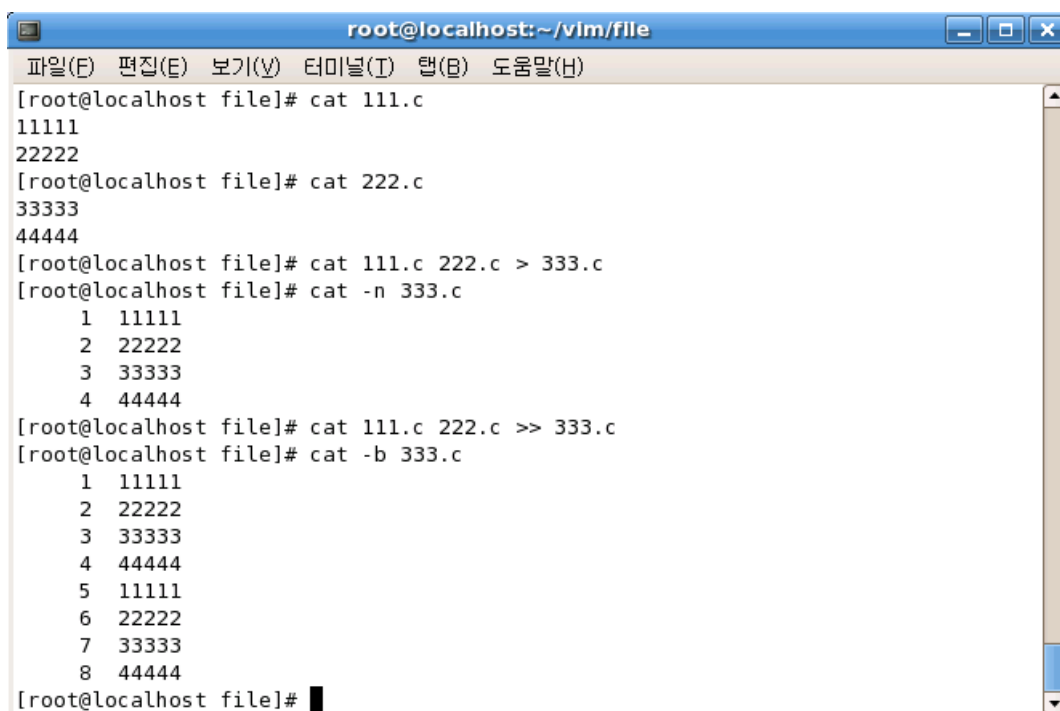
또한 디렉토리에 하위 디렉토리를 포함한 수많은 파일들에 대해서 일일이 소유권을 변경해야 한다면 -R 옵션을 사용하여 한 번에 소유권을 변경할 수 있다.

11. 파일 내용 확인 명령어 - cat, head, tail, more, less

가. cat

cat은 파일 내용을 보는 가장 기본적인 명령어이다. 당연한 이야기를 하나 하자면 보고자 하는 파일이 텍스트 파일일 경우 그 내용을 알아볼 수 있게 출력하지만, 바이너리 파일일 경우에는 내용이 깨져 보일 것이다.

```
# cat [옵션] [파일]
```



```
root@localhost:~/vim/file
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost file]# cat 111.c
11111
22222
[root@localhost file]# cat 222.c
33333
44444
[root@localhost file]# cat 111.c 222.c > 333.c
[root@localhost file]# cat -n 333.c
 1 11111
 2 22222
 3 33333
 4 44444
[root@localhost file]# cat 111.c 222.c >> 333.c
[root@localhost file]# cat -b 333.c
 1 11111
 2 22222
 3 33333
 4 44444
 5 11111
 6 22222
 7 33333
 8 44444
[root@localhost file]#
```

그림 43 cat 명령어

위의 예처럼 `cat 111.c 222.c > 333.c` 라고 했을 때, 333.c 라는 파일이 없었던 파일이라면 당연히 111.c 와 222.c 라는 두 파일의 내용이 합쳐진 결과가 나타나겠지만, 만약 333.c 가 기존에 다른 내용이 저장되었던 파일이라면 기존에 있었던 내용이 모두 사라지고 생성된 결과만이 들어갈 것이다. `>>`은 파일의 끝부분에 이어서 쓰라는 뜻이다 이렇게 입출력의 방향을 바꾸어 파일에 저장하거나 파일에서 불러 오는 것을 리디렉션이라 한다.

| | |
|--------|--------------------------------------|
| cat -b | 행 번호를 앞에 붙여서 출력한다 (빈 행은 번호를 붙이지 않는다) |
| cat -n | 행 번호를 앞에 붙여서 출력한다 (빈 행도 번호를 붙인다) |

표 12 cat 기본 옵션

나. head

파일 내용의 첫부분을 기준으로 출력하는 명령어이다. 특별히 옵션을 사용하지 않으면 맨 첫 번째

줄부터 10번째 줄까지 출력한다.

```
# head [옵션] [파일]
```

여기서 test.c 파일의 처음 3줄만 보기 위해서는 다음과 같이 입력한다.

```
# head -3 test.c
```

다. tail

파일 내용의 마지막 부분을 기준으로 출력하는 명령어이다.

```
# tail [옵션] [파일]
```

라. more, less

more와 less는 한 화면 단위로 넘어가며 텍스트 파일을 보여준다. cat의 경우 실행한 즉시 파일의 끝까지 한 번에 출력해 버려 내용이 많은 파일을 보기 쉽지 않았다. 그러나 more를 사용하면 한 화면 단위로 내용이 넘어가므로 내용을 자세히 볼 수 있으며, vi를 사용하는 것과 같은 방식으로 간단한 검색을 할 수도 있다.

```
# more [파일]
# less [파일]
```

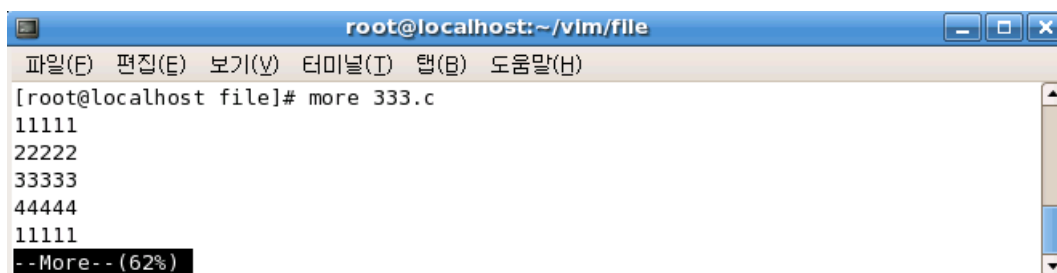


그림 44 more 명령어

[SpaceBar]를 누르면 한 페이지 단위로 이동하고 [Enter]키를 누르면 한 행 단위로 페이지가 넘어간다. more를 실행하던 중 e를 누르면 환경 변수에 설정된 기본 편집기를 사용하여 파일을 수정할 수도 있다. 그러나 이전 화면을 다시 볼 수는 없다.

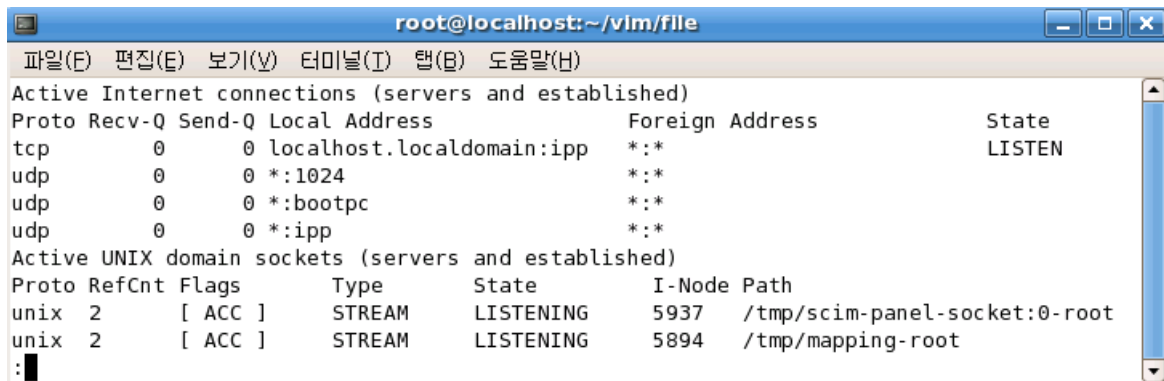


그림 45 less 명령어

less의 경우 more와 같이 한 페이지 이동시 [SpaceBar]를 이용하고, 뒤로 이동하려면 b키를 누르면 된다. 백분율로 표시되는 특정위치로 가려면 p키를 누르고 : 프롬프트에 백분율을 지정하면 움직일 수 있는 등 자유롭게 이동할 수 있다. 많은 경우 less 명령을 more라는 이름으로 심볼릭 링크를 만들어 사용하기도 한다.

12. df

이 명령은 각 파일 시스템 별로 남은 공간을 확인하여 보여 준다.

```
# df [옵션]
```

| | |
|-------|--------------|
| df -a | 모든 정보를 출력한다 |
| df -k | 용량을 kb로 보여준다 |

표 13 df 기본 옵션

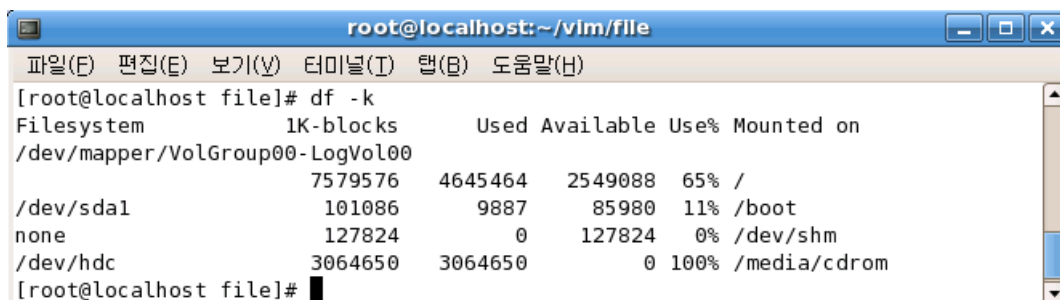


그림 46 df 명령어

이 명령어는 디스크의 남은 공간을 보여준다. 시스템이 멈춰버리거나 일반 사용자가 로그인 되지 않을 때 꼭 확인해 보아야 한다. 파티션이 일정 이상 채워지고 나면 일반 사용자는 시스템에 접근할 수 없다.

13. wc

wc 명령어는 파일 안의 글자 수, 단어 수, 줄 수를 세어 주는 명령어이다. 예컨대, 우리나라에서는 원고료를 원고지 몇 장, A4용지 몇 장과 같은 식으로 매기지만 외국에서는 A4용지 몇 장과 같은 식으로 계산하는 것은 물론 한 단어 당 얼마 정도 식으로 계산하는 경우도 있다. 이럴 때 일일이 단어 수를 계산하는 것보다 훨씬 능력적으로 사용할 수 있다.

```
# wc [옵션] [파일]
```

예컨대 어떤 파일에 대해 wc를 실행했을 때의 결과가 다음은 같다.

```
46 230 2018
```

이 결과에서 이 파일은 46줄 230단어 2018자로 이루어졌음을 알 수 있다. 몇 줄인지만 확인할 때에는 wc -l 명령을 사용한다. 바로 46이라는 결과를 볼 수 있다.

14. du

Disk Usage의 약어로 해당 디렉토리의 사용량을 출력한다. 소유자는 자신의 사용량만을 체크하지만 root는 모든 사용자 각각에 대해서 디스크 사용량을 출력한다.

```
# du [옵션] [디렉토리 또는 파일]
```

| | |
|--------|-------------------------|
| du -rm | 용량 표시를 MB로 표시 |
| du -a | 해당 경로에 대한 사용한 디스크 용량 표시 |
| du -s | 전체 사용량을 간단히 표시 |
| du -k | 용량 표기 단위를 KB로 표시 |

표 14 df 기본 옵션

du 명령어는 사용한 디스크 용량을 확인할 경우에 사용되는 명령어로 디스크에 남아 있는 용량을 확인하는 df 명령어와는 다소 차이가 있다. du 명령에서 일반적으로 디스크 용량을 확인하기 위해서는 -h 옵션을 지정하여 사용하는 것이 편리하다. 왜냐하면 디스크 용량 단위를 우리가 알고 있는 MB 또는 GB 단위로 표시하기 때문에 그 크기를 쉽게 알아볼 수 있다.

15. find

윈도우에서 파일을 찾을 때에는 [시작]→[찾기]에서 파일의 이름이나 형식, 마지막으로 변경한 날짜 등을 입력하여 간단히 찾을 수 있다. 리눅스에서는 find 명령에 다양한 옵션을 주어 같은 효과를 볼

수 있다.

```
# find [찾을 디렉토리 경로] [찾기 옵션] [찾은 후 행할 작업]
```

[찾을 디렉토리 경로]에는 다음과 같은 것들이 있다.

- . : 현재 디렉토리
- / : 루트 디렉토리 이하 (파일시스템 전체)
- ~ID : 특정 ID의 홈디렉토리 이하

[찾기 옵션]에는 다음과 같은 것들이 있다.

- -empty : 비어 있는 파일
- -gid [n] : 특정 gid을 갖는 파일 (n : 특정 gid)
- -group [gname] : 특정 group에 속한 파일 (gname : group명)
- -name : 지정한 형식을 갖는 파일 이름
- -newer : 특정 파일 이후에 생성된 파일
- -perm : 특정 허가모드를 가지고 있는 파일
- -uid [n] : 특정 uid를 갖는 파일 (n : 특정 uid)
- -used [n] : 최근에 n일 이후에 변경된 파일 (n : 일수)
- -user : 특정파일을 소유하고 있는 소유자의 파일

[찾은 후 행할 작업] 에 수행할 내용은 다음과 같은 것들이 있다.

- -print : 가장 많이 쓰는 옵션으로 찾은 파일을 보여준다.
- -exec : 찾은 파일들에 대한 특정 명령을 수행한다.

예컨대 다음과 같이 실행하면 전체 경로에서 root의 소유로 되어 있는 netbook이라는 파일을 찾는다.

```
# find / -user root -name netbook
```

16. touch

비어 있는 파일을 만들 때는 touch라는 명령어를 사용한다.

```
# touch abcd.txt
```

abcd.txt라는 파일이 생성되었으며 이 파일의 크기는 0이다. 또한 파일의 시간을 변경할 수도 있다.

17. 명령어 찾기 - which, whatis, whereis

find가 특정 파일을 찾아주는 명령어인데 비해 which라는 명령어는 사용자의 PATH 범위에서 특정 명령어의 위치가 어디인지를 찾아주는 명령어이다. 리눅스에서는 명령어의 취리를 모두 기억하기 어려우므로 이 명령어를 사용하여 찾고자 하는 명령어의 위치를 확인할 수 있다.

```
# which [명령어]
```

whereis라는 명령어도 이와 비슷한 일을 한다. which와 다른 점은 모든 디렉토리를 뒤져서 해당 명령어를 찾아준다는 것이다.

```
# whereis [명령어]
```

whatis라는 명령어가 있는데 which나 whereis가 명령어의 위치를 찾아주는데 비해 whatis는 해당 명령어가 무엇을 하는 것인지 간단하게 보여준다. 이것은 명령어 자체를 찾는 것이기 보다는 앞서 설명한 man이나 info와 비슷한 기능을 수행한다.

```
# whateis [명령어]
```

언뜻 보기에는 whereis가 which보다 성능이 뛰어나다고 생각할 수도 있으나 whereis는 개인의 PATH 변수와 명령어 그 자체만을 놓고 검색하는 것이 아닌, 더 넓은 범위에서 맨 페이지의 경로까지도 검색해 주기 때문이다. 그러나 검색 속도에서는 which가 우월하다. 또한 대부분의 실행 파일은 개인의 PATH 변수가 찾을 수 있는 경로 안에 있기 때문에 많은 경우 which를 사용하여 검색한다.

SECTION

03 프로세스 명령어

시스템의 여러 가지 상황에 대해서 알아볼 수 있는 명령어를 살펴본다. 이들 명령어를 통하여 시스템의 현 상황을 확인하고 이상이 있을 경우 적절한 조치를 취하게 된다.

프로세스란 명령어에 해당하는 실행 파일이 메모리에 적재되어 실제로 실행되는 상태가 된 것을 뜻한다. 모든 프로세스에는 각각 여러 가지 프로그램을 실제 실행하는데 필요한 정보가 있다. 이를 먼저 간단히 살펴보자.

- 프로세스 식별번호 (process ID, PID) : 프로세스가 시작될 때 할당되는 번호를 말한다. 프로세스ID는 오직 하나만 할당된다. (할당될 때 이미 죽은 프로세스의 식별번호를 재사용할 가능성은 있지만 두 개 이상의 프로세스가 같은 프로세스의 식별번호를 가질 수는 없다.)
- 사용자 식별번호 (user ID, UID) : 그 프로세스가 어떤 사용자에게 속해 있는지를 나타낸다. 이것은 누가 그 프로세스를 죽일(혹은 중단)할 수 있는지를 결정하고, 그 프로세스가 어떤 파일과 어떤 디렉토리에 대해 읽거나 쓰기 권한이 있는지 결정한다.
- 그룹 식별번호 (group ID, GID) : 프로세스가 어떤 그룹에 속하여 있는지를 나타낸다.
- 파일 기술자 (file descriptor) : 프로세스가 읽기 혹은 쓰기를 위해 어떤 파일을 열고 있는지, 또 그 각각의 파일 내에서 어떤 위치에 도달해 있는지를 기록한다.

1. ps

지금 실행하고 있는 프로세스를 볼 수 있는 명령이다.

```
# ps [옵션]
```

```
root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# ps -aux | more
Warning: bad syntax, perhaps a bogus '-?' See /usr/share/doc/procps-3.2.5/FAQ
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.2   1884    652 ?        S    22:58   0:00 init [5]

root         2  0.0  0.0      0      0 ?        SN   22:58   0:00 [ksoftirqd/0]
root         3  0.0  0.0      0      0 ?        S    22:58   0:00 [watchdog/0]
root         4  0.0  0.0      0      0 ?        S<   22:58   0:00 [events/0]
root         5  0.0  0.0      0      0 ?        S<   22:58   0:00 [khelper]
```

그림 47 ps 명령어

이 화면의 위 줄을 살펴보겠다. 각 항목의 내용은 다음과 같다.

| 항목 | 내용 |
|---------|---|
| USER | 프로세스 소유자의 사용자 이름 |
| PID | 프로세스의 식별번호 |
| %CPU | 프로세스가 사용하는 CPU의 백분율 |
| %MEM | 프로세스가 사용하는 메모리의 백분율 |
| VSZ | 페이지 단위의 가상 메모리 용량(KB 단위) |
| RSS | 실제 메모리 사용량 |
| TTY | 프로세스와 연결된 터미널 포트 ?는 시스템 부팅시 데몬에서 실행되었다는 것을 의미 |
| STAT | 현재 프로세스의 상태 R(Runnable) - 실행대기 상태, S(Sleeping)은 수면상태, D(in Disk wait) - 입출력을 기다리는 인터럽트가 불가능한 상태, T(sTopped) - 멈춰있거나 흔적이 남아있는상태, Z(Zombie) - 완전히 죽어있는 상태 |
| START | 프로그램이 시작된 시간 |
| TIME | 프로세스가 사용한 CPU 총 사용시간 |
| COMMEND | 프로세스의 실행 명령어 |

표 15 ps 항목별 내용

다음은 ps 명령과 함께 사용할 수 있는 옵션에 대한 설명이다.

| | |
|-------|----------------------|
| ps -a | 다른 사용자의 프로세스 현황도 표시 |
| ps -u | 실행한 유저와 실행 시간으로 표시 |
| ps -x | 터미널 제어 없이 프로세스 현황 표시 |
| ps -e | 모든 프로세스 선택하여 표시 |
| ps -f | 실행되는 모든 것을 표시 |
| ps -l | 긴 포맷으로 출력 |
| ps -j | "job"형식으로 표시 |

표 16 ps 기본 옵션

2. kill

일반적으로 프로세스들은 다음과 같은 세 가지 상황에서 활동을 중지한다.

- 상호 작용하는 프로세스에 [Ctrl]+z 가 입력되었을 때
- 사용자나 프로그램의 특별한 요청이 있을 때
- 우선순위가 낮은 프로세스가 제어 터미널에 접근하려 할 때

그 중 가장 잘 쓰이는 상황이 사용자나 프로그램에 특별한 요청을 하는 것으로써, 바로 kill 명령이다. kill 명령은 지정한 프로세스에 지정한 시그널(signal)을 보내어 프로세스를 종료시키도록 한다.

```
# kill [-시그널번호 또는 시그널] PID
```

```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# ps -ef | grep hanterm
root      2177      1  0 23:17 ?        00:00:00 hantermxf
root      2226    2083  0 23:18 pts/0    00:00:00 grep hanterm
[root@localhost ~]# kill -9 2177
[root@localhost ~]# ps -ef | grep hanterm
root      2228    2083  0 23:18 pts/0    00:00:00 grep hanterm
[root@localhost ~]#

```

그림 48 kill 명령어

kill 명령이 가지고 있는 시그널(signal) 종류를 알아보기 위해서는 kill -l 을 입력한다.

```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
5) SIGTRAP     6) SIGABRT     7) SIGBUS      8) SIGFPE
9) SIGKILL     10) SIGUSR1    11) SIGSEGV    12) SIGUSR2
13) SIGPIPE    14) SIGALRM    15) SIGTERM    17) SIGCHLD
18) SIGCONT    19) SIGSTOP    20) SIGTSTP    21) SIGTTIN
22) SIGTTOU    23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO
30) SIGPWR     31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1
36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4 39) SIGRTMIN+5
40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8 43) SIGRTMIN+9
44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13
52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9
56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6  59) SIGRTMAX-5
60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2  63) SIGRTMAX-1
64) SIGRTMAX
[root@localhost ~]#

```

그림 49 시그널의 종류

프로세스를 다시 실행하려면 -SIGHUP 또는 -1 옵션을 사용한다. 주로 이러한 경우는 시스템의 환경을 바꾸고 나서 데몬을 다시 띄워야 할 상황에서 시스템을 재시작하지 않고 데몬만 재시작 해주는 경우 유용하다. 시그널에 대해서는 추후 책의 뒷부분에 자세히 언급된다.

3. top

CPU 부하, 프로세스 로드, 메모리 사용량 같은 다양한 시스템 자원 정보를 실시간으로 제공한다.

```
# top [옵션]
```

아래 그림을 해석해 보면 다음과 같다. 23:25:53은 top을 실행한 시간으로써, 여기에서는 오후 11시 25분에 top 프로그램을 실행했다는 뜻이다. up 27min이라는 메시지는 시스템을 부팅한지 27분 지났다는 뜻이며, 3명이 접속해 있고, 평균 로드 수(load average)는 보이는 대로 0.07이다. Task: 82

total, 2 running, 80 sleeping, 0 stopped, 1 zombie라는 메시지는 83개의 프로세스가 메모리에 적재되어 있으며 80개는 활동이 잠시 멈춰진 상태이며 2개는 계속 활동하고 있다는 사실을 알려준다. 정지된 프로세스는 없고 좀비 프로세스가 1개 있다. 또한 실제 메모리와 가상 메모리 상태를 보여주며, 실행 중인 프로세스 중 많은 자원을 소모하는 상위 몇 개의 프로세스의 정보도 보여준다.

```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
top - 23:25:53 up 27 min,  3 users,  load average: 0.07, 0.06, 0.13
Tasks: 83 total,  2 running, 80 sleeping,  0 stopped,  1 zombie
Cpu(s):  1.7% us,  4.2% sy,  0.0% ni, 86.6% id,  7.3% wa,  0.1% hi,  0.0% si
Mem:   255652k total,  236688k used,   18964k free,    8144k buffers
Swap:  524280k total,    0k used,   524280k free,   121972k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 1851 root        16   0 35648  13m 4908 R   6.8   5.5   0:19.55 X
 1989 root        15   0 29492  10m 7568 S   3.4   4.0   0:02.72 metacity
 2261 root        15   0  1976   964  772 R   3.4   0.4   0:00.13 top
    1 root        16   0  1884   652  552 S   0.0   0.3   0:00.60 init
    2 root        34  19    0    0    0 S   0.0   0.0   0:00.00 ksoftirqd/0
    3 root         RT   0    0    0 S   0.0   0.0   0:00.00 watchdog/0
    4 root        10  -5    0    0    0 S   0.0   0.0   0:00.02 events/0
    5 root        10  -5    0    0    0 S   0.0   0.0   0:00.02 khelper
    6 root        11  -5    0    0    0 S   0.0   0.0   0:00.00 kthread
  
```

그림 50 top 명령어

top을 실행한 상태에서 k를 누른 다음 PID를 입력하면 해당 프로세스를 간단히 kill 할 수 있다. top 실행을 종료할 시에는 q를 누르기만 하면 된다.

| | |
|--------|------------------------|
| top -d | 스크린을 갱신할 때의 시간 출력 |
| top -p | 주어진 PID에 대한 프로세스만 모니터링 |
| top -q | 지연 시간 없이 스크린 갱신 |
| top -s | 보안 모드로 실행 |
| top -i | idle 및 zombie 프로세스 무시 |
| top -c | 명령 라인을 모두 보여줌 |

표 17 top 기본 옵션

4. nohup

```
# nohup [명령] &
```

nohup 명령은 부모 프로세스가 죽거나 종료되더라도 자식 프로세스는 계속 동작하도록 백그라운드 모드에서 프로세스를 실행하는 명령이다. 일반적으로 부모 프로세스가 종료되면 자식 프로세스도 종료되는데 이 명령을 실행하면 사용자가 로그아웃을 하더라도 자식 프로세스는 죽지 않고 실행되며, 이에 대한 정보는 nohup.out 파일에 기록된다.

SECTION

04 압축 명령어

리눅스에서 사용하는 많은 프로그램과 문서는 대개의 경우 각 상황에 맞는 적당한 압축 형식으로 구할 수 있다. 이 부분에서는 리눅스를 사용하기 위해 꼭 필요한 압축 명령어들을 살펴본다.

1. tar

tar는 여러 개의 파일을 하나의 파일로 묶고 묶여진 파일을 다시 푸는 유틸리티다. 또한 묶여진 파일을 풀기 전에 어떤 파일들이 들어있는지를 확인해 볼 수 있다.

이 명령어는 파일을 압축하는 것이 아니라, 디렉토리를 포함한 모든 파일을 단지 하나의 파일로 묶어주는 기능만을 한다. 처음에는 테이프에 백업할 데이터를 묶기 위해 사용되었지만, 요즘엔 대부분의 소스 코드와 바이너리를 배포할 때 이 포맷이 사용되고 있다. 기본적인 사용 방법은 다음과 같다.

```
# tar [옵션] [파일이름] [대상디렉토리 및 파일]
```

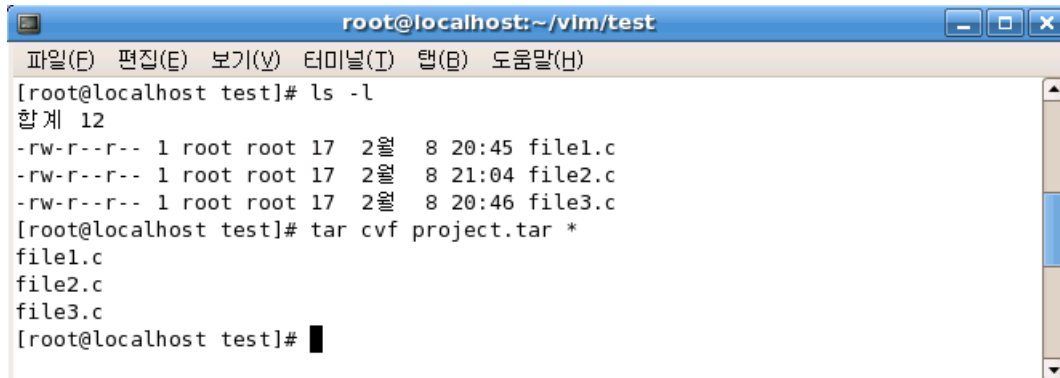
다음은 tar 명령과 함께 사용되는 옵션에 대한 설명이다.

| | |
|-------|-------------------------------------|
| tar c | 새로운 파일을 생성 함 |
| tar d | tar 파일과 해당 파일시스템 간의 차이점 출력 |
| tar r | tar 파일에 다른 파일들을 추가 |
| tar t | 묶여져 있는 tar 파일의 내용을 확인 |
| tar f | 읽거나 쓸 파일이름이 연속일 경우 붙여서 쓰게 함(반드시 사용) |
| tar p | tar 파일을 생성할 당시의 파일 퍼미션을 그대로 하여 풀어 줌 |
| tar v | 묶을 때나 풀어줄 때 파일들의 내용을 자세히 출력 |
| tar Z | 파일을 묶거나 풀 때 압축프로그램으로 압축 함 |
| tar z | 파일을 묶어나 풀 때 gzip으로 압축 함 |

표 18 tar 기본 옵션

일반적으로 tar로 작업을 하는 경우에는 다음의 3가지 경우가 있다. 첫 번째는 여러 개의 파일 및 디렉토리를 하나의 tar 파일로 묶는 경우이고, 두 번째는 이렇게 묶은 tar 파일을 다시 풀어준 경우이며, 세 번째가 묶여져 파일의 내용을 확인하는 것이다.

```
# tar cvf [압축파일.tar] [압축대상파일 및 디렉토리]   · 묶을 때
# tar vcf [압축파일.tar]                               · 확인할 때
# tar xvf [압축파일.tar]                               · 풀 때
```



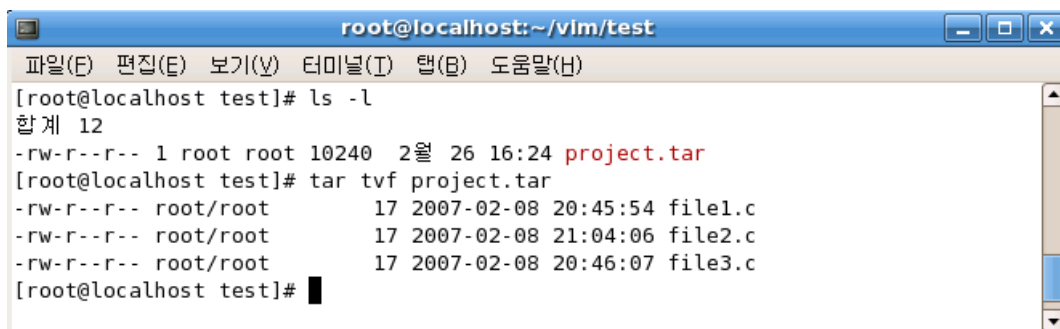
```

root@localhost:~/vim/test
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost test]# ls -l
합계 12
-rw-r--r-- 1 root root 17  2월  8 20:45 file1.c
-rw-r--r-- 1 root root 17  2월  8 21:04 file2.c
-rw-r--r-- 1 root root 17  2월  8 20:46 file3.c
[root@localhost test]# tar cvf project.tar *
file1.c
file2.c
file3.c
[root@localhost test]#

```

그림 51 tar로 파일 묶기

위 그림은 현재 디렉토리에 있는 모든 파일과 디렉토리를 project.tar라는 파일 하나로 묶는 모습을 보여주고 있다. 원본 파일은 그대로 존재하고 project.tar라는 파일이 새로 생성되어 그 파일에 지정 한 모든 파일 및 디렉토리가 들어가게 된다. c옵션을 사용해야 한다.



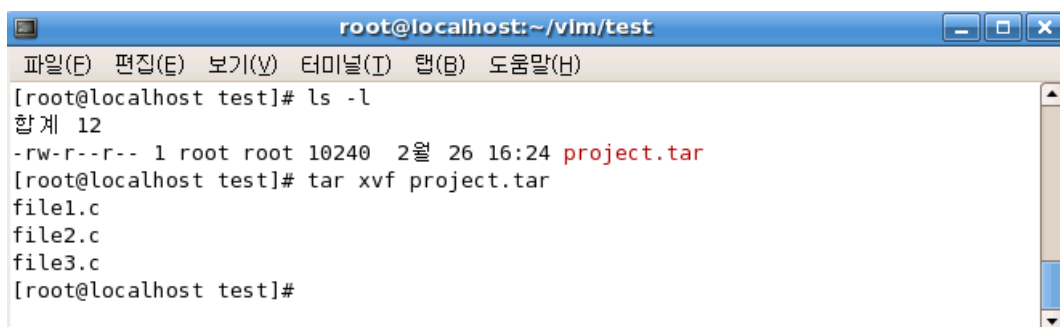
```

root@localhost:~/vim/test
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost test]# ls -l
합계 12
-rw-r--r-- 1 root root 10240  2월 26 16:24 project.tar
[root@localhost test]# tar tvf project.tar
-rw-r--r-- root/root      17  2007-02-08 20:45:54 file1.c
-rw-r--r-- root/root      17  2007-02-08 21:04:06 file2.c
-rw-r--r-- root/root      17  2007-02-08 20:46:07 file3.c
[root@localhost test]#

```

그림 52 tar 파일의 내용

위 그림은 project.tar라는 tar 파일에 어떤 파일들이 묶여져 있는지 확인해 본 것이다. t 옵션을 사용 한다.



```

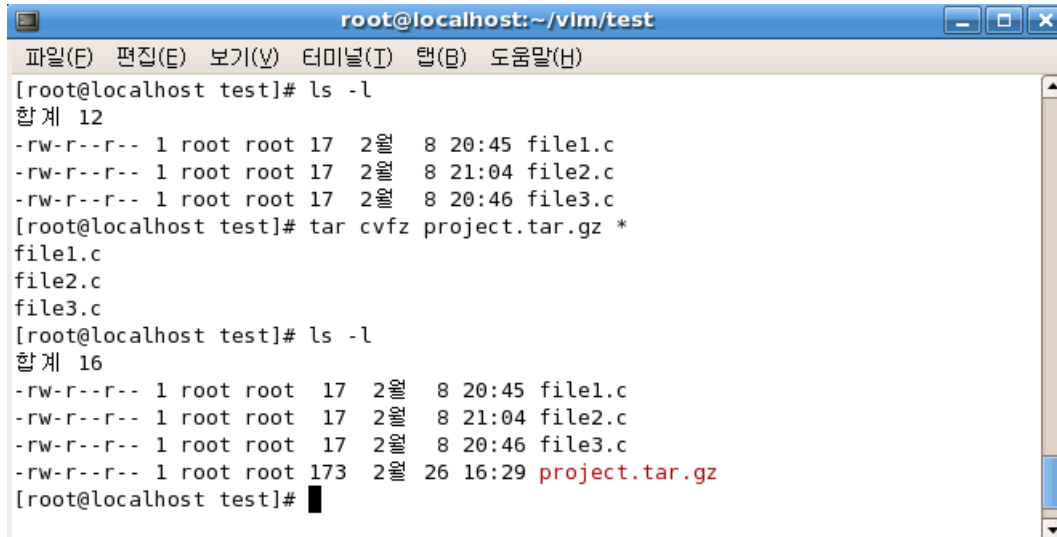
root@localhost:~/vim/test
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost test]# ls -l
합계 12
-rw-r--r-- 1 root root 10240  2월 26 16:24 project.tar
[root@localhost test]# tar xvf project.tar
file1.c
file2.c
file3.c
[root@localhost test]#

```

그림 53 tar 파일 풀기

위 그림은 project.tar에 묶여져 있는 파일들을 현재 디렉토리에 풀어준 것이다. x 옵션을 사용한다.

tar로 만들어진 파일은 압축한 상태가 아니므로 파일 크기가 상당히 클 수 있다. 효율적인 압축과 백업을 위해 보다 작은 크기의 파일을 만들어야 하는데, 이 경우 z 옵션을 사용하면 된다. z 옵션을 주면 tar 자체가 압축을 하는 것이 아니라 gzip 명령을 불러 압축을 시도한다.



```

root@localhost:~/vim/test
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost test]# ls -l
합계 12
-rw-r--r-- 1 root root 17  2월  8 20:45 file1.c
-rw-r--r-- 1 root root 17  2월  8 21:04 file2.c
-rw-r--r-- 1 root root 17  2월  8 20:46 file3.c
[root@localhost test]# tar cvfz project.tar.gz *
file1.c
file2.c
file3.c
[root@localhost test]# ls -l
합계 16
-rw-r--r-- 1 root root  17  2월  8 20:45 file1.c
-rw-r--r-- 1 root root  17  2월  8 21:04 file2.c
-rw-r--r-- 1 root root  17  2월  8 20:46 file3.c
-rw-r--r-- 1 root root 173  2월 26 16:29 project.tar.gz
[root@localhost test]#

```

그림 54 z 옵션 사용

z 옵션을 주어서 압축을 하면 용량이 많이 줄어든다. 그러나 이 파일을 풀어 놓으려 할 때, 앞서 설명한 일반적인 방법을 사용하면 다음과 같은 에러 메시지가 뜬다.



```

root@localhost:~/vim/test
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost test]# ls -l
합계 4
-rw-r--r-- 1 root root 173  2월 26 16:29 project.tar.gz
[root@localhost test]# tar xvf project.tar.gz
tar: Read 173 bytes from project.tar.gz
[root@localhost test]#

```

그림 55 tar 에러 메시지

위의 예에서 압축을 풀려면 tar xvfz project.tar로 하면 된다. 통상적으로 z 옵션으로 압축하여 사용할 때는 식별을 위해 tar cvfz project.tar.gz과 같이 파일명 뒤에 gz를 붙여주는 것이 좋다.

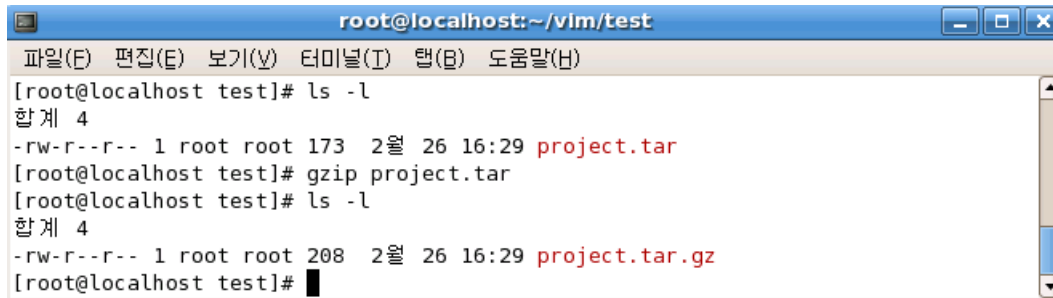
2. gzip

리눅스 환경에서 가장 많이 사용하는 압축 유틸리티이다. 파일의 크기를 작게 만들어 다른 네트워크로 전송할 때 시간을 절약해 준다. 압축할 때에는 다음과 같이 입력한다.

```
# gzip [압축할 파일명]
```

압축된 내용을 풀어 줄 때에는 다음과 같이 입력한다.

```
# gzip -d [압축된 파일명]
```



```

root@localhost:~/vim/test
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost test]# ls -l
합계 4
-rw-r--r-- 1 root root 173  2월 26 16:29 project.tar
[root@localhost test]# gzip project.tar
[root@localhost test]# ls -l
합계 4
-rw-r--r-- 1 root root 208  2월 26 16:29 project.tar.gz
[root@localhost test]#
  
```

그림 56 gzip 명령어

압축 해제에 경우에는 다음과 같이 사용할 수도 있다.

```
# gunzip [압축된 파일명]
```

gzip 명령은 옵션을 주어 압축할 수 있다. gzip -? [압축할 파일명] 의 형식으로 ?에는 1에서 9까지의 숫자가 들어간다. 1은 가장 빠르게 압축하는 대신 파일의 크기에는 거의 변화가 없고 9는 압축속도는 느리지만 가장 작은 크기로 압축한다. 그 사이의 숫자들은 각각 숫자가 커질수록 더 좋은 효율로 압축하게 된다. 옵션을 사용하지 않을 경우에는 기본값은 6으로 압축된다.

Chapter 3. 에디팅

SECTION

01 vi 개괄

1. vi의 역사

vi는 BSD의 쉘(C shell)을 개발한 빌 조이가 1976년에 ed의 기능을 확장시킨 ex(EXTended editor) 편집기를 개발하고 이를 확장시켜서 만들었다.

vi 이전에 사용되던 에디터로서 ed와 ex라는 라인 에디터가 있는데 ed를 사용해본 독자라면 라인 에디터가 얼마나 불편한지 알 수 있을 것이다. ed나 ex 같은 라인 에디터는 근대의 에디터와는 인터페이스가 많이 틀리다. 라인 에디터는 마치 명령을 내리듯이 문서를 편집하기 때문에 사용법이 복잡하고 다루기 힘든 편집기였다.

행 단위로 문서를 편집하던 시기에 마치 종이에 글을 쓰듯이 문서를 작성한다는 일은 매우 고무적인 일이었을 것이다. 믿겨지지 않겠지만 그래서 vi가 VIsual edit의 준말이다.

vi는 기존의 에디터에 대한 새로운 인터페이스뿐만 아니라 에디터의 통합을 가지고 왔다. vi 이전에는 각 벤드의 터미널마다 각각의 제어 코드가 틀려서 독립적인 에디터가 존재할 수 밖에 없었다. 그러나 각각의 서로 다른 터미널의 특성을 제어하기 위한 termcap이라는 데이터베이스를 이용하여 동일한 인터페이스로 제어를 할 수 있게 되었다. 세상의 많은 유닉스 시스템에서 vi를 표준으로 선택한 것은 당연한 일이었다.

2. vi의 장점

필자가 생각하는 vi의 최대 장점은 바로 키보드로 모든 것이 가능하다는 것이다. vi를 사용하면 마우스를 만질 필요가 없다. 키보드로 모든 기능을 다 사용할 수 있기 때문에 일에 집중할 수도 있고 또한 매우 빠르게 일을 처리할 수 있다.

vi는 마우스가 없는 시대에 만들어졌기 때문에 현대에 나오는 어떤 GUI 에디터보다 독특한 인터페이스를 가지고 있다. 이러한 독특한 인터페이스로 인해 vi를 처음 접하는 사용자는 많은 불편함을 느낀다. 그러나 익숙해지고 나면 vi가 정말 강력한 에디터라는 것을 알 수 있다.

SECTION

02 기본적인 vim 사용법

vi를 자유자재로 다룰 수 있기를 원한다면 각인해두어야 할 말이 있다.

“vi는 배우는 것이 아니라 몸으로 익히는 것이다”

vi는 습관이다. vi의 많은 명령들은 매우 단순하며 일관성이 있다. 먼저 그러한 일관성을 파악해야 되고, 일관성을 파악한 후에는 문서의 어떤 부분을 편집하려고 먹었다면 머릿속의 생각보다 빠르게 번개와 같은 손놀림으로 이미 편집하고 있을 때까지 부단히 사용해야 한다.

인터넷 게시판에서 글을 다 쓰고 습관적으로 vi의 저장 명령인 [ESC]+:wq!를 눌러서 그동안 작성한 내용이 전자 먼지로 사라져가는 것을 경험할 때쯤 되면 vi가 얼마나 괜찮은 에디터인지 알 수 있다. 그 전까지 vi는 단지 독특한 그리고 불편한 에디터일 뿐이다.

이번 장 역시 눈으로 보고 이해하는 장이 아니라 손으로 따라하는 장이라는 것을 기억하고 예제가 나오면 무조건 따라해 보기 바란다. 다시 한 번 말하지만 vi는 배우는 것이 아니라 몸이 익히는 것이다.

vi는 30년 된 공룡시대 에디터인 만큼 오랜 진화 과정을 거치면서 많은 클론을 낳았는데 앞으로 실습에서 사용할 vim이 바로 그것이다. vim은 Vi IMproved의 준말로써 vi의 기본 기능에 충실하면서 vi에는 없는 편리한 기능들이 추가되었다. 유닉스 시스템은 벤더에 따라 vi 또는 vim을 탑재한 시스템이 있는데 리눅스는 거의 vim이 탑재되어 있다. 전통적인 vi 보다는 vim이 인터페이스나 기능면에서 훨씬 월등하기 때문에 vim의 사용을 권한다. 그러므로 앞으로 vim을 기준으로 설명한다.

1. vim의 모드 이해

vim에 대해서 이해하고자 한다면 먼저 vim의 여러 가지 모드에 대해서 이해해야 한다. vim은 3가지 모드를 가지고 있는데 명령모드, 입력모드, ex모드가 바로 그것이다.

가. 명령모드

명령모드는 키 입력을 통해 vim에게 명령을 내리는 모드다. 명령모드에서 커서를 이동하거나, 삭제, 복사, 붙이기 등의 작업을 수행할 수 있다. 이 명령모드 때문에 대부분 처음 vim을 접하는 사람들은 생소함을 느낀다. 왜냐하면 vim을 실행하면 명령보드부터 시작하는데 명령모드에서는 아무리 타이핑해도 글자가 입력되지 않기 때문이다. 명령보드는 vim에게 명령을 내리기 위한 모드지 편집을 위한 모드가 아니라는 것을 기억하기 바란다.

나. 입력모드

입력모드는 실제로 문서를 편집하기 위한 모드다. 입력모드에서 글자를 타이핑하게 되면 실제로 화

면에 출력되면서 글자의 입력이 가능하게 된다. 다시 한 번 강조한다. vim은 명령모드와 입력모드가 있는데 명령모드는 키 입력으로 삭제, 복사, 붙이기 등의 작업을 수행하는 모드고 입력모드가 실제로 문서를 타이핑하여 편집하기 위한 모드라는 것을 기억하기 바란다. vim은 일반 에디터와 틀리게 명령모드와 입력모드가 분리되어 있다. 이러한 특징을 이해하지 못하고 접근하면 vim은 낯설고 이해하기 힘든 에디터가 된다.

다. ex모드

ex모드는 라인 에디터인 ex 에디터의 기능을 사용하는 모드다. vim은 ex 에디터를 기반으로 만들어진 에디터다. 그래서 ex 에디터의 기능을 그대로 사용할 수 있다. ex모드를 사용하면 특정 패턴들을 특정 문자열로 대체한다는 것과 같이 일괄적으로 처리해야 할 작업에 매우 효율적이다. 그리고 ex모드에서 ex 라인에디터의 기능만 구현되어 있는 것은 아니다. vim을 더욱 강력하게 하는 많은 기능들이 숨어있는 모드가 바로 ex모드다.

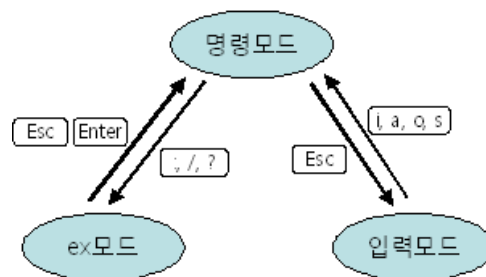


그림 57 모드 전환

각 모드의 전환은 위 그림과 같다. vim을 처음 실행시키면 명령모드에서부터 시작한다. 명령모드에서 I, i, A, a, O, o, s 중 한 키를 누르면 편집이 가능한 입력모드로 전환된다.(입력모드 전환키를 다 기억할 필요는 없다.)

입력보드에서 복사, 붙이기 등과 같은 작업을 수행하기 위해서는 명령모드로 다시 전환해야 한다. 입력모드에서 명령모드로의 전환은 [Esc]키를 누르면 된다. 그림에서 보는 바와 같이 입력모드에서 ex모드로 바로 전환하는 것은 불가능하다. 일단 명령모드로 전환한 후 ex모드로 전환해야 한다. 명령모드에서 ex모드로 전환하기 위해서는 :, /, ? 키 중 한 키를 누르면 된다.

: 키는 ex 편집기 명령과 vi의 ex모드 명령을 입력할 때 사용하고, /, ?는 특정한 패턴을 찾을 때 사용한다. / 키를 누른 후 찾고자 하는 패턴을 입력하면 패턴이 나오는 곳으로 커서가 이동한다. /는 정방향 탐색이고 ?는 역방향탐색을 할 때 이용된다.

vi의 기본 모드에 대해서 알았다면 이제는 에디터로써의 기본 기능에 대해서 알아보자. 에디터로써의 기본 기능에는 입력, 이동, 삭제, 복사, 붙이기, 찾기 그리고 치환, 저장 및 종료 등이 있다. 먼저 실행 방법과 저장, 종료에 대해서 알아보자.

2. 파일 열기, 저장, 종료

```
# vi hello.c
```

리눅스 셸 프롬프트 상에서 위 명령을 내리면 vim이 실행된다. '\$'는 셸 프롬프트를 의미한다. 실행시 이미 hello.c 파일이 존재한다면 hello.c 파일이 열리게 되고 hello.c 파일이 존재하지 않으면 새로 파일을 생성하게 된다.

만약 hello.c 파일이 있어서 기존의 파일을 편집하게 된다면 vim은 hello.c 파일에 대응하는 .hello.c.swp 파일을 생성한다.

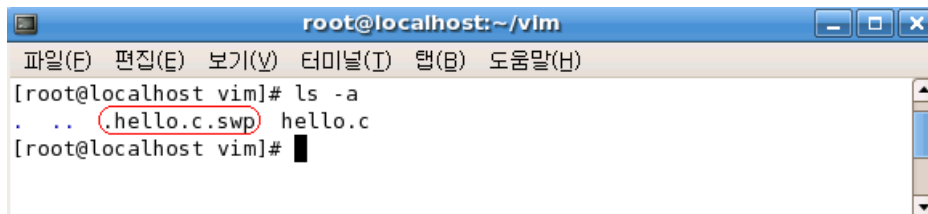


그림 58 .hello.c.swp 파일

.hello.c.swp 파일에는 파일의 소유자, 사용자 이름, 파일명과 경로, 수정 시간 등과 같은 각종 제어 정보와 함께 파일의 내용이 들어있다. 화면에 일단 편집하게 되면 이 .hello.c.swp 파일에 기록된 내용이 hello.c 파일에 반영된다. 파일이 존재하지 않아 새로 생성하는 경우 생성하는 파일의 임시 파일에 일단 기록하고 저장 명령이 내려졌을 때 임시 파일에 기록된 내용을 새로 생성할 파일에다 기록하게 된다.

이러한 .swp 임시 파일을 이용한 기록 방법은 정전 상황과 같이 시스템이 예상하지 못한 상황이 발생할 때를 대비하기 위함이다. 누구나 중대한 보고서나 소스를 작성하다가 갑자기 정전 혹은 예상치 못한 시스템 종료 발생하여 비명을 질러본 경험이 있을 것이다.

만약 vim을 사용하다가 정전이 되었다면 걱정할 필요가 없다. .swp 파일에 이미 작성한 내용이 다 남아있기 때문이다. 가끔은 .swp 파일이 남아있어서 vim을 실행할 때 그림과 같은 메시지가 출력되곤 한다.

주로 vim으로 파일을 편집하던 중에 vim을 정상 종료하지 않고 터미널을 종료하거나 vim이 이상 종료했을 때, 혹은 다른 곳에서 해당 파일을 편집하고 있을 때 .swp 파일이 남아서 그림과 같은 메시지가 출력된다. 이런 경우에는 rm 명령어로 .swp 파일을 수동으로 지워야 한다.

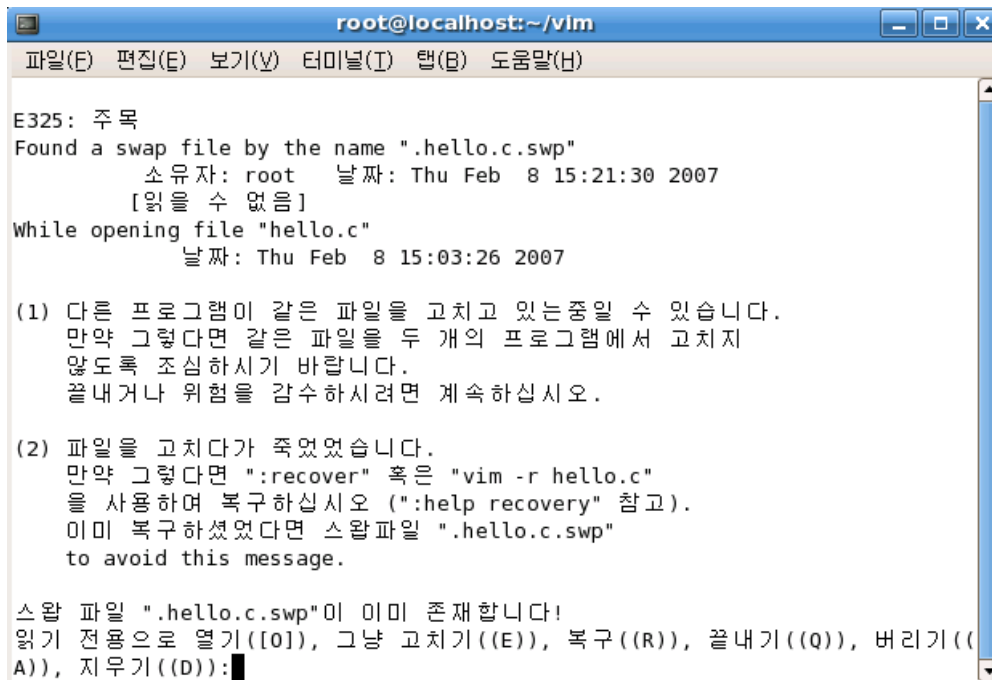


그림 59 기존의 .swp 파일 때문에 생긴 주의 메시지

이제 따라해보자. `vi hello.c` 명령으로 파일을 열었다면 기본적으로 명령모드에서 시작한다. `i` 키를 누르면 명령모드에서 입력모드로 전환되어 텍스트를 편집할 수 있게 된다.



그림 60 입력모드에서 텍스트 입력 후 저장

텍스트 입력이 끝나면 `[Esc]`키를 눌러 입력모드에서 명령모드로 전환한다. 현재까지는 `hello.c` 파일에 입력 내용이 반영되지 않고 단지 `.hello.c.swp` 파일에만 입력한 내용이 있는 상태다.

`hello.c` 파일에 저장하기 위해서는 `[Esc]`키를 누르고 `:` 키를 눌러 `ex`모드로 전환하여 그림과 같이 `w` 키를 누르고 `[Enter]`키를 치면 된다.

종료는 명령모드에서 `:` 키를 눌러 일단 `ex`모드로 전환하고 `q`키를 누르게 되면 종료하게 된다. (`[Esc]` : `[Enter]`키를 순차적으로 누르면 종료된다.) 일반적으로 저장 후 종료하는 것은 `ex`모드에서 다음과 같은 형식으로 사용한다.

```
.wq!
```

wq는 저장 후 종료를 의미하고 !는 강제 저장, 종료를 의미한다. 만약 사용자가 자신의 hello.c 파일을 수정 중이었는데 hello.c 파일의 퍼미션이 읽기 전용이었다면 저장하려고 할 때 읽기 전용 파일이라는 경고 메시지가 뜨면서 저장할 수 없다. 그때는 !를 붙여서 강제 저장해야 한다.

마찬가지로 문서에 수정을 가하고 나서 저장하지 않고 종료하기 위해서 :q 키를 누르게 되면 종료가 되지 않는다. 그럴 때도 :q!를 입력해서 강제로 종료하면 된다.

| 명령어 | 설명 |
|----------------|----------------------|
| :w | 저장 |
| :w file.txt | file.txt 파일로 저장 |
| :w >> file.txt | file.txt 파일에 덧붙여서 저장 |
| :q | vi 종료 |
| ZZ | 저장 후 종료 |
| :wq! | 강제 저장 후 종료 |
| :e file.txt | file.txt 파일을 불러옴 |
| :e | 현재 파일을 불러옴 |

표 19 저장 및 종료

일단 vim을 처음 시작하는 사람은 다음 명령만 반드시 기억하기로 하자.

- 저장 : [Esc] + :w
- 저장 후 종료 : [Esc] + :wq!
- 저장하지 않고 종료 : [Esc] + :q!
- file.txt를 불러옴 : [Esc] + :e file.txt

3. 입력

입력은 입력보드에서 가능하다. 명령모드에서 입력모드로 전환하기 위해서는 a, A, i, I, o, O, s, cc 등의 키 입력으로 가능하다. 독자들은 입력모드 키가 너무 많은 것이 의아스러울 것이다. 각 키에 대한 설명은 다음 표와 같다.

| | | | |
|---|---------------------|----|--------------------|
| a | 커서 위치 다음 칸부터 입력 | A | 커서 행의 맨 마지막부터 입력 |
| i | 커서의 위치에 입력 | I | 커서 행의 맨 앞에서부터 입력 |
| o | 커서의 다음 행에 입력 | O | 커서의 이전 행에 입력 |
| s | 커서 위치의 한 글자를 지우고 입력 | cc | 커서 위치의 한 행을 지우고 입력 |

표 20 입력모드 전환 키

위 키는 대표적인 키를 나열한 것이고 위에서 나열한 키 말고도 몇 가지 키가 더 있다. 위 키 중 특히 많이 사용하는 키의 경우에는 음영으로 처리했다.

위 키들을 모두 다 외우고 쓸 필요는 없다. 물론 많이 알면 그 만큼 더 편하겠지만 vim을 처음 시작

하는 지금으로써는 i(insert) 키만 기억하도록 하다. 안 그러면 앞으로 나올 수 많은 키들로 인해 꿈 속에서 키보드 괴물과 싸우게 될지도 모른다. 그리고 믿기지 않겠지만 나중에 익숙해지면 저 많은 키들을 모두 골고루 사용하게 될 것이다.

4. 이동

vim은 커서 이동과 관련하여 매우 재미있는 방식을 사용한다. 입력모드에서 이동은 일반 편집기와 같이 화살표 키를 이용하면 된다. 명령모드에서는 화살표 키로 이동할 수도 있고 h, j, k, l 키를 이용할 수도 있다. h, j, k, l 키는 다음 표와 같이 대응된다.

| h | j | k | l |
|---|---|---|---|
| ← | ↓ | ↑ | → |

표 21 명령모드 이동 키

처음엔 h, j, k, l 키를 이용한 커서 이동이 어딘가 모르게 어색할 것이다. 그러나 h, j, k, l 키에 익숙해지고 나면 정말 괜찮은 키 맵핑이라는 것을 알 수 있다. h, j, k, l 키를 이용함으로써 손이 키보드의 중심에서 벗어나지 않고 이동이 가능하기 때문이다.

명령모드에서 단어 단위의 이동은 w 키와 b 키로 가능하다. w 키를 누르면 다음 단어의 첫 글자로 이동하고 b키를 누르면 이전 단어의 첫 글자로 이동하게 된다.

같은 행 내에서 제일 처음 글자와 마지막 글자로 이동하는 것은 ^ 키와 \$ 키를 사용한다. ^ 키를 누르게 되면 행의 제일 첫 글자로 이동하고 \$를 누르게 되면 행의 제일 마지막 글자로 커서가 이동하게 된다. + 키와 - 키는 행을 변경할 때 이용된다. + 키는 다음 행의 첫 글자로 이동하고 - 키는 이전 행의 첫 글자로 이동한다. (키와) 키는 문장의 이동에 이용된다. vi에서 문장의 끝이란 마침표(.)가 나오는 지점을 의미한다. { 키와 } 키는 문단 단위의 이동에 이용된다. 문단은 공백인 행을 기준으로 나누어진다.

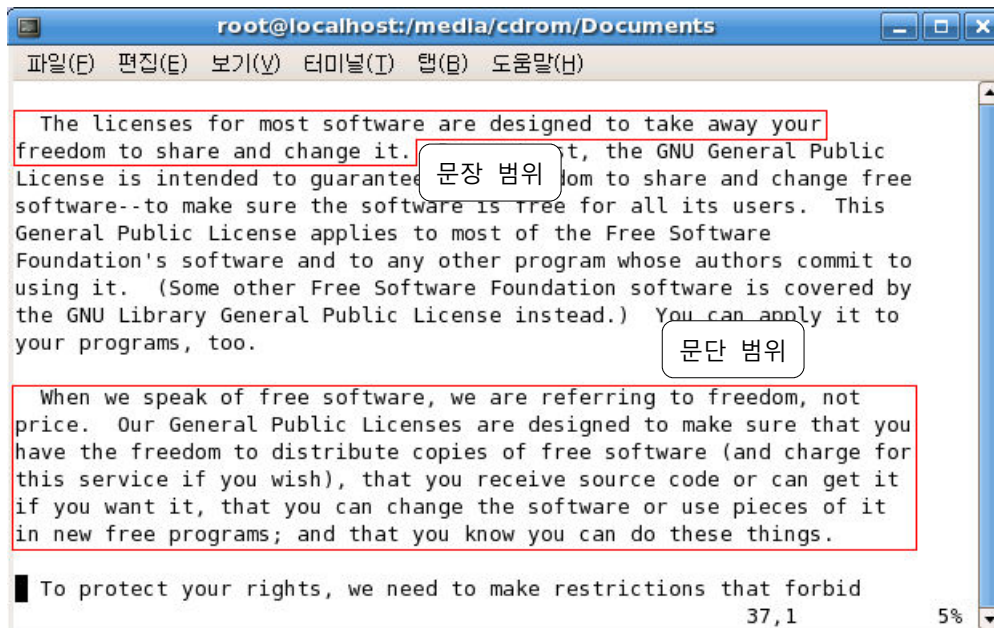


그림 61 vim에서 문장과 문단 범위

gg키는 문서의 맨 처음으로 이동할 때 사용되고 G 키는 문서의 마지막 위치로 이동할 때 이용된다. 지금까지 자주 사용되는 커서 이동키에 대한 설명이었다. 모든 커서 이동과 화면 이동에 관련된 키는 다음과 같다.

| | | | |
|----------|-----------------|----------|--------------------|
| h | 왼쪽으로 이동 | l | 오른쪽으로 이동 |
| j | 아래 행으로 이동 | k | 위 행으로 이동 |
| w or W | 다음 단어의 첫 글자로 이동 | b or B | 이전 단어의 첫 글자로 이동 |
| e or E | 단어의 마지막 글자로 이동 | <CR> | 다음 행의 첫 글자로 이동 |
| ^ | 그 행의 첫 글자로 이동 | \$ | 그 행의 마지막 글자로 이동 |
| + | 다음 행의 첫 글자로 이동 | - | 위 행의 첫 글자로 이동 |
| (| 이전 문장의 첫 글자로 이동 |) | 다음 문장의 첫 글자로 이동 |
| { | 이전 문단으로 이동 | } | 다음 문단으로 이동 |
| H | 커서를 화면의 맨 위로 이동 | z<CR> | 현재 행을 화면의 맨위로 이동 |
| M | 커서를 화면의 중앙으로 이동 | z. | 현재 행을 화면의 중앙으로 이동 |
| L | 커서를 화면 최하단으로 이동 | z- | 현재 행을 화면의 최하단으로 이동 |
| [n]H | 커서를 위에서 n행으로 이동 | [n]L | 커서를 아래에서 n행으로 이동 |
| Ctrl + u | 반 화면 위로 스크롤 | Ctrl + d | 반 화면 아래로 스크롤 |
| Ctrl + b | 한 화면 위로 스크롤 | Ctrl + f | 한 화면 아래로 스크롤 |
| gg, 1G | 문서의 맨 첫 행으로 이동 | G | 문서의 맨 마지막 행으로 이동 |
| [n]G | n행으로 이동 | : [n] | n행으로 이동 |

표 22 명령모드 이동키 종합 (<CR>은 vim내에서 [Enter]키의 의미, [n]은 임의의 번호)

굉장히 많은 키들로 인해 기가 질릴지도 모르겠다. 하지만 역시 다 익힐 필요는 없다. 사실 위에 나오는 키를 하나도 사용하지 않고 단지 화살표 키만으로 사용할 수도 있다. 지금은 h, j, k, l 키와 단어 단위의 이동키인 w와 b, 문단 단위의 이동키인 {와 } 그리고 행의 시작과 끝으로 이동하는 ^와 \$, 특정한 행으로 이동하는 :[n], 마지막으로 문서의 시작과 끝으로 이동하는 gg와 G 정도만 기억하기로 하자. 그리고 나머지는 이 책의 부록 부분에 있는 vim 명령어 요약을 컴퓨터 옆에 붙여놓고 필요할

때 하나씩 익히면 된다.

- 한 글자씩 이동 : h, j, k, l
- 단어 단위 이동 : w, b
- 문단 단위 이동 : { }
- 행의 시작과 끝 : ^, \$
- n행으로 이동 : :[n]
- 문서의 시작과 끝 : gg, G

5. 편집

가. 삭제

명령모드에서 x를 누르면 현재 커서 위치의 한 글자를 삭제한다. dw는 커서 위치의 한 단어를 삭제하고 dd는 한 행을 삭제한다. vi의 명령에는 규칙이 하나있는데 명령 앞에 숫자를 넣게 되면 그 명령을 앞에 누른 숫자 만큼 반복한다는 의미가 있다.

그래서 10x를 누르게 되면 현재의 커서부터 10개의 글자가 삭제된다. 마찬가지로 10dw하게 되면 10개의 단어가, 10dd하게 되면 10개의 행이 삭제된다.

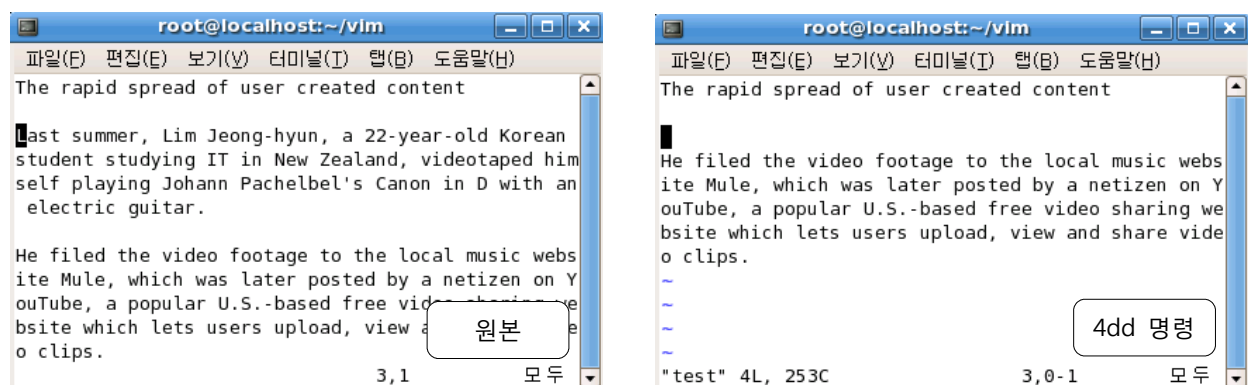


그림 62 4dd 명령 화면

주로 많이 사용하는 삭제 명령은 위에서 설명한 x, dw, dd 정도가 있고 기타 삭제와 관련된 다른 방법은 다음 표와 같다.

| | |
|--------|----------------------|
| x, dl | 커서 위치의 글자 삭제 |
| X, dh | 커서 바로 앞의 글자 삭제 |
| dw | 한 단어를 삭제 |
| d0 | 커서 위치부터 행의 처음까지 삭제 |
| D, d\$ | 커서 위치부터 행의 끝까지 삭제 |
| dd | 커서가 있는 행을 삭제 |
| dj | 커서가 있는 행과 그 다음 행을 삭제 |
| dk | 커서가 있는 행과 그 앞행을 삭제 |

표 23 삭제와 관련된 키

나. 복사 & 붙이기

yw는 현재 커서 위치의 한 단어를 복사한다. yy는 현재 커서 위치의 한 행을 복사한다. 이렇게 복사한 문자들은 p로 붙여 넣을 수 있다. 앞서 설명한 vi의 명령 규칙대로 10yw는 현재 커서 위치에서 10개의 단어를 복사하고, 10yy는 10개의 행을 복사한다. 마찬가지로 2p를 하게 되면 복사한 내용을 2번 붙여 넣게 된다.

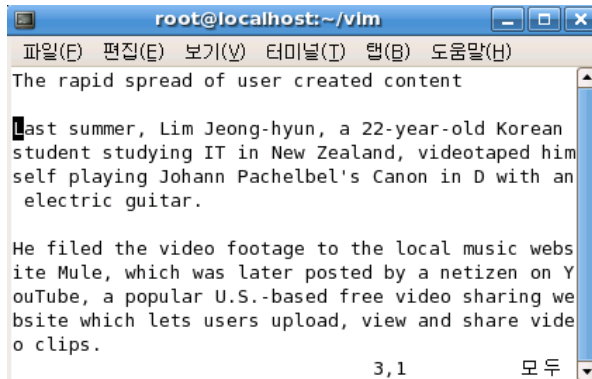
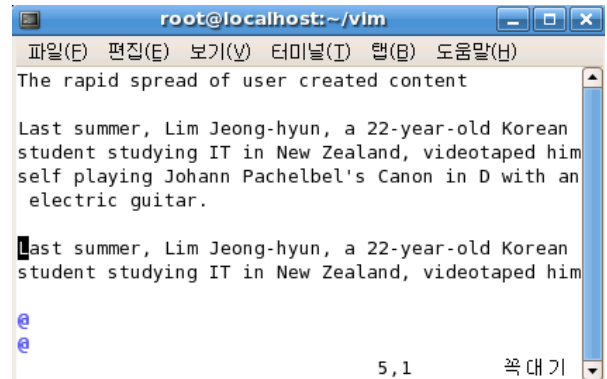


그림 63 22yy로 두행을 복사한 후 p로 붙여 넣기



다. 잘라내기

잘라내기에 대해서 이해하려면 먼저 vi의 레지스터에 대해서 알아야 한다. vi는 총 17개의 레지스터를 가지고 있는데 일단 dd 명령어로 한 행을 지웠다면 화면에 지워짐과 동시에 그 내용은 "1 레지스터에 들어가게 된다. 그리고 다시 dd 명령어로 한 행을 지우게 되면 "1 레지스터에 들어 있던 내용은 "2 레지스터로 옮겨가고 "1 레지스터에 새로 지운 내용이 들어가게 된다.

| "1 | "2 | "3 | "4 | "5 | "6 | "7 | "8 | "9 |
|-----|-----|-----|----|----|----|----|----|----|
| aaa | | | | | | | | |
| "1 | "2 | "3 | "4 | "5 | "6 | "7 | "8 | "9 |
| bbb | aaa | | | | | | | |
| "1 | "2 | "3 | "4 | "5 | "6 | "7 | "8 | "9 |
| ccc | bbb | aaa | | | | | | |

그림 64 레지스터에 들어가는 형식

이러한 레지스터는 ex모드에서 :reg 명령으로 볼 수 있다. 위 그림은 레지스터의 내용을 본 것이고 아래 그림은 레지스터의 이름을 나타낸 것이다.

| "" | "0 | "1 | "2 | "3 | "4 | "5 | "6 | "7 | "8 | "9 | "- | ". | :" | %" | "# | "/ |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

그림 65 레지스터 버퍼

"" 레지스터는 바로 이전에 지워진 내용이 항상 들어간다. "1부터 "9까지 레지스터는 지워지는 내용

이 큐 형식으로 들어가게 된다.

"1부터 "9 외에 나머지 레지스터는 특수한 목적이 있다. ". 레지스터에는 최근까지 타이핑한 내용이 들어간다. 그래서 아래 그림을 통해 최근에 'ABC'를 타이핑했다는 것을 알 수 있다. "% 레지스터 버퍼에는 현재 편집하는 파일명이 들어간다. 그리고 마지막으로 "/" 레지스터에는 가장 최근에 검색한 문자열이 들어간다.

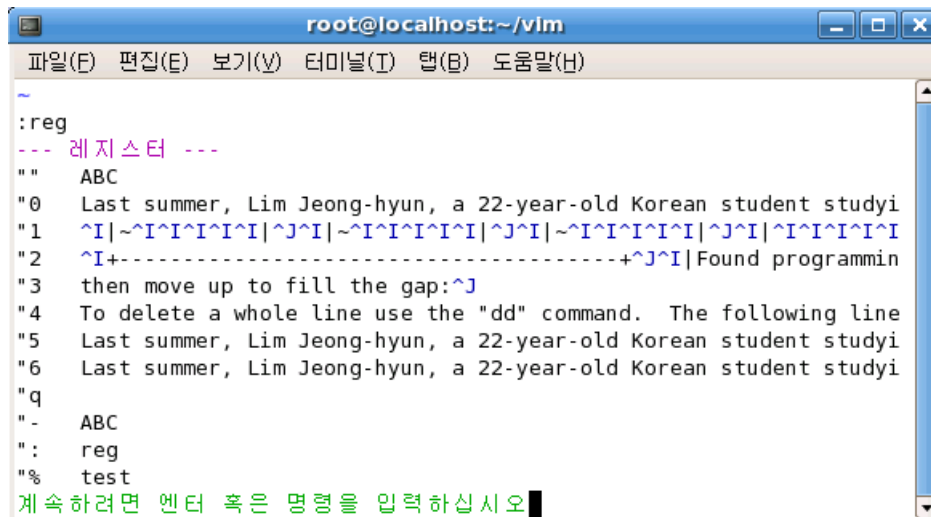


그림 66 :reg 명령어로 본 레지스터의 내용

바로 이전에 지워 레지스터에 들어있는 내용은 p 명령으로 붙여넣기 할 수 있다. 복사 역시 마찬가지다. yy를 눌러 한 행을 복사했다면 그 내용은 일단 레지스터에 들어가게 된다. 그리고 p 명령어를 통해서 레지스터에 있는 내용을 다시 출력할 수 있다. 그래서 vi에서 잘라내기란 일단 dw나 dd로 지우고 p 명령어로 붙여넣기 하면 되는 것이다.

p 명령으로 붙여넣기 할 때 특정 레지스터를 선택할 수 있다. "2p하게 되면 2번 레지스터에 있는 내용이 붙여 넣어진다. 만약 이전에 지우긴 했는데 그 내용이 어떤 레지스터에 있는지 모르겠다면 ex 모드에서 :reg 명령을 입력하여 확인할 수 있고 또 다른 방법으로는 "1pu.u.u. 명령으로 각 레지스터에 있는 내용을 하나씩 되돌리면서 붙여넣기 할 것을 선택할 수 있다. "1p는 1번 레지스터의 내용을 붙여는 명령이고 u는 되돌리기 명령이다. 그리고 .은 바로 이전에 했던 작업을 반복하는 명령이다.

라. 블록지정

지금까지 여러 단어 혹은 여러 행을 지우거나 복사하기 위해서 명령 앞에 숫자를 붙였다. 예를 들면 4개의 단어를 지우기 위해서는 5dw를 사용했고 5행을 복사하기 위해서는 5yy를 사용했다. 그러나 여러 단어나 행을 지우거나 복사하기 위해서 일일이 단어의 개수나 행 수를 세기는 매우 귀찮은 일일 것이다. 그럴 때는 블록을 지정하여 직접 개수를 세지 않고도 쉽게 원하는 작업을 수행할 수 있다.

다음 그림과 같이 원하는 위치에서 v를 누르고 j 키로 커서를 아래로 이동시키면 역상으로 블록이 형성되는 것을 볼 수 있다. 일단 원하는 범위만큼 블록을 지정했다면, 지우고 싶을 경우에는 d를 누

르면 지워진다. 또한 지정된 블록을 복사하고 싶은 경우 y를 누르면 지정된 블록에 대한 복사가 일어난다.



그림 67 블록지정 모습

만약 다음 그림과 같은 문자열에서 uid 부분만 지우고자 한다면 일반 블록지정으로는 지우기가 힘들다. 이때는 Ctrl + v 키를 사용한다. Ctrl 키를 누른 상태에서 v 키를 누르게 되면 커서의 위치를 기점으로 사각형의 블록을 형성할 수 있다. 원하는 만큼 지정한 후 삭제(d), 복사(y)를 하면 된다.

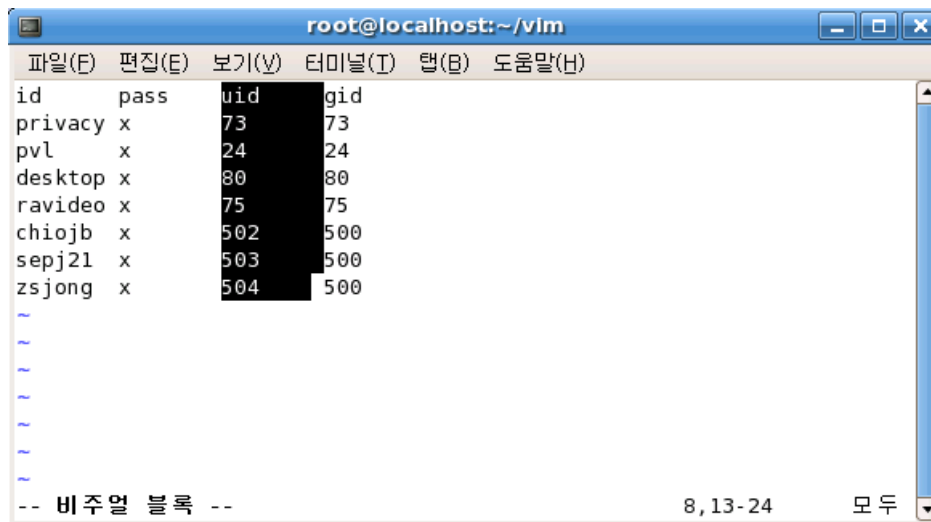


그림 68 블록지정 모습 2

| | | | |
|---|-------------------|---|-----------|
| ~ | 대소문자 전환 | d | 삭제 |
| y | 복사 | c | 치환 |
| > | 행 앞에 탭 삽입 | < | 행 앞에 탭 제거 |
| : | 선택된 영역에 대하여 ex 명령 | J | 행을 합침 |
| U | 대문자로 만듦 | u | 소문자로 만듦 |

표 24 블록지정 후 사용 가능한 키

6. 되돌리기와 되살리기

되돌리기와 되살리기 기능이 vim에 없었다면 vim은 정말 정신 건강에 해로운 에디터였을 것이다. 혹은 있다고 하더라도 기능을 사용할 줄 모른다면 정신 건강을 해치는 것은 마찬가지다. vim에서 되돌리기 기능은 u 명령을 사용한다. 그리고 되살리기는 Ctrl + r 키를 이용하면 된다. 원래 u 명령의 의미는 되돌리기가 아니라 명령 취소의 의미다. 그러나 결국 되돌리기가 되는 것은 사실이기 때문에 되돌리기 기능이라고 말할 수 있다.

7. 문자열 탐색

vim에서 특정한 문자열을 찾는 방법은 매우 쉽다. 그냥 단순히 한 문자열만 찾기 위해서는 아래와 같이 명령모드에서 '/' 키(역방향 탐색은 ? 키)를 누른 후 찾기 원하는 문자열을 입력하기만 하면 된다.

/[찾고자 하는 문자열] 또는 /[찾고자 하는 문자열]

다음은 vim에서 파일을 열어 'to'라는 문자열을 탐색한 예다.

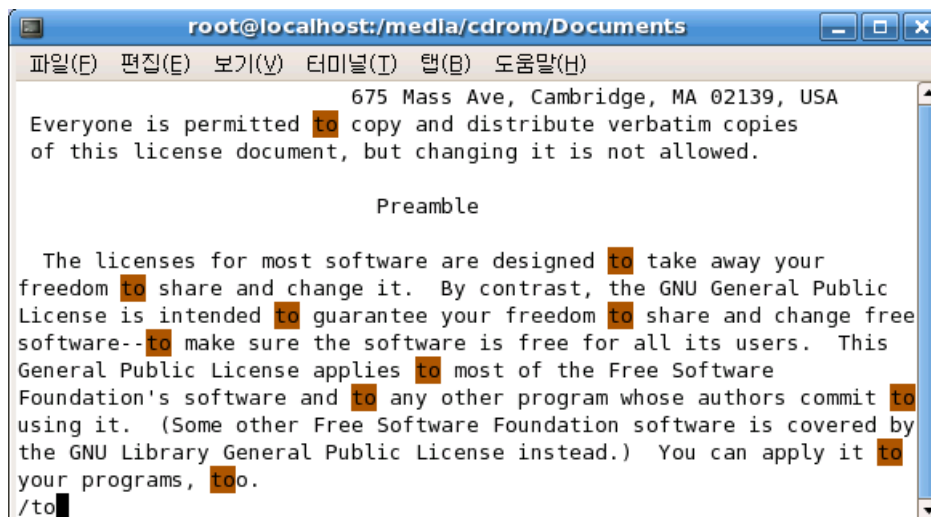


그림 69 문자열 탐색 모습

위 화면에서 n키를 누르면 커서가 다음에 매칭되는 'to' 문자열로 이동하고 대문자 N키를 누르면 이전에 매칭한 'to' 문자열로 이동한다.

간단한 문자열의 경우 위와 같은 /, ?를 사용한 명령형식으로 찾을 수 있다. 그러나 보다 복잡한 문자열의 경우 정규 표현식을 사용해야 한다.

8. 문자열 치환

문자열 치환 역시 매우 쉽다. ex모드에서 아래 명령을 내리면 문서에 있는 모든 old 문자열을 new로 바꾼다.

```
:%s/old/new/g
```

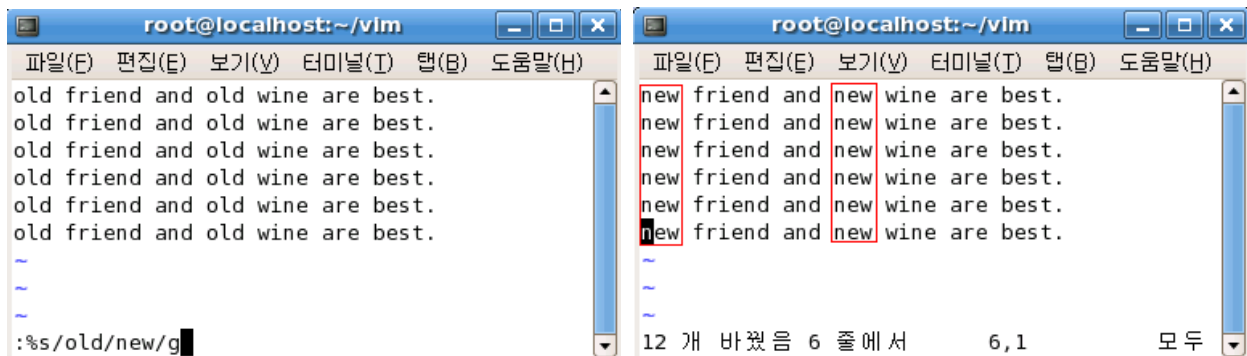


그림 70 치환 전과 치환 후

위 명령은 슬래시(/)를 구분자로 총 4개의 부분으로 나누어져 있다.

```
:[범위]/[매칭 문자열]/[치환 문자열]/[행 범위]
```

먼저 범위는 매칭 문자를 치환할 문서상의 범위를 의미한다. %s는 문서 전체를 의미한다. 그래서 :%s/old/new/g하게 되면 문서 전체에 있는 old 문자열이 new 문자열로 치환되는 것이다. 아래 예제 들은 범위 지정의 예다.

문서 전체에서 제일 처음 매칭되는 행의 문자열만 치환한다.

```
:s/old/new/g
```

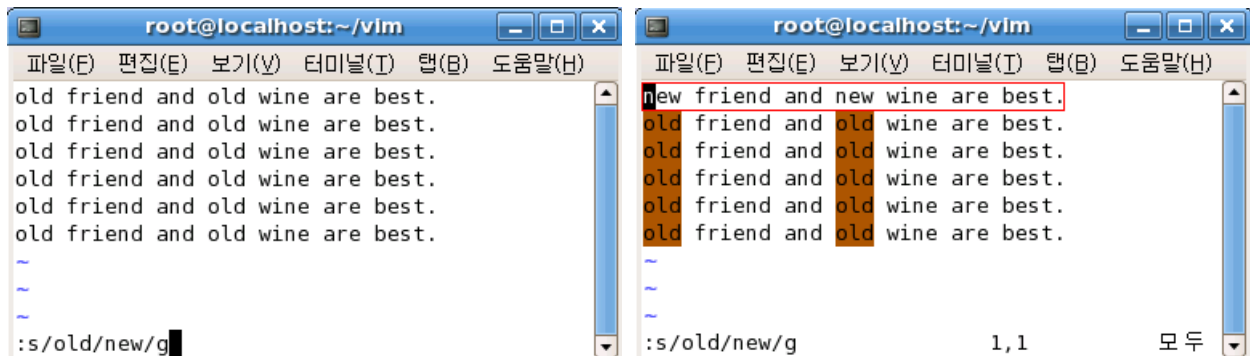


그림 71 치환 전과 치환 후

2행에서 4행 사이에 매칭되는 문자열을 치환한다.

```
:2,4s/old/new/g
```

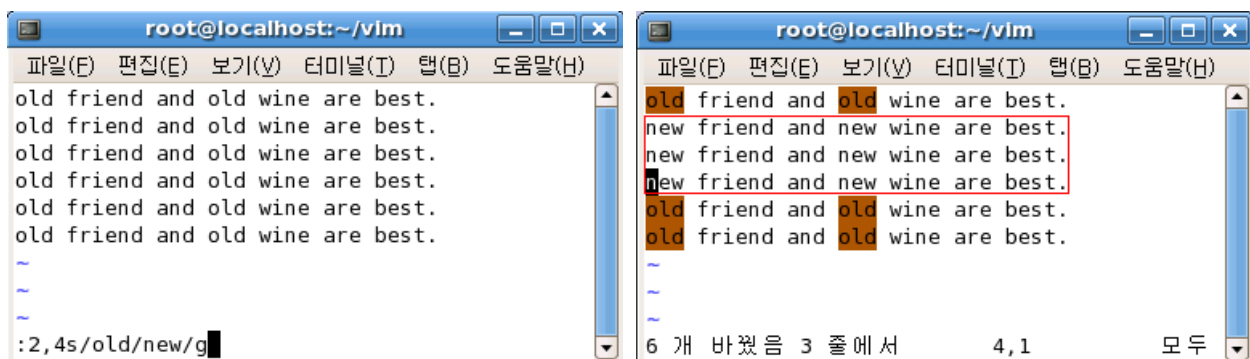


그림 72 치환 전과 치환 후

현재 커서의 위치에서 위로 1행 아래로 3행 되는 범위에 매칭되는 문자열을 치환한다.

```
:-1,+3s/old/new/g
```

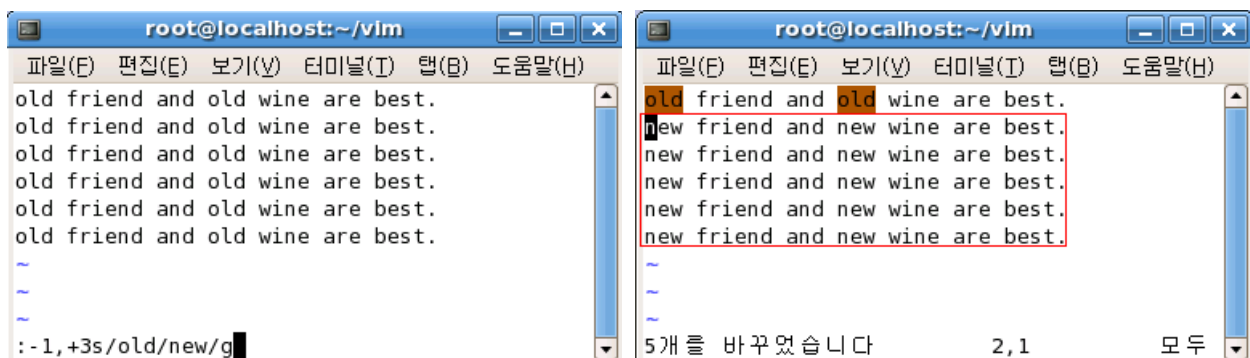


그림 73 치환 전과 치환 후

치환 문자열은 매칭 문자열을 만났을 때 치환할 문자열을 의미하고 마지막으로 행 범위는 치환될 행의 범위를 의미한다. g는 행 전체에 걸쳐서 치환하라는 의미고 만약 행 범위에 g를 쓰지 않고 명령을 내리면 한 행에서 여러 개의 매칭 문자열이 존재한다고 해도 한번만 치환이 일어나게 된다.

```
:%s/old/new
```

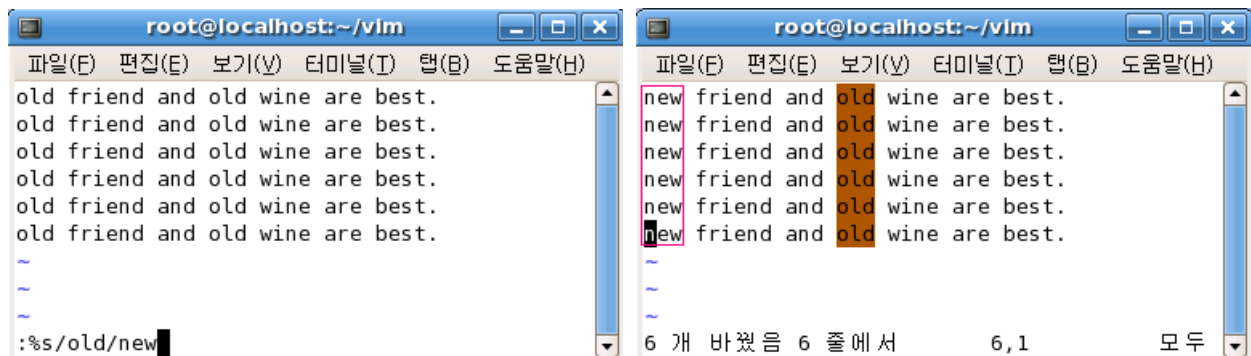


그림 74 치환 전과 치환 후

행 범위에 아래와 같이 c를 두게 되면 매칭되는 문자열에 대해 치환할 것인지 사용자에게 물어본다.

```
:%s/old/new/gc
```

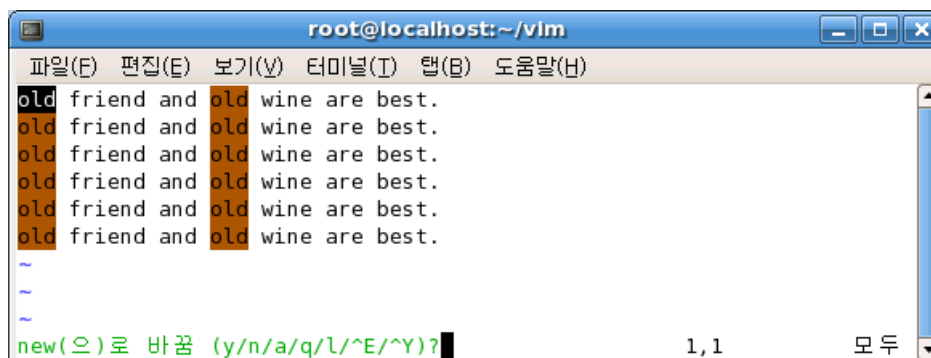


그림 75 치환 명령 질의

만약에 특정한 패턴이 있는 행에 대해서만 치환되게 하려면 다음과 같은 형식으로 사용한다.

```
:[범위]/[패턴]/[매칭 문자열]/[치환문자열]/[행 범위]
```

위 명령어는 매칭 문자에 대한 치환을 조절할 수 있기 때문에 유용하다. 예를 들면 다음 예제와 같

은 소스코드가 있을 때 주석 부분은 손대지 않고 실제 코드 부분만 test 변수를 str 변수로 전부 바꾸고자 한다면 이전에 배운 내용대로라면 주석의 test도 치환된다.

예제 1 테스트용 소스코드 내용

```
#include <stdio.h>

////////////////////
// test source code
////////////////////

int main()
{
    char test[] = "Network";
    printf( "%s", test );
    return 0;
}
```

그러나 아래처럼 명령을 내리면 주석 부분은 건드리지 않고 변수명을 바꿀 수 있다.

```
:g/\(^[^\n/].*test\|^test\)/s/test/str/g
```

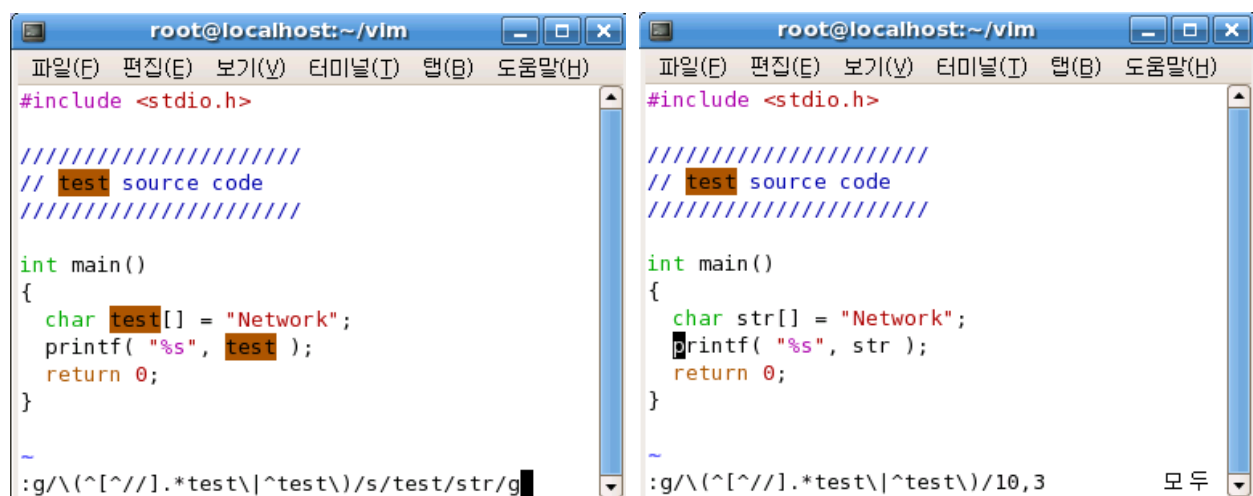


그림 76 치환 전과 치환 후

`\(^[^\n/].*test\|^test\)`는 정규 표현식¹⁾으로써 `//`으로 시작하지 않는 `test` 문자열과 `test`로 시작하는 문자열이라는 의미를 갖는다. 그래서 위 명령을 내리게 되면 `//`으로 시작하는 주석에 있는 `test`는 문자열 치환이 되지 않고 실제 소스코드 부분만 치환이 일어난 것을 볼 수 있다. 이러한 치환 명령을 요약해서 정리해보면 아래 표와 같다.

1) 정규 표현식(Regular Expressions)

참고문헌 : 정규 표현식 완전 해부와 실습- 제프리 프리들 저, 서환수 역, 한빛미디어

정규 표현식 기초 - <http://eldercrow.i-i.st/temptemp/regexp.htm>

| | |
|------------------------|--|
| :s/old/new | 현재 행의 처음 old를 new로 교체 |
| :s/old/new/g | 현재 행의 모든 old를 new로 교체 |
| :10,20s/old/new/g | 10행부터 20행까지 모든 old를 new로 교체 |
| :-3,+4s/old/new/g | 현재 커서 위치에서 3행 위부터 4행 아래까지 old를 new로 교체 |
| :%s/old/new/g | 문서 전체에서 old를 new로 교체 |
| :%s/old/new/gc | 문서 전체에서 old를 new로 확인하며 교체 |
| :g/pattern/s/old/new/g | pattern이 있는 모든 행의 old를 new로 교체 |
| :/pattern/s//new/g | :%s/old/new/g와 동일 |

표 25 치환 명령 요약

SECTION

03 vim을 강력하게 하는 고급 테크닉

지금까지는 에디터로써 기본적인 기능을 설명했다. 지금까지의 내용만으로는 vim이 다른 에디터에 비해 많이 다르다는 것은 알 수 있었겠지만 어째서 vim이 다른 에디터와 비교해서 강력한지는 알 수 없었을 것이다. vim의 색다른 기능들과 특징을 알아본다.

1. 여러 파일을 편집하는 방법

보통 한 프로젝트의 소스 파일은 하나의 파일로 이루어져 있지 않다. 코딩을 하다보면 이 파일에서 저 파일로 옮겨가면서 수정해야 되는 일이 대부분이다. 보통 GUI 에디터는 여러 파일의 편집을 지원하기 위해서 여러 탭으로 구분하여 파일을 편집하는 형식으로 이루어져 있다. 지금 수정하는 파일에서 다른 파일을 수정하기 위해 마우스나 단축키를 사용해서 다른 파일의 탭을 열고 수정하면 된다. vim도 여러 파일을 편집하는 모드를 지원한다. 여러 파일을 편집하는데 있어서 vim 만큼 편리한 에디터도 드물다. 그렇지만 vim의 기본 설정 상태에서는 편집하기가 다소 불편하다. 여러 파일을 편리하게 편집하기 위해서는 몇 가지 설정을 해야 한다.

먼저 여러 파일을 여는 방법을 알아보자. 다음 그림과 같이 현재 작업 디렉토리에 file1.c, file2.c, file3.c 세 개의 파일이 있다. 각 파일이 프로젝트에서 각 부분을 담당한다고 생각하면 된다.

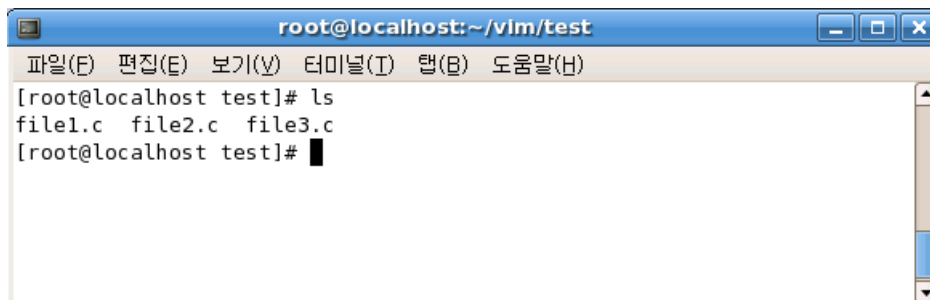


그림 77 현재 디렉토리의 파일

이렇게 여러 파일들이 있을 때 한꺼번에 여러 파일을 여는 방법은 아래 명령어로 가능하다.

```
$vi file1.c file2.c file3.c
```

위 명령어와 같이 여러 파일명을 일일이 지정해도 되고 아래와 같이 셸 메타 기호를 사용해서 한꺼번에 지정해도 된다.


```
$vi * 또는 vi *.c
```

위 명령을 내리면 다음 그림과 같이 file1.c 파일이 열린 것을 확인할 수 있다.

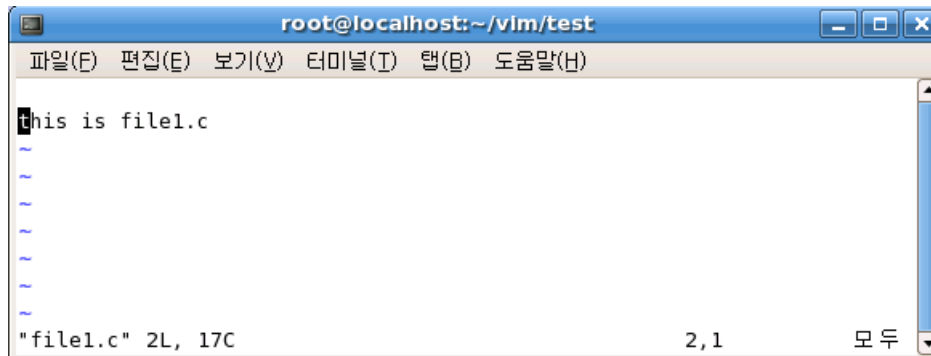


그림 78 file1.c의 내용

보기에는 file1.c 파일 밖에 열리지 않은 것으로 보인다. 그렇지만 vim 내부적으로는 세 개의 파일이 다 열려 있는 상태다. 열려 있는 각 파일은 vim의 파일 버퍼에 들어 있게 된다. 각 파일 버퍼에 들어 있는 내용은 :ls 명령어로 확인할 수 있다. 다음 그림과 같이 file1.c 파일이 열려 있는 화면을 파일 버퍼라고 부르기도 한다. 그럼 파일 버퍼 간 전환을 해보자.

```
:b2
```

ex모드에서 위 명령을 내리게 되면 열려있는 2번 파일로 전환한다. :b2라는 명령어의 의미는 '파일 버퍼 2번'의 의미다.

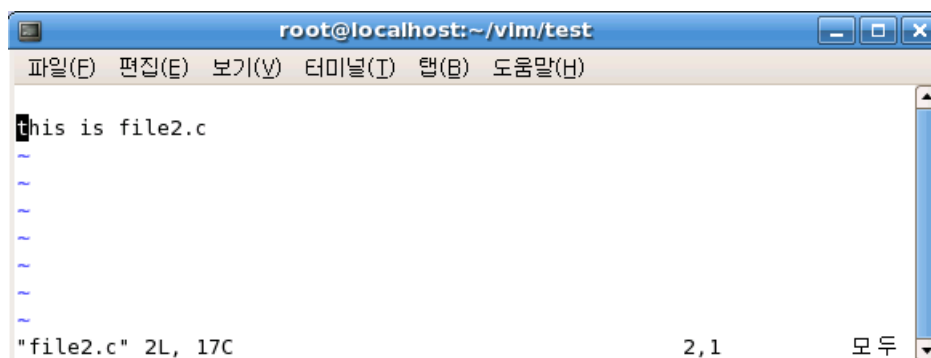


그림 79 file2.c의 내용

마찬가지로 3번 파일 버퍼로 전환하기 위해서는 :b3이라고 명령을 내리면 된다. 열려 있는 파일을 닫는 명령은 :bw다. :bw하게 되면 현재의 버퍼를 닫게 된다. 버퍼에 관련된 명령은 다음과 같다.

| | |
|---------------|------------------------------------|
| :buffers | 버퍼의 내용을 나열 |
| :files 또는 :ls | 버퍼의 내용을 나열 |
| :b[n] | n번 버퍼로 이동 |
| :bd[n] | n번 버퍼를 삭제(n이 없으면 현재의 버퍼를 완전 삭제) |
| :bw[n] | n번 버퍼를 완전 삭제(n이 없으면 현재의 버퍼를 완전 삭제) |
| :bp[n] | 이전 버퍼로 이동. n을 붙이면 n번만큼 이전 버퍼로 이동 |
| :bn[n] | 다음 버퍼로 이동. n을 붙이면 n번만큼 다음 버퍼로 이동 |
| :sb[n] | 창을 수평 분할하여 n번 버퍼를 로드 |
| :bf | 첫 번째 버퍼로 이동 |
| :bl | 마지막 버퍼로 이동 |
| :al | 현재 열려 있는 모든 버퍼를 수평 분할 창에 로드 |

표 26 파일 버퍼 관련 명령 요약

위에서 사용한 방법으로 파일을 전환하는 것은 코딩 과정 내내 빈번하게 발생하는 파일 전환을 고려해볼 때 다소 불편한 면이 있다. 이제 vim의 설정을 변경하여 보다 편지하게 단축키를 지정해놓자.

```
:map ,1 :b!1<CR>
:map ,2 :b!2<CR>
:map ,3 :b!3<CR>
```

map 명령은 단축키를 매칭(mapping)할 때 사용된다. 위 명령을 ex모드에서 차례로 내린 후 명령모드에서 1 키를 누르게 되면 1번 버퍼로 전환하고, 2를 누르게 되면 2번 버퍼로 전환한다. 이때 타이밍이 중요하다. , 키를 누르고 댄 후 숫자 키를 눌러야 하는데 , 키를 누른 후 한참 있다가 1번을 누르게 되면 정상적인 명령어로 인식하지 않는다.

b1이라고 하지 않고 b!1이라고 한 이유는 만약 2번 버퍼에 편집 작업을 수행한 후 파일을 저장하지 않고는 1번 버퍼로 이동하려고 할 때, vim이 파일이 편집되었는데 저장하지 않았다고 경고를 발생시키는데, 이를 무시하고 강제로 버퍼를 전환하기 위해서다. <CR>은 vim 내에서 [Enter]키의 의미가 있다. 현재 vim에 매칭되어 있는 모든 매핑 리스트는 단순히 ex모드에서 :map이라고 입력하면 화면에 출력된다.

이제 파일 버퍼를 전환할 때는 ,[숫자]를 누르면 된다. 그런데 매번 위와 같이 :map 명령어로 키를 매핑해야 한다면 정말 불편한 일이 아닐 수 없다. 보다 편리하게 하기 위해서는 vim 설정 파일에 map 명령을 등록해야 한다.

가. vim 설정 파일에 키 매핑 추가

사용자의 홈 디렉토리에 .vimrc 파일을 생성하게 되면 vim은 실행할 때 ~/.vimrc 파일을 읽어서 실행하게 된다. .vimrc 파일은 vim 설정 파일로써 vim이 실행될 때 ~/.vimrc 파일에 적힌 명령을 차례로 수행하게 되어 있다.

~/.vimrc 파일을 열거나 생성하여 다음 내용을 추가해보자.

예제 2 ~/.vimrc 파일 내용

```
map ,1 :b!1<CR>
map ,2 :b!2<CR>
map ,3 :b!3<CR>
map ,4 :b!4<CR>
map ,5 :b!5<CR>
map ,6 :b!6<CR>
map ,7 :b!7<CR>
map ,8 :b!8<CR>
map ,9 :b!9<CR>
map ,0 :b!0<CR>
map ,w :bw<CR>
```

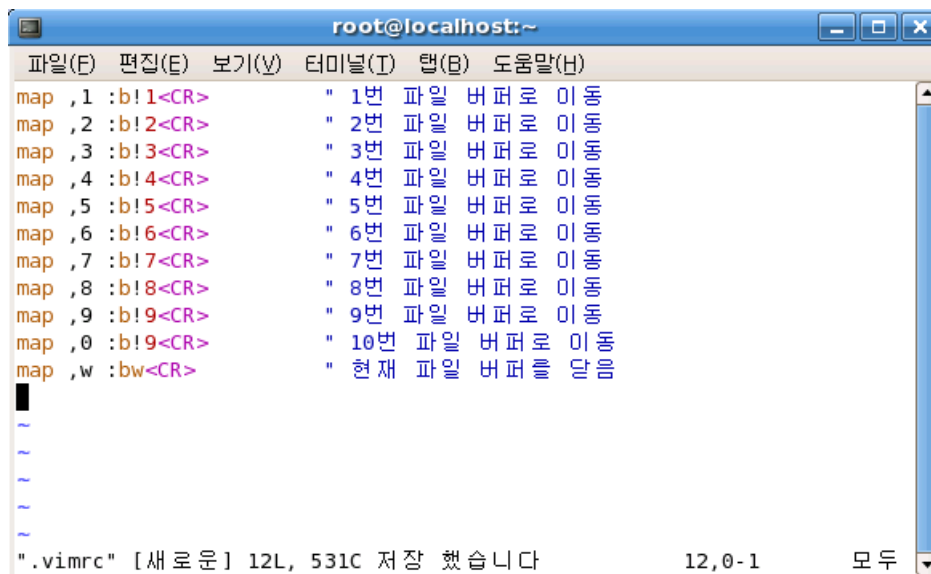


그림 80 ~/.vimrc 파일의 내용

그리고 다시 여러 파일을 열었다면 명령모드에서 ,1 또는 ,2와 같은 단축키로 파일 버퍼를 전환할 수 있다. 지금 설정에서는 10개의 파일 버퍼를 단축키로 지정했다. 그러나 vim에서 사용 가능한 파일 버퍼는 이보다 훨씬 많다.(1,000 단위가 넘어갈 정도다.) 10개 이상의 파일 버퍼를 지정하고 싶다면 추가적으로 지정하면 된다. map 명령어로 단축키를 매핑할 때는 주의할 점이 있는데, map 명령어로 매핑할 때는 명령모드에서 사용하지 않는 단축키를 사용해야 한다는 것이다. 위 .vimrc 파일에서 "(큰 따옴표)는 주석을 의미한다.

2. 반복되는 문자열을 저장해서 쓰기

이전에 잘라내기에서 레지스터에 대해 잠깐 언급했다. 이전에 설명한 17개의 레지스터 외에 vim은 a부터 z까지 26개의 레지스터를 더 지원한다. 이러한 레지스터를 네임 레지스터라고 하는데 이 레지스터는 반복되는 문자열을 저장해서 쓰기에 편리하다.

가령 C 코딩을 하다보면 다음 예제와 같이 printf로 디버깅 메시지를 찍어보는 경우가 빈번하다.

예제 3 printf로 디버깅 메시지를 찍어보는 예제 소스

```
int main()
{
    int i;
    i = 2;

#ifdef DEBUG
    printf( "DEBUG : i = %d\n", i );
#endif

    return 0;
}
```

이러한 디버깅 메시지를 네임 레지스터에 저장하고 사용하면 된다. 저장하는 명령은 명령모드에서 아래 같이 하면 된다.

"a3yy

"a는 a레지스터에 저장하라는 의미고 3yy는 현재 커서 위치에서 세 행을 복사해서 저장하라는 의미다. 이렇게 명령을 내리면 현재의 커서 위치에서 아래로 3행을 복사하여 a 레지스터에 저장하게 된다. 저장된 내용을 붙여넣는 방법은 원하는 위치에서 아래의 명령을 내리면 된다.

"ap

"a로 a 레지스터를 지정하고 p로 붙여넣기 했다.

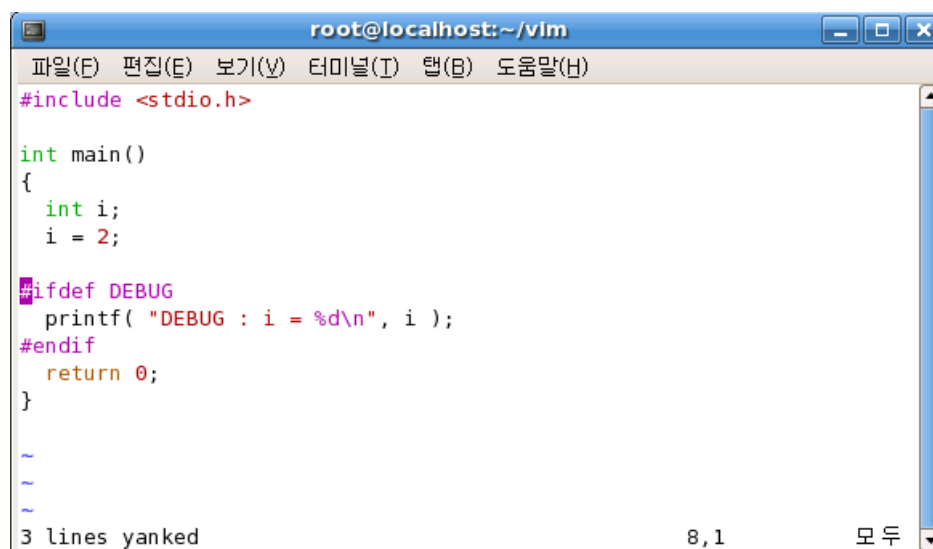


그림 81 "a3yy 명령을 내린 화면

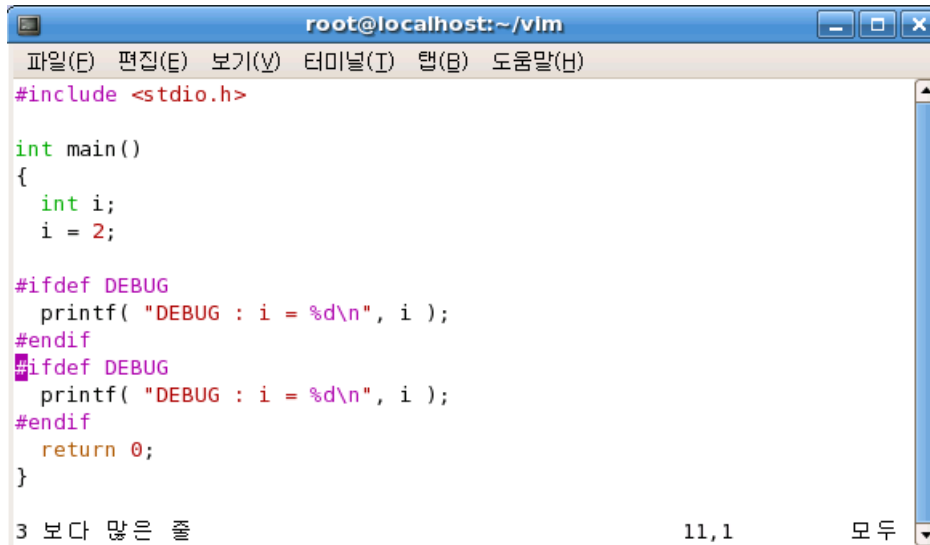


그림 82 "ap" 명령을 내린 화면

a부터 z까지 총 26개의 네임 레지스터를 지원하므로 넉넉하게 원하는 문자열을 저장해서 사용할 수 있다.

3. 매크로의 사용

vim에서 매크로란 키 입력을 네임 레지스트에 저장해서 사용하는 방법을 말한다. 이해하기 쉽게 예제를 보면서 설명하겠다.

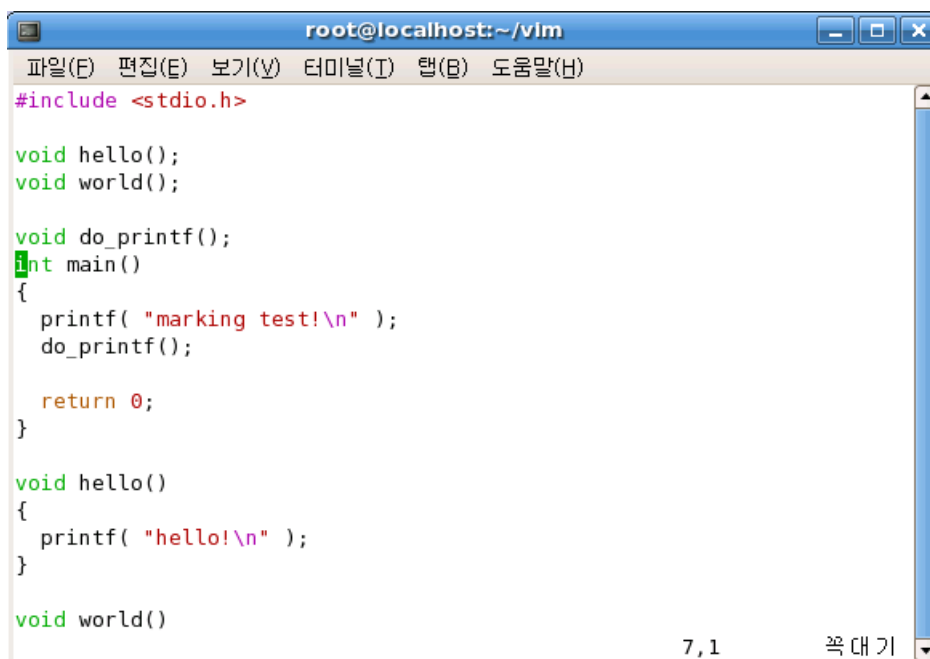


그림 83 매크로 사용 설명을 위한 예제 소스

가령 위 그림과 같은 소스 파일이 있다고 가정했을 때 현재 커서가 있는 행에 주석을 넣기 위해서는 키 입력으로 `^i // [SpaceBar] [Esc] [Enter]` 키를 순차적으로 누르면 된다. `^` 기호는 커서를 현재 행의 처음으로 이동시키는 명령이다. `i` 키를 사용하여 입력모드로 전환한 후 `//`와 공백을 입력한다. 그리고 `[Esc]`키를 사용하여 명령모드로 전환한 후 `[Enter]`키를 눌러 다음 행의 처음으로 커서를 이동하는 것이다.

이러한 키 입력을 저장해서 필요할 때 사용하는 기능이 있다면 반복적으로 키를 눌러야 하는 작업에 있어서 편리할 것이다. 이런 경우 사용하는 기능이 바로 매크로 기능이다. 매크로를 사용하는 방법은 다음과 같다.

먼저 `q[네임 레지스터명]`을 눌러 매크로를 등록할 레지스터를 지정한다. 가령 `b` 레지스터를 사용하고자 한다면 `qb`라고 입력하면 된다.

qb

`qb`라고 입력하면 화면 하단에 '기록 중'이란 글자가 나온다. 이후에 누르는 키들은 `b` 레지스터에 저장되게 된다.

`^ i // [SpaceBar] [Esc] [Enter]`키를 차례로 누른다. 그리고 `q`를 누르면 화면 하단에 '기록 중'이란 메시지가 사라지면서 `b` 레지스터에 이제까지 누른 키들이 등록된다.

이제 `b` 레지스터에 등록된 매크로는 `@b`를 누름으로써 사용할 수 있다. `@b`를 누르게 되면 `^ i // [SpaceBar] [Esc] [Enter]`키를 순차적으로 누르는 것과 같은 결과를 가져오게 된다. 7@b하게 되면 `@b`를 일곱 번 사용한 결과가 나타난다.

7@b

다음 그림은 `b` 레지스터에 `^ i // [SpaceBar] [Esc] [Enter]` 키를 등록하여 7@b 명령어로 일곱 번 반복했을 때의 결과다.

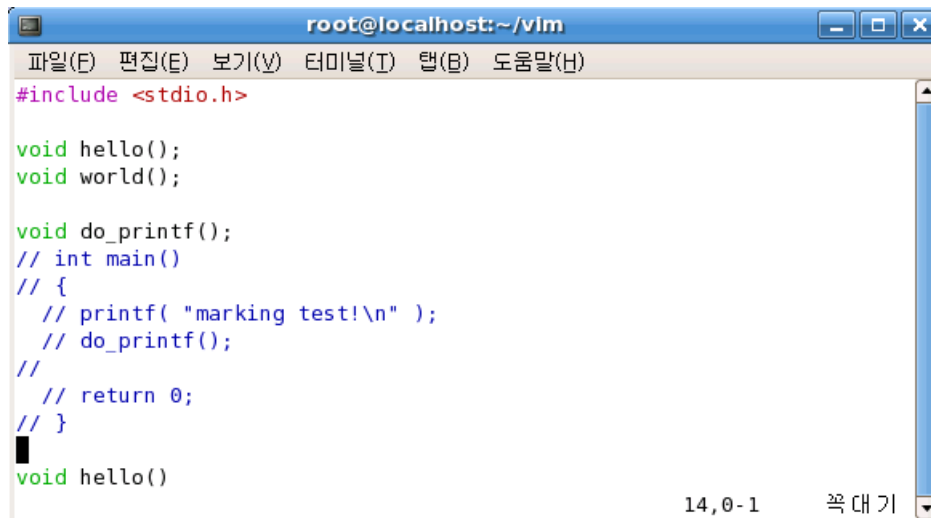


그림 84 7@b 명령 후

마찬가지로 행 주석을 없애는 매크로를 c 레지스터에 저장하기 위해서는 qc를 입력해서 매크로를 저장할 레지스터를 지정한 후 ^3x [Enter]키를 입력한 후 q를 누르면 된다.

^ 기호는 행의 처음을 의미하고 3x를 사용하여 현재 커서의 위치에서 세 글자를 지운 후 [Enter]키를 눌러 다음 행의 처음으로 이동한다는 의미다.

4. 다중 창 사용하기

프로그래밍을 하거나 소스를 분석하다보면 다른 파일에 작성된 구조체의 타입이나 소스를 참조해야 되는 경우가 많이 발생한다. 이런 경우 다중 창 기능을 사용하면 유용하다. 다중 창 기능이란 vim의 화면을 여러 개로 분할하여 사용하는 방법을 말한다.

명령모드에서 Ctrl + w n(Ctrl 키를 누른 상태에서 w를 누르고 Ctrl 키를 떼고 n키를 누른다. n은 new의 의미) 키를 누르면 vim 창이 다음 그림과 같이 가로로 분할되는 것을 볼 수 있다.

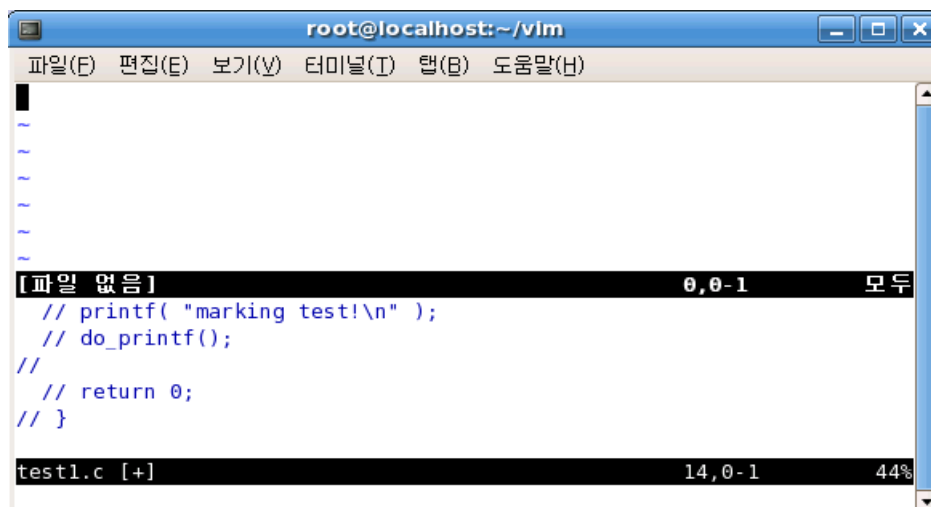


그림 85 Ctrl + w n 입력 후 모습

이렇게 창이 분할되면 :e [파일명] 명령어로 파일을 불러올 수 있다. 다음 그림은 파일을 불러오는 모습을 나타내었다.

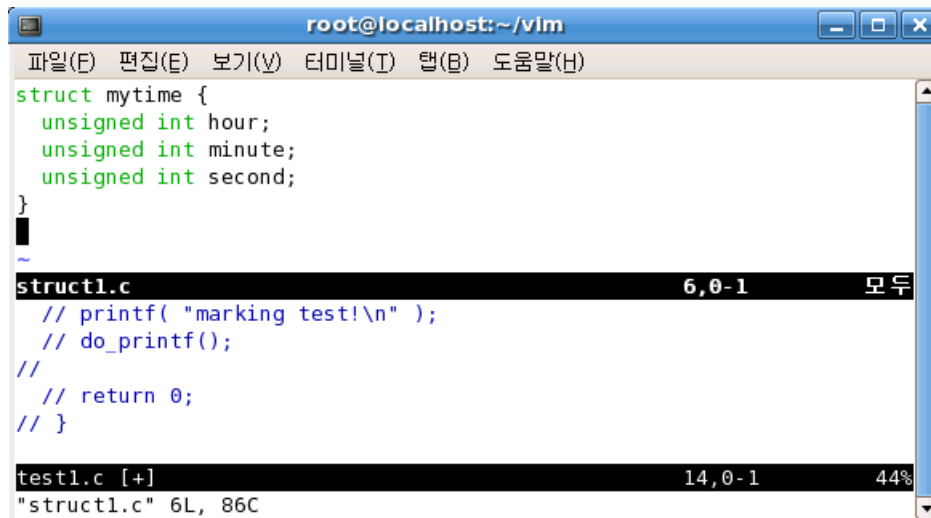


그림 86 :e struct1.c 명령어로 새로 열린 창에 struct1.c 파일을 불러온 모습

창간 커서 전환은 Ctrl + ww(Ctrl을 누른 상태에서 w를 두 번 누름) 키로 할 수 있다. 반복해서 Ctrl + w n 키를 누르면 창을 계속 분할할 수 있다. 그리고 이렇게 분할한 창은 Ctrl + w q(quit) 키를 누르거나 ex모드에서 :q!를 입력함으로써 닫을 수 있다.

Ctrl + w s 키는 현재의 파일을 가지고 분할한다. 무슨 말이나 하면 현재 file1.c 파일이 열려 있는 창에서 Ctrl + w s(split) 키를 누르게 되면 창이 수평 분할되면서 file1.c의 내용이 분할된 창에 들어가게 된다. Ctrl + w v(vertical split) 키를 누르게 되면 창이 수직 분할한다. 다음 그림은 Ctrl + w v 키로 창을 수직 분할한 예다.

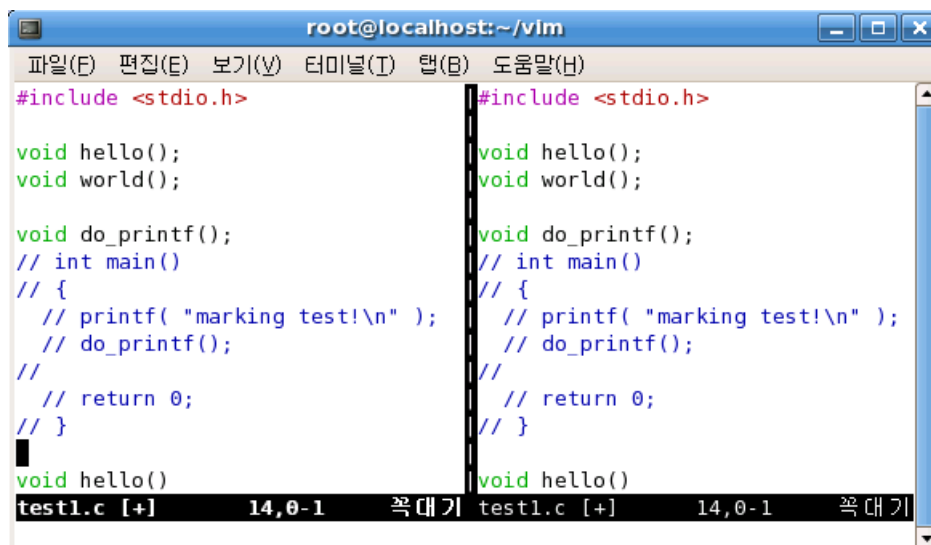


그림 87 Ctrl + w v 키로 수직 분할한 모습

수직분할 역시 창간 전환은 Ctrl + ww 키로 하며 주로 두 파일을 비교할 때 사용된다. 분할한 창의 사이즈를 조절하고 싶을 때가 있다. 가령 헤더 파일에 구조체를 참고하다가 당장은 필요가 없어졌지만 빈번하게 참고해야 할 때가 있을 것이다. 이런 경우 창의 크기를 조정하면 된다. 수정 중인 소스 파일의 창은 최대로 하고 헤더 파일의 창은 최소로 만들면 되는 것이다. 현재 작업 중인 창을 최대로 하는 명령은 수평 분할일 경우 Ctrl + w _ 키고 수직 분할일 경우 Ctrl + w | 키다.

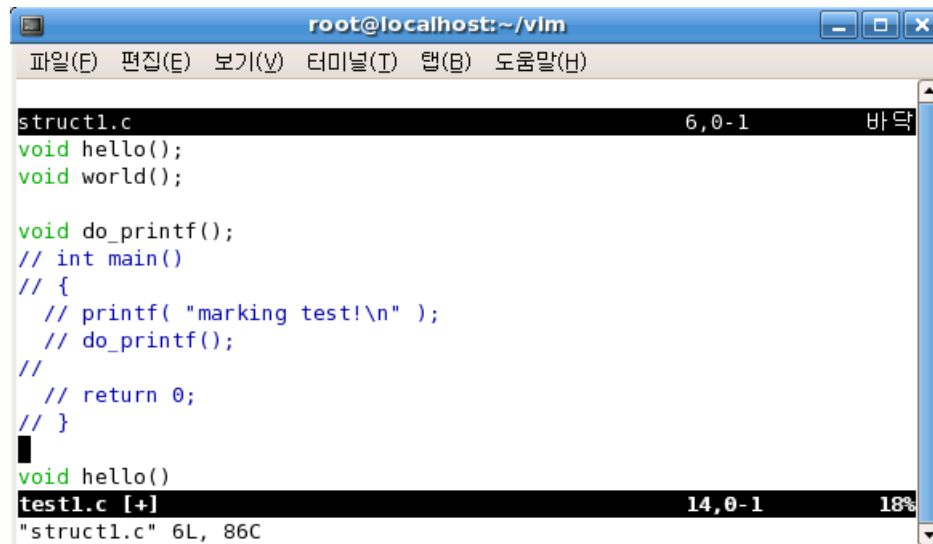


그림 88 Ctrl + w _ 키로 최대화한 모습

위와 같이 최대화한 창을 원래대로 균등하게 분할된 상태로 만들고 싶을 때는 Ctrl + w = 키를 사용하면 된다. 최대화되었던 창이 원래의 균등 분할된 상태로 돌아온다.

다음은 이러한 창 분할과 관련된 명령들을 요약해 놓은 표다.

| 명령모드 | ex모드 | 결과 |
|-----------------|-------------|---------------------------------|
| 창 생성 | | |
| Ctrl + w s | :sp[plit] | 현재 파일을 두 개의 수평 창으로 나눔 |
| Ctrl + w v | :n]vs[plit] | 새로운 수직 창 생성, n이 붙으면 n칸 크기의 창 분할 |
| Ctrl + w n | :new | 새로운 수평 창 생성 |
| Ctrl + w ^ | | 수평 창으로 나누고 이전 파일 오픈 |
| Ctrl + w f | | 창을 수평으로 나누고 커서 위치의 파일을 오픈 |
| Ctrl + w i | | 커서 위치의 단어가 정의된 파일을 오픈 |
| 창 삭제 | | |
| Ctrl + w q | :q[uit]! | 현재 커서의 창을 종료 |
| Ctrl + w c | :clo[se] | 현재 커서의 창을 닫기 |
| Ctrl + w o | :on[ly] | 현재 커서의 창만 남기고 모든 창 삭제 |
| 창 위치 바꾸기 | | |
| Ctrl + wr | | 순차적으로 창의 위치를 순환 |
| Ctrl + wx | | 이전 창과 위치를 바꿈 |

| 창 이동 | | |
|-----------|--|--------------------|
| Ctrl + wh | | 왼쪽 창으로 커서 이동 |
| Ctrl + wj | | 아래쪽 창으로 커서 이동 |
| Ctrl + wk | | 위쪽 창으로 커서 이동 |
| Ctrl + wl | | 오른쪽 창으로 커서 이동 |
| Ctrl + ww | | 창을 순차적으로 이동 |
| Ctrl + wp | | 가장 최근에 이동한 방향으로 이동 |
| Ctrl + wt | | 최상위 창으로 이동 |
| Ctrl + wb | | 최하위 창으로 이동 |

| 창 크기 조정 | | |
|---------------|--|--------------------------------------|
| Ctrl + w = | | 창의 크기를 모두 균등하게 함 |
| Ctrl + w _ | | 수평 분할에서 창의 크기를 최대화 |
| Ctrl + w | | 수직 분할에서 창의 크기를 최대화 |
| Ctrl + w [n]+ | | 창의 크기를 n행만큼 증가 (Ctrl + w +는 1행만큼 증가) |
| Ctrl + w [n]- | | 창의 크기를 n행만큼 감소 (Ctrl + w -는 1행만큼 감소) |
| Ctrl + w [n]> | | 창의 크기를 오른쪽으로 n칸만큼 증가 (수직 창에 한함) |
| Ctrl + w [n]< | | 창의 크기를 왼쪽으로 n칸만큼 증가 (수직 창에 한함) |

표 27 다중 창 명령 요약 (ex모드에서 []내에 문자는 생략 가능함을 의미)

위 명령들을 자세히 살펴보면 규칙성이 있다는 것을 발견할 수 있다. 일단 Ctrl + w를 누르게 되어 있다. 그리고 창 생성의 경우 분할(split), 새 창(new), 수직 분할(vertical split)의 영문자 첫 글자로 이루어져 있다. 창 삭제의 경우도 마찬가지다.

일단 Ctrl + w를 누른다. 그리고 종료(quit), 닫기(close), 단일(only)의 영문자 첫 자를 누르면 된다. 창 이동의 경우 기본 이동키인 h, j, k, l 키와 연관이 있는 것을 알 수 있고(h, j, k, l 키 대신 화살표 키를 사용해도 된다.) 최상위(top), 최하위(bottom) 등도 첫 자로 이루어진 것을 알 수 있다.

이러한 일관성만 파악한다면 아무리 많은 창 관련 명령도 외우기 쉬울 것이다. 그리고 항상 말하듯이 vi는 배우는 것이 아니라 익히는 것이기 때문에 이해되었다고 해서 따라해보지 않으면 안된다. 몸이 기억할 수 있게 여러 번 반복해서 사용해야 한다.

5. 마킹으로 이동하기

지금까지 vim의 기본적인 커서 이동에 관해서 알아보았다. 이제는 조금 더 고차원적인 마킹으로 이동하는 방법에 대해서 알아보자. 마킹은 의미대로 커서의 특정 위치를 문자로 마킹하여 나중에 한번에 표시한 위치로 돌아가기 위한 방법이다. 이러한 마킹 기법은 소스가 매우 긴 파일이거나 여러 곳을 동시에 수정할 때 많이 사용된다. 그리고 특히 커널과 같이 거대한 프로젝트 소트를 분석할 때 사용하면 유용하다. 마킹은 다음 명령어로 가능하다.

m[임의의 알파벳]: 마킹할 때
'[마킹한 알파벳]: 마킹한 위치로 돌아갈 때

마킹에는 세 가지 종류가 있다. A-Z까지 대문자로 마킹한 전역 마킹과 a-z까지 소문자로 마킹한 지역 마킹 그리고 ~/.viminfo 파일에서 자동으로 지정한 0-9로 이루어진 파일 마킹이 그것이다. 알파벳으로 마킹하는 전역, 지역 마킹은 언제든지 사용 가능하나 숫자로 이루어진 파일 마킹은 vim이 실행될 때나 혹은 종료될 때 자동으로 지정되기 때문에 사용할 수 없다. 파일 마킹은 이전에 vim으로 편집한 파일의 경로와 파일 내에 위치를 vim이 알아서 마킹하게 된다. 전역 마킹과 지역 마킹의 차이점은 다음과 같다.

전역 마킹(A-Z) : 현재 파일을 포함한 다른 파일간의 마킹이 가능하다.

지역 마킹(a-z) : 현재 파일 내에서만 마킹이 가능하다.

즉, 대문자로 마킹하게 되면 파일이 틀리더라도 마킹한 위치로 이동할 수 있지만 소문자로 마킹하게 되면 현재 파일 내에서만 마킹이 유효하게 된다.

위 그림과 같은 소스가 있을 때 main() 함수에 do_print() 함수가 사용되었는데 이전에 do_print() 함수의 원형이 선언되지 않았다. 그래서 위 소스는 컴파일되긴 하나 컴파일할 때 경고 메시지가 발생하게 된다.

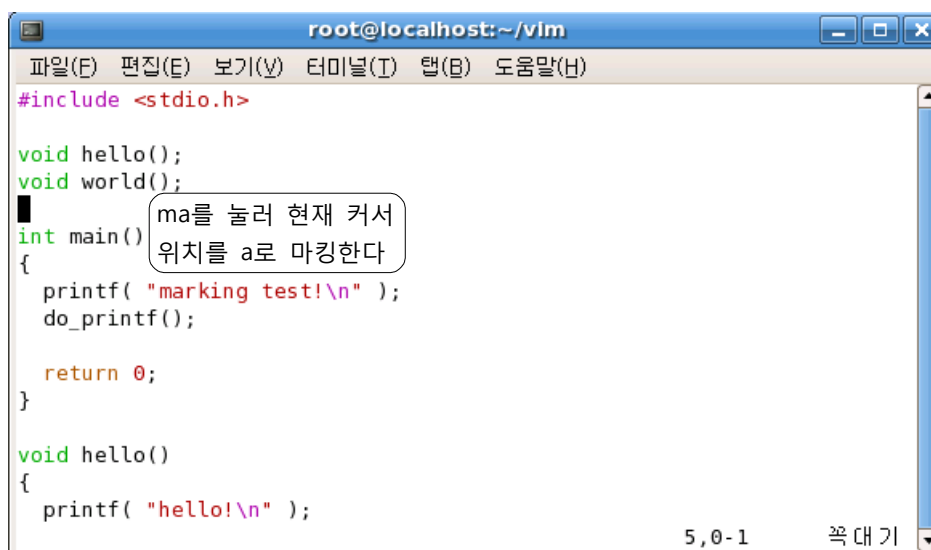


그림 89 마킹 사용법 예제 1

경고 메시지는 결코 달가운 것이 아니기에 main() 함수 앞에 do_print() 함수의 원형을 선언하여 경고 메시지를 없애기로 했다. do_print() 함수는 물론 현재 파일의 아래 부분에 있고 필자는 do_print() 함수를 모르는 상태라고 가정하자.

명령모드에서 탐색 명령 /do_print를 사용하면 do_printf() 함수를 찾을 수 있을 것이고 yy 명령어로 do_printf() 함수의 정의 부분에서 원형을 복사할 수 있다. 하지만 돌아오는 것이 문제다. 파일의 소스가 매우 길 경우 원하는 위치로 돌아오는 것이 만만치 않을 것이다. 물론 위 소스는 매우 짧고 그리고 돌아오기 원하는 위치가 파일의 최상단에 있기 때문에 gg(파일의 처음으로 이동) 명령으로 돌아오면 될 것이다. 그러나 만약 돌아오기 원하는 위치가 파일의 임의의 위치라면 마킹을 사용하면 한

결 수월하다.

먼저 현재 위치를 마킹해야 한다. 현재 위치에서 `ma`(maring a)라고 입력하게 되면 현재 위치가 `a`로 마킹된다. 그러 다음 `/do_print` 명령어로 `do_print()` 함수가 정의되어 있는 곳을 찾는다. 찾았으면 `yy`로 복사한다.

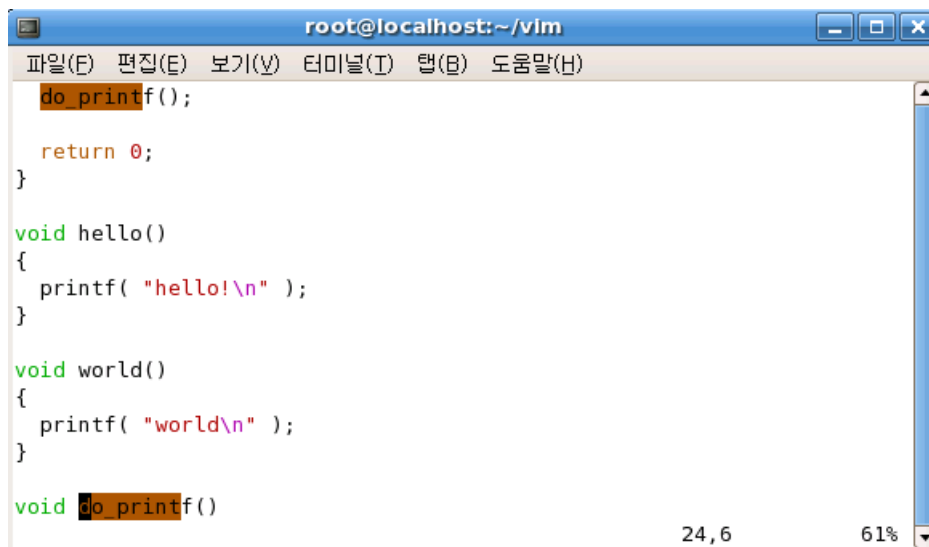


그림 90 마킹 사용법 예제 2

그리고 다시 `a`로 마킹된 곳으로 돌아가면 된다. '`a`' 기호는 backtick을 의미)를 입력하면 `a`로 마킹된 지점으로 돌아간다. 그리고 `p` 명령어로 `do_print()` 함수의 원형을 붙여넣고 편집하면 된다.

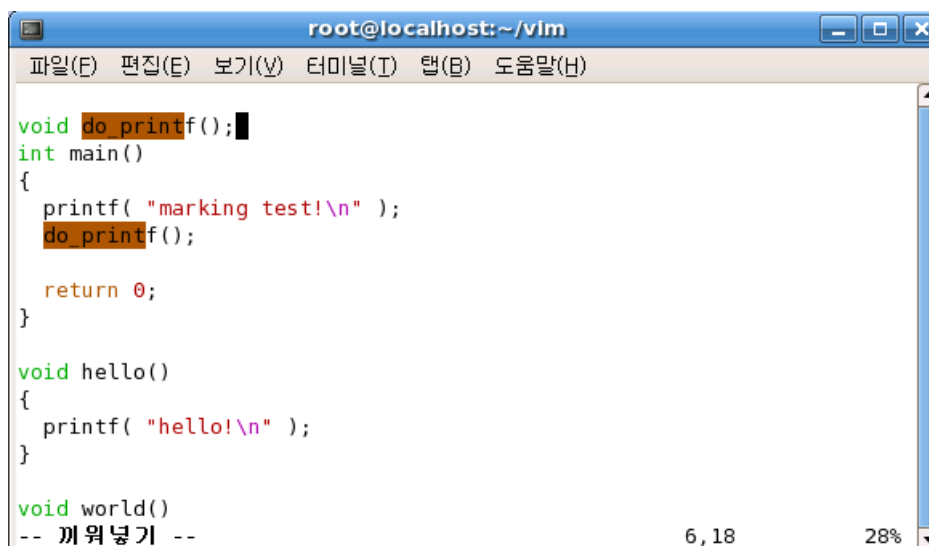


그림 91 마킹 사용법 예제 3

vim에서 마킹된 모든 정보는 `:marks` 명령을 통해서 확인할 수 있고 마킹과 관련된 명령은 다음 표와 같다.

| 명령어 | 내용 | 명령어 | 내용 |
|-----|--|-----|--------------------------------------|
| ma | a로 마킹, mb는 b로 마킹 | | |
| `a | a로 마킹된 위치로 돌아감 | `a | a로 마킹된 행의 처음으로 돌아감 |
| `` | 바로 전에 커서가 위치하던 행의 처음 | `` | 바로 전의 커서 위치로 이동 |
| `,` | 이전에 vim으로 현재 파일을 닫았을 때 커서가 있던 행의 처음으로 이동 | `,` | 이전에 vim으로 현재 파일을 닫았을 때 커서가 있던 위치로 이동 |

6. 셸 명령어 사용

vim에서 작업을 하다가 잠깐 셸로 돌아가 어떤 명령을 실행해야 할 때가 있다. 예를 들면 ifconfig 명령을 입력하여 작업 PC의 IP를 알아내거나 아니면 현재의 프로그램을 컴파일하고 실행시켜 결과를 알아보고 싶은 경우가 있다. 그럴 때마다 매번 vim을 종료하고 셸로 돌아가서 명령을 내린 후 다시 vim을 실행한다면 작업이 비효율적일 것이다.

이렇게 셸 명령어를 입력하거나 아니면 셸로 잠깐 빠져 나가기 위한 방법이 있다. ex모드에서 :![명령어]를 입력하면 잠시 vim으로 빠져나갔다가 명령이 수행된 후 다시 vim으로 돌아오게 된다. 예를 들면 :!ls를 입력하게 되면 vim에서 ls 명령이 수행되고 다시 vim으로 돌아올 수 있다.

```
:[명령어]
```

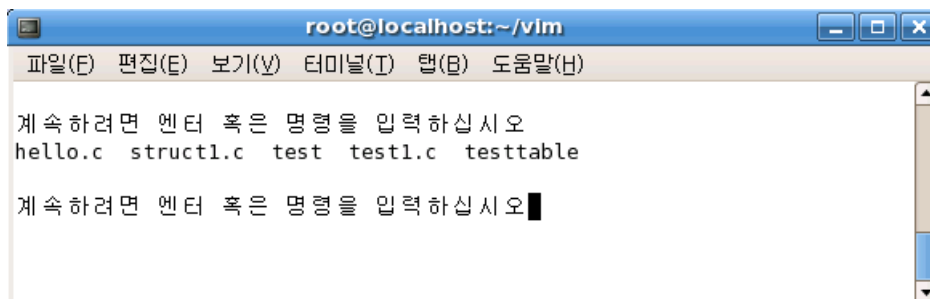


그림 92 :!ls 명령어 수행 결과

마찬가지로 셸을 수행할 수도 있다. :!bash라고 입력하면 새로운 셸이 열리면서 vim을 빠져 나가게 된다. 셸에서 원하는 명령을 수행하고 다시 vim으로 돌아오고자 한다면 셸에서 exit 명령을 내려 셸을 종료하게 되면 다시 vim으로 돌아오게 된다.

명령을 내리고 난 후 화면에 출력되는 결과가 필요할 때에는 :r![명령어]와 같은 형식으로 명령을 내리면 명령이 수행된 결과를 현재 커서의 위치에 끼워넣게 된다. 가령 ls 명령을 내렸을 때 화면에 출력되는 결과를 현재 커서 위치에 끼워넣고 싶다면 :r!ls라고 명령을 내리면 된다.

```
:r![명령어]
```

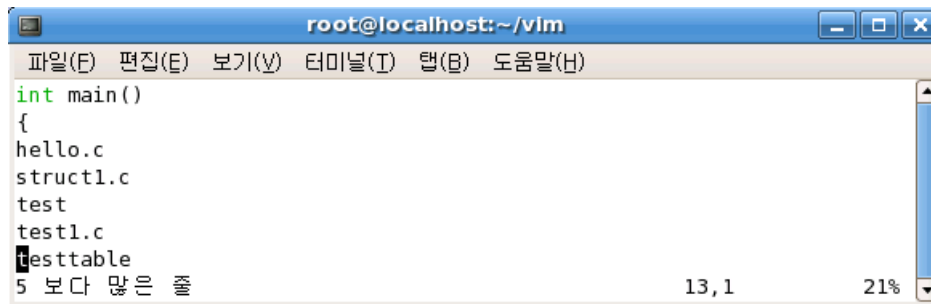


그림 93 :r!ls 명령어 수행 결과

원래 :r[파일명] 명령은 현재 커서의 위치에 파일의 내용을 끼워넣기 위해서 사용한다. 그래서 file1.c 파일 수정 중에 :r file2.c 명령을 내리게 되면 현재 커서의 위치에 file2.c의 내용을 끼워넣게 된다.

```
!r[파일명]
```

Chapter 4. 리눅스 프로그래밍 환경

SECTION

01 네트워크 환경

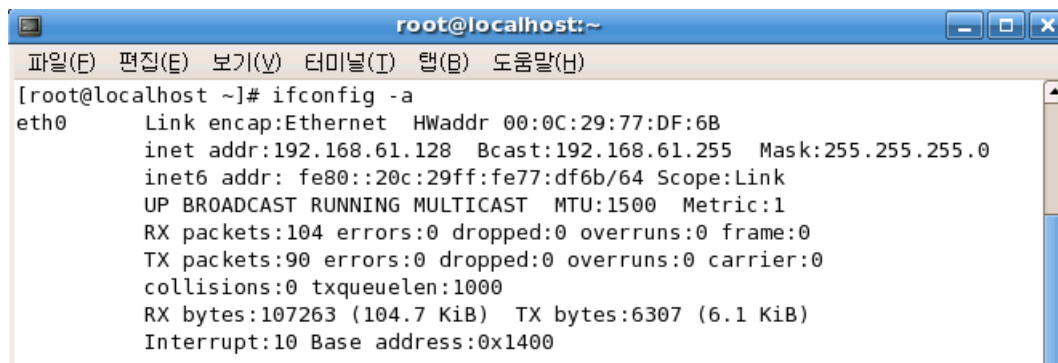
자신의 IP 주소, 도메인 네임 서버 정보, 게이트웨이 주소, 다른 호스트와의 연결 확인 등 현재 네트워크 정보를 알아내는데 사용하는 유닉스 명령문들을 소개한다. 리눅스에서는 네트워크 환경을 시스템 관리자(root)만이 변경할 수 있는 경우가 많다.

1. 네트워크 서비스 시작

자신이 사용하는 호스트가 인터넷에 연결되어 동작 중인지 확인하려면 `ifconfig` 명령을 사용한다.

```
#ifconfig [옵션]
```

`ifconfig` 명령(interface configuration)은 네트워크 디바이스의 상태를 보여주며, `-a` 옵션은 설치되어 있는 모든 네트워크 디바이스의 상태를 보여준다.



```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 랩(B) 도움말(H)
[root@localhost ~]# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:0C:29:77:DF:6B
          inet addr:192.168.61.128  Bcast:192.168.61.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe77:df6b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:104 errors:0 dropped:0 overruns:0 frame:0
          TX packets:90 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:107263 (104.7 KiB)  TX bytes:6307 (6.1 KiB)
          Interrupt:10 Base address:0x1400
  
```

그림 94 `ifconfig` 명령어

자신의 호스트가 통신이 가능한 상태인지를 확인하려면 위의 출력 결과에서 "UP"을 찾아보면 된다. "UP"이 보이지 않는다면 호스트가 통신 불가능한 상태임을 나타낸다.

만일 호스트가 네트워크를 지원하지 않고 있으면 다음과 같이 네트워크 설정 파일들의 내용을 적절히 작성한 후 "network" 명령으로 네트워크를 활성화시킨다. 수정해야 할 환경 설정 파일들은 아래와 같다.


```

/etc/sysconfig/network
/etc/sysconfig/network-script/ifcfg-eth0
/etc/resolv.conf

```

이 파일들을 아래와 같이 호스트 환경에 맞추어 각각 수정한 후에 "/etc/init.d/network restart" 명령을 사용하여 네트워크 서비스를 재시작 한다.

```

root@localhost:/etc/sysconfig
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
NETWORKING=yes                # 호스트가 네트워크를 지원함
HOSTNAME=localhost.localdomain # 호스트의 도메인 이름 등록
~
-- 끼워넣기 --                2,70-60    모두

```

그림 95 /etc/sysconfig/network 설정 파일

```

root@localhost:/etc/sysconfig/network-scripts
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
DEVICE=eth0                    # 이더넷 네트워크 카드 이름
BOOTPROTO=dhcp                 # 자동IP(dhcp) / 고정IP(static) 설정
HWADDR=00:0C:29:77:DF:6B      # 이더넷 네트워크 카드 MAC 주소
ONBOOT=yes                     # 부팅시 자동활성화 여부
TYPE=Ethernet                  # 네트워크 카드 종류
~
-- 끼워넣기 --                5,45-53    모두

```

그림 96 /etc/sysconfig/network-scripts/ifcfg-eth0 설정 파일

```

root@localhost:/etc
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
: generated by /sbin/dhclient-script
search localhost
nameserver 192.168.61.2
~
"resolv.conf" 3L, 80C          1,1      모두

```

그림 97 /etc/resolv.conf 설정 파일

```

root@localhost:/etc/init.d
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost init.d]# ./network restart
Shutting down interface eth0:          [ OK ]
Shutting down loopback interface:      [ OK ]
Bringing up loopback interface:        [ OK ]
Bringing up interface eth0:
eth0에 관한 IP 정보를 얻고 있습니다... 완료.
[ OK ]
[root@localhost init.d]#

```

그림 98 /etc/init.d/network restart 명령 수행 결과

SECTION

02 네트워크 명령어

리눅스는 강력한 네트워크 기능을 제공한다. 본격적인 사용이나 다양한 네트워크 서비스는 뒤에 다루겠지만, 네트워크 상태 확인에 필요한 기본 명령어를 숙지하고 사용해 보도록 한다.

1. ping

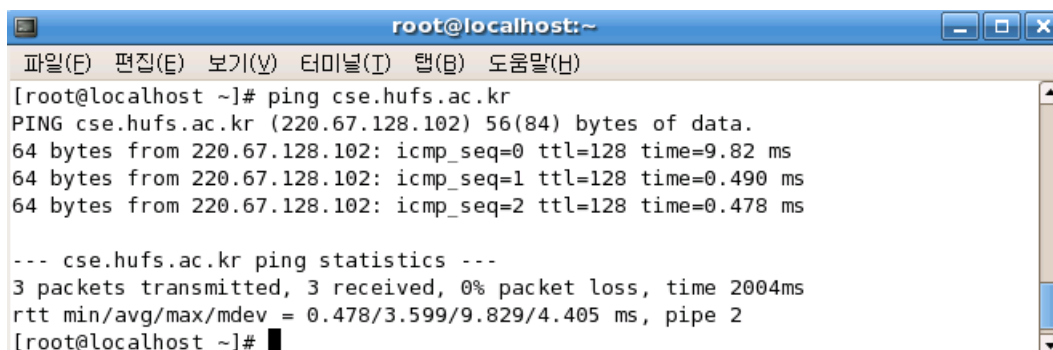
원격 시스템이 제대로 네트워크에 연결되어 있는지 확인하는 명령어이다.

```
# ping [hostname 또는 ip주소]
```

네트워크 관련 명령어 중 가장 흔하게 사용하는 것이다. 이 명령어는 지정한 대상 호스트로 ICMP 패킷을 보내어 그 응답을 통해 통신 상태를 점검한다. 네트워크 연결이 느려질 때 병목 현상이 일어나는 곳을 알아야 바르게 조치를 취할 수 있기 때문에, 네트워크가 느려질 때 흔히 사용하는 명령어이다. 또한 시스템의 네트워크 부분을 설치한 후 제대로 설정되었는지 확인할 때에도 많이 사용한다.

ping이라는 네트워크 명령어를 통하여 알 수 있는 것은 대상 호스트의 통신 가능 여부와 이 테스트를 하기 위해 ICMP 프로토콜을 사용한다는 것이다. 즉, ICMP 에코 요구를 네트워크를 통해 목표 호스트에게 보내는데, ping은 목표 호스트와의 통신 가능 여부와 데이터 패킷이 목표 호스트까지 갔다 오는데 걸리는 시간을 제공한다.

ping 다음에 테스트하려는 서버의 도메인이나 IP주소를 입력하면 해당 주소로 ICMP 패킷을 보내게 된다. ping 테스트를 끝낼 때는 [Ctrl]+c를 입력한다.



```
root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# ping cse.hufs.ac.kr
PING cse.hufs.ac.kr (220.67.128.102) 56(84) bytes of data.
64 bytes from 220.67.128.102: icmp_seq=0 ttl=128 time=9.82 ms
64 bytes from 220.67.128.102: icmp_seq=1 ttl=128 time=0.490 ms
64 bytes from 220.67.128.102: icmp_seq=2 ttl=128 time=0.478 ms

--- cse.hufs.ac.kr ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.478/3.599/9.829/4.405 ms, pipe 2
[root@localhost ~]#
```

그림 99 ping 명령어

위 그림과 같이 [Ctrl]+c를 입력하고 나면 ping 테스트의 결과를 볼 수 있다. 테스트 할 서버로 보낸

패킷이 3개, 응답으로 받은 패킷도 3개, 손실된 패킷은 없는 것을 볼 수 있다. ping 명령도 다양한 옵션을 조합하여 사용할 수 있다.

| | |
|---------|--|
| ping -s | ping 테스트시에 사용할 패킷사이즈 설정 |
| ping -q | ping 테스트 결과를 지속적으로 보여주지 않고 종합 결과만을 출력 |
| ping -i | ping 테스트시 사용할 interval(지연시간) 설정 |
| ping -b | ping 테스트를 하는 서버와 동일한 네트워크에 있는 모든 호스트로 패킷을 보냄 |
| ping -c | ping 테스트시에 보낼 패킷 수를 지정 |
| ping -t | ping 테스트를 멈추지 않고 연속하여 실행 |

표 29 ping 기본 옵션

2. traceroute

패킷이 목적 호스트까지 전달되는 경로를 조사하여 화면에 출력하는 명령어가 traceroute이다. traceroute를 이용하여 패킷이 어느 경로에서 유실되는지를 확인할 수 있으며, 어느 네트워크에서 트래픽이 발생하는지를 점검할 수 있다. traceroute의 일반적인 사용법은 다음과 같다.

```
# traceroute [테스트 할 주소]
```

예를 들어 내 컴퓨터에서 daum.net으로 가는 과정을 추적해 보자. 내 컴퓨터에서 출발한 패킷은 여러 호스트를 거쳐서 원하는 서버로 전송된다. 이 명령을 실행하면 네트워크 상태가 좋지 않거나 특정 사이트에 접속이 잘 되지 않을 때 어디에서 문제가 생겼는지 확인할 수 있다. 테스트한 결과에서 다음과 같은 내용을 파악할 수 있다.

1. 대상 호스트와의 통신은 잘 이루어지고 있는가?
2. 대상 호스트까지 몇 개의 hop 수를 거쳐서 통신이 되고 있는가?
3. 대상 호스트까지의 통신에 있어서 지연되는 구간은 없는가? 만약, 지연되는 구간이 있다면 어느 구간인가?
4. 대상 호스트까지의 통신 경로에 있는 특정 구간의 정보 확인.

네트워크 인터페이스(랜카드 등)가 하나만 있을 때에는 문제가 없지만, 여러 개의 랜 카드를 사용할 때에는 경로 추적이 잘 되지 않을 때가 있다. 이때는 패킷을 전달하는 인터페이스를 지정해주면 정상적으로 작동한다. 여러 개의 인터페이스에서 기본 라우터로 설정된 인터페이스를 통하여 경로추적을 할 때는 다음과 같은 명령을 내린다.

```
# traceroute -i [인터페이스 이름 또는 호스트 주소]
```

3. route

route는 현재 사용 중인 서버의 라우팅 경로를 설정하기 위한 것으로 특정 네트워크 인터페이스에 라우팅을 설정하는 명령어이다.

```
# route
```

```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags   Metric  Ref    Use  Iface
192.168.61.0    *               255.255.255.0   U        0        0      0   eth0
169.254.0.0     *               255.255.0.0    U        0        0      0   eth0
default         192.168.61.2   0.0.0.0         UG       0        0      0   eth0
[root@localhost ~]#

```

그림 100 route 명령어

특정 네트워크 인터페이스의 기본 게이트웨이를 설정하는 명령어 형식은 다음과 같다.

```
# route add default gw [게이트웨이 IP주소] dev [네트워크 인터페이스 장치]
```

다음은 네트워크 인터페이스에 외부와 통신이 가능하도록 하기 위하여 기본 게이트웨이를 설정하는 명령어 형식이다.

```
# route add -net [게이트웨이 IP주소] netmask [넷마스크] dev [네트워크 인터페이스 장치]
```

4. netstat

netstat은 route 명령보다 더 자세한 라우팅 테이블 정보와 네트워크 상태를 체크하는 도구이다. 일반적으로 route 명령보다는 netstat 명령을 많이 사용한다. netstat 명령의 일반적인 사용법은 다음과 같다.

```
# netstat -nr
```

다음은 netstat에서 쓸 수 있는 옵션들이다.

| | |
|------------|---|
| netstat -a | --all과 같으며 listen되는 소켓 정보와 listen되지 않는 소켓 정보를 모두 출력 |
| netstat -n | --numeric과 같으며 10진수의 수치 정보로 결과를 출력 |
| netstat -r | --route와 같으며 설정된 라우팅 정보를 출력 |
| netstat -p | --program과 같으며 실행되고 있는 각 프로그램과 PID정보를 출력 |
| netstat -i | --interface=iface와 같으며 모든 네트워크 인터페이스 정보를 출력하거나 특정 네트워크 인터페이스 정보를 출력 |
| netstat -c | --continous와 같으며 netstat 결과를 연속적으로 출력 |
| netstat -l | --listening과 같으며 현재 listen 되고 있는 소켓 정보를 출력 |
| netstat -s | --statistic과 같으며 각 프로토콜에 대한 통계 정보를 출력 |

표 30 netstat 기본 옵션

netstat은 라우팅 테이블 정보를 조회하는 목적 외에도 서버에서 사용 중인 포트와 그 포트를 사용하는 프로그램에 대한 정보를 얻는 데에도 사용된다.

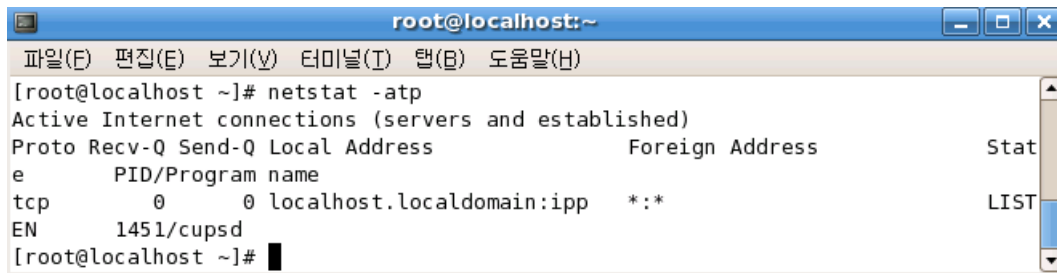
```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# netstat -s
Ip:
  361 total packets received
  4 with invalid addresses
  0 forwarded
  0 incoming packets discarded
  357 incoming packets delivered
  295 requests sent out
Icmp:
  4 ICMP messages received
  0 input ICMP message failed.
  ICMP input histogram:
    echo requests: 1
    echo replies: 3
  1 ICMP messages sent
  0 ICMP messages failed
  ICMP output histogram:
    echo replies: 1
Tcp:
  9 active connections openings
  1 passive connection openings
  0 failed connection attempts
  0 connection resets received
  0 connections established
  320 segments received
  268 segments send out
  1 segments retransmited
  0 bad segments received

```

그림 101 netstat -s 명령어

위 그림을 보면 s 옵션을 사용한 것으로 리눅스 서버에서 사용할 수 있는 프로토콜에 대한 통계 정보를 확인할 수 있다.



```

root@localhost:~
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost ~]# netstat -atp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost.localdomain:ipp  *:*                     LISTEN
           PID/Program name
           1451/cupsd
[root@localhost ~]#

```

그림 102 netstat -atp 명령어

atp 옵션을 사용하여 현재 응답할 수 있는(서버에서 열려있는) 포트 번호와 각 제몬들, 그리고 그 포트에서 사용하는 프로그램에 대한 정보를 상세히 점검해 볼 수 있다.

SECTION

03 gcc를 사용해 컴파일 하기

GCC는 원래 GNU C compiler의 줄임말로 사용되었지만, 1999년 4월에 "GNU Compiler Collection"로 바뀌었다. GNU C 컴파일러는 리처드 스톨먼 등에 의해 만들어졌는데, 이 컴파일러는 품질이 매우 좋으며, 이식성이 좋은 C, C++ 컴파일러이다. C, C++, JAVA, FORTRAN 등 여러 가지 컴파일러들이 포함되어 있으나 이 책에서는 C 컴파일러를 이용해 모든 소스를 다루고 실행한다.

1. 가장 단순한 컴파일 명령

가장 단순한 컴파일 명령은 다음과 같다.

```
# gcc hello.c
```

위와 같이 명령을 내리면 hello.c가 a.out이라는 이름으로 컴파일된다.

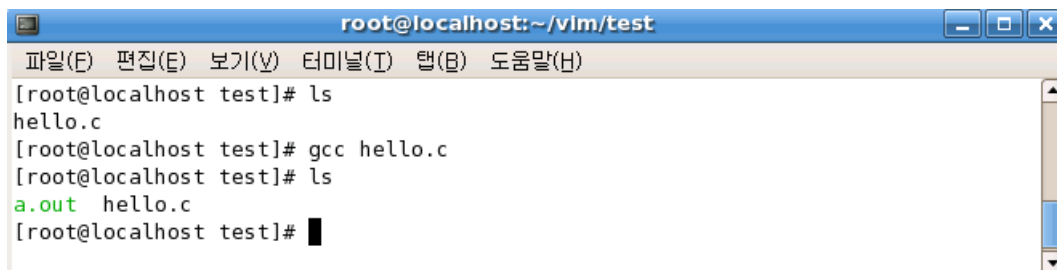


그림 103 gcc hello.c로 컴파일한 모습

가장 단순하게 출력 파일의 이름조차 지정하지 않은 컴파일 방법이다. 위와 같은 컴파일 방법은 간단한 소스 테스트에서 자주 사용하고 보통은 아래와 같은 형식으로 사용한다.

```
# gcc -W -Wall -O2 -o hello hello.c
```

-Wall 옵션은 모든 모호한 코딩에 대해서 경고를 보내는 옵션이고 -W 옵션은 합법적이지만 모호한 코딩에 대해서 경고를 보내는 옵션이다. 그래서 -W -Wall 옵션을 붙이게 되면 아주 사소한 모호성에 대해서도 경고가 발생한다.

-O2는 최적화 레벨을 2로 설정한다. 최적화 레벨 2는 거의 대부분의 최적화를 시도한다. -o hello는 컴파일된 파일명을 hello로 하라는 의미다. 위와 같은 기본 컴파일 명령에 다음부터 설명하는 옵션들

을 바탕으로 필요한 다른 옵션들을 추가하면 된다.

2. gcc 옵션

가. gcc 전역 옵션

| 옵션 | 설명 |
|--------------|----------------------------------|
| -E | 전처리 과정을 화면에 출력 |
| -S | 어셈블리 파일 생성 |
| -c | 오브젝트 파일 생성 |
| -v | 컴파일 과정을 화면에 출력 |
| --save-temps | 컴파일시 생성되는 중간 파일 저장 |
| -da | 컴파일 과정에서 생성되는 중간 코드 생성(RTL 파일 등) |

표 31 gcc 전역 옵션

나. 전처리기(cpp0) 옵션

| 옵션 | 설명 |
|---------------------|--|
| -I[경로] | 헤더 파일을 탐색할 디렉토리 지정 예) -I/opt/include |
| -include [헤더 파일 경로] | 해당 헤더 파일을 모든 소스 내에 추가 예) -include /root/my.h |
| -D[매크로] | 외부에서 #define 지정 예) -DDEBUG |
| -D[매크로]=[매크로 값] | 외부에서 해당 매크로를 정의하고 값을 지정 예) -DDEBUG=1 |
| -U[매크로] | 외부에서 #undef 지정 예) -UDEBUG |
| -M 또는 -MM | make 기술 파일을 위한 소스 파일의 종속 항목 출력 |
| -nostdinc | 표준 C 헤더 파일을 포함하지 않음 |
| -C | 전처리 과정에서 주석을 제거하지 않음 |
| -Wp,[옵션 리스트] | 옵션 리스트를 전처리기에 바로 전달 |

표 32 전처리기(cpp0) 옵션

다. C 컴파일러(cc1) 옵션

| 옵션 | 설명 |
|--------------|---|
| -ansi | ANSI C 문법으로 문법 검사 |
| -std=[C 표준] | 지정한 C 표준으로 문법 검사(c89, c99, gnu89, gnu99 등) |
| -traditional | K&R C 문법으로 문법 검사 |
| -fno-asm | asm, inline, typeof 키워드를 사용하지 않음(gnu89 문법 기준) |

표 33 C 언어 옵션

| 옵션 | 설명 |
|------------------|---|
| -Wall -W | 모든 경고 메시지 출력 |
| -w | 모든 경고 메시지 제거 |
| -Werror | 모든 경고를 오류로 취급하여 컴파일 중단 |
| -pedantic | C89 표준에서 요구하는 모든 경고 메시지를 표시 |
| -pedantic-errors | C89 표준에서 요구하는 모든 오류 메시지를 표시 |
| -Wtraditional | ANSI C와 K&R C 간에 서로 다른 결과를 가져올 수 있는 부분 경고 |

표 34 경고 옵션

| 옵션 | 설명 |
|-----------|-----------------|
| -O0 | 아무런 최적화를 수행치 않음 |
| -O1 또는 -O | 최적화 레벨 1 수행 |
| -O2 | 최적화 레벨 2 수행 |
| -O3 | 최적화 레벨 3 수행 |
| -Os | 사이즈 최적화 수행 |

표 35 최적화 옵션

| 옵션 | 설명 |
|-----|--------------------|
| -g | 바이너리 파일에 디버깅 정보 삽입 |
| -pg | 프로파일을 위한 코드 삽입 |

표 36 디버깅 옵션

라. 어셈블리(as) 옵션

| 옵션 | 설명 |
|--------------|----------------------|
| -Wa,[옵션 리스트] | 어셈블러에게 옵션 리스트를 바로 전달 |
| -Wa,-al | 어셈블된 코드와 인스트럭션을 보임 |
| -Wa-as | 정의된 심볼을 보임 |

표 37 어셈블리(as) 옵션

마. 링크(ld) 옵션

| 옵션 | 설명 |
|--------------|--------------------------------|
| -L[경로] | 라이브러리 탐색 디렉토리 지정 예) -L/opt/lib |
| -l[라이브러리 이름] | 해당 라이브러리를 링크 예) -lm |
| -shared | 공유 라이브러리를 우선하여 링크 |
| -static | 정적 라이브러리를 우선하여 링크 |
| -nostdlib | 표준 C 라이브러리를 사용하지 않음 |
| -M 또는 -MM | make 기술 파일을 위한 소스 파일의 종속 항목 출력 |
| -nostdinc | 표준 C 헤더 파일을 포함하지 않음 |
| -Wl,[옵션 리스트] | 옵션 리스트를 링크에 바로 전달 |

표 38 링크(ld) 옵션

Chapter 5. 소켓 프로그래밍 기초

SECTION

01 소켓의 동작 원리와 이해

1. 소켓의 기본 동작 방식

먼저, 소켓이 무엇인지에 대한 정의부터 알아보자. 소켓을 간단히 말하면 “**프로그래머가 네트워크 프로그램을 조금 더 쉽게 작성할 수 있도록 도와주기 위한 도구(API)**”인데, 운영체제에서는 네트워크 송수신에 관련된 함수들을 만들어 놓고 시스템 콜 형태로 제공해 주고 있다. 이런 시스템 콜은 C 언어에서 함수를 호출하듯이 호출하기만 하면 된다. 그러므로 네트워크에 대해서 잘 모르는 초보자라도 운영체제에서 제공하는 네트워크와 관련된 시스템 콜의 사용 방법만 알고 있다면 네트워크 프로그래밍을 할 수 있다.

전화 통화와 비유해서 설명하면 쉽게 이해할 수 있을 것이다. 우리들은 친구들과 연락을 하기위해서 전화를 많이 사용하지만 전화기가 어떻게 만들어지는지 전화라인을 타고 음성 데이터가 어떻게 전송되는지에 대한 것은 모르고 있다. 단지 전화기의 버튼을 누르는 것만으로 친구와 전화통화를 하면서 친구에게 음성 데이터를 보내게 된다. 소켓은 이렇게 네트워크에 대한 자세한 지식이 없더라도 데이터를 쉽게 송수신 할 수 있게 도와주는 역할을 한다. 이처럼 소켓은 ‘통신’을 한다는 점에서 그리고 사용하는 방법에 있어서 전화기와 매우 유사하므로 일상에서 자주 사용하는 전화기와 소켓의 사용 방법을 비교하면서 소켓의 사용법에 대해서 알아볼 것이다.

전화기를 이용해서 A와 B가 통화를 한다고 가정할 때 다음과 같은 전형적인 과정을 통해서 통화가 이루어질 것이다.

1. A와 B 사이에는 우선 전화기가 준비되어 있어야 한다.
2. A: 수화기를 들고 버튼을 눌러서 연결을 시도한다.
3. B: 신호가 울리고 수화기를 들어서 연결을 받아들인다.
4. A: “B씨가 맞나요?” (상대방을 확인한다)
 - B: “네 맞습니다. 실례지만 누구시죠?” (B 역시 상대방을 확인해야 한다)
 - A: “아, 전 A입니다 반갑습니다.”
5. 이제 서로에 대한 확인도 되었으니 이런저런 얘기를 나눈다 (잠시 후 모든 얘기가 다 끝났다).
6. B: “그럼 다음에 뵙도록 하겠습니다. 안녕히 계세요.”
 - A: “네 안녕히 계세요.”
7. 수화기를 내려놓고 전화를 끊는다.

위의 과정과 소켓을 통한 데이터 통신 과정의 3단계를 비교해 보면 다음과 같다.

| 전화 | 소켓 |
|--|---|
| 1. 한쪽에서 연결을 시도하고, 다른 쪽에서는 연결을 대기한다. | 1. 한쪽에서는 연결을 시도하고, 다른 한쪽에서는 연결을 받아들인다. |
| 2. 연결이 이루어진 후 통화를 한다. | 2. 데이터 통신을 한다. |
| 3. 연결을 종료한다. | 3. 통신을 끊는다. |

표 39 전화와 소켓의 통신 과정 3단계 비교

이제부터 위의 과정을 구현하는 데 필요한 소켓 함수에 대해서 알아보겠다. 여기에서는 단순히 함수를 소개하는 선에서 그칠 것이며 자세한 내용은 이 장의 마지막 부분에서 예제를 통해 실제 함수를 이용해서 익힌다.

가. 전화기와 비교한 소켓의 통신 과정

1) 일단 전화기를 준비한다

전화기가 있어야 한다. 앞서 말했듯이 전화기는 '소켓'이다. 통화를 위한 다양한 서비스를 이용하려면 전화기가 있어야 하듯이 네트워크 통신을 위한 다양한 기능을 이용하려면 먼저 소켓을 만들어야 한다. 그 다음 연결, 통신, 종료와 관련된 모든 일은 이 소켓을 통해서 이루어진다.



그림 104 전화기와 유사한 소켓

소켓은 socket 함수를 호출해서 생성하는데, 자세한 함수 사용 방법은 추후 예제를 보면서 설명하겠다.

2) 상대방에게 연결하기 위해서 전화번호를 준비한다

수화기를 들었다면 이제 전화번호를 눌러 주어야 한다. 전화번호를 눌러야 원하는 상대방의전화번호로 정확하게 신호를 보낼 수 있기 때문이다. 인터넷에서는 통신할 상대방 컴퓨터의 위치를 찾기 위해서 IP 주소를 사용한다는 것을 이미 배웠다.

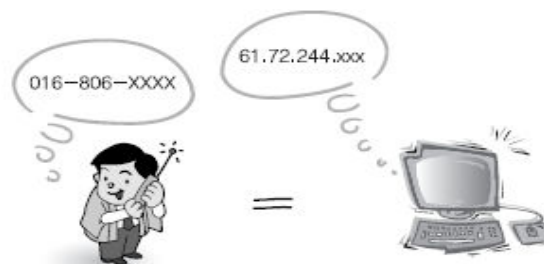


그림 105 인터넷의 전화번호 = 인터넷주소

소켓에서는 이러한 인터넷 주소를 부여하기 위해서 bind 함수를 사용한다. 이 함수를 이용해서 우리가 통신하고자 하는 컴퓨터의 인터넷 주소를 설정할 수 있게 된다.

3) 전화번호를 누르고 연결을 시도한다

이제 수화기도 들었고, 전화번호도 준비했으니 버튼을 누르고 신호를 보내야 할 것이다. 이를테면 “지금 당신에게 전할 말이 있으니 수화기를 들어 주세요”라고 상대방에게 요청하는 과정이다. 소켓에서는 connect 함수를 통해서 인터넷 내에서 멀리 떨어져 있는 다른 컴퓨터에 연결을 받아줄 것을 요청할 수 있다.

4) 연결을 기다리고, 연결을 받아들인다

통신을 하려면 연결되는 상대가 있어야 하므로 최소한 2명의 인원이 필요한데, 보통 한 명은 전화를 기다리고 한 명은 전화를 거는 형식을 취하게 된다. 전화를 기다리는 쪽에서는 상대방으로부터 신호가 도착하면 수화기를 들어서 연결을 받아들이게 된다.

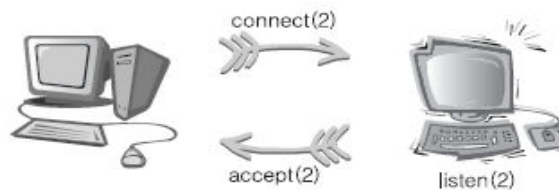


그림 106 소켓 함수를 이용한 컴퓨터간의 연결

지금까지는 소켓에서 연결을 요청하는 쪽을 기준으로 connect 함수를 이용해서 연결을 요청하는 과정을 설명했다. 그렇지만 connect 함수로 연결을 요청했다고 무조건 연결이 되는 것은 아니다. 상대방 소켓에서도 기다리고 있다가 연결 요청이 들어오면 받아들이는 부분이 있어야 한다. 이러한 작업을 위해서 소켓은 listen 함수와 accept 함수를 제공한다. listen 함수는 연결 요청을 기다리기 위해, accept 함수는 연결 요청이 들어왔을 때 요청을 받아들이기 위해서 사용한다.

5) 연결이 되었다면 이제 대화를 한다

전화벨 소리를 확인해서 양쪽 모두 수화기를 들었다면, 이제 본격적인 통화가 이루어지게 된다. 이때 서로 간에 주고받는 정보는 ‘언어’로 이루어져 있을 것이다. 소켓으로 되돌아가 보면 connect 함수를 이용해서 연결을 요청하면 반대쪽은 accept 함수를 통해서 연결 요청을 받아들이게 되고 이것으로 컴퓨터간의 통신을 위한 준비가 끝나게 된다. 컴퓨터 간에 주고받는 데이터는 컴퓨터에서 사용하는 것과 동일하게 비트로 된 일련의 연속된 값들이다.

이제 통신을 해야 하는데, 수화기 역할을 할 수 있는 함수를 사용해야 할 것이다. 제대로 된 통화를 원한다면 전송과 수신을 모두 필요로 하므로 최소한 두 개의 함수가 준비되어야 한다. 보통송신(쓰기)을 위해서는 write 함수를, 수신(읽기)을 위해서는 read 함수를 사용한다. 그리고read/write 함수 외에도 송신을 위한 sendto 함수와 수신을 위한 recvfrom 함수 등 다른 함수들도 있다. 유닉스 프로그램을 해본 독자들은 read/write가 네트워크 데이터를 송수신하는 데 사용한다고 하면 이상하다고 생각할 수도 있을 것이다. 우리가 유닉스 프로그램에서는 read/write를 파일 입출력 함수라고 생각하

기 때문이다. 하지만 read/write는 특정 기능에 한정된 함수가 아니라 운영체제에서 제공하는 시스템 콜이다. 때문에 어떤 상황에서 사용하느냐에 따라서 동작하는 내부루틴이 달라지게 된다.

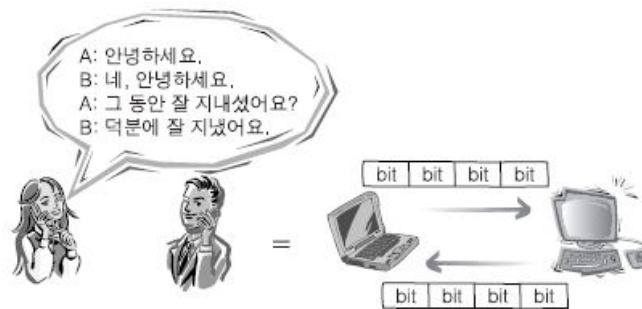


그림 107 비트로 이루어진 데이터 교환을 위한 컴퓨터간의 통신

6) 모든 대화를 마쳤다면 통화를 끊어야 한다

필요한 모든 대화를 나눴다면 서로 인사를 하고 전화를 끊게 된다. 이 인사에는 “이제 통화를 끊겠습니다”란 의미도 함께 전달하게 된다. 만약, 이러한 과정이 없이 그냥 수화기를 내려놓게 된다면 상대방은 분명 ‘기본적인 예의도 모르는 사람’이라고 생각할 것이다.

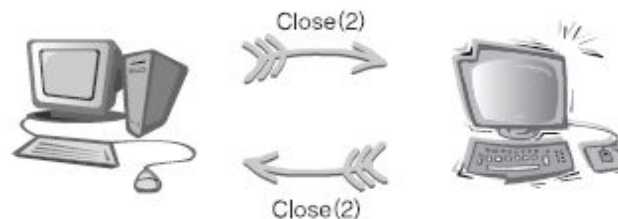


그림 108 close 함수를 이용한 컴퓨터간 통신 종료

소켓을 통한 네트워크 통신에서도 이러한 과정이 필요하다. 모든 통신이 끝났다면 상대방에게 연결을 끊을 것임을 알려주고 자신도 연결을 끊어야 한다. 이처럼 두 번의 연결종료 작업(수화기 내려놓기)이 이루어짐을 알 수 있다. 이러한 연결종료의 작업을 위해서 소켓은 close함수를 제공한다.

2. 서버/클라이언트 개념

소켓을 통해 통신이 어떻게 이루어지는지 대략적인 과정을 전화 통화의 예를 통해 알아보았는데, 전화 통화의 경우든 소켓을 통한 인터넷 통신의 경우든 하나의 공통된 모델을 따른다는 것을 알 수 있다. 즉, 무엇인가 요청하는 쪽(전화를 거는 쪽)과 요청을 받아서 처리하는 쪽(전화를 받는 쪽)의 존재다. 이처럼 요청을 하는 객체와 요청을 처리하는 객체가 하나의 쌍을 이루는 모델을 서버/클라이언트(server/client 줄여서 c/s) 모델이라고 부른다. 우리나라 말로 번역해 보자면 ‘고객/일꾼 모델’ 정도가 될 것이다.

마찬가지로 일상생활의 예를 통해 서버/클라이언트의 관계를 살펴보자. 서버/클라이언트 모델이 적용되는 가장 확실한 예는 전화를 통한 고객지원센터다.

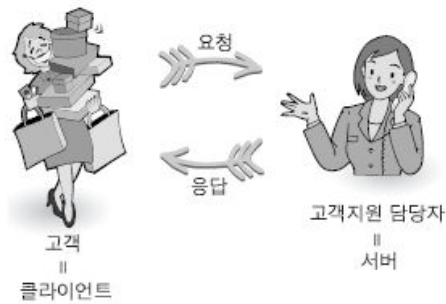


그림 109 서버/클라이언트 모델의 예

인터넷에서도 위의 모델은 그대로 적용된다. 즉, 어떤 데이터를 요청하는 호스트와 데이터의 요청을 받아서 처리하는 호스트의 쌍으로 이루어지는 서버/클라이언트 모델을 그대로 따르게 된다. 인터넷 내에서 데이터를 요청하는 호스트는 '클라이언트 호스트', 요청을 처리하는 호스트는 '서버 호스트'라고 부르며, 보통은 호스트 글자를 빼고 '클라이언트'와 '서버'로 부른다.

여기서 호스트는 컴퓨터라는 하드웨어를 나타내며 (한마디로 껍데기다) 실제로 요청을 하고 처리를 하는 것은 하드웨어에 설치되어 있는 프로그램이 맡아서 한다. 클라이언트에 설치되어 요청하는 프로그램을 클라이언트 프로그램, 서버에 설치되어 요청을 처리하는 프로그램을 서버프로그램이라고 부른다.

인터넷에서 서버/클라이언트 모델을 따르는 가장 유명한 서비스는 WWW다. 여기에서 클라이언트 프로그램은 인터넷 익스플로러, 넷스케이프, 모질라와 같은 웹 브라우저이며, 서버 프로그램은 IIS, 아파치와 같은 웹 서버 프로그램이다. 아래의 그림을 보면 웹 브라우저와 웹 서버의 관계를 쉽게 이해할 수 있을 것이다.



그림 110 인터넷에서 적용되는 서버/클라이언트 모델

클라이언트는 보통 일반 PC가 될 것이다. WWW 서비스를 이용하기 위해서 우리는 클라이언트 프로그램인 익스플로러를 실행하고, 서버에 웹 페이지를 보여달라고 요청하게 된다. 서버는 서버 웹 프로그램인 아파치 등이 설치되어 있어서 클라이언트 프로그램인 익스플로러의 요청을 받으면 웹 페이지를 읽어서 이를 서버에 전달하게 된다. 응답을 받은 익스플로러는 보기 좋은 화면으로 만들어서 모니터에 뿌려주게 된다.

이러한 서버/클라이언트 모델은 웹 서버인 WWW에서 뿐만 아니라 FTP, 이메일 등을 비롯한 많은 수의 온라인 게임 등의 서비스를 위해 널리 사용되고 있다. 서버/클라이언트 모델 외에도 몇 가지 다른 모델도 있긴 하지만 네트워크 프로그램의 90% 이상이 서버/클라이언트 모델을 사용하고 있으며, 우리가 흔히 네트워크 프로그램을 만든다고 하면 서버/클라이언트 모델을 따르는 프로그램을 작성한다고 생각하면 된다.

3. 소켓으로 작성하는 서버/클라이언트 프로그램의 동작 방법

지금부터는 개념적으로만 설명했던 서버/클라이언트 프로그램의 작성 방법을 함수별로 설명하면서 자세히 알아보겠다. 우선 서버 프로그램 동작순서를 단계별로 정리하면 다음과 같다.

1. 소켓생성(socket 함수)
2. 생성된 소켓에 인터넷 주소 부여(bind 함수)
3. 데이터 수신 대기(listen 함수)
4. 데이터 수신(accept 함수)
5. 데이터 읽기(read 함수)
6. 데이터 쓰기(write 함수)
- 7 .3번으로돌아간다(listen 함수)

서버 프로그램이 하는 역할은 AS 센터에서 전화 상담원이 하는 일과 같다. 전화 상담원은 전화를 준비한 후(socket), 전화번호를 부여받는다(bind). 전화기가 준비가 되면 고객이 전화를 해서 전화벨이 울릴 때까지 전화 앞에서 대기한다(listen). 전화벨이 울리면 전화를 받고(accept)고객의 불만 사항을 접수한다(read). 고객의 불만 사항을 접수한 전화 상담원은 고객의 불만을처리한후고객에게처리결과를알려준다(write). 그리고 다시 전화벨을 기다린다(listen).

이처럼 서버 프로그램은 수동적인 입장에서 클라이언트 프로그램의 접속을 기다리고 클라이언트 프로그램이 접속을 하면 데이터를 수신한 후에 처리한 결과 데이터를 다시 클라이언트에게 전송해 주는 역할을 하게 된다. 이런 서버/클라이언트 프로그램의 구조를 간단하게 구성해보면 다음과 같은 구조가 된다.

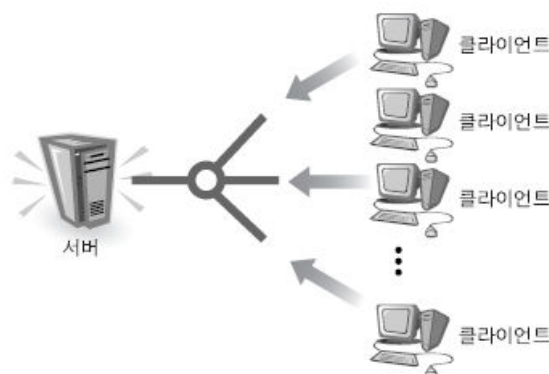


그림 111 서버/클라이언트 구조

위 그림을 보고 짐작하는 독자도 있겠지만 서버/클라이언트 구조는 서버에 여러 대의 클라이언트가 연결하는 구조로 되어 있다. 그렇기 때문에 서버는 여러 클라이언트의 동시 접속을 빠른 시간 내에 처리하기 위해서 하드웨어적인 성능이 뛰어나야 한다. 그리고 앞으로 여러분이 서버 프로그램을 하게 되면 얼마나 빠른 시간과 적은 컴퓨터 자원을 사용해서 접속하는 클라이언트의 요구를 처리할 것인지에 대해서 무수한 밤을 지새우면서 고민하게 될 것이다.

이제 클라이언트 프로그램의 순서를 알아보자. 클라이언트 프로그램은 위에서 설명했던 것처럼 서버에 접속해서 데이터를 요청하고 처리된 데이터를 받는 프로그램이다. 다음은 클라이언트 프로그램이 서버에 접속하기 위한 순서를 단계별로 정리한 것이다.

1. 소켓 생성(socket 함수)
2. 서버에 연결(connect 함수)
3. 데이터 쓰기(write 함수)
4. 데이터 읽기(read 함수)
5. 연결 종료(close 함수)

클라이언트 프로그램이 하는 일은 AS 센터에 전화를 하는 고객이 하는 일과 같다. 우선 AS 센터에 전화하기 위해서 전화를 준비한다(socket). AS 센터의 전화번호를 눌러서 연결을 하고(connect) 전화 상담원이 전화를 받으면 불만 사항을 접수하고(write) 전화 상담원이 처리한 내용을 듣게 된다(read).

서버 프로그램과 비교하면 클라이언트 프로그램에서 하는 일은 비교적 간단하다고 할 수 있다. 클라이언트 프로그램은 서버 프로그램과 다르게 요청을 기다리는 것이 아니라 서버에 연결한 다음에 자신이 필요로 하는 요청을 보내고 원하는 데이터를 받는 역할만 하면 되기 때문이다. 클라이언트 프로그램에서 신경 써야 하는 부분은 서버 프로그램에서의 많은 연결을 처리하기 위한 알고리즘이나 구조가 아니라 사용자에게 얼마나 더 편리한 그래픽 사용자 인터페이스(GUI)를 제공할 것인지가 될 것이다.

지금까지의 내용을 정리하자면 클라이언트 프로그램은 다음과 같은 순서를 따른다.

소켓 생성→서버의 인터넷 주소 설정→연결 요청→통신(읽기/쓰기) →연결 종료

서버 프로그램은 위의 과정에서 연결을 기다리는 listen 과정과 연결 되었을 때 받아들이는 accept 과정이 추가된다. 정리해 보면 서버 프로그램은 다음과 같은 순서를 따른다.

소켓 생성→인터넷 주소 부여→연결 수신 대기→연결 수신→
통신(읽기/쓰기) →연결 수신 대기(이후 과정 반복)

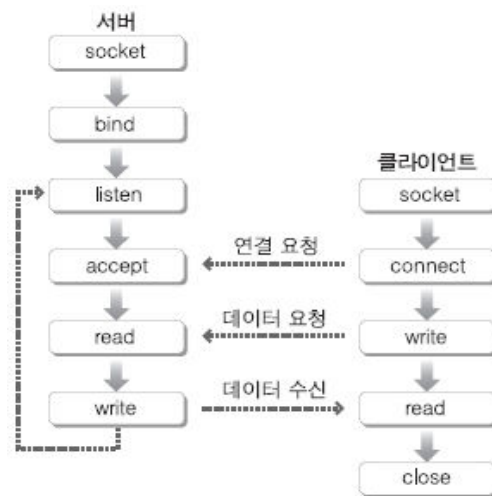


그림 112 클라이언트 프로그램 작성을 위한 프로그램의 흐름

위 그림을 보면 왼쪽에 있는 클라이언트 프로그램은 socket, connect, read, write, close와 같이 다섯 개의 함수가 사용되고 있다. “정말 다섯 개의 소켓 함수만으로 클라이언트 프로그램을 작성할 수 있을까?”라고 반문할 수도 있을 것이다. 하지만 정말 위의 다섯 개의 함수만 이용하면 제대로 동작하는 클라이언트 프로그램을 작성할 수 있다. 모든 클라이언트 프로그램의 뼈대는 위의 구조를 유지하며 나머지 작업은 여기에 살을 가져다 붙이는 작업(GUI 화면을 입히고, 데이터를 처리하는 작업 등)이기 때문이다.

정리해보면 클라이언트 프로그램은 다음과 같은 소켓함수의 흐름을 따르게 된다.

socket → connect → read/write → close

서버 프로그램의 경우에는 클라이언트 프로그램에서 필요 없는 ‘연결을 기다리고’, ‘연결을 받아들이는’ 부분이 추가된다. 나머지 부분은 클라이언트 프로그램과 전혀 다를 게 없다. 그렇지만 클라이언트 프로그램의 작성에서 사용되었던 다섯 개의 함수들 외에 listen과 accept가 추가되어 있음을 알 수 있다. 나머지 구성이나 전반적인 흐름은 클라이언트와 동일하다. 정리하면 서버 프로그램은 다음과 같은 소켓 함수의 흐름으로 작성된다.

socket → **bind** → **listen** → **accept** → read/write → **listen**(반복)

굵게 표시된 함수는 클라이언트 프로그램과 비교해서 추가된 함수다. 이들 함수가 추가된 외에는 클라이언트 프로그램과 구성이 동일하다. 다음 장에서 위의 뼈대를 토대로 실제 간단한 클라이언트 프로그램을 만들어서 작동시킬 것인데, 충분히 기대해도 좋을 것이다.

SECTION

02 바이트 순서 바꾸기

1. 호스트 바이트 순서

바이트 순서(byte order)에는 호스트 바이트 순서와 네트워크 바이트 순서 두 가지가 있다. 호스트 바이트 순서는 컴퓨터가 내부 메모리에 숫자를 저장하는 순서를 말하는 것으로써, CPU에 따라 다르다.

예를 들어, 두 바이트로 구성된 십진수 50146의 경우 16진수로 표현하면 0xC3E2인데 이것은 80x86 계열의 CPU에서는 E2, C3의 순서로(즉, 하위 바이트부터) 메모리에 저장되고 MC68000 계열의 CPU에서는 C3, E2의 순서로 메모리에 저장된다.

전자를 하위 바이트(즉 little end)가 메모리에 먼저 저장된다고 하여 little-endian이라고 하고 후자를 상위 바이트(즉 big end)가 메모리에 먼저 저장된다고 하며 big-endian이라고 부른다.

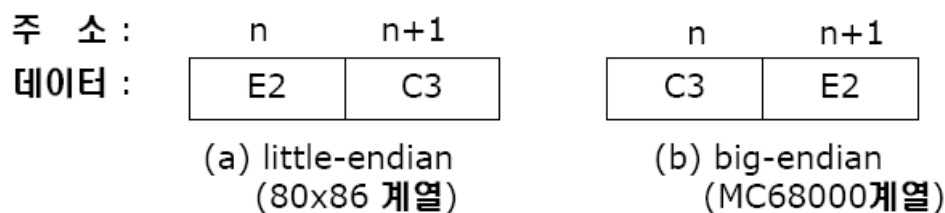


그림 113 0xC3E2(십진수 50146)의 호스트 바이트 순서 비교

2. 네트워크 바이트 순서

네트워크 바이트 순서는 포트번호나 IP 주소와 같은 정보를 바이트 단위로 네트워크로 전송하는 순서를 말한다. 네트워크 바이트 순서는 high-order 바이트부터 전송한다.(즉 big-endian이다.)

예를 들어, 2 바이트 수 0xC3E2의 경우 C3, E2의 순서로 상위 바이트부터 전송한다. 이는 80x86 계열의 CPU가 사용하는 호스트 바이트 순서는 네트워크 바이트 순서와 다르기 때문에 80x86 계열의 컴퓨터에서 아무런 처리 없이 네트워크를 통하여 전송한 숫자를 MC68000 계열에서 수신하면 바이트 순서가 바뀌게 된다.

이러한 문제를 해결하기 위하여 컴퓨터 내부에서 사용하던 숫자(sockaddr_in 구조체 내의 포트번호 등)를 네트워크로 전송하기 전에 htons() 함수²⁾를 사용하여 모두 네트워크 바이트 순서로 바꾸어야 하고, 반대로 네트워크로부터 수신한 숫자는 ntohs() 함수를 사용하여 자신의 호스트 바이트 순서로 바꾸어야 한다. 모든 컴퓨터가 네트워크 바이트 순서를 지켜 숫자를 전송하도록 하는 것이다.

2) host to network short의 약자이다.

위에서 MC68000 계열의 CPU에서는 호스트 바이트 순서와 네트워크 바이트 순서가 같았으므로 이러한 호스트에서의 htons()와 ntohs() 함수는 실제로 아무런 작업도 하지 않게 된다.

바이트 순서를 바꾸는 함수에는 변환할 바이트 길이가 2 바이트인 경우 또는 4 바이트인 경우에 따라 다음과 같이 두 가지 종류가 있다.

- unsigned short integer 변환 (2 바이트 크기)
 - htons() : host-to-network 바이트 변환
 - ntohs() : network-to-host 바이트 변환
- unsigned long integer 변환 (4 바이트 크기)
 - htonl() : host-to-network 바이트 변환
 - ntohl() : network-to-host 바이트 변환

주의할 것은, 바이트 순서를 맞추는 같은 포트번호와 같이 어떤 숫자를 네트워크로 전송할 때이며 텍스트 문서나 바이너리 파일 등 일반 데이터는 바이트 변환이 필요 없다는 것이다. 일반 데이터는 메모리(버퍼)에 저장되었다가 바이트 단위로 메모리 앞 주소부터 차례대로 전송되며 수신측에서도 수신된 데이터를 메모리에 바이트 순서대로 차례로 저장하기 때문이다.

3. 바이트 순서 바꾸기 프로그램

4 바이트로 표현된 숫자의 바이트 순서를 바꾸는 프로그램을 작성해 보자. 예를 들어 16진수 0x12345678을 입력하면 0x78563412가 출력되도록 하는 것이다.

가. 커맨드 라인 인수 사용법

C로 제작된 프로그램의 시작지점인 main() 함수에서 커맨드 라인 명령을 얻어오는 방법을 알아보자. 커맨드 창에서 입력한 명령을 얻어오려면 main()함수의 인자를 아래와 같이 정의하면 된다.

```
int main( int argc, char* argv[] );
```

argc에는 실행파일명을 포함한 인자의 개수가 저장된다. 아무런 인자가 없이 실행되었을 때에는 1이 저장된다. argv는 실행파일명을 포함한 인자가 char* 형식의 스트링이 배열로 저장된다. argv[0]은 실행파일명에 대한 스트링을 가리킨다.

예를 들어 커맨드 창에서 "cp srcfilename destfilename" 명령을 내리게 되면 각 변수에는 아래와 같은 값이 입력되게 된다.

```

· argc == 3
· argv[0] == "cp"
· argv[1] == "srcfilename"
· argv[2] == "destfilename"

```

나. 문자열 변환 함수

커맨드 창을 통해 입력된 16진수는 문자열이기 때문에 unsigned long형 변수를 인자로 취하는 htonl() 함수에서는 바로 사용될 수 없다. 따라서 char*형 문자열을 unsigned long형 숫자로 바꿔주는 함수를 만들어 사용하도록 한다.

예제 4 char*형 문자열을 unsigned long형으로 바꿔주는 함수

```

int str2num( char* str )
{
    int nInt = 0;
    int nSeed = 0;
    char cChar;
    int i;

    for( i = 0; str[i]; i++ )
    {
        nInt = nInt << 4;
        switch( str[i] )
        {
            case 'F':
                nSeed = 15;
                break;
            case 'E':
                nSeed = 14;
                break;
            case 'D':
                nSeed = 13;
                break;
            case 'C':
                nSeed = 12;
                break;
            case 'B':
                nSeed = 11;
                break;
            case 'A':
                nSeed = 10;
                break;
            default:
                cChar = str[i];
                nSeed = cChar - 48;
                break;
        }
        nInt += nSeed;
    }

    return nInt;
}

```

다. htonl() 함수를 이용해 바이트 순서 변환

실제 htonl() 함수를 이용해 바이트 순서를 변환하여 반환하는 함수이다.

예제 5 htonl() 함수를 이용해 바이트 순서를 변환하는 함수

```
unsigned int convert_order32( unsigned int before )
{
    unsigned int nReturn = htonl( before );
    return nReturn;
}
```

라. 전체 소스

위의 str2num() 함수와 convert_order32() 함수를 이용한 전체 소스이다.

예제 6 바이트 순서 변환 전체 소스

```
// 포함 파일
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

// 함수 프로토타입
unsigned int convert_order32( unsigned int before );
int str2num( char* str );

// 메인 함수
int main( int argc, char* argv[] )
{
    int nChanged = convert_order32( str2num(argv[1]) );
    printf( "%x\n", nChanged );
    return 0;
}

// char*형 문자열을 unsigned long형으로 바꿔주는 함수
int str2num( char* str )
{
    int nInt = 0;
    int nSeed = 0;
    char cChar;
    int i;

    for( i = 0; str[i]; i++ )
    {
        nInt = nInt << 4;
        switch( str[i] )
        {
            case 'F':
                nSeed = 15;
                break;
            case 'E':
                nSeed = 14;

```

```

        break;
    case 'D':
        nSeed = 13;
        break;
    case 'C':
        nSeed = 12;
        break;
    case 'B':
        nSeed = 11;
        break;
    case 'A':
        nSeed = 10;
        break;
    default:
        cChar = str[i];
        nSeed = cChar - 48;
        break;
    }
    nInt += nSeed;
}

return nInt;
}

// htonl() 함수를 이용해 바이트 순서를 변환하는 함수
unsigned int convert_order32( unsigned int before )
{
    unsigned int nReturn = htonl( before );
    return nReturn;
}

```

마. 실행 화면



```

root@localhost:~/network
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./2-1 12345678
78563412
[root@localhost network]# ./2-1 ABCDEF12
12efcdab
[root@localhost network]# ./2-1 7890
90780000
[root@localhost network]#

```

4. 호스트 바이트 순서 확인 프로그램

위의 바이트 순서 바꾸기 프로그램을 응용하면 호스트 컴퓨터의 바이트 순서를 확인할 수 있는 프로그램의 작성이 가능하다. 네트워크 바이트 순서는 big-endian이기 때문에 임의의 수를 htonl() 함수의 인자로 입력하여 같은 수가 반환되면 호스트 컴퓨터의 바이트 순서 또한 big-endian이고, 다른 수가 반환되면 little-endian으로 판단한다.

가. 전체 소스

예제 7 바이트 순서 확인 전체 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

unsigned int convert_order32( unsigned int before );
int str2num( char* str );

int main( int argc, char* argv[] )
{
    int before = str2num( "12345678" );
    int after = convert_order32( before );
    if( before == after )
        printf( "이 컴퓨터는 big-endian을 사용합니다.\n" );
    else
        printf( "이 컴퓨터는 little-endian을 사용합니다.\n" );

    return 0;
}

int str2num( char* str )
{
    int nInt = 0;
    int nSeed = 0;
    char cChar;
    int i;

    for( i = 0; str[i]; i++ )
    {
        nInt = nInt << 4;
        switch( str[i] )
        {
            case 'F':
                nSeed = 15;
                break;
            case 'E':
                nSeed = 14;
                break;
            case 'D':
```



```

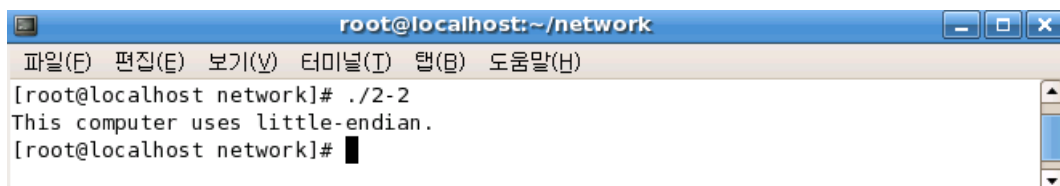
        nSeed = 13;
        break;
    case 'C':
        nSeed = 12;
        break;
    case 'B':
        nSeed = 11;
        break;
    case 'A':
        nSeed = 10;
        break;
    default:
        cChar = str[i];
        nSeed = cChar+48;
        break;
    }
    nInt += nSeed;
}

return nInt;
}

unsigned int convert_order32( unsigned int before )
{
    unsigned int nReturn = htonl( before );
    return nReturn;
}

```

나. 실행 화면



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./2-2
This computer uses little-endian.
[root@localhost network]#

```

SECTION

03 도메인이름과 IP주소 상호 변환하기

1. IP 주소 변환

4바이트(32비트)의 IP 주소는 편의를 위해 cse.hufs.ac.kr과 같은 도메인 이름 또는 203.253.64.1과 같은 dotted decimal 방식으로 표현되어 사용되어 진다. dotted decimal 표현은 4개의 숫자 변수가 아니라 15개의 문자로 구성된 문자열 변수가 사용된다. 소켓 프로그램에서는 이들 주소 표현법을 상호 변환할 수 있는 함수를 제공하고 있는데 아래 그림은 이들 세 가지 인터넷 주소 표현 방법과 이들을 상호 변환해 주는 네 개의 주소변환 함수들을 나타낸다. 실제 소켓 프로그램에서는 IP 데이터그램을 네트워크로 전송할 때 IP 헤더의 4바이트 바이너리 IP주소만 사용한다는 점을 유의해야 한다.

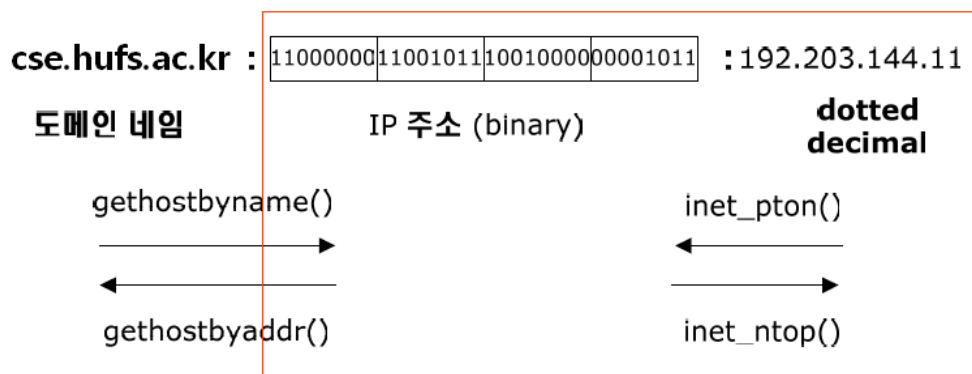


그림 116 IP주소 표현의 세 가지 방법 및 이들의 상호변환 함수

예를 들어, 위 그림에서 dotted decimal로 표현된 192.203.144.11을 32비트의 IP주소로 변환하려면 inet_pton() 함수를 사용하고, IP주소를 dotted decimal로 변환하려면 inet_ntop()를 사용한다. ntop은 numerical to presentation의 약자이다. 다음은 이 함수들의 사용 문법이다.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

const char* inet_ntop( int af, const void* src, char* dst, size_t cnt );
int inet_pton( int af, const char* src, void* dst );
```

위와 비슷한 기능을 수행하는 함수로 inet_addr(), inet_ntoa(), inet_aton() 등이 과거에 사용되었는데, inet_addr()은 255.255.255.255 주소를 처리하지 못하는 문제가 있고 inet_ntoa()는 멀티스레딩 환경에서 스레드가 정지된 후 재진입이 불가능하다는 단점이 있다. 또한 inet_addr(), inet_ntoa(), inet_aton()

들은 IPv4에서만 사용할 수 있지만 `inet_ntop()`, `inet_pton()`은 IPv6의 주소까지 처리할 수 있다.

아래의 예제는 dotted decimal로 표현된 주소(예: 210.115.36.23)를 명령문 인자로 입력하면 이것을 `inet_pton()` 함수를 이용하여 4바이트의 IP주소(16진수로는 E72473D2)로 바꾸어 화면에 출력하고, 이 IP 주소로부터 `inet_ntop()`를 호출하여 다시 dotted decimal 주소를 얻는 프로그램이다.

가. 소스 코드

예제 8 ASCII(dotted decimal)로 표현된 주소를 4바이트 IP주소로 변환

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    struct in_addr inaddr;      // 32비트 IP 주소 구조체
    char buf[20];

    if(argc < 2)
    {
        printf("사용법 : %s IP 주소(dotted decimal) \n", argv[0]);
        exit(0);
    }

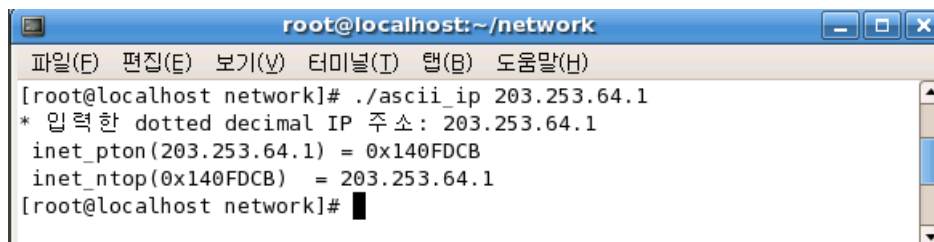
    printf("* 입력한 dotted decimal IP 주소: %s\n", argv[1]);

    inet_pton(AF_INET, argv[1], &inaddr.s_addr);
    printf(" inet_pton(%s) = 0x%X\n", argv[1], inaddr.s_addr);

    inet_ntop(AF_INET, &inaddr.s_addr, buf, sizeof(buf));
    printf(" inet_ntop(0x%X) = %s\n", inaddr.s_addr, buf);

    return 0;
}
```

나. 실행 화면



```
root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./ascii_ip 203.253.64.1
* 입력한 dotted decimal IP 주소: 203.253.64.1
inet_pton(203.253.64.1) = 0x140FDCB
inet_ntop(0x140FDCB) = 203.253.64.1
[root@localhost network]#
```

2. 도메인 주소 변환

앞에서 사용한 주소변환 함수들은 단순히 계산만 하는 함수이다. 즉, 십진수의 dotted decimal 주소와 4바이트의 이진수를 계산에 의하여 바꾸는 작업만 하면 되며 이러한 함수는 즉시 처리가 가능하다. 그러나 도메인 네임으로부터 IP 주소를 얻거나 IP 주소로부터 해당 호스트의 도메인 네임을 얻으려면 DNS(Domain Name Service) 서버의 도움을 받아야 하며 이러한 함수는 처리가 즉시 이루어지지 않을 수도 있다. `gethostbyname()`은 도메인 네임으로부터 IP 주소를 얻는 함수이고, `gethostbyaddr()`은 반대로 IP 주소로부터 도메인 네임을 얻는 함수이다. 이들 두 함수는 호스트의 각종 정보를 저장한 구조체 `hostent`의 포인터를 리턴하는데 이 두 함수의 사용 문법은 아래와 같다.

```
#include <netdb.h>
struct hostent* gethostbyname( const char* hname );
struct hostent* gethostbyaddr( const char* in_addr, int len, int family );
```

3. gethostbyname()

위에서 `gethostbyname()`은 도메인 네임 `hname`을 스트링 형태로 입력받고 그 이름에 해당하는 호스트의 각종 정보를 가지고 있는 `hostent` 구조체의 포인터를 리턴한다.

`gethostbyaddr()`은 IP 주소를 포함하고 있는 구조체 `in_addr`의 포인터와 이 주소의 길이, 주소 타입을 입력하여 해당 호스트의 정보를 가지고 있는 `hostent` 구조체의 포인터를 리턴한다. 이와 같이 IP 주소로부터 호스트 정보(도메인 네임 등)를 얻기 위해서는 Reverse DNS가 수행되어야 하는데 최근에는 보안 문제로 대부분의 DNS 서버가 Reverse DNS 기능을 제공하지 않고 있다.

구조체 `hostent`의 정의는 다음과 같다. `h_addr`은 호스트의 첫 번째 IP 주소 즉, 호스트의 대표 주소인 `h_addr_list[0]`을 가리킨다.

```
struct hostent {
    char* h_name;           // 호스트 이름
    char** h_aliases;       // 호스트 별명들
    int h_addrtype;         // 호스트 주소의 종류 (AF_INET=2 등)
    int h_length;           // 주소의 크기 (바이트 단위이며 IPv4에서는 4임)
    char** h_addr_list;     // IP 주소 리스트
};
#define h_addr h_addrlist[0] // 첫 번째(대표) 주소
```

도메인 네임으로부터 해당 호스트의 `hostent` 구조체 정보를 구한 후 `hostent` 구조체의 내용을 모두

출력해 보는 예제 프로그램을 소개한다.

가. 소스 코드

예제 9 도메인 이름을 IP 주소로 변환

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>          // memcpy 함수 선언
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <errno.h>

int main(int argc, char *argv[])
{
    struct hostent *hp;
    struct in_addr in;
    int i;
    char    buf[20];

    if(argc<2)
    {
        printf("Usage: %s  hostname\n",argv[0]);
        exit(1);
    }

    hp = gethostbyname(argv[1]);
    if(hp==NULL)
    {
        printf("gethostbyname fail\n");
        exit(0);
    }

    printf("호스트 이름           : %s\n", hp->h_name);
    printf("호스트 주소타입 번호: %d\n", hp->h_addrtype);
    printf("호스트 주소의 길이   : %d 바이트\n", hp->h_length);

    for( i=0; hp->h_addr_list[i]; i++)
    {
        memcpy(&in.s_addr, hp->h_addr_list[i],sizeof(in.s_addr));
        inet_ntop(AF_INET, &in, buf, sizeof(buf));
        printf("IP 주소(%d 번째)       : %s\n",i+1,buf);
    }

    for( i=0; hp->h_aliases[i]; i++)
        printf("호스트 별명(%d 번째) : %s ",i+1, hp->h_aliases[i]);

    puts("");
}
```

```

    return 0;
}

```

프로그램은 먼저 목적지 호스트의 도메인 네임을 명령 인자로 받고 `gethostbyname()`을 이용하여 목적지 호스트의 `hostent` 구조체를 얻는다. 다음에는 `hostent` 내에 있는 호스트 이름, 별명(있다면), 주소 체계, dotted decimal 인터넷 주소를 화면에 출력한다.

나. 실행 화면

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./get_hostent cse.hufs.ac.kr
호스트 이름      : cse.hufs.ac.kr
호스트 주소타입 번호 : 2
호스트 주소의 길이  : 4 바이트
IP 주소 (1 번째)   : 220.67.128.102
[root@localhost network]#

```

4. gethostbyaddr()

입력된 dotted decimal 주소로부터 `gethostbyaddr()`을 이용하여 해당 호스트를 찾아 도메인 네임을 얻는 프로그램을 아래에 소개한다. 한편, 현재 사용 중인 자신의 컴퓨터의 도메인 이름을 얻으려면 `uname()`이나 `gethostname()`을 이용하면 된다.

가. 소스 코드

예제 10 `hostent` 구조체 내용 출력 프로그램

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    struct hostent  *myhost;
    struct in_addr  in;

    if(argc < 2)
    {
        printf("사용법 : %s ip_address \n", argv[0]);
        exit(0);
    }

    inet_pton(AF_INET, argv[1], &in.s_addr); // dotted decimal->32bit 주소

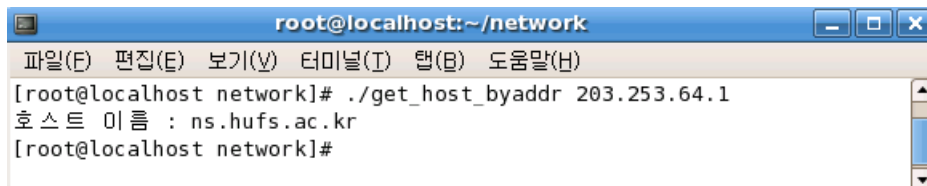
```

```
myhost = gethostbyaddr((char *)&(in.s_addr), sizeof(in.s_addr), AF_INET);

if (myhost == NULL)
{
    printf("Error at gethostbyaddr() \n");
    exit(0);
}

printf("호스트 이름 : %s\n", myhost->h_name);
return 0;
}
```

나. 실행 화면



The screenshot shows a terminal window titled "root@localhost:~/network". The window has a menu bar with options: 파일(F), 편집(E), 보기(V), 터미널(T), 탭(B), 도움말(H). The terminal content shows the execution of a program that takes an IP address as input and returns the host name. The input is "203.253.64.1" and the output is "호스트 이름 : ns.hufs.ac.kr".

```
root@localhost:~/network
[root@localhost network]# ./get_host_byaddr 203.253.64.1
호스트 이름 : ns.hufs.ac.kr
[root@localhost network]#
```

SECTION

04 TCP 클라이언트 프로그램

여기서는 TCP 클라이언트 프로그램의 일반적인 작성 절차를 설명하고, 유닉스 서버가 제공하는 daytime 및 에코 서비스를 이용하는 클라이언트 프로그램 tcp_daytimecli.c와 echocli.c를 각각 소개한다.

1. TCP 클라이언트 프로그램 작성 절차

다음 그림은 TCP 소켓을 사용하는 클라이언트 프로그램의 일반적인 작성 절차를 나타낸 것이다. 클라이언트는 먼저 `socket()`으로 소켓을 개설하고 `connect()`를 호출하여 서버에게 연결을 요청한다. 연결이 이루어지면 `send()`와 `recv()`를 사용하여 데이터를 송수신하고 작업이 종료되면 `close()`로 소켓을 닫는다.

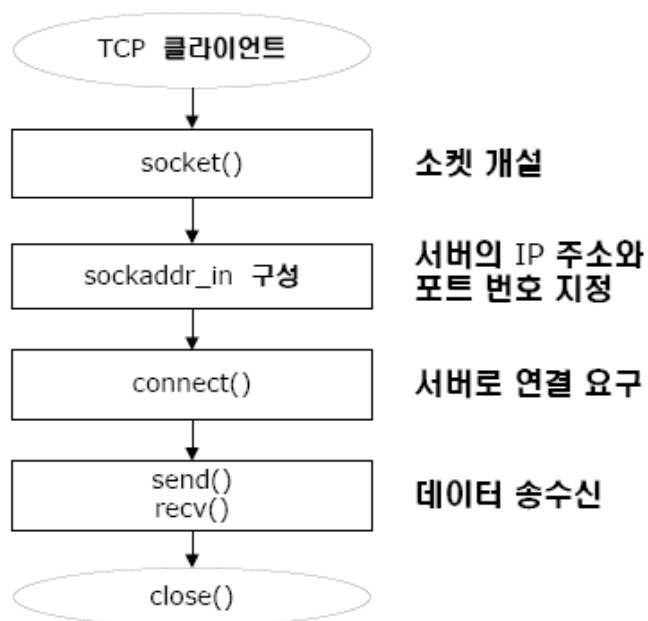


그림 120 TCP(연결형) 클라이언트 프로그램
작성 절차

가. `socket()`, 소켓 개설

`socket()`으로 소켓을 개설할 때 연결형(TCP) 또는 비연결형(UDP) 소켓을 선택해야 하며 연결형 소켓을 개설하려면 서비스 type 인자는 `SOCK_STREAM`으로 선택한다.

소켓을 이용한 통신프로그램에서는 사용할 전송 프로토콜, 자신의 IP 주소와 포트번호, 상대방의 IP 주소와 포트번호 등 다섯 개의 정보가 지정되어야 한다고 하였다. 소켓을 만들 때에는 이 중에서 전송 프로토콜만을 지정하게 된다. 밑의 그림은 `socket()` 수행 시 내부적으로 일어나는 동

작을 나타내는데, 응용 프로그램이 프로토콜을 지정하여 socket()을 호출하면 소켓 인터페이스가 새로 생성된 소켓의 소켓번호를 리턴하는 것을 보여주고 있다.

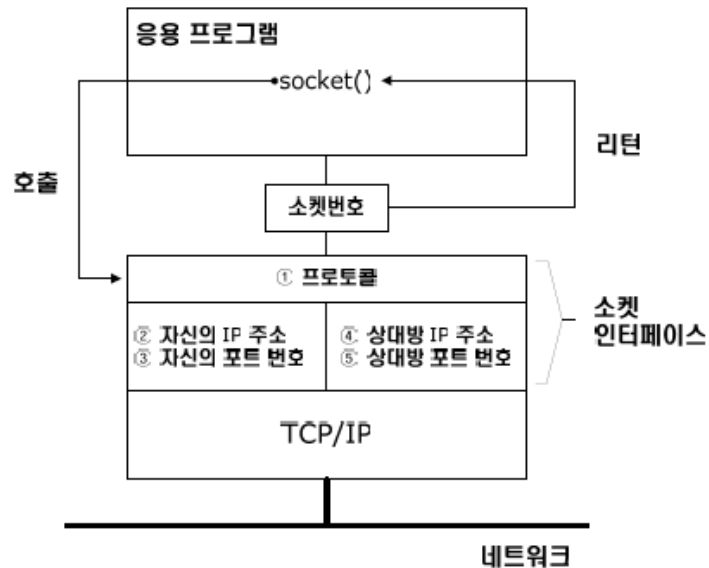


그림 121 socket() 호출시 소켓번호와
소켓 인터페이스의 관계

나. connect(), 서버에 연결 요청

클라이언트는 connect()를 호출하여 서버에게 연결요청을 하며 이때 3-way 핸드셰이크가 시작된다. connect()의 사용 문법은 다음과 같다.

```
int connect (
    int s,                // 서버와 연결시킬 소켓번호
    const struct sockaddr* addr, // 상대방 서버의 소켓주소 구조체
    int addrlen );        // 구조체 *addr의 크기
```

인자 addr는 연결한 서버의 IP 주소와 2바이트의 포트번호를 포함하고 있는 소켓주소 구조체 sockaddr_in을 지정한다. 인자 s는 소켓번호이며 연결이 이루어지고 나면 앞으로 이 소켓번호를 사용해서 서버와 통신을 하게 된다. 다음 그림에 connect()의 수행 내용을 나타냈다.

3-way 핸드셰이크가 성공하여 서버와 연결이 되면 connect() 는 0을 리턴한다. 실패하면 -1을 리턴하며 전역변수 errno에 에러코드가 들어 있게 된다. 서버와 연결이 되려면 서버측에서는 listen()과 accept()를 호출해 두고 있어야 한다.

connect() 호출 중에 에러가 발생하였을 때는 바로 다시 connect()를 호출하지 말고 해당 소켓을 close()로 닫고 새로운 소켓을 socket()으로 만든 후 사용하는 것이 안전하다.

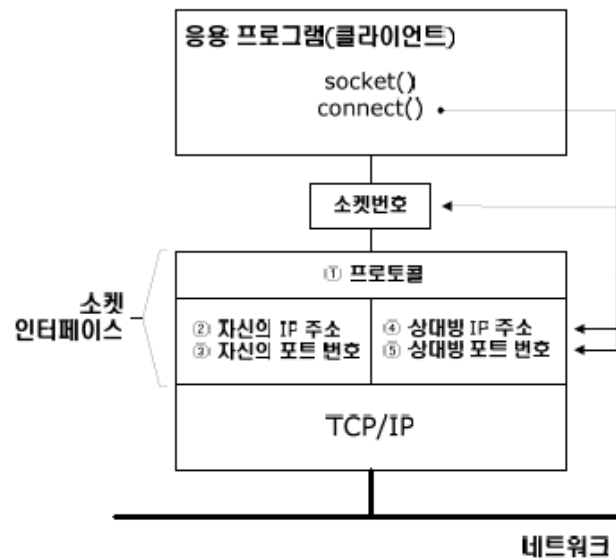


그림 122 connect() 호출시 소켓번호와
소켓주소의 관계

다. send(), recv(), 데이터 송수신

클라이언트가 서버와 연결되면 send(), recv() 또는 write(), read()를 사용하여 서버와 데이터를 송수신할 수 있는데 이 함수들을 표에 정리하였다.

| 문법 | 인자 | |
|---|--------|-----------------|
| int send(int s, char* buf, int length, int flags); | s | 소켓번호 |
| | buf | 전송할 데이터가 저장된 버퍼 |
| | length | buf 버퍼의 크기 |
| | flags | 보통 0 |
| int write(int s, const void* buf, int length); | s | 소켓번호 |
| | buf | 전송할 데이터가 저장된 버퍼 |
| | length | buf 버퍼의 길이 |
| int recv(int s, char* buf, int length, int flags); | s | 소켓번호 |
| | buf | 수신 데이터를 저장한 버퍼 |
| | length | buf 버퍼의 길이 |
| | flags | 보통 0 |
| int read(int s, void* buf, int length); | s | 소켓번호 |
| | buf | 수신 데이터를 저장할 버퍼 |
| | length | buf 버퍼의 길이 |

표 40 TCP(스트림) 소켓의 데이터 송수신 함수

send()와 write()는 스트림형(TCP) 소켓을 통하여 데이터를 송신하는 함수이며 데이터를 전송할 소켓 번호(s), 송신할 데이터 버퍼(buf), 전송할 데이터 크기(length)를 지정해야 한다. send(), write() 두 함수의 동작은 같다. 단 send()는 flags 인자를 추가로 사용할 수 있다는 점만 다르다. 두 함수 모두 실제로 전송된 데이터 크기를 바이트 단위로 리턴한다.

recv()와 read()는 스트림형 소켓을 통하여 데이터를 수신하는 함수이며 데이터를 수신할 소켓번호(s),

수신버퍼(buf), 읽을 데이터의 크기(length)를 지정해야 한다. 두 함수 모두 실제로 읽은 데이터 크기를 바이트 단위로 리턴한다.

send()와 recv()함수에서는 flags 옵션을 지정할 수 있는데 이는 일반 데이터를 다룰 때에는 0으로 선택한다. flags는 데이터 송수신시에 특별한 옵션을 지정할 때 사용된다. 예를 들어, recv() 호출 시에 flags를 MSG_PEEK로 설정하면 수신된 데이터를 읽은 후에 데이터를 수신버퍼에 그대로 남겨둔다. 즉, 다음번 read()에서도 같은 데이터를 다시 읽을 수 있도록 한다. flags 사용에 대해서는 추후 소켓 옵션에서 설명한다.

한 IP 데이터그램에 실어서 전송할 수 있는 최대 데이터 크기인 최대 세그먼트 크기(MSS: Maximum Segment Size)보다 큰 데이터를 write()나 send()로 보내면 전체 데이터가 MSS 크기로 분할되어 여러번에 나누어 전송된다. 주의할 것은 TCP에서는 세그먼트의 순서 확인과 데이터그램 분실을 수신측에서 검사하고 재전송을 함으로써 신뢰할만한 통신을 제공하나 UDP를 사용하면 이러한 예러제어는 제공되지 않는다는 것이다.

TCP 소켓에서 write()나 send()를 실행하면 데이터는 먼저 TCP 계층에 있는 송신버퍼(send buffer)로 들어간다. 만일 송신버퍼가 비어 있지 않아 데이터를 이곳에 쓸 수 없으면 프로세스는 블록 상태로 가며 프로그램은 write() 문에서 기다리게 된다(이것은 소켓 동작 모드가 블록형일 때이며 소켓을 넌 블록형 모드로 바꾸었을 때에는 블록되지 않는다). write() 문이 블록된 경우에 송신버퍼에 먼저 들어있던 데이터가 전송되고 write한 데이터가 송신버퍼로 모두 이동하면 write()문이 리턴된다. 즉, write()문이 리턴되었다는 것은 데이터가 이제 비로소 자신의 TCP에 있는 송신버퍼에 들어갔다는 것을 의미하며 데이터가 목적지에 전달된 것이 아님을 주의하여야 한다.

라. close(), 소켓 닫기

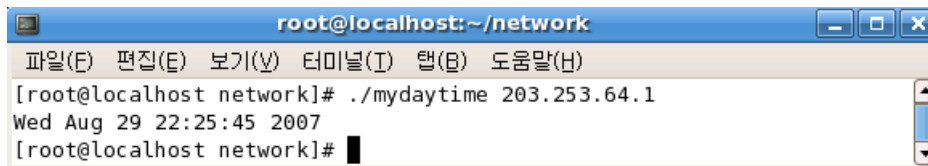
소켓의 사용을 마치려면 해당 소켓번호를 지정하여 close()를 호출하여 소켓을 닫아야 한다. close()는 클라이언트나 서버 중 누구나 먼저 호출할 수 있다. close()를 호출한 시점에 송신버퍼에 들어있으나 아직 전송되지 못한, 또는 네트워크 내에서 전달 중인 데이터가 있을 수 있는데, 기본적으로는 이러한 데이터가 모두 전달된 후에 TCP 연결이 종료된다. 소켓 옵션을 변경하여 미전송된 데이터를 모두 버리고 종료하도록 할 수도 있다(소켓 옵션에 대해서는 추후 다루게 된다).

2. TCP 클라이언트 예제 프로그램

가. daytime 클라이언트

유닉스 서버가 제공하는 daytime 서비스를 이용하는 클라이언트 프로그램을 작성한다. daytime 서비스를 제공하는 서버의 dotted decimal IP 주소를 명령문 인자로 입력하면 오늘 날짜와 현재 시각을 리턴한다.

다음은 프로그램을 실행한 결과이다.



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./mydaytime 203.253.64.1
Wed Aug 29 22:25:45 2007
[root@localhost network]#

```

이 프로그램에서는 사용할 소켓을 아래와 같이 개설하는데, 프로토콜 체계로는 인터넷을 , 서비스는 연결형(TCP)을 지정하고 있다.

```
s = socket( PF_INET, SOCK_STREAM, 0 );
```

다음에는 연결할 서버의 소켓주소 구조체를 만들어야 하는데 주소 체계를 인터넷(AF_INET)으로 지정하고 서버의 IP 주소와 포트번호를 소켓주소 구조체 servaddr에 기록하였다.

```

struct sockaddr_int servaddr;           // 서버의 소켓주소 구조체
servaddr.sin_family = AF_INET;         // 주소 체계 선택
inet_pton( AF_INET, argv[1], &servaddr.sin_addr ); // 32비트 IP 주소로 변환
servaddr.sin_port = htons(13);         // daytime 서비스 포트번호

```

위에서 inet_pton() 함수는 사용자가 명령 인자로 입력한 dotted decimal 형태의 IP 주소 문자열 argv[1]을 32비트 IP 주소로 변환하는 함수이다. daytime 서비스를 받기 위하여 well-known 포트 번호 13번을 지정하였다.

이상과 같이 서버의 주소 정보를 모두 servaddr 구조체에 기록한 다음에는 서버에 연결을 요청하기 위해 connect() 함수를 다음과 같이 호출한다.

```
connect( s, (struct sockaddr*)&servaddr, sizeof(servaddr) );
```

서버와 연결된 후, 서버가 보내오는 문자열(날짜와 시간)을 수신하기 위해 클라이언트는 read()를 아래와 같이 호출하는데, 소켓을 통한 입출력도 파일 입출력과 유사하게 이루어지는 것을 알 수 있다.

```
n = read( s, buf, sizeof(buf) ); // recv(s, buf, sizeof(buf), 0 ); 도 같은 동작은 한다.
```

1) 소스 코드

예제 11 daytime 서비스를 요청하는 TCP(연결형) 클라이언트

```
#include <stdio.h>
```

```

#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MAXLINE 127

int main(int argc, char *argv[])
{
    int s, nbyte;
    struct sockaddr_in servaddr;
    char buf[ MAXLINE+1 ];

    if(argc != 2) {
        printf("Usage: %s ip_address\n", argv[0]);
        exit(0);
    }

    // 소켓 생성
    if((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(1);
    }

    // 서버의 소켓주소 구조체 servaddr을 '\0'으로 초기화
    bzero((char *)&servaddr, sizeof(servaddr));

    // servaddr의 주소 지정
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    servaddr.sin_port = htons(13);

    // 연결요청
    if(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("connect fail");
        exit(1);
    }

    // 서버가 보내오는 daytime 데이터의 수신 및 화면출력
    if((nbyte=read(s, buf, MAXLINE )) < 0) {
        perror("read fail");
        exit(1);
    }

    buf[nbyte] = 0;
    printf("%s", buf);
}

```

```

    close(s);
    return 0;
}

```

위에서 `bzero()`는 어떤 포인터가 가리키는 구조체의 내용을 모두 0으로 설정하기 위해 사용하였는데 `bzero()` 대신 `memset()`을 사용할 수도 있다. 참고로 C 언어에서는 문자열 즉, string을 다룰 때 문자열의 끝에 NULL 문자를 자동으로 추가하므로 통신 프로그램에서 사용하는 정보(예: 2바이트로 된 포트번호, 4바이트로 된 IP 주소 등)를 C 스트링으로 다루면 안된다(즉, `strcpy()` 등을 사용할 수 없다).

`connect()` 문에서 소켓주소 구조체를 나타내는 함수 인자로 `servaddr`를 바로 사용하지 않고 (`struct sockaddr*`)&`servaddr`와 같이 캐스팅(casting)하여 사용한 것을 볼 수 있다. 이렇게 하는 이유는 대부분의 인터넷 소켓 프로그램에서는 인터넷 주소를 편리하게 다루기 위하여(IP 주소와 포트번호를 직접 쓰거나 읽을 수 있도록) `sockaddr_in` 구조체를 사용하고 있으나, `connect()` 함수를 비롯한 각종 소켓 함수의 정의에서는 일반적인 소켓주소 구조체인 `sockaddr`를 사용하도록 정의되어 있기 때문이다. 따라서 `sockaddr_in` 구조체를 `sockaddr` 타입으로 캐스팅 하는 것이 필요하다. `sockaddr`와 `sockaddr_int` 두 구조체 모두 16바이트로 구성되어 있어 크기가 같다.

한편 프로그램을 실행시켰을 때 아래와 같은 에러 메시지가 출력되는 경우가 있다.

```

$ tcp_daytimecli 203.252.65.3
    connect fail: Connection refused

```

이것은 목적지 호스트(203.252.65.3)에서 (주로 보안상의 이유로) daytime 서비스를 제공하지 않기 때문이다. 즉, 13번 포트가 비활성 상태이므로 연결이 거부된 것이다. 만약 목적지 주소를 127.0.0.1로 하였을 때에도 "Connection refused" 메시지가 출력된다면 로컬 호스트가 daytime 서비스를 하고 있지 않은 것이다. 로컬 호스트에서 daytime 서비스를 하고 있다면 아래와 같이 출력될 것이다.

```

$ tcp_daytimecli 127.0.0.1
Wed Jan 1 09:00:21 2007

```

daytime 서비스가 13번 포트번호를 사용하는 것은 다음과 같이 `/etc/services` 파일을 통해 확인할 수 있다.

```

$ cat /etc/services | grep daytime
    daytime      13/tcp
    daytime      13/udp

```

나. TCP 에코 클라이언트

TCP 클라이언트 프로그램의 두 번째 예로 에코 서비스를 이용하는 클라이언트 프로그램을 소개한다. 표준 인터넷 서비스인 에코는 well-known 포트 7번을 통해 제공되며 클라이언트가 보낸 문자열을 다시 클라이언트로 전송한다. 다음은 프로그램을 실행한 결과 예이다.

```
$ tcp_echo 210.115.36.231
입력: abcdefghijklmn
수신메시지: abcdefghijklm
```

이 프로그램도 앞에서 설명한 daytime 프로그램과 거의 유사하게 작성된다. 한 가지 차이점은 이 프로그램에서는 에코 서비스를 이용해야 하므로 서버의 소켓주소 구조체 servaddr의 포트번호를 아래와 같이 7로 바꾸어야 한다.

```
servaddr.sin_port = htons( 7 );
```

connect()를 호출한 후 서버와 연결이 완료되면 서버로 전송할 문자열을 키보드를 통해 아래와 같이 입력받는다.

```
fgets( buf, sizeof(buf), stdin );
```

fgets() 함수는 스트링 문자열을 입력받는 함수로 첫 번째 인자(buf)는 문자열을 저장할 버퍼이고, 두 번째 인자는 버퍼의 크기, 그리고 세 번째 인자는 입력 스트림 포인터이다. 입력 스트림 포인터로 stdin을 지정하였는데 stdin은 FILE 포인터 타입의 전역변수로 표준 입력 파일인 키보드를 가리킨다.

fgets()로 키보드에서 입력받은 문자열을 write()를 이용해 서버로 전송하고 서버가 에코시켜준 문자열을 read()로 수신하여 화면에 출력한다.

1) 소스 코드

예제 12 에코 서비스를 요청하는 TCP(연결형) 클라이언트

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```

#include <netinet/in.h>
#include <unistd.h>

#define MAXLINE 127

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr;
    int s, nbyte;
    char buf[MAXLINE+1];

    if(argc != 2) {
        printf("usage: %s ip_address\n", argv[0]);
        exit(0);
    }

    if((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }
    // 에코 서버의 소켓주소 구조체 작성
    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    servaddr.sin_port = htons(7);

    // 연결요청
    if(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr))
        < 0) {
        perror("connect fail");
        exit(0);
    }

    printf("입력: ");
    if (fgets(buf, sizeof(buf), stdin) == NULL)
        exit(0);
    nbyte = strlen(buf);
    // 에코 서버로 메시지 송신
    if (write(s, buf, nbyte) < 0) {
        printf("write error\n");
        exit(0);
    }
    // 수신된 에코 데이터 화면출력
    printf("수신 : ");
    if( (nbyte=read(s, buf, MAXLINE)) < 0) {
        perror("read fail");
        exit(0);
    }
    buf[nbyte]=0;
    printf("%s", buf);
}

```

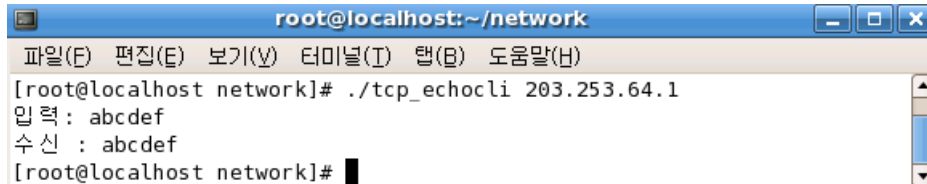


```

    close(s);
    return 0;
}

```

2) 실행 화면



```

root@localhost:~/network
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./tcp_echocli 203.253.64.1
입력: abcdef
수신 : abcdef
[root@localhost network]#

```

다. 포트번호 배정

클라이언트에서는 소켓에 특정 포트번호를 배정(bind)하지 않고 시스템이 임의로 배정하는 포트번호를 사용한다고 하였다. 포트번호는 TCP 소켓인 경우는 connect() 호출이 성공한 후에, UDP 소켓의 경우는 첫 번째 메시지를 보내는 sendto() 함수가 성공한 후에 배정된다. 아래는 이것을 확인하는 예제 프로그램인데 TCP와 UDP 클라이언트에서 시스템이 배정한 포트번호를 getsockname()을 이용하여 얻은 후 화면에 출력하고 있다.

참고로 getsockname()은 자신의 호스트에 있는 소켓 정보를 알아내는 함수인데 TCP로 연결된 상대방의 소켓 정보를 얻으려면 getpeername()을 사용하면 된다.

이 프로그램이 실행되기 위해서는 서버에서 TCP 및 UDP 에코 서비스가 동작하고 있어야 한다.

예제 13 시스템이 자동으로 배정한 포트번호를 출력하는 프로그램

```

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

#define MSG "Test Message"

int main() {
    int sd1, sd2 ;
    int addrlen;
    struct sockaddr_in servaddr, cliaddr; // 소켓주소 구조체
    unsigned short port1, port2; // 포트번호

    // 소켓주소 구조체 초기화
    servaddr.sin_family = AF_INET ;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY) ;
    servaddr.sin_port = htons(7);

```

```

sd1=socket(PF_INET, SOCK_STREAM, 0);
sd2=socket(PF_INET, SOCK_DGRAM, 0);

// TCP 소켓의 포트번호 얻기
if( connect(sd1, (struct sockaddr*)&servaddr,
    sizeof(servaddr))<0) {
    perror("connect fail");
    exit(1);
}

getsockname(sd1, (struct sockaddr*)&cliaddr, &addrlen) ;
port1 = ntohs(cliaddr.sin_port);

// UDP 소켓의 포트번호 얻기
sendto(sd2, MSG, strlen(MSG), 0,
    (struct sockaddr*)&servaddr, sizeof(servaddr));
getsockname(sd2, (struct sockaddr*)&cliaddr, &addrlen) ;
port2 = ntohs(cliaddr.sin_port);

printf("스트림 소켓 포트번호 = %d\n", port1) ;
printf("데이터그램 소켓 포트번호 = %d\n", port2) ;
close(sd1) ;
close(sd2) ;
return 0;
}

```

SECTION

05 TCP 서버 프로그램

1. TCP 서버 프로그램 작성 절차

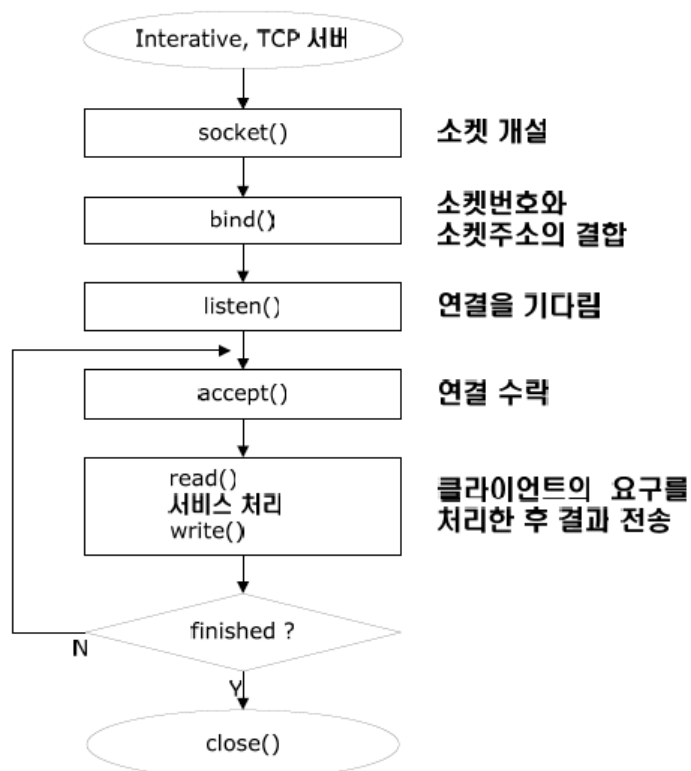


그림 125 Iterative 모델의 TCP 서버

가. socket(), 소켓의 생성

- 1) 서버도 클라이언트와 통신을 하기 위해 소켓을 생성한다.

```
socket( PH_INET, SOCK_STREAM, 0 );
```

나. bind

- 1) socket()으로 생성된 소켓의 소켓번호는 응용 프로그램만 알고 있다.
가) 컴퓨터 외부와 통신하기 위해 소켓번호와 소켓주소를 연결해 두어야 한다.

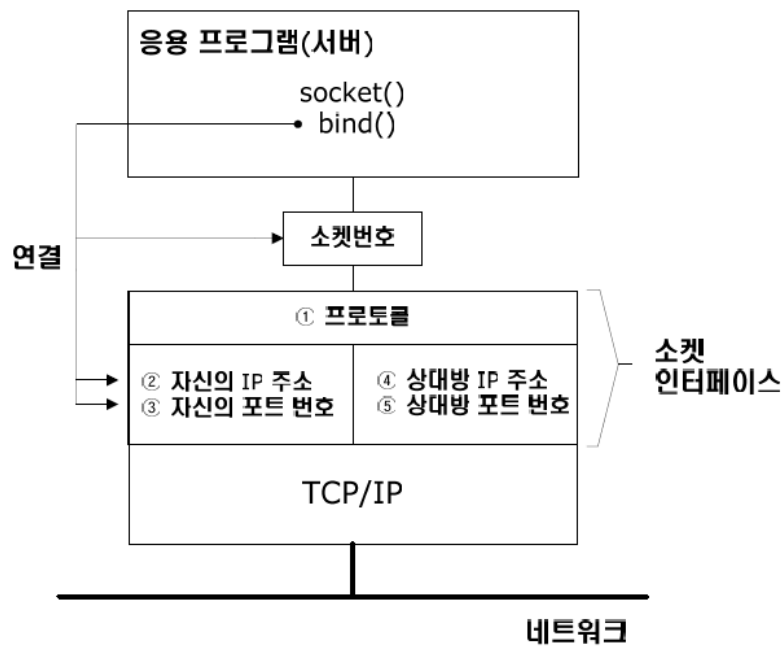


그림 126 bind() 호출시 소켓번호와 소켓주소의 관계

2) bind()가 필요한 이유

가) 임의의 클라이언트가 서버 프로그램의 특정 소켓으로 접속을 하려면 서버는 자신의 소켓번호와 클라이언트가 알고 있는 자신의 IP 주소 및 포트번호를 미리 연결해 두어야 한다.

3) bind() 사용 문법

```
int bind( int s, struct sockaddr* addr, int len );
```

가) int s : 소켓번호

나) struct sockaddr* addr : 서버 자신의 소켓주소 구조체 포인터

다) int len : addr 구조체의 크기

4) bind()는 성공시 0, 실패시 -1을 리턴

다. bind() 사용 예

예제 14 bind() 사용 예

```
#define SERV_IP_ADDR "210.115.49.220"
#define SERV_PORT 5000

// 소켓 생성
s=socket(PF_INET, SOCK_STREAM, 0);
struct sockaddr_in server_addr;

// 소켓 구조체 내용
server_addr.sin_family=AF_INET;
```

```
server_addr.sin_addr.s_addr=inet_addr(SERV_IP_ADDR);
server_addr.sin_port=htons(SERV_PORT);

// 소켓번호와 소켓 주소를 bind
bind(s, (struct sockaddr *)&server_addr, sizeof(server_addr));
```

라. listen()

1) 사용 문법

```
int listen( int s, int backlog );
```

가) int s : 소켓 번호

나) int backlog : 연결을 기다리는 클라이언트의 최대 수

2) 동작 순서

가) 클라이언트가 listen()을 호출해 둔 서버 소켓을 목적지로 connect()를 호출한다.

(1) 3-way 핸드셰이크 연결설정의 시작

나) 시스템이 핸드셰이크를 마친 후에는 서버 애플리케이션이 설정된 연결을 받아들인다.

(1) accept()가 사용 됨

다) accept()는 한 번에 하나의 연결만 가져간다.

(1) 여러 연결요청이 동시에 오면 시스템은 설정된 연결들을 accept 큐에 넣고 대기한다.

(2) backlog 인자는 대기시킬 수 있는 연결의 최대 수

3) listen은 소켓을 단지 수동 대기모드로 바꾸어 주는 역할만 한다.

가) 성공시 0, 실패시 -1을 리턴

마. accept()

1) 서버가 listen()의 호출 이후 클라이언트와 설정된 연결을 실제로 받기 위해 사용한다.

2) 사용문법

```
int accept( int s, struct sockaddr* addr, int* addrlen );
```

가) int s : 소켓번호

나) struct sockaddr* addr : 연결요청을 한 클라이언트의 소켓주소 구조체

다) int* addrlen : *addr 구조체 크기의 포인터

3) 특징

가) 수행 성공시 클라이언트와의 통신에 사용할 새로운 소켓이 생성된다.

(1) 실패시에는 -1이 리턴

나) 서버는 클라이언트와 통신하기 위해 새로 만들어진 소켓번호를 사용한다.

다) 연결된 클라이언트의 소켓주소 구조체와 소켓주소 구조체의 길이의 포인터를 리턴한다.

- (1) addr과 addrlen 인자로 리턴한다.
- (2) 서버는 addr 소켓주소 내용으로 연결된 클라이언트의 IP 주소를 알 수 있다.

2. TCP 에코 서버 프로그램

예제 15 TCP 에코 서버 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

#define MAXLINE 127

int main(int argc, char *argv[])
{
    struct sockaddr_in servaddr, cliaddr;
    int listen_sock, accp_sock,    // 소켓번호
        addrlen=sizeof(cliaddr),  // 주소구조체 길이
        nbyte;
    char buf[MAXLINE+1];

    if(argc != 2) {
        printf("usage: %s port\n", argv[0]);
        exit(0);
    }

    // 소켓 생성
    if((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }

    // servaddr을 '\0'으로 초기화
    bzero((char *)&servaddr, sizeof(servaddr));

    // servaddr 세팅
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(atoi(argv[1]));

    // bind() 호출
    if(bind(listen_sock, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail");
        exit(0);
    }
}
```

```
// 소켓을 수동 대기모드로 세팅
listen(listen_sock, 5);
// iterative 에코 서비스 수행
while(1) {
    puts("서버가 연결요청을 기다림..");
    // 연결요청을 기다림
    accp_sock = accept(listen_sock, (struct sockaddr *)&cliaddr, &addrlen);
    if(accp_sock < 0) {
        perror("accept fail");
        exit(0);
    }
    puts("클라이언트가 연결됨..");
    nbyte = read(accp_sock, buf, MAXLINE);
    write(accp_sock, buf, nbyte);
    close(accp_sock);
}
close(listen_sock);
return 0;
}
```

실행 화면은 다음과 같다.

```
root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./tcp_echoserv 7
서버가 연결요청을 기다림..
클라이언트가 연결됨..
서버가 연결요청을 기다림..
```

3. 연습문제 풀이

- 가. bind() 함수를 잘못 사용하는 경우 발생하는 에러코드를 확인해보시오. 아래의 테스트 항목별로 bind() 함수를 실행하고 bind() 함수가 실패한 경우 perror() 함수를 사용하여 에러의 내용을 출력하시오.

```
int bind(int s, struct sockaddr* addr, int len);
```

예제 16 bind() 함수 에러코드 확인

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

#define MAXLINE 127
```

```

int main(int argc, char *argv[])
{
    struct sockaddr_in servaddr, cliaddr;
    int listen_sock, accp_sock, addrlen=sizeof(cliaddr), nbyte;
    char buf[MAXLINE+1];

    if(argc != 2)
    {
        printf("usage: %s port\n", argv[0]);
        exit(0);
    }

    if((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket fail");
        exit(0);
    }

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

    if(bind(listen_sock, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
        perror("bind fail");
        exit(0);
    }

    if( bind( 4, (struct sockaddr*)&servaddr, sizeof(servaddr) ) < 0 )
        perror( "test1 bind fail: " );

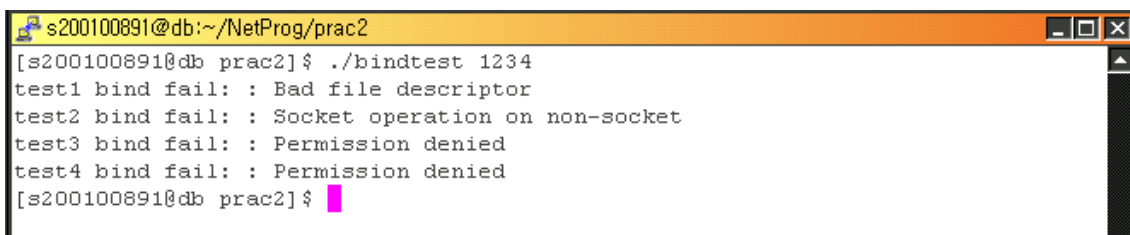
    if( bind( 1, (struct sockaddr*)&servaddr, sizeof(servaddr) ) < 0 )
        perror( "test2 bind fail: " );

    servaddr.sin_port = htons(80);
    if( bind( listen_sock, (struct sockaddr*)&servaddr, sizeof(servaddr) ) < 0 )
        perror( "test3 bind fail: " );

    servaddr.sin_port = htons(200);
    if( bind( listen_sock, (struct sockaddr*)&servaddr, sizeof(servaddr) ) < 0 )
        perror( "test4 bind fail: " );

    return 0;
}

```



```

s200100891@db:~/NetProg/prac2
[s200100891@db prac2]$ ./bindtest 1234
test1 bind fail: : Bad file descriptor
test2 bind fail: : Socket operation on non-socket
test3 bind fail: : Permission denied
test4 bind fail: : Permission denied
[s200100891@db prac2]$

```


나. TCP 에코 서버 프로그램과 TCP 에코 클라이언트 프로그램을 응용하여, 클라이언트 2개의 접속을 받아 메시지를 교환하는 프로그램을 작성하시오.

1) 문제 해결

- 가) 서버 프로그램에서 accept() 함수를 두 번 호출하여 클라이언트 2개의 접속을 기다린다.
- 나) 클라이언트가 접속하면 각각 accp_sock1, accp_sock2에 값을 저장한다.
- 다) client1에서 메시지를 서버로 전송한다.
- 라) server는 메시지를 받아 client2에게 그 메시지를 보낸다.
- 마) client2는 받은 메시지를 출력한다.
- 바) client2에서 메시지를 서버로 전송한다.
- 사) server는 메시지를 받아 client1에게 그 메시지를 보낸다.
- 아) client1은 받은 메시지를 출력한다.
- 자) 프로그램 종료.

예제 17 서버 프로그램 소스 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

#define MAXLINE 127

int main(int argc, char *argv[])
{
    struct sockaddr_in servaddr, cliaddr;
    int listen_sock, accp_sock1, accp_sock2, addrlen=sizeof(cliaddr), nbyte;
    char buf[MAXLINE+1];

    if(argc != 2)
    {
        printf("usage: %s port\n", argv[0]);
        exit(0);
    }
    if((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket fail");
        exit(0);
    }

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

    if(bind(listen_sock, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
```

```

        perror("bind fail");
        exit(0);
    }
    listen(listen_sock, 5);

    puts( "Server is starting." );
    accp_sock1 = accept( listen_sock, (struct sockaddr*)&cliaddr, &addrlen );
    if( accp_sock1 < 0 )
    {
        perror( "accept socket1 fail" );
        exit( 0 );
    }
    puts( "Client1 Connected." );

    accp_sock2 = accept( listen_sock, (struct sockaddr*)&cliaddr, &addrlen );
    if( accp_sock2 < 0 )
    {
        perror( "accept socket2 fail" );
        exit( 0 );
    }
    puts( "Client2 Connected." );

    nbyte = read(accp_sock1, buf, MAXLINE);
    write(accp_sock2, buf, nbyte);
    nbyte = read(accp_sock2, buf, MAXLINE);
    write(accp_sock1, buf, nbyte);

    close( accp_sock1 );
    close( accp_sock2 );
    close(listen_sock);

    return 0;
}

```

예제 18 클라이언트 프로그램 소스 코드

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <unistd.h>

#define MAXLINE 127

int main(int argc, char *argv[])
{
    struct sockaddr_in servaddr;
    int s, nbyte;
    char buf[MAXLINE+1];

    if(argc != 2)

```

```

{
    printf("usage: %s ip_address\n", argv[0]);
    exit(0);
}

if((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("socket fail");
    exit(0);
}

bzero((char *)&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
servaddr.sin_port = htons(atoi(argv[1]));

if(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
{
    perror("connect fail");
    exit(0);
}

if (fgets(buf, sizeof(buf), stdin) == NULL)
    exit(0);
nbyte = strlen(buf);

printf("입력: ");
if (write(s, buf, nbyte) < 0)
{
    printf("write error\n");
    exit(0);
}

printf("수신 : ");
if( (nbyte=read(s, buf, MAXLINE)) <0)
{
    perror("read fail");
    exit(0);
}
buf[nbyte]=0;
printf("%s", buf);

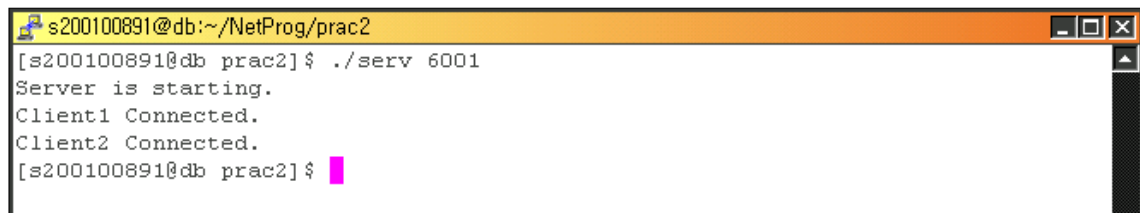
close(s);

return 0;
}

```

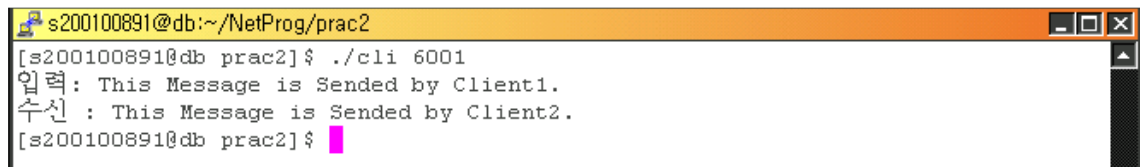
2) 실행화면

가) 서버 프로그램



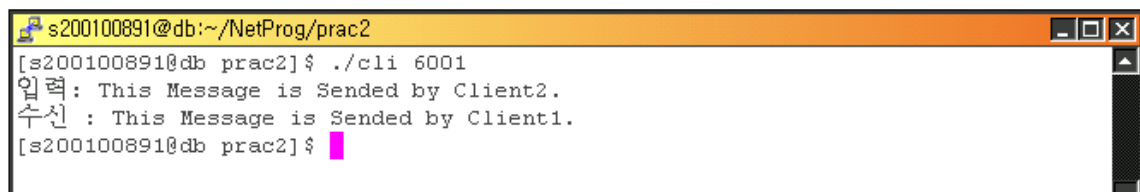
```
s200100891@db:~/NetProg/prac2
[s200100891@db prac2]$ ./serv 6001
Server is starting.
Client1 Connected.
Client2 Connected.
[s200100891@db prac2]$
```

나) 클라이언트1



```
s200100891@db:~/NetProg/prac2
[s200100891@db prac2]$ ./cli 6001
입력: This Message is Sended by Client1.
수신 : This Message is Sended by Client2.
[s200100891@db prac2]$
```

다) 클라이언트2



```
s200100891@db:~/NetProg/prac2
[s200100891@db prac2]$ ./cli 6001
입력: This Message is Sended by Client2.
수신 : This Message is Sended by Client1.
[s200100891@db prac2]$
```

SECTION

06 UDP 프로그램

1. UDP 프로그램 작성 절차

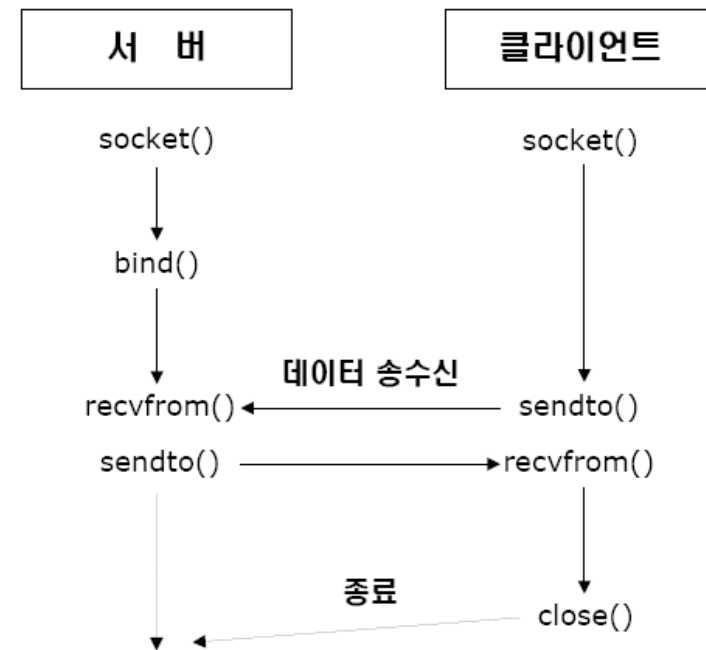


그림 132 UDP 소켓 프로그래밍 절차

가. 특징

- 1) type 인자로 `SOCK_DGRAM` 을 지정
- 2) UDP 소켓은 특정 호스트와의 일대일 통신이 아니라 임의의 호스트와 데이터그램 송수신
- 3) 비연결형 소켓
 - 가) 연결설정을 위한 `connect()` 시스템 콜을 사용할 필요 없음
 - 나) 소켓 개설 후 바로 임의의 상대방과 데이터 송수신 가능
- 4) 데이터 송수신시 각 데이터그램마다 목적지의 IP 주소와 포트번호를 항상 함수 인자로 줌

나. `sendto()` `recvfrom()` 와 함수의 사용법

- 1) `sendto()`
 - 가) UDP-소켓을 통한 데이터의 송신 함수
- 2) `recvfrom()`
 - 가) UDP 소켓을 통한 데이터의 수신 함수
 - 나) 성공적으로 수행되면 `from` 구조체에 데이터그램을 보낸 상대방의 소켓주소가 들어감

다) 데이터를 보낸 발신자에게 데이터를 보낼 때

(1) from에 있는 소켓주소를 sendto()의 to로 복사하여 사용

라) fromlen에는 from 구조체의 길이가 정수형 포인터로 리턴됨

| 문법 | 인자 | |
|---|---------|-----------------|
| <pre>int sendto(int s, char* buf, int length, int flags sockaddr* to, int tolen);</pre> | s | 소켓번호 |
| | buf | 전송할 데이터가 저장된 버퍼 |
| | length | buf 버퍼의 크기 |
| | flags | 보통 0 |
| | to | 목적지의 소켓주소 구조체 |
| | tolen | to 버퍼의 크기 |
| <pre>int recvfrom(int s, char* buf, int length, int flags, sockaddr* from, int* fromlen);</pre> | s | 소켓번호 |
| | buf | 수신 데이터를 저장할 버퍼 |
| | length | buf 버퍼의 길이 |
| | flags | 보통 0 |
| | from | 발신자의 소켓주소 구조체 |
| | fromlen | from 버퍼의 크기 |

표 41 sendto()와 recvfrom() 함수의 사용법

2. UDP 에코 프로그램

예제 19 echo 서비스를 수행하는 UDP 클라이언트

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

#define MAXLINE 511

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr;
    int s, nbyte, addrlen = sizeof(servaddr);
    char buf[MAXLINE+1];

    if(argc != 3) {
        printf("usage: %s ip_address port_number\n", argv[0]);
        exit(0);
    }

    if((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }
}
```

```

// 에코 서버의 소켓주소 구조체 작성
bzero((char *)&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr(argv[1]);
servaddr.sin_port = htons(atoi(argv[2]));

// 키보드 입력을 받음
printf("입력 : ");
if (fgets(buf, MAXLINE, stdin)==NULL) {
    printf("fgets 실패");
    exit(0);
}

// 에코 서버로 메시지 송신
if (sendto(s, buf, strlen(buf), 0,
          (struct sockaddr *)&servaddr, addrlen) < 0) {
    perror("sendto fail");
    exit(0);
}

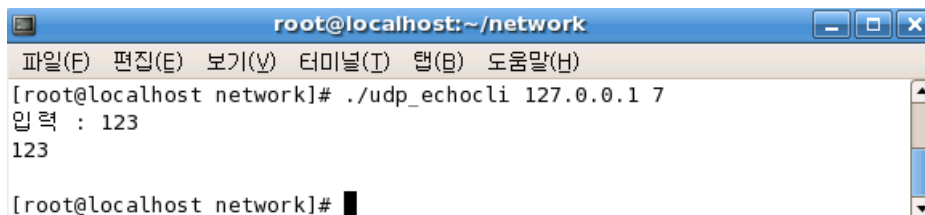
// 수신된 에코 메시지
if ( (nbyte=recvfrom(s, buf, MAXLINE, 0,
                    (struct sockaddr *)&servaddr, &addrlen)) < 0) {
    perror("recevfrom fail");
    exit(0);
}

buf[ nbyte ]=0;
printf("%s\n", buf);
close(s);

return 0;
}

```

실행 화면은 다음과 같다.



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./udp_echocli 127.0.0.1 7
입력 : 123
123
[root@localhost network]#

```

3. UDP 에코 서버

예제 20 echo 서비스를 수행하는 UDP 서버

```

#include <stdio.h>
#include <stdlib.h>

```

```

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

#define MAXLINE 511

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr, cliaddr;
    int s, nbyte, addrlen = sizeof(struct sockaddr);
    char buf[MAXLINE+1];

    if(argc != 2) {
        printf("usage: %s port\n", argv[0]);
        exit(0);
    }

    // 소켓 생성
    if((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }

    // servaddr을 '\0'으로 초기화
    bzero((char *)&servaddr, addrlen);
    bzero((char *)&cliaddr, addrlen);

    // servaddr 세팅
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(atoi(argv[1]));

    // bind() 호출
    if(bind(s, (struct sockaddr *)&servaddr, addrlen) < 0) {
        perror("bind fail");
        exit(0);
    }

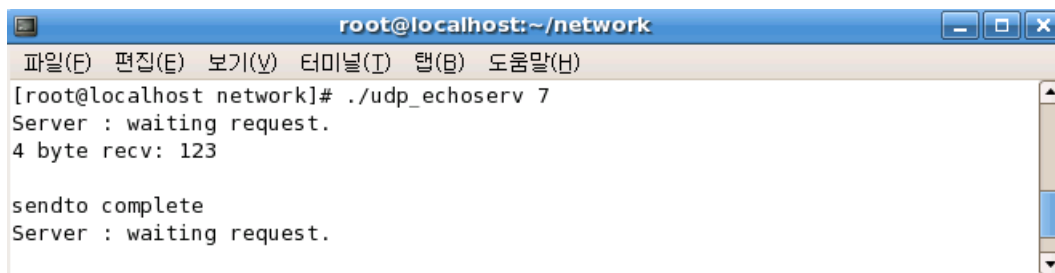
    // iterative 에코 서비스 수행
    while(1)
    {
        puts("Server : waiting request.");
        nbyte = recvfrom(s, buf, MAXLINE, 0,
            (struct sockaddr *)&cliaddr, &addrlen);
        if(nbyte < 0) {
            perror("recvfrom fail");
            exit(1);
        }
    }
}

```



```
    buf[nbyte] = 0;
    printf("%d byte recv: %s\n", nbyte, buf);
    if (sendto(s, buf, nbyte, 0,
               (struct sockaddr *)&cliaddr, addrlen) < 0) {
        perror("sendto fail");
        exit(1);
    }
    puts("sendto complete");
}
}
```

실행 화면은 다음과 같다.

A terminal window titled 'root@localhost:~/network' with standard window controls. The terminal shows the execution of a program. The prompt is '[root@localhost network]# ./udp_echoserv 7'. The output is 'Server : waiting request.' followed by '4 byte recv: 123' on the next line. Then, 'sendto complete' is printed, followed by 'Server : waiting request.' on the next line. The terminal has a menu bar with '파일(F)', '편집(E)', '보기(V)', '터미널(T)', '탭(B)', and '도움말(H)'. A scrollbar is visible on the right side of the terminal window.

```
root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./udp_echoserv 7
Server : waiting request.
4 byte recv: 123

sendto complete
Server : waiting request.
```

Chapter 6. 고급 소켓 프로그래밍

SECTION

01 소켓의 동작모드

1. 블록 모드

가. 소켓에 대해 시스템 콜 호출시 시스템이 동작 완료할 때까지 시스템 콜에서 프로세스가 멈추어 있는 모드

나. 소켓을 처음 생성할 때 디폴트로 블록 모드가 됨

다. 블록 모드의 소켓을 사용하는 프로그램에서 프로세스가 영원히 블록 상태가 될 수 있음

라. 소켓 관련 시스템 콜 중에 블록될 수 있는 예

1) listen(), connect(), accept(), recv(), send(), read(), write(), recvfrom(), sendto(), close()

마. 응용 프로그램이 일대일 통신을 하거나 한 가지 작업만 할 경우에 사용

2. 년블록 모드

가. 소켓 관련 시스템 콜에 대하여 시스템이 즉시 처리할 수 있으면 바로 결과를 리턴

나. 즉시 처리할 수 없는 경우 시스템 콜이 바로 리턴되어 응용 프로그램이 블록되지 않음

다. 폴링을 주로 사용

1) 다중화를 위해 시스템 콜이 성공적으로 실행될 때까지 계속 확인하는 방법

라. 통신상대가 여럿이거나 여러 작업을 병행하여 실행할 경우 사용

3. 비동기 모드

가. 소켓에서 어떤 I/O 변화 발생시 그 사실을 응용 프로그램이 알 수 있도록 함

1) 이 때 입출력 처리 등 원하는 동작을 함

나. 소켓을 비동기로 바꾸는 방법

1) select() 함수를 이용

가) I/O 변화가 발생할 수 있는 소켓들 전체를 대상으로 select()를 호출해 둠

나) 이 중 임의의 소켓에서 I/O 변화가 발생시 select()문이 리턴되어 원하는 작업을 함

2) fcntl()를 사용하여 소켓을 signal-driven I/O 모드로 바꾸는 방법

가) 특정 소켓에서 I/O 변화가 발생시 SIGIO 시그널을 발생시킴

나) 응용 프로그램에서는 이 시그널을 받아 필요한 작업을 함

다. 통신상대가 여럿이거나 여러 작업을 병행하여 실행할 경우 사용

SECTION

02 다중처리 기술

다중 입출력 서치를 포함해 네트워크 프로그램에서 네트워크 프로그램에서 동시에 여러 작업을 처리하는 기술

1. 멀티태스킹

여러 작업을 병행하여 처리하는 기법 (멀티프로세스, 멀티스레드)

가. 멀티프로세스

- 1) 유닉스에서 멀티태스킹을 위해 프로세스를 여러 개 실행시키는 방법
- 2) 독립적으로 처리해야 할 작업의 수만큼 프로세스를 만드는 방법
- 3) 각 프로세스들이 독립적으로 작업을 처리하므로 구현이 간편함
- 4) 다중처리 할 작업이 상당히 독립적으로 진행되어야 할 경우에 적합
- 5) 병렬처리 해야 할 작업 수만큼 프로세스를 생성하는 단점
 - 가) 프로세스 수가 증가하면 메모리 사용량이 증가하고 프로세스 스케줄링 횟수가 많아짐
 - 나) 동시에 개설했을 수 있는 프로세스의 수를 제한함
- 6) 프로세스간 데이터를 공유하기가 불편함
 - 가) 운영체제의 도움을 받아 프로세스간 통신(IPC)을 하므로 프로그램 구현이 복잡해짐

나. 멀티스레드

- 1) 유닉스에서 멀티태스킹을 위해 스레드를 여러 개 실행시키는 방법
- 2) 프로세스 내에서 독립적으로 실행하는 스레드를 여러 개 실행시킴
 - 가) 외부에서는 이 스레드들 전체가 하나의 프로세스처럼 취급됨
- 3) 프로세스에서 스레드를 생성하면 새로 생성된 스레드는 원래 프로세스의 이미지를 같이 사용
- 4) 새로 생성된 스레드용 스택 영역은 스레드별로 별도로 배정되며 스택을 공유하지 않음
- 5) 멀티프로세스 방식보다 필요로 하는 메모리량이 적고 스레드 생성 시간이 짧음
- 6) 스레드간 스케줄링도 프로세스간 스케줄링보다 빠르게 이루어 짐
- 7) 다중 처리할 작업들이 서로 밀접한 관계로 데이터 공유가 많이 필요한 경우에 적합
- 8) 동기화 문제 발생
 - 가) 한 프로세스 내에서 생성된 스레드들이 이미지를 공유하므로 전역 변수를 같이 사용함
 - 나) 스레드들은 쉽게 데이터를 공유할 수 있음
 - 다) 한 스레드가 어떤 변수의 값을 변경하는 도중에 다른 스레드가 동시에 이 변수를 액세스
 - 라) 변수의 값이 명확하게 사용되지 않음

2. 다중화

한 프로세스 또는 스레드 내에서 이루어지는 다중처리 방법

가. 폴링

처리해야 할 작업들을 순차적으로 돌아가면서 처리하는 방법

- 1) 서버가 여러 클라이언트와 통신할 때 각 클라이언트로부터의 데이터 수신을 순차적으로 처리
- 2) 입출력 함수가 어느 한 곳에서 블록되지 않아야 하므로 파일이나 소켓을 넌블록 모드로 설정
- 3) 넌블록 모드의 파일이나 소켓
 - 가) 블록될 수 있었던 입출력 함수 호출시 시스템이 즉시 처리하면 결과를 바로 리턴
 - 나) 즉시 처리할 수 없는 경우엔 함수가 즉시 리턴 되어 프로그램이 블록 되지 않는 모드
- 4) 여러 클라이언트들이 고르게 트래픽을 발생시키는 경우에 서버에서 사용하기에 적합
 - 가) 서버는 폴링을 할 때마다 수신할 데이터가 거의 항상 기다리고 있기 때문에 프로그램이 효율적으로 실행됨

나. 인터럽트

프로세스가 어떤 작업을 처리하는 도중에 특정한 이벤트가 발생하면 해당 이벤트를 처리하는 방법

- 1) 하드웨어인터럽트
 - 가) 키보드 등을 사용하여 입력하였을 때 발생하는 인터럽트
- 2) 소프트웨어인터럽트
 - 가) 프로세스 사이에 이벤트 발생을 알려주기 위한 시그널
- 3) 인터럽트처리
 - 가) CPU에서 하드웨어 인터럽트를 인터럽트 처리 루틴에서 처리함
 - 나) 시그널을 사용하여 데이터 입출력을 인터럽트 방식으로 처리함

다. 선택팅

폴링의 반대 개념

- 1) 어떤 클라이언트로부터 데이터가 도착하면 서버는 데이터가 도착한 클라이언트와의 입출력 처리
 - 2) 클라이언트로부터의데이터도착이불규칙적인경우에적합
 - 3) 유닉스에서 선택팅을 사용하기 위해 `select()` 함수를 이용
 - 가) 입출력을 처리할 파일이나 소켓들을 비동기 모드로 바꿈
-

SECTION

03 비동기형 채팅 프로그램

1. 채팅 서버 프로그램 구조

가. 연결용 소켓

- 1) 서버에서 `socket()`을 호출하여 채팅 참가를 접수할 소켓을 개설
- 2) 이 소켓을 자신의 소켓주소와 `bind()` 함

나. 통신용 소켓

- 1) 연결용 소켓을 대상으로 `select()`를 호출하여 새로운 참가 요청을 처리
- 2) `accept()`가 리턴하는 소켓번호를 채팅참가자 리스트에 등록
- 3) 서버는 연결용 소켓과 통신용 소켓을 대상으로 다시 `select()` 호출

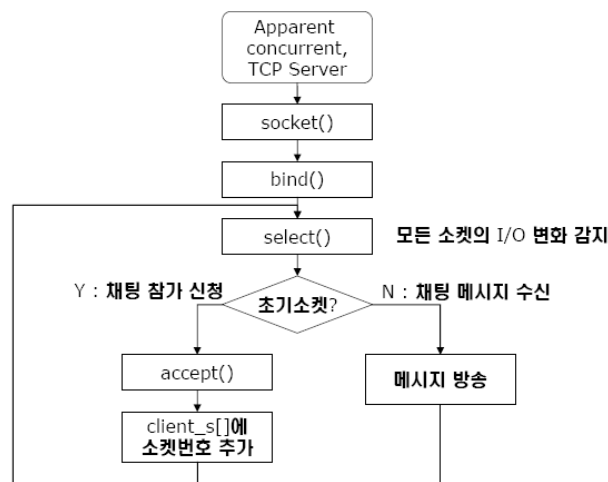


그림 135 select()를 이용한 비동기형 채팅 서버

다. 채팅 서버와 클라이언트의 연결 관계

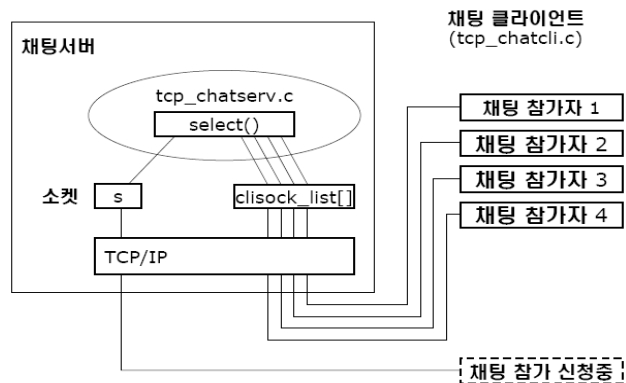


그림 136 채팅 서버와 클라이언트의 연결 관계

1) select()

가) 소켓에서 발생하는 I/O변화를 기다리다가 지정된 I/O변화 발생시 리턴

나) 응용 프로그램에서는 select() 리턴시 어떤 소켓에서 어떤 I/O변화인지 확인하고 작업 처리

2) 서버

가) 연결용 소켓과 채팅 참가자 리스트를 대상으로 select()를 호출하고 대기

나) select()문이 리턴시 새로운 참가자의 연결요청인지 기존 채팅 참가자 중 누군가 채팅 메시지를 보낸 것인지 구분하여 필요한 작업 수행

2. select()

가. 사용문법

```
int select( int amxdfp1, fd_set* readfds, fd_set* writefds, fd_set* exceptfds,
            struct timeval* tvptr );
```

1) maxfdp1 : 현재 개설된 소켓번호 중 가장 큰 소켓 번호 +1의 값

2) fd_set 타입의 인자 : 각각 읽기, 쓰기, 예외발생 같은 I/O변화시 이를 감지할 대상이 되는 소켓들을 지정하는 배열형 구조체

가) 이 세 구조체를 통해 어떤 소켓에서 어떤 종류의 I/O변화가 발생하는지, 감지할지를 선택

3) tvptr : select()가 기다리는 시간을 지정

가) NULL인 경우 지정한 I/O변화가 발생할 때까지 무한히 기다림

나) 0인 경우 기다리지 않고 바로 리턴

다) 그 외엔 지정된 시간만큼 또는 도중에 I/O변화가 발생할 때까지 기다림

나. fd_set 타입의 구조체와 소켓번호와의 관계

| | | | | | | |
|-----------|---|---|---|---|-----|---|
| readfds | 1 | 0 | 0 | 1 | ... | 0 |
| writefds | 0 | 1 | 0 | 1 | ... | 0 |
| exceptfds | 0 | 0 | 0 | 0 | ... | 0 |

maxfdp1-1

그림 137 fd_set 타입의 구조체와 소켓번호와의 관계

1) 동작방식

가) fd_set 타입 구조체에 I/O변화를 감지할 소켓이나 파일을 1로 set

나) select()를 호출해 두면 해당 조건이 만족되는 순간 select()문이 리턴

2) readfds

가) 0, 3번이 세트되어 있음

나) 키보드(표준입력 0), 소켓번호 3에서 어떤 데이터가 입력시 프로그램이 이를 읽을 수 있는 상태가 되면 select()문이 리턴

3) writefds

가) 1, 3번이 세트되어 있음

나) 파일기술자(표준출력 1)나 소켓번호 3번이 write를 할 수 있는 상태로 변할시 select()문이 리턴

다. fd_set을 사용하기 위한 매크로

1) fd_set 타입 구조체의 배열 값을 편리하게 지정하기 위해 매크로가 제공됨

가) FD_ZERO(fd_set* fdset)

(1) fdset의 모든 비트를 지움

나) FD_SET(int fd, fd_set* fdset)

(1) fdset 중 소켓 fd에 해당하는 비트를 1로 함

다) FD_CLR(int fd, fd_set* fdset)

(1) fdset 중 소켓 fd에 해당하는 비트를 0으로 함

라) FD_ISSET(int fd, fd_set* fdset)

(1) fdset 중 소켓 fd에 해당하는 비트가 set되어 있으면 양수 값을 리턴

3. 채팅 서버 프로그램

예제 21 채팅 참가자 관리, 채팅 메시지 수신 및 방송 서버

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXLINE 511
#define MAX_SOCKET 1024 // 솔라리스의 경우 64

char *EXIT_STRING = "exit"; // 클라이언트의 종료요청 문자열
char *START_STRING = "Connected to chat_server \n"; // 클라이언트 환영 메시지

int maxfdp1; // 최대 소켓번호 +1
int num_chat = 0; // 채팅 참가자 수
int clisock_list[MAX_SOCKET]; // 채팅에 참가자 소켓번호 목록
int listen_sock; // 서버의 리슨 소켓

// 새로운 채팅 참가자 처리
void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax(); // 최대 소켓 번호 찾기
void removeClient(int s); // 채팅 탈퇴 처리 함수
int tcp_listen(int host, int port, int backlog); // 소켓 생성 및 listen
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char *argv[]) {
```



```

struct sockaddr_in cliaddr;
char buf[MAXLINE+1];
int i, j, nbyte, accp_sock, addrlen = sizeof(struct sockaddr_in);
fd_set read_fds; // 읽기를 감지할 fd_set 구조체

if(argc != 2) {
    printf("사용법 :%s port\n", argv[0]);
    exit(0);
}

// tcp_listen(host, port, backlog) 함수 호출
listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
while(1) {
    FD_ZERO(&read_fds);
    FD_SET(listen_sock, &read_fds);
    for(i=0; i<num_chat; i++)
        FD_SET(clisock_list[i], &read_fds);
    maxfdp1 = getmax() + 1; // maxfdp1 재 계산
    puts("wait for client");
    if(select(maxfdp1, &read_fds, NULL, NULL, NULL)<0)
        errquit("select fail");

    if(FD_ISSET(listen_sock, &read_fds)) {
        accp_sock=accept(listen_sock, (struct sockaddr*)&cliaddr, &addrlen);
        if(accp_sock == -1)
            errquit("accept fail");
        addClient(accp_sock,&cliaddr);
        send(accp_sock, START_STRING, strlen(START_STRING), 0);
        printf("%d번째 사용자 추가.\n", num_chat);
    }

    // 클라이언트가 보낸 메시지를 모든 클라이언트에게 방송
    for(i = 0; i < num_chat; i++) {
        if(FD_ISSET(clisock_list[i], &read_fds)) {
            nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
            if(nbyte<= 0) {
                removeClient(i); // 클라이언트의 종료
                continue;
            }
            buf[nbyte] = 0;
            // 종료문자 처리
            if(strstr(buf, EXIT_STRING) != NULL) {
                removeClient(i); // 클라이언트의 종료
                continue;
            }
            // 모든 채팅 참가자에게 메시지 방송
            for (j = 0; j < num_chat; j++)
                send(clisock_list[j], buf, nbyte, 0);
            printf("%s\n", buf);
        }
    }
}

```

```

    }
} // end of while
return 0;
}

// 새로운 채팅 참가자 처리
void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET, &newcliaddr->sin_addr, buf, sizeof(buf));
    printf("new client: %s\n", buf);
    // 채팅 클라이언트 목록에 추가
    clisock_list[num_chat] = s;
    num_chat++;
}

// 채팅 탈퇴 처리
void removeClient(int s) {
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
    num_chat--;
    printf("채팅 참가자 1명 탈퇴. 현재 참가자 수 = %d\n", num_chat);
}

// 최대 소켓번호 찾기
int getmax() {
    // Minimum 소켓번호는 가정 먼저 생성된 listen_sock
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

// listen 소켓 생성 및 listen
int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }
    // servaddr 구조체의 내용 세팅
    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);

```

```

    if (bind(sd , (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }
    // 클라이언트로부터 연결요청을 기다림
    listen(sd, backlog);
    return sd;
}

```

실행 화면은 다음과 같다.

```

root@localhost network
[root@localhost network]# ./tcp_chatserv 9999
wait for client
new client: 127.0.0.1
1번째 사용자 추가.
wait for client
[cse1] :Hello

wait for client
채팅 참가자 1명 탈퇴. 현재 참가자 수 = 0
wait for client
new client: 127.0.0.1
1번째 사용자 추가.
wait for client
[cse2] :Hi~!

wait for client
채팅 참가자 1명 탈퇴. 현재 참가자 수 = 0
wait for client

```

4. 통신용 소켓 구분

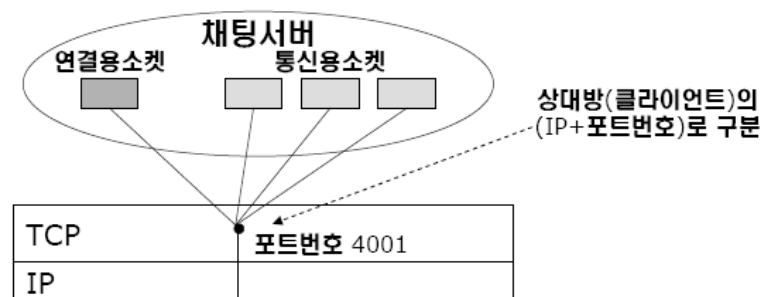


그림 139 TCP서버에서의 다수의 통신용 소켓을 구분하는 방법

가. 채팅 서버는 클라이언트와 통신에 사용하기 위해 다수의 통신용 소켓을 개설

- 1) 모든 소켓은 같은 포트번호를 사용함
- 2) 클라이언트들은 서버의 포트번호만 알고 있음
- 3) 서버에서 다른 포트번호를 사용하면 통신이 안 됨

나. 한 개의 포트번호만으로 각 클라이언트와 연결할 수 있는 이유

- 1) 클라이언트의 IP 주소와 포트번호를 내부적인 키로 사용

- 2) 채팅 서버에 도착하는 데이터그램의 포트번호는 모두 같음
- 3) 서버는 클라이언트의 IP 주소와 포트번호를 키 값으로 사용하여 해당 통신용 소켓으로 메시지 전달

다. 두 개의 telnet 클라이언트가 서버에 접속시

- 1) 서버에 두 개의 통신용 소켓이 생성됨
- 2) 통신용 소켓은 모두 23번을 포트번호로 사용함
- 3) 각 클라이언트를 구분하기 위해 클라이언트의 IP주소와 포트번호를 키로 사용

5. 채팅 클라이언트 프로그램

예제 22 서버에 접속한 후 키보드의 입력을 서버로 전달하고, 서버로부터 오는 메시지를 화면에 출력

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAXLINE    1000
#define NAME_LEN    20

char *EXIT_STRING= "exit";

// 소켓 생성 및 서버 연결, 생성된 소켓리턴
int tcp_connect(int af, char *servip, unsigned short port);
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char *argv[]) {
    char bufall[MAXLINE+NAME_LEN], // 이름+메시지를 위한 버퍼
        *bufmsg;                    // bufall 에서 메시지부분의 포인터

    int maxfdp1,                    // 최대 소켓 디스크립터
        s,                          // 소켓
        namelen;                    // 이름의 길이

    fd_set read_fds;

    if(argc != 4) {
        printf("사용법 : %s sever_ip port name \n", argv[0]);
        exit(0);
    }

    sprintf(bufall, "[%s] :", argv[3]); // bufall 의 앞부분에 이름을 저장
    namelen= strlen(bufall);
```

```

    bufmsg = bufall+namelen; // 메시지 시작 부분 지정

    s = tcp_connect(AF_INET, argv[1], atoi(argv[2]));
    if(s==-1)
        errquit("tcp_connect fail");

    puts("서버에 접속되었습니다.");
    maxfdp1 = s + 1;
    FD_ZERO(&read_fds);

    while(1) {
        FD_SET(0, &read_fds);
        FD_SET(s, &read_fds);
        if(select(maxfdp1, &read_fds, NULL, NULL, NULL) < 0)
            errquit("select fail");
        if (FD_ISSET(s, &read_fds)) {
            int nbyte;
            if ((nbyte = recv(s, bufmsg, MAXLINE, 0)) > 0) {
                bufmsg[nbyte] = 0;
                printf("%s \n", bufmsg);
            }
        }
        if (FD_ISSET(0, &read_fds)) {
            if(fgets(bufmsg, MAXLINE, stdin)) {
                if (send(s, bufall, namelen+strlen(bufmsg), 0) < 0)
                    puts("Error : Write error on socket.");
                if (strstr(bufmsg, EXIT_STRING) != NULL ) {
                    puts("Good bye.");
                    close(s);
                    exit(0);
                }
            }
        }
    } // end of while
}

int tcp_connect(int af, char *servip, unsigned short port) {
    struct sockaddr_in servaddr;
    int s;
    // 소켓 생성
    if ((s = socket(af, SOCK_STREAM, 0)) < 0)
        return -1;

    // 채팅 서버의 소켓주소 구조체 servaddr 초기화
    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = af;
    inet_pton(AF_INET, servip, &servaddr.sin_addr);
    servaddr.sin_port = htons(port);

    // 연결요청

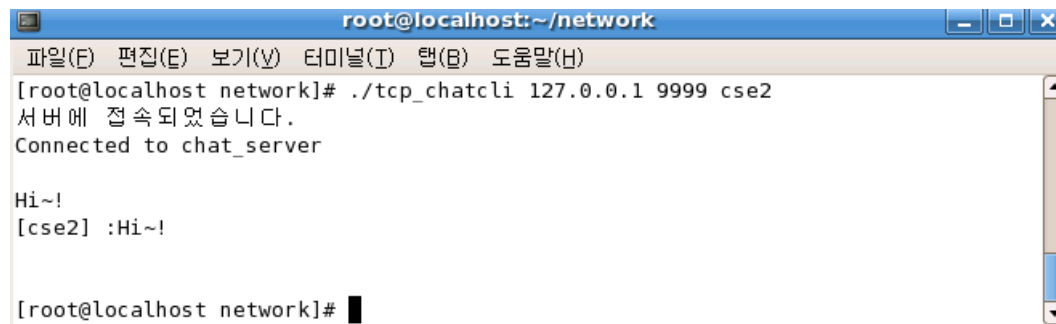
```

```

    if(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr))
        < 0)
        return -1;
    return s;
}

```

실행 화면은 다음과 같다.



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./tcp_chatcli 127.0.0.1 9999 cse2
서버에 접속되었습니다.
Connected to chat_server

Hi~!
[cse2] :Hi~!

[root@localhost network]#

```

SECTION

04 폴링형 채팅 프로그램

소켓을 년블록 보드로 설정하고 채팅 메시지 수신여부를 폴링 방식으로 점검

1. `fcntl()`

소켓을 년블록 모드나 비동기 모드로 바꿈

가. `file control`을 의미함나. `fcntl()`의 동작

- 1) 소켓을 년블록 모드로 설정
- 2) 소켓을 비동기 모드로 설정
- 3) 소켓의 소유자를 설정하거나 현재 소유자를 얻어옴

다. 년블록 모드 설정

1) 문법

가) `fd` : 모드 변경을 원하는 소켓 디스크립터

나) `cmd(명령)`

- (1) `F_SETFL` : 플래그 세트
- (2) `F_GETFL` : 플래그 읽기
- (3) `F_SETOWN` : 소켓의 소유자 설정
- (4) `F_GETOWN` : 소켓의 소유자 얻기

다) `flag`

- (1) 구체적인 옵션을 설정
- (2) `O_NONBLOCK` : 년블록 모드로 설정
- (3) `O_ASYNC` : `SIGIO` 시그널에 의해 구동되도록 비오기 모드로 설정

라. 설정코드

- 1) 소켓을 년블록 모드로 설정하는 코드
- 2) 기존의 플래그 값을 유지
 - 가) `F_GETFL` 명령으로 얻은 후 이를 `O_NONBLOCK`와 OR연산

마. 비동기 모드 설정

- 1) 비동기 모드로 설정하는 이유
 - 가) `select()` 함수 자체는 블록형 함수임
 - (1) 아무 입출력 변화가 없으면 프로그램은 `select()`에서 블록 됨
 - 나) 인터럽트형 다중화에서는 `SIGIO` 등의 시그널이 발생할 때 입출력을 처리할 수 있음

바. 소켓의 소유자를 설정하거나 현재 소유자를 얻어옴

- 1) 소켓의 소유자를 설정하는 이유
 - 가) 소켓을 처음 생성하면 소유자가 없음
 - 나) 프로세스에서 나중에 시그널을 수신하도록 하려면 소켓의 소유자가 필요
 - (1) 소켓의 소유자에게 SIGIO나 SIGURG 등의 시그널이 전달됨
 - 다) accept()가 리턴하는 통신용 소켓은 연결용 소켓의 소유자를 상속 받음

2. 폴링형 채팅 서버

폴링형 채팅 서버 프로그램에서는 소켓들을 년블록 모드로 설정한 후 무한 루프로 입출력 폴링

가. 년블록 모드

- 1) 소켓에 대해 read(), write() 등의 입출력 함수를 호출하면 함수는 바로 리턴
 - 가) 함수의 리턴 값을 보고 원하는 작업의 실행 여부를 확인
 - (1) 정상인 경우 0, 에러인 경우 -1이 리턴
 - 나) 에러인 경우 에러 코드 값을 보고 함수 자체의 에러인지 소켓이 즉시 리턴된 것인지 확인
 - (1) 소켓이 즉시 리턴된 것은 소켓이 년블록 모드이므로 리턴됨
 - (2) 이 경우 에러코드는 EWOULDBLOCK
- 2) recv() 를 호출했을 때 EWOULDBLOCK 이외의 에러
 - 가) 클라이언트와의 연결 종료 또는 클라이언트가 리셋을 보낸 경우
 - (1) 클라이언트를 채팅 목록에서 제거하는 등의 에러처리를 함
- 3) accept()
 - 가) 년블록 모드의 소켓에 accept() 를 호출한 경우
 - (1) accept() 가 리턴한 소켓은 디폴트로 블록 모드임
 - (2) 필요한 경우 명시적으로 년블록 모드로 변환해야 함

나. 소켓의 모드 확인

- 1) val = fcntl(sockfd, F_GETFL, 0);
 - 가) 기존의 플래그 값을 얻어옴
- 2) if(val & O_NONBLOCK)
 - 가) 년블록 모드인지 확인
 - 나) 년블록 모드일 경우 0을 리턴, 아니면 -1을 리턴

3. 폴링형 채팅 서버 프로그램

예제 23 년블록 모드의 채팅 서버

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```



```
#include <sys/types.h>
#include <sys/file.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#define MAXLINE 511
#define MAX_SOCKET 1024 //솔라리스는 64

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server\n";
int maxfdp1; // 최대 소켓번호 +1
int num_chat = 0; // 채팅 참가자 수
int clisock_list[ MAX_SOCKET ]; // 채팅에 참가자 소켓번호 목록
int listen_sock;

// 새로운 채팅 참가자 처리
void addClient(int s, struct sockaddr_in *newcliaddr);
void removeClient(int); // 채팅 탈퇴 처리 함수
int set_nonblock(int sockfd); // 소켓을 년블록으로 설정
int is_nonblock(int sockfd); // 소켓이 년블록 모드인지 확인
int tcp_listen(int host, int port, int backlog); // 소켓 생성 및 listen
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char *argv[]) {
    char buf[MAXLINE];
    int i, j, nbyte, count;
    int accp_sock, clilen;
    struct sockaddr_in cliaddr;

    if(argc != 2) {
        printf("사용법 :%s port\n", argv[0]);
        exit(0);
    }

    listen_sock = tcp_listen(INADDR_ANY,atoi(argv[1]),5);
    if(listen_sock==-1)
        errquit("tcp_listen fail");
    if(set_nonblock(listen_sock) == -1)
        errquit("set_nonblock fail");

    for(count=0; ;count++) {
        if(count==100000) {
            putchar('.') ;
            fflush(stdout); count=0;
        }
        addrlen = sizeof(cliaddr);
        accp_sock= accept(listen_sock, (struct sockaddr *)
            &cliaddr, &clilen);
```

```

    if(accp_sock == -1 && errno!=EWOULDBLOCK)
        errquit("accept fail");
    else if(accp_sock >0) {
        // 채팅 클라이언트 목록에 추가
        clisock_list[num_chat] = accp_sock;
        // 통신용 소켓은 년블록 모드가 아님
        if(is_nonblock(accp_sock)!=0 && set_nonblock
            (accp_sock)<0 )
            errquit("set_nonblock fail");
        addClient(accp_sock,&cliaddr);
        send(accp_sock, START_STRING, strlen(START_STRING), 0);
        printf("%d번째 사용자 추가.\n", num_chat);
    }

    // 클라이언트가 보낸 메시지를 모든 클라이언트에게 방송
    for(i = 0; i < num_chat; i++) {
        errno = 0;
        nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
        if(nbyte==0) {
            removeClient(i);    // abrupt exit
            continue;
        }
        else if( nbyte==-1 && errno == EWOULDBLOCK)
            continue;
        // 종료문자 처리
        if(strstr(buf, EXIT_STRING) != NULL) {
            removeClient(i);    // abrupt exit
            continue;
        }
        // 모든 채팅 참가자에게 메시지 방송
        buf[nbyte] = 0;
        for (j = 0; j < num_chat; j++)
            send(clisock_list[j], buf, nbyte, 0);
        printf("%s\n", buf);
    }
}

// 새로운 채팅 참가자 처리
void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET,&newcliaddr->sin_addr,buf,sizeof(buf));
    printf("new client: %s\n",buf);
    // 채팅 클라이언트 목록에 추가
    clisock_list[num_chat] = s;
    num_chat++;
}

// 채팅 탈퇴 처리
void removeClient(int i) {
    close(clisock_list[i]);
}

```

```

    if(i != num_chat-1)
        clisock_list[i] = clisock_list[num_chat-1];
    num_chat--;
    printf("채팅 참가자 1명 탈퇴. 현재 참가자 수 = %d\n", num_chat);
}

// 소켓이 nonblock 인지 확인
int is_nonblock(int sockfd) {
    int val;
    // 기존의 플래그 값을 얻어온다
    val=fcntl(sockfd, F_GETFL,0);
    // 년블록 모드인지 확인
    if(val & O_NONBLOCK)
        return 0;
    return -1;
}

// 소켓을 년블록 모드로 설정
int set_nonblock(int sockfd) {
    int val;
    // 기존의 플래그 값을 얻어온다
    val=fcntl(sockfd, F_GETFL,0);
    if(fcntl(sockfd, F_SETFL, val | O_NONBLOCK) == -1)
        return -1;
    return 0;
}

// listen 소켓 생성 및 listen
int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }
    // servaddr 구조체의 내용 세팅
    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr))
        < 0) {
        perror("bind fail"); exit(1);
    }
    // 클라이언트로부터 연결요청을 기다림
    listen(sd, backlog);
    return sd;
}

```

4. 연습문제 풀이

가. 하나의 채팅 클라이언트 프로그램에서 최대 50개의 TCP 소켓을 개설하고 이들을 각각 서버에 연결하는 busy_connect 프로그램을 작성하시오(채팅 메시지는 보내지 않음). 이때 연결의 수는 main() 함수의 인자로 받도록 하시오. 예를 들어 아래와 같이 실행하면 21.0211.123.123/9999의 호스트에 40개의 TCP 접속을 한다.

```
$ busy_connect 210.211.123.123. 9999 40
```

예제 24 TCP 소켓을 개설하고 이들을 각각 서버에 연결하는 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAXLINE      1000
#define NAME_LEN      20

char* EXIT_STRING = "exit";

int tcp_connect( int af, char* servip, unsigned short port );
void errquit( char* msg ) { perror(msg); exit(1); }

int main( int argc, char* argv[] )
{
    char bufall[ MAXLINE + NAME_LEN ]; // buffer for name + msg
    char* bufmsg;                      // pointer of msg part in bufall
    int maxfdp1;                      // maximum value of socket desc.
    int s;                            // socket
    int namelen;                      // length of name
    fd_set read_fds;

    int connectno;                    // number of connection
    int* aSocket;                     // array of sockets
    int i;

    if( argc != 5 )
    {
        printf( "Usage : %s server_ip port name connectno \n", argv[0] );
        exit( 0 );
    }

    sprintf( bufall, "[%s] : ", argv[3] );
    namelen = strlen( bufall );
    bufmsg = bufall + namelen;

    // assign array of sockets
    connectno = atoi( argv[4] );
    aSocket = (int*)malloc( sizeof(int) * connectno );
```

```

// make connects
for( i = 0; i < connectno; i++ )
{
    aSocket[i] = tcp_connect( AF_INET, argv[1], atoi(argv[2]) );
    if( aSocket[i] == -1 )
        errquit( "failed to connect" );
    printf( "Connected to server (socket no : %d)\n", i );
    maxfdp1 = aSocket[i] + 1;
}

FD_ZERO( &read_fds );

while( 1 )
{
    FD_SET( 0, &read_fds );
    if( select( maxfdp1, &read_fds, NULL, NULL, NULL ) < 0 )
        errquit( "failed to select()" );
    if( FD_ISSET( 0, &read_fds ) )
    {
        if( fgets( bufmsg, MAXLINE, stdin ) )
        {
            if( strstr( bufmsg, EXIT_STRING ) != NULL )
            {
                puts( "Good bye." );
                for( i = 0; i < connectno; i++ )
                {
                    close( aSocket[i] );
                    printf("Disconnected (socket no : %d)\n", i);
                }
                exit( 0 );
            }
        }
    }
}

free( aSocket );
return 0;
}

int tcp_connect( int af, char* servip, unsigned short port )
{
    struct sockaddr_in servaddr;
    int s;

    if( ( s = socket( af, SOCK_STREAM, 0 ) ) < 0 )
        return -1;

    bzero( (char*)&servaddr, sizeof(servaddr) );
    servaddr.sin_family = af;
    inet_pton( AF_INET, servip, &servaddr.sin_addr );
    servaddr.sin_port = htons( port );

    if( connect( s, (struct sockaddr*)&servaddr, sizeof(servaddr) ) < 0 )
        return -1;

    return s;
}

```

```

s200100891@db:~/NetProg/prac4
[s200100891@db prac4]$ ./q1 127.0.0.1 6666 xissy 20
Connected to server (socket no : 0)
Connected to server (socket no : 1)
Connected to server (socket no : 2)
Connected to server (socket no : 3)
Connected to server (socket no : 4)
Connected to server (socket no : 5)
Connected to server (socket no : 6)
Connected to server (socket no : 7)
Connected to server (socket no : 8)
Connected to server (socket no : 9)
Connected to server (socket no : 10)
Connected to server (socket no : 11)
Connected to server (socket no : 12)
Connected to server (socket no : 13)
Connected to server (socket no : 14)
Connected to server (socket no : 15)
Connected to server (socket no : 16)
Connected to server (socket no : 17)
Connected to server (socket no : 18)
Connected to server (socket no : 19)
exit
Good bye.
Disconnected (socket no : 0)
Disconnected (socket no : 1)
Disconnected (socket no : 2)
Disconnected (socket no : 3)
Disconnected (socket no : 4)
Disconnected (socket no : 5)
Disconnected (socket no : 6)
Disconnected (socket no : 7)
Disconnected (socket no : 8)
Disconnected (socket no : 9)
Disconnected (socket no : 10)
Disconnected (socket no : 11)
Disconnected (socket no : 12)
Disconnected (socket no : 13)
Disconnected (socket no : 14)
Disconnected (socket no : 15)
Disconnected (socket no : 16)
Disconnected (socket no : 17)
Disconnected (socket no : 18)
Disconnected (socket no : 19)
[s200100891@db prac4]$

```

나. `fork & exec`를 이용해서 문제 4-1의 `busy_connect.c` 프로그램을 여러 개 실행시킬 수 있는 `bomb.c` 프로그램을 작성하시오. 이제 `bomb.c`를 사용하여 임의의 채팅 서버(예:비동기형 채팅서버)에 총 몇 명까지 접속할 수 있는지를 확인하시오.(주의:서버가 다운될 수 있음)

예제 25 `busy_connect.c` 프로그램을 여러 개 실행시킬 수 있는 `bomb.c` 프로그램

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

void child_start();

```

```

int main( int argc, char* argv[] )
{
    pid_t pid;
    int child_status, child_return;
    int nCount;

    nCount = 1;
    while( 1 )
    {
        if( ( pid = fork() ) < 0 )
        {
            perror( "fork failed : " );
            exit( 0 );
        }
        else if( pid == 0 )
            child_start();
        else if( pid > 0 )
            printf( "\n\t** parent pid: %d, child pid = %d,
                    Start %dth child program\n", getpid(), pid, nCount );

        nCount++;
    }

    return 0;
}

void child_start()
{
    char* args[] = { "q1", "127.0.0.1", "6666", "xissy", "20", NULL };
    execv( "./q1", args );

    perror( "exec error at child: " );
    exit( 0 );
}

```

```

s200100891@db:~/NetProg/prac4
[s200100891@db prac4]$ ./q2
Connected to server (socket no : 0)
Connected to server (socket no : 1)
Connected to server (socket no : 2)
Connected to server (socket no : 3)
Connected to server (socket no : 4)
Connected to server (socket no : 5)
Connected to server (socket no : 6)
Connected to server (socket no : 7)

    ** parent pid: 27714, child pid = 27715, Start 1th child program

    ** parent pid: 27714, child pid = 27716, Start 2th child program
Connected to server (socket no : 8)
Connected to server (socket no : 9)
Connected to server (socket no : 10)
Connected to server (socket no : 11)
Connected to server (socket no : 12)
Connected to server (socket no : 0)
Connected to server (socket no : 13)
Connected to server (socket no : 14)
Connected to server (socket no : 15)
Connected to server (socket no : 16)
Connected to server (socket no : 17)
Connected to server (socket no : 18)
Connected to server (socket no : 19)
Connected to server (socket no : 1)
Connected to server (socket no : 2)
Connected to server (socket no : 3)
Connected to server (socket no : 4)
Connected to server (socket no : 5)
Connected to server (socket no : 6)
Connected to server (socket no : 7)
Connected to server (socket no : 8)
Connected to server (socket no : 9)
Connected to server (socket no : 10)
Connected to server (socket no : 11)
Connected to server (socket no : 12)
Connected to server (socket no : 13)
Connected to server (socket no : 14)
Connected to server (socket no : 15)
Connected to server (socket no : 16)
Connected to server (socket no : 17)
Connected to server (socket no : 18)
Connected to server (socket no : 19)

```

tcp_chatserv 프로그램에서 1020명까지 접속할 수 있다. 이는 프로세스 마다 생성할 수 있는 파일/소켓 디스크립트의 갯수가 1024개로 제한되어있기 때문이다. fd0~2와 listen 소켓을 제외한 1020의 소켓을 생성하여 클라이언트의 접속을 받을 수 있다.

다. 네트워크 관련 에러코드에는 아래와 같은 것들이 있는데 strerror() 함수를 이용해서 아래 각 errno에 대한 에러 내용을 출력해보시오.

(힌트: 아래와 같은 define문을 사용하면 타이핑을 줄일 수 있다. 예를 들어, net_errmsg(EAGAIN)을 호출하면 "EAGAIN = xxxxxxxxxx"와 같은 형태로 출력된다.)


```
#define net_errmsg(no) printf("%s=%s\n", #no, strerror(no))
```

예제 26 네트워크 관련 에러코드 출력 프로그램

```
#include <stdio.h>
#include <string.h>
#include <errno.h>

#define net_errmsg(no)      printf( "%s : \t\t%s\n", #no, strerror(no) )

int main()
{
    printf( "Print ERROR Messages\n" );

    net_errmsg( EAGAIN );
    net_errmsg( ENOTSOCK );
    net_errmsg( EDESTADDRREQ );
    net_errmsg( EMSGSIZE );
    net_errmsg( EPROTOTYPE );
    net_errmsg( ENOPROTOOPT );
    net_errmsg( EPROTONOSUPPORT );
    net_errmsg( ESOCKTNOSUPPORT );
    net_errmsg( EOPNOTSUPP );
    net_errmsg( ESHUTDOWN );
    net_errmsg( ETOOMANYREFS );
    net_errmsg( ETIMEDOUT );
    net_errmsg( ECONNREFUSED );
    net_errmsg( EHOSTDOWN );
    net_errmsg( EHOSTUNREACH );
    net_errmsg( EWOULDBLOCK );
    net_errmsg( EALREADY );
    net_errmsg( EINPROGRESS );
    net_errmsg( EPNOSUPPORT );
    net_errmsg( EADDRINUSE );
    net_errmsg( EADDRNOTAVAIL );
    net_errmsg( ENETDOWN );
    net_errmsg( ENETUNREACH );
    net_errmsg( ENETRESET );
    net_errmsg( ECONNABORTED );
    net_errmsg( ECONNRESET );
    net_errmsg( ENOBUFS );
    net_errmsg( EISCONN );
    net_errmsg( ENOTCONN );

    return 0;
}
```

```

s200100891@db:~/NetProg/prac4
[s200100891@db prac4]$ ./q3
Print ERROR Messages
EAGAIN : Resource temporarily unavailable
ENOTSOCK : Socket operation on non-socket
EDESTADDRREQ : Destination address required
EMSGSIZE : Message too long
EPROTOTYPE : Protocol wrong type for socket
ENOPROTOPT : Protocol not available
EPROTONOSUPPORT : Protocol not supported
ESOCKTNOSUPPORT : Socket type not supported
EOPNOTSUPP : Operation not supported
ESHUTDOWN : Cannot send after transport endpoint shutdown
ETOOMANYREFS : Too many references: cannot splice
ETIMEDOUT : Connection timed out
ECONNREFUSED : Connection refused
EHOSTDOWN : Host is down
EHOSTUNREACH : No route to host
EWOULDBLOCK : Resource temporarily unavailable
EALREADY : Operation already in progress
EINPROGRESS : Operation now in progress
EPPFNOSUPPORT : Protocol family not supported
EADDRINUSE : Address already in use
EADDRNOTAVAIL : Cannot assign requested address
ENETDOWN : Network is down
ENETUNREACH : Network is unreachable
ENETRESET : Network dropped connection on reset
ECONNABORTED : Software caused connection abort
ECONNRESET : Connection reset by peer
ENOBUFS : No buffer space available
EISCONN : Transport endpoint is already connected
ENOTCONN : Transport endpoint is not connected
[s200100891@db prac4]$

```

라. 비동기형 채팅 프로그램에서 서버는 `select()` 함수에서 블록상태가 된다. 이때 클라이언트가 CTRL+C를 누르면 서버의 `select()`가 에러를 리턴하는지를 확인하시오. 에러가 발생하지 않았다면 `select()` 이후 호출되는 `recv()`에서는 에러가 발생하는지 확인하시오. `recv()`나 `select()`에서 에러가 발생하면 에러의 원인을 찾아내고 에러가 발생하지 않도록 프로그램을 수정하시오.

(힌트: 클라이언트 프로그램에서 CTRL+C를 누르면 비정상적으로 프로그램이 종료되므로 TCP는 리셋(RST)을 서버로 전송하고 서버측 TCP는 리셋을 수신하면 `errno`에 `ECONNRESET` 값을 설정한다.)

예제 27 네트워크 관련 에러코드 출력 프로그램

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXLINE 511
#define MAX_SOCK 1024

```

```

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server \n";
int maxfdp1;
int num_chat = 0;
int clisock_list[MAX SOCK];
int listen_sock;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char *argv[]) {
    struct sockaddr_in cliaddr;
    char buf[MAXLINE+1];
    int i, j, nbyte, accp_sock, addrlen = sizeof(struct
        sockaddr_in);
    fd_set read_fds;

    if(argc != 2) {
        printf("Usage : %s port\n", argv[0]);
        exit(0);
    }

    listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
    while(1) {
        FD_ZERO(&read_fds);
        FD_SET(listen_sock, &read_fds);
        for(i=0; i<num_chat; i++)
            FD_SET(clisock_list[i], &read_fds);
        maxfdp1 = getmax() + 1;
        puts("wait for client");
        if(select(maxfdp1, &read_fds, NULL, NULL, NULL)<0)
        {
            //errquit("select fail");
            perror("select fail");
            exit(0);
        }

        if(FD_ISSET(listen_sock, &read_fds)) {
            accp_sock=accept(listen_sock, (struct sockaddr
                *)&cliaddr, &addrlen);
            if(accp_sock == -1)
                errquit("accept fail");
            addClient(accp_sock, &cliaddr);
            send(accp_sock, START_STRING, strlen(START_STRING), 0);
            printf("Connected %dth chatter. \n", num_chat);
        }

        for(i = 0; i < num_chat; i++) {
            if(FD_ISSET(clisock_list[i], &read_fds)) {
                nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
                if(nbyte<= 0) {
                    perror("recv fail");
                    removeClient(i);
                    continue;
                }
                buf[nbyte] = 0;
            }
        }
    }
}

```

```

        if(strstr(buf, EXIT_STRING) != NULL) {
            removeClient(i);
            continue;
        }
        for (j = 0; j < num_chat; j++)
            send(clisock_list[j], buf, nbyte, 0);
        printf("%s\n", buf);
    }
}
return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET, &newcliaddr->sin_addr, buf, sizeof(buf));
    printf("new client: %s\n", buf);
    clisock_list[num_chat] = s;
    num_chat++;
}

void removeClient(int s) {
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
    num_chat--;
    printf("A chatter leaved. Current chatters = %d\n", num_chat);
}

int getmax() {
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }
    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }
    listen(sd, backlog);
    return sd;
}

```

Chapter 7. 소켓 옵션

SECTION

01 소켓 옵션 종류

1. SO_KEEPALIVE

TCP 연결이 정상적으로 지속되고 있는지 주기적으로 확인

가. 사용방법

```
int set = 1;
setsockopt( s, SOL_SOCKET, SO_KEEPALIVE, (char*)&set, sizeof(int) );
```

나. 동작방식

- 1) 옵션이 셋팅 되어 있으면 TCP는 확인시간(예: 2시간)동안 데이터 송수신을 기다림
 - 가) TCP 없을시 연결이 아직 살아있는지 질문(keep alive prove) 을 보냄
- 2) Keep alive 질문에 대해 중간 라우터가 ICMP 메시지로 host unreachable error 또는 network unreachable error를 보낸 경우
 - 가) EHOSTENREACH 또는 ENETUNREACH 에러가 리턴

다. keep alive prove의 응답

- 1) 상대방이 ACK를 보냄
 - 가) TCP 연결이 정상적으로 동작 중이므로 확인 시간 이후에 질문을 다시 보냄
- 2) 상대방이 RST 에러를 보냄
 - 가) 상대방 호스트가 꺼진 후 재부팅 된 상태
 - 나) TCP는 소켓을 닫으며 ECONNRESET의 에러코드를 가짐
- 3) 아무 응답이 없음
 - 가) 질문을 몇 번 더 보낸 후 연결을 종료
 - 나) 에러코드는 ETIMEDOUT

라. 사용용도

- 1) 비정상적인 TCP연결 종료를 찾아 소켓을 종료하기 위해 사용

2. SO_LINGER

close()를 호출 후에 상대방에서도 정상적으로 종료 절차가 이루어지는지 확인

가. 사용방법

```

struct linger{
    int l_onoff;
    int l_linger;
};

struct linger ling;
ling.l_onoff = 1;
ling.l_linger = 0;

if( setsockopt( sockd, SOL_SOCKET, SO, LINGER, (void*) &ling, sizeof(struct linger) ) != 0 ) {
    perror(" 소켓옵션지정 실패 ");
    exit(1);
}

```

나. 동작방식

- 1) close()를 호출한 이후에 상대방에서도 정상적으로 종료 절차가 이루어 졌는지 사용
- 2) TCP 연결을 종료할 때 close()를 호출하여 종료함
 - 가) close() 는 즉시 리턴 됨
 - 나) 소켓 송신버퍼에 전송할 데이터가 있어도 close() 는 리턴 되며 모두 전송 후 연결 종료
 - 다) close() 가 리턴되어도 모든 데이터의 전송 여부는 보장되지 않음
- 3) close() 는 옵션에서 지정한 linger 시간 또는 정상 종료 때까지 블록
 - 가) 정상 종료 전에 linger 시간이 먼저 타임 아웃되면 ETIMEDOUT 에러 발생
- 4) 송신버퍼에 남아있던 데이터를 전송하기까지 기다리는 최대 시간을 제한하는데 사용
 - 가) close()는 linger 시간 이내에 데이터를 전송하지 못하면 에러를 리턴하고 통신이 종료되기 때문

다. 사용방법

- 1) 송신버퍼에 데이터가 남아 있어도 타임아웃이 되면 데이터를 폐기하도록 옵션을 지정
- 2) l_onoff는 LINGER 옵션의 사용 여부를 지정
 - 가) l_onoff = 0 일 때 LINGER 옵션을 사용하지 않음
 - 나) close() 는 원래대로 동작하여 송신버퍼에 전송할 데이터가 남아있어도 즉시 리턴
 - 다) 송신버퍼에 데이터가 남아 있었다면 커널에 의해 전송이 마무리됨
 - 라) 전송이 실패시 프로그램에서 확인할 수 없음
 - 마) 연결 종료후가시간동안 TCP 2MSL TIME-WAIT 상태가 됨
- 3) l_linger 는 기다리는 시간을 지정
 - 가) l_linger = 0 일 때 송신버퍼에 남아 있는 데이터는 파기
 - 나) 정상적인 종료 과정을 거치지 않고 RST을 전송하여 TCP 연결을 즉시 리셋
 - 다) 이 때는 TCP가 TIME-WAIT 상태에도 머무르지 않음
 - 라) l_linger 이 양수 값일 때 송신버퍼에 데이터가 있는 경우 , close() 는 데이터를 모두 전송하고 ACK를 수신할 때까지 또는 l_linger 시간이 지날 때까지 블록
 - 마) 소켓이 nonblock 모드일 경우 close() 함수는 즉시 리턴
 - 바) l_linger 시간이 먼저 지나서 close() 가 리턴된 경우에는 송신버퍼에 남은 데이터는 폐기

3. shutdown()

close() 대신 상대방이 FIN을 보낼 때까지 기다리는 방법

가. 사용방법

```
int shutdown( int s, int how );
```

나. 동작방식

- 1) 호스트 A가 호스트 B로 데이터를 보낸 후 close()를 호출한 시점에 아직 호스트 B의 수신버퍼에 호스트 A가 보낸 데이터가 읽히지 않고 대기하고 있는 경우
 - 가) 호스트 B는 수신버퍼에 아직 읽지 않은 데이터가 있어도 호스트 A가 보내온 FIN에 대한 ACK를 보냄
 - 나) 호스트 A의 close() 함수가 정상적으로 리턴 되어도 호스트 B의 응용 프로그램이 호스트 A가 보낸 모든 데이터를 읽었는지는 모름
 - 다) 이 경우 shutdown() 을 사용하여 상대방이 FIN을 보낼 때까지 대기

다. 사용방법

- 1) int s
 - 가) 소켓번호
- 2) int how
 - 가) 양방향 스트림에 대한 동작을 지정하는 인자
 - 나) SHUT_WR : 송신 스트림만 닫음
 - 다) SHUT_RD : 수신 스트림만 닫음
 - 라) SHUT_RDWR : 송신 및 수신의 양방향 스트림을 닫음

4. SO_RCVBUF와 SO_SNDBUF

소켓의 송신버퍼와 수신버퍼의 크기를 변경할 때 사용

가. 동작방식

- 1) 송신버퍼
 - 가) TCP 의 경우 응용 프로그램에서 write() 함수를 호출시 커널은 데이터를 소켓 송신버퍼로 복사
 - 나) 데이터가 송신버퍼에 모두 복사되면 데이터를 전송하기 시작
 - 다) 전송한 후에 바로 송신버퍼를 지우지 않고 재전송의 가능성을 위해 저장하고 있다가 ACK를 받은 후에 송신버퍼를 지움
 - 라) 송신버퍼가 가득 차게 되면 write() 함수는 블록됨
- 2) 수신버퍼
 - 가) TCP 는 흐름제어를 위해 수신버퍼의 여유 공간을 상대방에게 알려줌
 - 나) 여유 공간의 크기를 윈도우라고 함

- 다) TCP에서는 수신버퍼의 용량보다 큰 데이터를 받게 되면 나머지는 버려짐
- 라) UDP에서는 윈도우를 이용한 흐름제어가 없어 큰 데이터 전송시 주의해야함
- 마) UDP의 수신버퍼의 크기는 보통 40000바이트 정도이며 최소한 576바이트 이상은 보장

나. 버퍼의 크기 변경 시기

- 1) 클라이언트 : connect() 함수 호출 이전
- 2) 서버 : listen() 함수 호출 이전

5. SO_REUSEADDR

동일한 소켓주소의 중복 사용을 허용

가. 사용방법

```
int set = 1;
setsockopt( udp_sock1, SOL_SOCKET, SO_REUSEADDR, (char*)&set, sizeof(int) );
```

나. 동작방식

- 1) 동일한 소켓주소를 여러 프로세스나 한 프로세스 내의 여러 소켓에서 중복 사용을 허용
- 2) 기본적으로는 한 호스트 내에서 같은 포트번호를 중복하여 사용할 수 없음
- 3) 주로 서버에서 사용됨

다. TIME-WAIT 상태에서의주소재사용

- 1) active close를 하면 TCP가 TIME-WAIT상태에서 2MSL 시간 동안 기다림
 - 가) 이 동안 사용 중이던 포트번호를 중복하여 사용할 수 없음
- 2) bind() 를 호출하기 전에 SO_REUSEADDR 옵션을 지정하여 TIME-WAIT 상태에서 포트번호를 중복하여 사용함
- 3) 단 , TIME-WAIT 상태에 있을 때만 주소 재사용이 가능함
 - 가) close 소켓이 되지 않은 상태에서는 소켓주소를 중복하여 사용할 수 없음
- 4) TCP가 TIME-WAIT 상태 동안에도 서비스를 즉시 재시작하기 위해 사용

라. 자식 프로세스가 서버인 경우

- 1) TCP 서버 프로그램에서 자식 프로세스가 서비스 처리를 담당하던 중 부모 프로세스가 재시작시
 - 가) 포트 번호 사용 중 에러가 발생
 - 나) 과거의 자식 프로세스가 해당 포트번호를 사용하기 때문에 발생
- 2) 소켓 개설후 bind() 호출전 옵션 지정

마. 멀티홈 서버의 경우

- 1) 멀티홈 호스트
 - 가) 호스트가 두 개 이상의 랜에 접속된 경우에 호스트가 두 개 이상의 IP 주소를 가짐
- 2) 멀티홈 호스트에서 두 개 이상의 IP 주소가 같은 포트번호를 사용하는 것이 필요한 경우

- 가) 소켓주소 재사용 옵션 설정
- 나) 같은 포트번호를 사용하면서 둘 이상의 다른 IP 주소를 사용하는 서버 프로그램들을 한 호스트 내에서 실행시킬 때 소켓주소 재사용 옵션 설정이 필요
- 3) 하나의 프로세스 내에서 여러 소켓을 개설하고 각 소켓들이 같은 포트번호 다른 IP 주소를 사용할 때도 옵션 설정이 필요
- 가) 각 소켓별로 각각 다른 IP 주소를 지정하여 bind() 함

바. 완전중복바인딩

- 1) 여러 프로세스에서 동일한 IP 주소와 동일한 포트번호를 중복하여 bind() 하는 것
 - 가) UDP 소켓에서만 사용 가능함
- 2) 사용 예
 - 가) 멀티 캐스트 데이터그램을 수신하는 경우
- 3) 하나의 호스트 내에서 여러 멀티캐스트 가입자들이 데이터를 수신하려고 할 때
 - 가) 각 가입자들이 동일한 데이터그램을 수신하기 위해 완전 중복 바인딩 된 소켓주소를 사용

6. 기타 옵션

가. SO_OOBINLINE

- 1) 대역외 데이터를 일반 데이터의 수신버퍼에 같이 저장되도록 함
 - 가) 동등한 순서로 처리됨
 - 나) 일반적으로 대역외 데이터는 일반 데이터보다 우선순위가 높은 버퍼에 저장됨

나. SO_RCVLOWAT 와 SO_SNDLOWAT

- 1) 수신버퍼 최소량은 기본적으로 TCP와 UDP에서 한 바이트임
 - 가) select() 함수가 데이터 읽기 조건이 만족되어 리턴하기 위해 한 바이트 이상의 데이터가 수신 버퍼에 도착해야함
 - 나) 한 바이트 이상의 데이터가 수신되면 select() 함수가 리턴됨
- 2) 송신버퍼의 최소량은 기본적으로 2048바이트임

다. SO_DONTROUTE

- 1) send(), sendto(), sendmsg() 호출 시에 사용
- 2) 라우팅 테이블이 잘못 되었을때 라우팅 테이블을 무시할 때 사용
 - 가) 데이터그램을 로컬 인터페이스에 있는 장비로 전송

라. SO_BROADCAST

- 1) 브로드캐스트는 UDP 소켓에서만 사용 가능함
- 2) 이더넷 같은 방송형 서브 네트워크에서만 가능함

마. TCP_NODELAY

- 1) Nagle's 알고리즘이 디폴트로 실행되므로 이를 취소해야 할 때 사용
- 2) Nagle's 알고리즘

- 가) 앞에 전송된 데이터 ACK를 받기 전까지는 작은 크기의 데이터는 전송하지 않고 기다림
- 나) ACK를 기다리는 동안 작은 데이터들을 모아서 전송
 - (1) 너무 작은 데이터 세그먼트들이 자주 발생하지 않게 하여 전송 효율 높임
 - (2) ACK가 빨리 오면 작은 데이터 송신도 이에 비례하므로 통신 처리가 늦어지지 않음
- 3) delayed ACK 알고리즘
 - 가) 데이터를 수신한 경우 그 데이터에 대해 ACK를 즉시 보내지 않고 약간의 지연을 둠
 - 나) piggyback
 - (1) 지연시간동안 상대방에게 보낼 데이터에 ACK 비트를 세트함으로 별도의 ACK를 생략
 - (2) 지연 타이머는 500ms 이하이며 보통 50 ~ 200ms를 사용

바. TCP_MAXSEG

- 1) 최대 세그먼트 크기의 지원은 모든 시스템이 지원하지 않음
 - 가) 리눅스에서는 지원됨
 - 2) 최대 세그먼트 크기 변경은 연결 설정 이전에 함
 - 3) 너무 큰 값으로 변경되었을시 연결 설정 과정에서 지정값보다 작게 재지정됨
-

SECTION

02 소켓 옵션 변경

1. 소켓 옵션 변경 함수

accept()를 호출하기 전에 소켓 옵션을 지정

가. 사용방법

```
int getsockopt(int s, int level, int opt, const char *optval, int *optlen)
int setsockopt(int s, int level, int opt, const char *optval, int optlen)
```

- 1) int s
 - 가) 옵션 내용을 읽을 또는 변경할 소켓 번호
- 2) int level
 - 가) 프로토콜 레벨을 지정
 - 나) 소켓 레벨의 옵션 : SOL_SOCKET
 - 다) IP 프로토콜에 관한 옵션 : IPPROTO_IP
 - 라) TCP에 관한 옵션 : IPPROTO_TCP
- 3) int opt
 - 가) 변경할 또는 읽을 옵션을 지정
- 4) const char *optval
 - 가) 지정하려는 또는 읽을 옵션 값을 가리키는 포인터
- 5) int *optlen, int optlen
 - 가) *optval의 크기를 나타냄
 - 나) getsockopt()에서는 optlen을 setsockopt()에서는 *optlen을 사용

나. 소켓 옵션의 종류

| 레벨 | 옵션 | 의미 |
|------------|-------------------|--|
| SOL_SOCKET | SO_BROADCAST | 방송형 메시지 전송 허용 |
| | SO_DEBUG | DEBUG 모드를 선택 |
| | SO_REUSEADDR | 주소의 재사용 선택 |
| | SO_LINGER | 소켓을 닫을 때 미전송된 데이터가 있어도 지정된 시간만큼 기다렸다가 소켓을 닫음 |
| | SO_KEEPALIVE | TCP의keep-alive동작 선택 |
| | SO_OOBINLINE | OOB 데이터를 일반 데이터처럼 읽음 |
| | SO_RCVBUF | 수신버퍼의 크기 변경 |
| | SO_SNDBUF | 송신버퍼의 크기 변경 |
| IPPROTO_IP | IP_TTL | Time To Live 변경 |
| | IP_MULTICAST_TTL | 멀티캐스트 데이터그램의 TTL 변경 |
| | IP_ADD_MEMBERSHIP | 멀티캐스트 그룹에 가입 |

| | | |
|-------------|--------------------|-----------------------------|
| IPPROTO_TCP | IP_DROP_MEMBERSHIP | 멀티캐스트 그룹에서 탈퇴 |
| | IP_MULTICAST_LOOP | 멀티캐스트 데이터그램의 loopback 허용 여부 |
| | IP_MULTICAST_IF | 멀티캐스트 데이터그램 전송용 인터페이스 지정 |
| | TCP_KEEPAIVE | keep-alive 확인 메시지 전송 시간 지정 |
| | TCP_MAXSEG | TCP의 MSS(최대 메시지 크기) 지정 |
| | TCP_NODELAY | Nagle 알고리즘의 선택 |

표 42 소켓 옵션의 종류

2. 소켓 옵션 변경 예제 프로그램

예제 28 소켓 옵션을 사용하여 수신버퍼의 크기 변경

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main() {
    int s;
    int val, len;

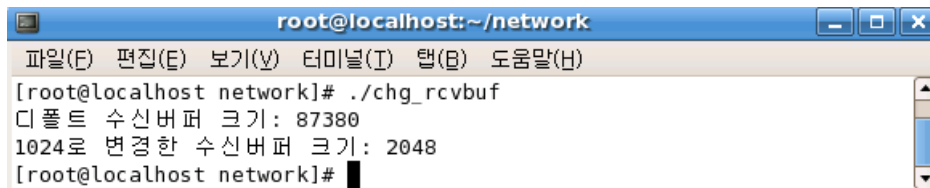
    // 소켓 생성
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(1);
    }

    len = sizeof(val);
    // 수신 버퍼의 크기 값을 얻어 옴
    if (getsockopt(s, SOL_SOCKET, SO_RCVBUF, &val, &len) < 0) {
        perror("socket fail");
        exit(1);
    }

    // 현재 디폴트로 설정된 수신 버퍼의 크기를 출력
    printf("디폴트 수신버퍼 크기: %d\n", val);

    val = 1024;
    // 수신 버퍼를 val 사이즈 만큼 변경
    setsockopt(s, SOL_SOCKET, SO_RCVBUF, &val, sizeof(val));
    // 변경된 수신 버퍼의 값을 얻어 옴
    getsockopt(s, SOL_SOCKET, SO_RCVBUF, &val, &len);
    printf("1024로 변경한 수신버퍼 크기: %d\n", val);
    return 0;
}
```

실행화면은 다음과 같다.



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./chg_rcvbuf
디폴트 수신버퍼 크기: 87380
1024로 변경한 수신버퍼 크기: 2048
[root@localhost network]#

```

수신 버퍼의 크기는 커널이 두 배의 값으로 변경한다. 1024로 설정하여도 출력에서는 2048이 나온다.

3. 연습문제 풀이

가. TCP 에코 클라이언트에서 메시지를 전송한 후 MSG_PEEK 옵션을 주고 recv() 함수를 호출하여 수신버퍼에 메시지가 얼마나 도착했는지를 확인하는 프로그램을 작성하시오(아래의 동작 순서 참고.)

- 1) 에코 클라이언트에서 메시지를 전송한 후 MSG_PEEK 옵션을 주고 recv()를 호출한다. 그러면 서버가 메시지를 에코할 때까지 대기 상태가 된다.
- 2) 서버는 즉시 에코를 하고 2초간 sleep한다.
- 3) 클라이언트는 recv()함수가 리턴한 후 4초간 sleep한다.
- 4) 서버는 클라이언트보다 먼저 깨어나며 클라이언트가 sleep하는 동안 "end of message"를 전송한다.
- 5) 클라이언트는 깨어난 후에 MSG_PEEK 옵션을 주고 recv()를 호출하여 도착한 메시지가 더 있는지를 확인한다.
- 6) 클라이언트는 MSG_PEEK 옵션 없이 recv()를 호출하여 도착한 메시지를 모두 읽는다.

예제 29 서버 프로그램 소스

```

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

#define MAXLINE 127

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr, cliaddr;
    int listen_sock, accp_sock,
        addrlen=sizeof(cliaddr),
        nbyte;
    char buf[MAXLINE+1];

    if(argc != 2) {
        printf("usage: %s port\n", argv[0]);
        exit(0);
    }

    if((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {

```

```

    perror("socket fail");
    exit(0);
}

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

if(bind(listen_sock, (struct sockaddr *)&servaddr,
        sizeof(servaddr)) < 0) {
    perror("bind fail");
    exit(0);
}

listen(listen_sock, 5);

while(1) {
    puts("Server is waiting client.....");

    accp_sock = accept(listen_sock,
                       (struct sockaddr *)&cliaddr, &addrlen);
    if(accp_sock < 0) {
        perror("accept fail");
        exit(0);
    }
    puts("Client connected");
    nbyte = read(accp_sock, buf, MAXLINE);
    write(accp_sock, buf, nbyte);

    sleep(2);
    sprintf(buf, "end of server message\n");
    write(accp_sock, buf, strlen(buf));

    close(accp_sock);
}

close(listen_sock);
return 0;
}

```

예제 30 클라이언트 프로그램 소스

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define MAXLINE 127

```

```

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr;
    int s, nbyte;
    char buf[MAXLINE+1];

    if(argc != 2) {
        printf("usage: %s ip_address\n", argv[0]);
        exit(0);
    }

    if((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    servaddr.sin_port = htons(6666);

    if(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("connect fail");
        exit(0);
    }

    printf("Input: ");
    if (fgets(buf, sizeof(buf), stdin) == NULL)
        exit(0);
    nbyte = strlen(buf);

    if (write(s, buf, nbyte) < 0) {
        printf("write error\n");
        exit(0);
    }

    printf("First Peek Recv : ");

    sleep(1);
    nbyte = recv( s, buf, MAXLINE, MSG_PEEK | MSG_DONTWAIT );
    if( nbyte < 0 )
    {
        perror("first recv fail");
        exit(0);
    }
    buf[nbyte]=0;
    printf("%s", buf);

    sleep(4);

    printf("Second Peek Recv : ");
    nbyte = recv( s, buf, MAXLINE, MSG_PEEK | MSG_DONTWAIT );
    if( nbyte < 0 )
    {
        perror("second recv fail");
    }
}

```



```

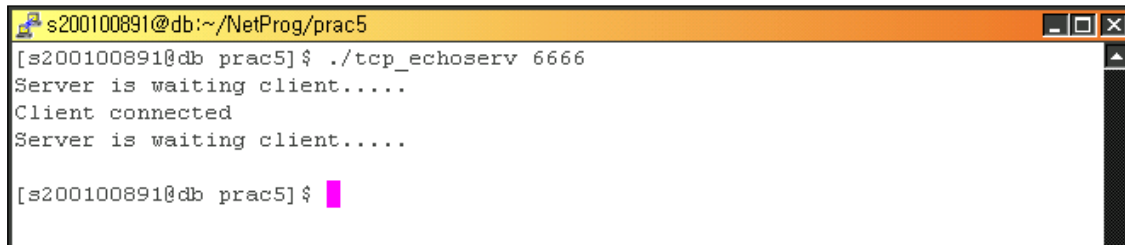
        exit(0);
    }
    buf[nbyte]=0;
    printf("%s", buf);

    printf("First Non-peek Recv : ");
    if( (nbyte=read(s, buf, MAXLINE)) < 0)
    {
        perror("read fail");
        exit(0);
    }
    buf[nbyte]=0;
    printf("%s", buf);

    printf("Second Non-peek Recv : ");
    if( (nbyte=read(s, buf, MAXLINE)) < 0)
    {
        perror("read fail");
        exit(0);
    }
    buf[nbyte]=0;
    printf("%s\n", buf);

    close(s);
    return 0;
}

```

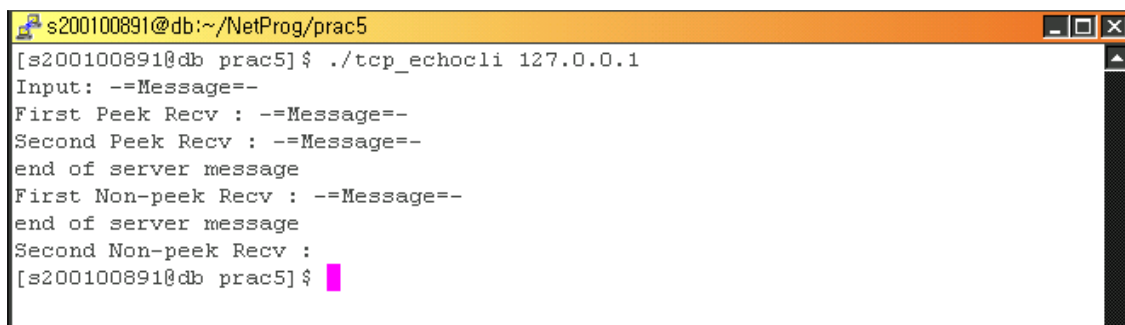


```

s200100891@db:~/NetProg/prac5
[s200100891@db prac5]$ ./tcp_echoserv 6666
Server is waiting client.....
Client connected
Server is waiting client.....

[s200100891@db prac5]$

```



```

s200100891@db:~/NetProg/prac5
[s200100891@db prac5]$ ./tcp_echocli 127.0.0.1
Input: --Message--
First Peek Recv : --Message--
Second Peek Recv : --Message--
end of server message
First Non-peek Recv : --Message--
end of server message
Second Non-peek Recv :
[s200100891@db prac5]$

```

MSG_PEEK 옵션을 이용해 recv()하면 버퍼의 내용이 지워지지 않고 그대로 남아 있지만, MSG_PEEK 옵션을 사용하지 않으면 읽어온 내용이 버퍼에서 사라짐을 확인할 수 있다.

나. 위의 내용을 UDP 소켓에서 작성하고 TCP 소켓의 경우와 어떤 차이가 있는지를 설명하시오.

예제 31 서버 프로그램 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MAXLINE 511

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr, cliaddr;
    int s, nbyte, addrlen = sizeof(struct sockaddr);
    char buf[MAXLINE+1];

    if(argc != 2) {
        printf("usage: %s port\n", argv[0]);
        exit(0);
    }

    if((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }

    bzero((char *)&servaddr, addrlen);
    bzero((char *)&cliaddr, addrlen);

    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(atoi(argv[1]));

    if(bind(s, (struct sockaddr *)&servaddr, addrlen) < 0) {
        perror("bind fail");
        exit(0);
    }

    while(1)
    {
        puts("Server : waiting request.");
        nbyte = recvfrom(s, buf, MAXLINE, 0,
                        (struct sockaddr *)&cliaddr, &addrlen);
        if(nbyte < 0) {
            perror("recvfrom fail");
            exit(1);
        }
        buf[nbyte] = 0;
        printf("%d byte recv: %s\n", nbyte, buf);

        if (sendto(s, buf, nbyte, 0, (struct sockaddr *)&cliaddr, addrlen) < 0 )
        {

```

```

        perror("sendto fail");
        exit(1);
    }

    sleep(2);
    sprintf( buf, "end of server message\n" );
    if (sendto(s, buf, strlen(buf), 0, (struct sockaddr*)&cliaddr, addrlen) < 0
)
    {
        perror("sendto fail");
        exit(1);
    }

    puts("sendto complete");
}

return 0;
}

```

예제 32 클라이언트 프로그램 소스

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXLINE 511

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr;
    int s, nbyte, addrlen = sizeof(servaddr);
    char buf[MAXLINE+1];

    if(argc != 3) {
        printf("usage: %s ip_address port_number\n", argv[0]);
        exit(0);
    }

    if((s = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);
    servaddr.sin_port = htons(atoi(argv[2]));

    printf("Input : ");
    if (fgets(buf, MAXLINE, stdin)==NULL) {

```

```

    printf("fgets fail");
    exit(0);
}

if (sendto(s, buf, strlen(buf), 0,
          (struct sockaddr *)&servaddr, addrlen) < 0) {
    perror("sendto fail");
    exit(0);
}

sleep(1);
if ( (nbyte=recvfrom(s, buf, MAXLINE, MSG_PEEK | MSG_DONTWAIT,
                    (struct sockaddr *)&servaddr, &addrlen)) < 0) {
    perror("First recvfrom fail");
    exit(0);
}
buf[ nbyte ]=0;
printf("First Peek recvfrom : %s\n", buf);

sleep(4);

if ( (nbyte=recvfrom(s, buf, MAXLINE, MSG_PEEK | MSG_DONTWAIT,
                    (struct sockaddr *)&servaddr, &addrlen)) < 0) {
    perror("Second recvfrom fail");
    exit(0);
}
buf[ nbyte ]=0;
printf("Second Peek recvfrom : %s\n", buf);

if ( (nbyte=recvfrom(s, buf, MAXLINE, 0,
                    (struct sockaddr *)&servaddr, &addrlen)) < 0) {
    perror("First Non-peek recvfrom fail");
    exit(0);
}
buf[ nbyte ]=0;
printf("First Non-peek recvfrom : %s\n", buf);

if ( (nbyte=recvfrom(s, buf, MAXLINE, 0,
                    (struct sockaddr *)&servaddr, &addrlen)) < 0) {
    perror("Second Non-peek recvfrom fail");
    exit(0);
}
buf[ nbyte ] =0;
printf("Second Non-peek recvfrom : %s\n", buf);

close(s);
return 0;
}

```

```
s200100891@db:~/NetProg/prac5
[s200100891@db prac5]$ ./udp_echoserv 6666
Server : waiting request.
12 byte recv: --Message--

sendto complete
Server : waiting request.

[s200100891@db prac5]$
```

```
s200100891@db:~/NetProg/prac5
[s200100891@db prac5]$ ./udp_echocli 127.0.0.1 6666
Input : --Message--
First Peek recvfrom : --Message--

Second Peek recvfrom : --Message--

First Non-peek recvfrom : --Message--

Second Non-peek recvfrom : end of server message

[s200100891@db prac5]$
```

TCP는 스트림 서비스를 제공하여 연속적으로 도착한 데이터의 경계를 구분하지 못하기 때문에 MSG_PEEK으로 버퍼에 남아있는 내용과 그 이후에 버퍼로 들어온 내용이 함께 출력된다. 하지만 UDP는 데이터그램 단위로 데이터를 읽을 수 있기 때문에 MSG_PEEK으로 읽은 데이터 크기만큼 다음번 recvfrom()에서 처리하는 것을 볼 수 있다.

다. TCP 에코 클라이언트 프로그램의 마지막 부분에서 close() 함수를 호출하면 서버와 FIN 패킷을 주고받게 된다. SO_LINGER 옵션을 사용해서 RST(reset)이 전송되는 tcp_reset_cli.c 프로그램을 작성하시오. 이 때 서버에서 RST를 수신한 것을 확인할 수 있도록 recv() 함수의 리턴 값을 검사하고 perror()함수로 에러의 원인을 출력하시오.

(힌트: l_onoff 값은 1로, l_linger 값을 0으로 설정하여 SO_LINGER 옵션을 지정한 후에 close()함수를 호출하면 된다.)

예제 33 서버 프로그램 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

#define MAXLINE 127

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr, cliaddr;
```

```

    int listen_sock, accp_sock,
        addrlen=sizeof(cliaddr),
        nbyte;
char buf[MAXLINE+1];

    if(argc != 2) {
        printf("usage: %s port\n", argv[0]);
        exit(0);
    }

    if((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }

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

    if(bind(listen_sock, (struct sockaddr *)&servaddr,
        sizeof(servaddr)) < 0) {
        perror("bind fail");
        exit(0);
    }

    listen(listen_sock, 5);

    while(1) {
        puts("Server is waiting client.....");

        accp_sock = accept(listen_sock,
            (struct sockaddr *)&cliaddr, &addrlen);
        if(accp_sock < 0) {
            perror("accept fail");
            exit(0);
        }
        puts("Client connected");
        nbyte = read(accp_sock, buf, MAXLINE);
        write(accp_sock, buf, nbyte);

        sleep(1);
        nbyte = recv(accp_sock, buf, MAXLINE, 0 );
        if( nbyte < 0 )
        {
            perror("recv fail");
        }

        close(accp_sock);
    }
    close(listen_sock);
    return 0;
}

```

예제 34 클라이언트 프로그램 소스

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define MAXLINE 127

int main( int argc, char* argv[] )
{
    struct sockaddr_in servaddr;
    int s, nbyte;
    char buf[MAXLINE+1];

    struct linger ling;

    if( argc != 2 ) {
        printf("usage: %s ip_address\n", argv[0]);
        exit(0);
    }

    if((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket fail");
        exit(0);
    }

    ling.l_onoff = 1;
    ling.l_linger = 0;
    if(setsockopt(s, SOL_SOCKET, SO_LINGER, (void*)&ling, sizeof(struct linger)) != 0)
    {
        perror( "setsockopt fail " );
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    servaddr.sin_port = htons(6666);

    if(connect(s, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("connect fail");
        exit(0);
    }

    printf("Input: ");
    if (fgets(buf, sizeof(buf), stdin) == NULL)
        exit(0);
    nbyte = strlen(buf);

    if (write(s, buf, nbyte) < 0) {

```

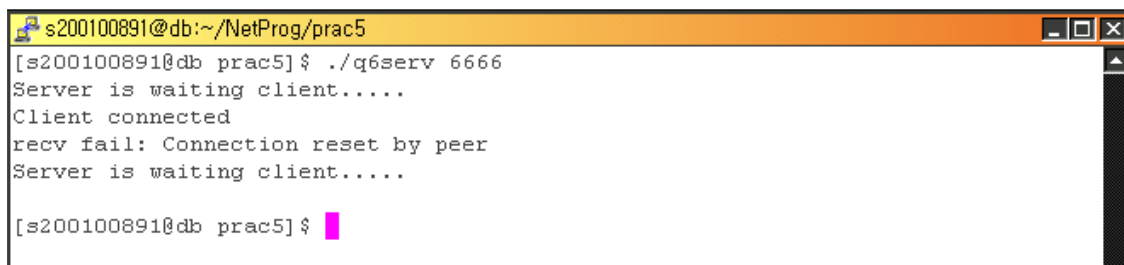
```

        printf("write error\n");
        exit(0);
    }

    printf("Recv : ");
    if( (nbyte=read(s, buf, MAXLINE)) < 0) {
        perror("recv fail");
        exit(0);
    }
    buf[nbyte]=0;
    printf("%s", buf);

    close(s);
    return 0;
}

```

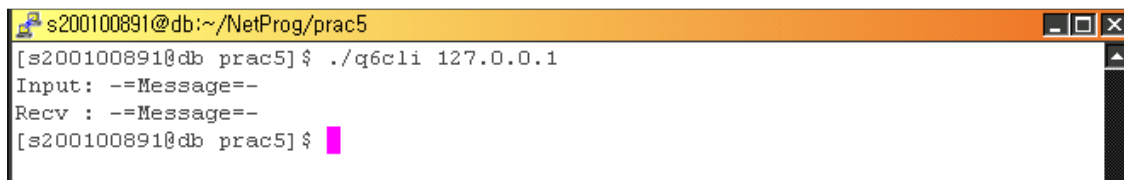


```

s200100891@db:~/NetProg/prac5
[s200100891@db prac5]$ ./q6serv 6666
Server is waiting client.....
Client connected
recv fail: Connection reset by peer
Server is waiting client.....

[s200100891@db prac5]$

```



```

s200100891@db:~/NetProg/prac5
[s200100891@db prac5]$ ./q6cli 127.0.0.1
Input: --Message--
Recv : --Message--
[s200100891@db prac5]$

```

서버 프로그램 실행화면에서 RST을 수신한것을 "recv fail: Connection reset by peer"을 통해 알 수 있다.

Chapter 8. 프로세스

SECTION

01 프로세스의 이해

1. 프로세스

가. OS상에서 실행되는 개개의 프로그램

- 1) task라고도 함
- 2) 기계어로 이루어진 수동적인 프로그램이 아님
- 3) 시스템 자원을 할당받아 동작되고 있는 능동적인 프로그램
- 가) CPU의 레지스터, 프로그램 포인터, 스택 메모리 등 프로그램 실행에 필요한 자원을 할당 받음

2. 프로세스 식별자

가. 커널에서 관리하기 위해 내부 프로세스 테이블에 등록

- 1) 이 등록번호를 프로세스 식별자(PID)라 함

나. 프로세스 식별자를 통해 Linux 내부의 모든 프로세스를 통제함

- 1) 프로세스 식별자를 통해 프로세스에게 신호를 전달
- 2) 생성과 동시에 발생하며 종료시 커널에 반환

3. 프로세스 식별자 확인

가. getpid()

- 1) 호출 프로세스의 프로세스 ID를 반환, 입력 값은 없음

나. getpgrp()

- 1) 호출 프로세스의 그룹ID 반환, 입력 값은 없음

다. getppid()

- 1) 부모 프로세스ID 반환

라. getpgid()

- 1) 프로세스ID를 입력 값으로 가짐
- 2) 성공시 부모 프로세스의 그룹ID 반환
- 3) 에러시 pid - 1를 반환

4. 프로세스 그룹

가. 어떤 하나의 작업을 수행하는데 관계된 프로세스들의 집합

나. 프로세스 그룹 ID

- 1) 동일 프로세스 그룹 내에 있는 프로세스들은 동일한 프로세스 그룹 ID를 소유

다. 프로세스 그룹 생존시간

- 1) 프로세스 그룹의 생성에서 모든 프로세스의 종료까지의 시간 간격
- 2) 프로세스 그룹 리더에 의해 프로세스에서의 프로세스를 생성, 종료
- 3) 같은 그룹 내의 모든 프로세스가 없어질 때까지 그 그룹은 존재함

SECTION

02 프로세스의 생성과 종료

1. 프로세스의 생성 - fork()

가. 사용방법

```
int PID = fork();

if (PID == 0)
{
    child_work();
}
else if (PID > 0)
{
    parent_work();
}
else
{
    error();
}
```

나. 리눅스에서 유일하게 프로세스를 생성하는 함수

다. 성공시 프로세스 ID를 반환함

라. 동작방법

- 1) fork()를 실행한 프로세스로부터 새 프로세스가 복제됨
- 2) fork()를 실행한 프로세스는 부모 프로세스라 함
- 3) 새 프로세스는 자식 프로세스라 함

2. 프로세스의 종료

가. exit()

- 1) 정상적인 종료
 - 가) main() 함수의 묵시적인 종료
 - 나) main() 함수의 return 사용
 - 다) exit() 호출
- 2) 사용방법
 - 가) 종로의 상태로 사용할 정수 값을 인자로 받아 해당 상태로 종료됨

나. abort()

- 1) 비정상적인 종료
 - 가) 신호에 의한 종료
 - 나) abort() 호출

SECTION

03 데몬 서버 구축 방법

1. 데몬 프로세스

백그라운드로 실행되는 프로세스로 특정 터미널 제어와 관계없이 실행되는 프로세스

가. 데몬 프로세스 생성

1) 백그라운드로 실행되는 프로세스

가) 터미널에서 데몬 실행 후 사용자가 로그아웃 해도 데몬은 종료되지 않음

나) 데몬은 kill 명령이나 SIGKILL 시그널을 보냄으로 종료

나. 사용방법

```
struct sigaction sact;
sigset_t mask;

// 부모 프로세스를 종료시키고 자식 프로세스에서 실행 됨
if((pid = fork()) != 0)
    exit(0);

// 스스로 세션 리더가 됨
setsid();

// SIGHUP 시그널을 무시, 손자 프로세스와 터미널의 연관을 끊음
sact.sa_handler = SIG_IGN;
sact.sa_flags = 0;
sigemptyset(&sact.sa_mask);
sigaddset(&sact.sa_mask, SIGHUP);
sigaction(SIGHUP, &sact);

// 손자 프로세스를 생성하고 부모 프로세스를 종료
if((pid = fork()) != 0)
    exit(0);

// 작업 디렉토리를 루트 디렉토리로 변경
chdir("/");
// 새로 생성되는 파일이 임의의 소유 권한을 가지게 함
umask(0);
// 혹시 개설되어 있을 소켓을 닫음
for(i=0 ; i<MAXFD ; i++)
    close(1);
```

1) 위의 코드를 프로그램 앞부분에 추가

2. 데몬 서버 종류

httpd, sendmail, named 등과 같이 항상 실행되고 있으면서 서비스를 제공하는 데몬

예제 35 독립형 데몬 서버 구축

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <syslog.h>
#include <stdarg.h>
#include <unistd.h>
#include <sys/stat.h>

#define MAXLINE    511
#define MAXFD      64

int  tcp_listen(int host, int port, int backlog);

int main(int argc, char *argv[]) {
    struct sockaddr_in cliaddr;
    int listen_sock, accp_sock; // 소켓 번호
    int i, addrlen, nbyte;
    pid_t pid;
    char buf[MAXLINE+1];
    struct sigaction sact;

    if (argc != 2) {
        printf("사용법: %s port\n", argv[0]);
        exit(0);
    }

    // 프로그램을 데몬 프로세스로 실행시키는 코드
    if((pid = fork()) != 0)
        exit(0); // 부모 프로세스는 종료시킨다
                // 즉, 자식 프로세스만 아래 부분을 실행한다
    setsid();   // 스스로 세션 리더가 된다

    // SIGHUP 시그널(hang up)을 무시한다
    sact.sa_handler = SIG_IGN;
    sact.sa_flags = 0;
    sigemptyset(&sact.sa_mask);
    sigaddset(&sact.sa_mask, SIGHUP);
    sigaction(SIGHUP, &sact, NULL);

    if ((pid = fork()) != 0) // 다시 자식 프로세스(손자)를 만든다
        exit(0);           // 부모 프로세스는 종료시킨다
```

```

chdir ("/");          // 디렉토리 변경
umask(0);             // umask 설정
for (i = 0; i < MAXFD ; i++)
    close(i);         // 혹시 개설되어 있을지 모르는 소켓을 닫는다

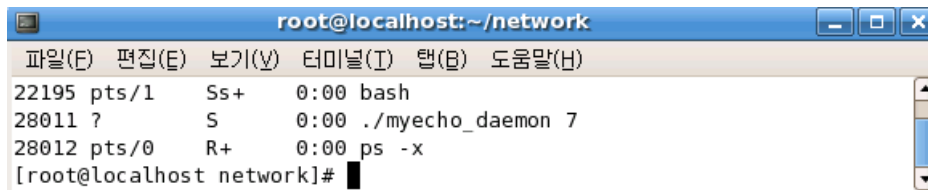
listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]),5);
// interactive 에코 서비스 수행
while(1) {
    addrlen = sizeof(cliaddr);
    // 연결요청을 기다림
    if ((accp_sock = accept(listen_sock,
                            (struct sockaddr *)&cliaddr, &addrlen)) < 0)
        exit(0);
    if ((nbyte = read(accp_sock, buf, MAXLINE)) < 0)
        exit(0);
    write(accp_sock, buf, nbyte);
    close(accp_sock);
}
close(listen_sock);
}

// listen 소켓 생성 및 listen
int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }
    // servaddr 구조체의 내용 세팅
    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) <
        0) {
        perror("bind fail");    exit(1);
    }
    // 클라이언트로부터 연결요청을 기다림
    listen(sd, backlog);
    return sd;
}

```

프로그램을 실행하면 데몬 서버가 구동됨을 확인할 수 있다.



3. 연습문제 풀이

가. 앞 부분에서 소개한 임의의 채팅 서버를 데몬 형태로 동작시키고 동작을 확인하시오.

(힌트: 채팅 서버를 시작하기 전에 데몬 프로세스가 되기 위한 초기화 작업을 하면 된다. 데몬은 화면에 출력할 수 없으므로 모든 화면 출력 메시지를 특정 파일에 기록해야 한다.)

예제 36 서버 프로그램 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <signal.h>
#include <syslog.h>
#include <stdarg.h>
#include <sys/stat.h>

#define MAXLINE 511
#define MAX_SOCK 1024

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server \n";

int maxfdp1;
int num_chat = 0;
int clisock_list[MAX_SOCK];
int listen_sock;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char *argv[])
{
    struct sockaddr_in cliaddr;
    char buf[MAXLINE+1];
    int i, j, nbyte, accp_sock, addrlen = sizeof(struct sockaddr_in);
```

```

fd_set read_fds;

pid_t pid;
struct sigaction sact;
FILE* fLog;
char szMsg[1024];

if(argc != 2) {
    printf("Usage :%s port\n", argv[0]);
    exit(0);
}

if( ( pid = fork() ) != 0 )
    exit( 0 );
setsid();

sact.sa_handler = SIG_IGN;
sact.sa_flags = 0;
sigemptyset( &sact.sa_mask );
sigaddset( &sact.sa_mask, SIGHUP );
sigaction( SIGHUP, &sact, NULL );

if( ( pid = fork() ) != 0 )
    exit( 0 );
umask( 0 );

for( i = 0; i < MAX_SOCKET; i++ )
    close( i );

listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
while(1) {
    FD_ZERO(&read_fds);
    FD_SET(listen_sock, &read_fds);
    for(i=0; i<num_chat; i++)
        FD_SET(clisock_list[i], &read_fds);
    maxfdp1 = getmax() + 1;

    fLog = fopen( "./daemon.log", "a" );
    sprintf( szMsg, "wait for client\n" );
    fwrite( szMsg, strlen(szMsg), 1, fLog );
    fclose( fLog );

    if(select(maxfdp1, &read_fds, NULL, NULL, NULL)<0)
        errquit("select fail");

    if(FD_ISSET(listen_sock, &read_fds)) {
        accp_sock=accept(listen_sock, (struct sockaddr
            *)&cliaddr, &addrlen);
        if(accp_sock == -1)
            errquit("accept fail");
        addClient(accp_sock,&cliaddr);
        send(accp_sock, START_STRING, strlen(START_STRING), 0);

        fLog = fopen( "./daemon.log", "a" );

```



```

        sprintf( szMsg, "Added %dth User.\n", num_chat );
        fwrite( szMsg, strlen(szMsg), 1, fLog );
        fclose( fLog );
    }

    for(i = 0; i < num_chat; i++) {
        if(FD_ISSET(clisock_list[i], &read_fds)) {
            nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
            if(nbyte <= 0) {
                removeClient(i);
                continue;
            }
            buf[nbyte] = 0;

            if(strstr(buf, EXIT_STRING) != NULL) {
                removeClient(i);
                continue;
            }

            for (j = 0; j < num_chat; j++)
                send(clisock_list[j], buf, nbyte, 0);

            fLog = fopen( "./daemon.log", "a" );
            sprintf( szMsg, "%s\n", buf );
            fwrite( szMsg, strlen(szMsg), 1, fLog );
            fclose( fLog );
        }
    }
} // end of while

return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    FILE* fLog;
    char szMsg[1024];
    char buf[20];
    inet_ntop(AF_INET, &newcliaddr->sin_addr, buf, sizeof(buf));

    fLog = fopen( "./daemon.log", "a" );
    sprintf( szMsg, "new client: %s\n", buf );
    fwrite( szMsg, strlen(szMsg), 1, fLog );
    fclose( fLog );

    clisock_list[num_chat] = s;
    num_chat++;
}

void removeClient(int s) {
    FILE* fLog;
    char szMsg[1024];

    close(clisock_list[s]);
    if(s != num_chat-1)

```

```

        clisock_list[s] = clisock_list[num_chat-1];
        num_chat--;

        fLog = fopen( "./daemon.log", "a" );
        sprintf( szMsg, "Leave a User. Current Users = %d\n", num_chat );
        fwrite( szMsg, strlen(szMsg), 1, fLog );
        fclose( fLog );
    }

    int getmax() {
        int max = listen_sock;
        int i;
        for (i=0; i < num_chat; i++)
            if (clisock_list[i] > max )
                max = clisock_list[i];
        return max;
    }

    int tcp_listen(int host, int port, int backlog) {
        int sd;
        struct sockaddr_in servaddr;

        sd = socket(AF_INET, SOCK_STREAM, 0);
        if(sd == -1) {
            perror("socket fail");
            exit(1);
        }

        bzero((char *)&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(host);
        servaddr.sin_port = htons(port);
        if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
            perror("bind fail"); exit(1);
        }

        listen(sd, backlog);
        return sd;
    }
}

```



```

s200100891@db:~/NetProg/prac6
[s200100891@db prac6]$ ./q1
Usage :./q1 port
[s200100891@db prac6]$ ./q1 6666
[s200100891@db prac6]$

```

```
s200100891@db:~/NetProg/prac6
wait for client
new client: 127.0.0.1
Added 1th User.
wait for client
new client: 127.0.0.1
Added 2th User.
wait for client
[xissy] :Hi~ taeho!

wait for client
[taeho] :Hello~ xissy!

wait for client
Leave a User. Current Users = 1
wait for client
Leave a User. Current Users = 0
wait for client
~
~
~
~
~
~
"daemon.log" 17L, 297C 1,1 All
```

```
s200100891@db:~/NetProg
[s200100891@db NetProg]$ ./tcp_chatcli 127.0.0.1 6666 xissy
서버에 접속되었습니다.
Connected to chat_server

Hi~ taeho!
[xissy] :Hi~ taeho!

[taeho] :Hello~ xissy!

exit
Good bye.
[s200100891@db NetProg]$
```

```
s200100891@db:~/NetProg
[s200100891@db NetProg]$ ./tcp_chatcli 127.0.0.1 6666 taeho
서버에 접속되었습니다.
Connected to chat_server

[xissy] :Hi~ taeho!

Hello~ xissy!
[taeho] :Hello~ xissy!

exit
Good bye.
[s200100891@db NetProg]$
```

Chapter 9. 시그널

SECTION

01 시그널 종류

1. 시그널의 종류

가. 시그널의 정의

- 1) 소프트웨어 인터럽트
- 2) 운영체제 또는 커널에서 일반 프로세스로 보냄
- 3) 일반 프로세스에서 커널의 도움을 받아 다른 프로세스로 시그널을 넘김

나. 시그널의 발생

- 1) kill() 시스템 콜

```
int kill( pid_t pid, int sig );
```

- 가) 다른 프로세스에게 시그널이 발생되도록 함
- 나) 프로세스 pid인 프로세스에게 sig 시그널이 발생

- 2) raise()

```
int raise( int sig );
```

- 가) 프로세스가 자신에게 시그널을 보냄

다. 시그널의 종류

- 1) 리눅스 시그널의 이름은 SIG로 시작
- 2) signal.h 파일에 소개되어 있음

| 시그널 | 발생 조건 |
|---------|---|
| SIGINT | 인터럽트키(CTRL-C)를 입력했을 때 발생 |
| SIGKILL | 강제 종료 시그널로서, 프로세스에서 이 시그널을 무시하거나 블록할 수 없음 |
| SIGIO | 비동기 입출력이 발생했을 때 전달 |
| SIGPIPE | 닫힌 파이프나 소켓에 데이터르 쓰거나 읽기를 시도할 때 발생 |
| SIGCHLD | 프로세스가 종료되거나 취소될 때 부모 프로세스에게 전달 |
| SIGPWR | 전원의 중단 및 재시작 시에 init 프로세스로 전달 |
| SIGTSTP | 사용자가 키보드에서 중지키(CTRL-Z)를 눌렀을 때 발생 |
| SIGSYS | 잘못된 시스템 호출 시에 발생 |
| SIGURG | 대역외 데이터를 수신 시에 발생 |
| SIGUSR1 | 사용자가 임의의 목적으로 사용할 수 있는 시그널 |
| SIGUSR2 | |

| | |
|-----------|-------------------------------------|
| SIGHUP | 터미널과 연결이 끊어졌을 때 세션 리더에게 보내짐 |
| SIGQUIT | 종료키(CTRL-\\)를 눌렀을 때 전달 |
| SIGILL | 프로세스가 규칙에 어긋난 명령을 수행하려고 할 때 전달 |
| SIGTRAP | 프로그램이 디버깅 지점에 도달하면 전달 |
| SIGABRT | abort() 함수를 호출하면 발생 |
| SIGFPE | 숫자를 0으로 나누거나 연산 에러 시 발생 |
| SIGVTALRM | setitimer() 함수에 의한 가상 타이머 시간 만료를 알림 |

표 43 시그널의 종류

SECTION

02 시그널 처리

1. 시그널 처리 기본 동작

가. 정해진 기본 동작 수행

1) 프로세스가 시그널을 수신시 각 시그널에 따라 처리할 기본 동작이 미리 정해져 있음

가) 대부분 프로세스 종료임

나) SIGKILL, SIGSTOP 시그널

(1) 기본 동작으로 프로세스 종료임

(2) 무시나 시그널 핸들러 등록이 허용되지 않음

나. 사용자가 지정한 작업 수행

1) 시그널 핸들러 수행

가) 미리 정해진 함수를 수행

2) 시그널 무시

가) 프로세스는 시그널을 무시하므로 아무 영향이 없음

3) 시그널 블록

가) 해당 시그널의 수신시 시그널 대기큐에 들어감

(1) 현재의 동작을 계속하고 나중에 시그널을 처리

나) bitmask 형태의 시그널 블록 마스크에 추가해서 블록함

다) 블록된 시그널은 중복되지 않음

2. 시그널 핸들러

가. 시그널 집합

1) 여러 개의 시그널을 한 번에 표시하기 위해 sigset_t를 사용

2) sigset_t 변수에는 여러 개의 시그널을 bitmask 형태로 누적하여 표현

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

3) sigemptyset()

가) 시그널 집합을 비어 있는 상태로 초기화

4) sigfillset()

가) 시스템에 정의되어 있는 모든 시그널들을 시그널 집합에 넣음

5) sigaddset()

가) 시그널 집합에 특정 시그널을 추가

6) sigdelset()

가) 시그널 집합에서 특정 시그널을 제거

7) sigismember()

가) 시그널 집합 안에 특정 시그널이 존재하는지 여부를 확인

나. 시그널 핸들러 설정

1) sigaction() 함수

가) 시그널 핸들러를 등록하기 위해 sigaction() 함수를 사용

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

나) 인자 signum

(1) 처리할 대상이 되는 시그널

(2) SIGKILL과 SIGSTOP를 제외한 모든 시그널이 가능함

다) 인자 act

(1) signum 시그널 발생시 동작을 결정에 참조되는 구조체

(2) 시그널 핸들러 이름, 실행도중 블록시킬 시그널 집합, 환경설정용 flag인자 포함

2) sigaction 구조체

```
struct sigaction {
    void(*sa_handler)(int);
    void(*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void(*sa_restore)(void);
}
```

가) sa_handler, sa_sigaction

(1) 시그널 핸들러를 가리킴

(2) 일반적으로 sa_handler를 사용하고 sa_sigaction은 부가적인 정보를 이용할 때 사용

나) sa_mask

(1) 시그널 핸들러가 동작되는 동안 블록시킬 시그널 집합

(2) 시그널 핸들러가 실행되는 도중에 sa_mask 없는 시그널 도착시

(가) 시그널 핸들러는 인터럽트 되고 새로 도착한 시그널에 대한 처리가 진행됨

(나) 인터럽트를 받지 않는게 안전하므로 sigfillset(&act.sa_mask)로 모두 포함시킴

다) sa_restore

(1) 사용되지 않음

라) sa_flags

(1) SA_RESTART

(가) 인터럽트 되었던 시스템 콜이 시그널 핸들러 처리한 후에 계속 수행되도록 함

(2) SA_NOCLDWAIT

(가) 좀비 프로세스를 만들지 않도록 하기 위한 플래그

(나) SIGCHLD 시그널 처리시 주로 사용됨

(3) SA_NODEFER

(가) 도착한 시그널의 수만큼 핸들러가 수행됨

(4) SA_SIGINFO

(가) 미설정시 시그널이 발생하면 sa_handler에 명시된 핸들러 함수가 호출됨

(나) 설정시 시그널이 발생하면 sa_sigaction에 명시된 핸들러 함수가 호출됨

(5) SA_ONESHOT or SA_RESETHAND

(가) 시그널 핸들러가 한 번 실행된 후에 시그널 처리 기본 동작으로 되돌아감

(나) 설치한 시그널 핸들러는 한 번만 실행되고 이후의 시그널은 SIG_DFL에 의해 수행

3) signal_unsafe 함수

가) 결과를 예측할 수 없는 함수

(1) malloc(), free(), alarm(), sleep(), fcntl(), fork()등

(2) 시그널 핸들러 외부의 작업과 충돌을 일으킬 가능성이 있음

(3) 정적인 변수나 전역 변수를 사용

나) 호출해야 할 경우

(1) 호출보다 핸들러 내에서는 특정 플래그만 설정

(2) 핸들러가 종료된 후에 플래그를 확인하고 signal_unsafe한 함수를 핸들러 밖에서 호출

4) signal() 함수

가) sigaction()이 소개되기 이전에 사용되던 함수

나) 시그널 핸들러를 등록하거나 시그널 무시를 설정함

다) 현재 사용하지 않기를 권장함

3. 시그널 처리 예

예제 37 SIGINT 시그널에 대한 핸들러 등록하는 예제 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

int count;

// 시그널 핸들러
void catch_sigint(int signum) {
    printf("\n(count=%d) CTRL-C pressed!\n", count);
    return;
}
```

```

int main(int argc, char *argv[]) {
    struct sigaction act;
    sigset_t masksets;
    int i;
    char buf[10];
    sigfillset(&masksets);

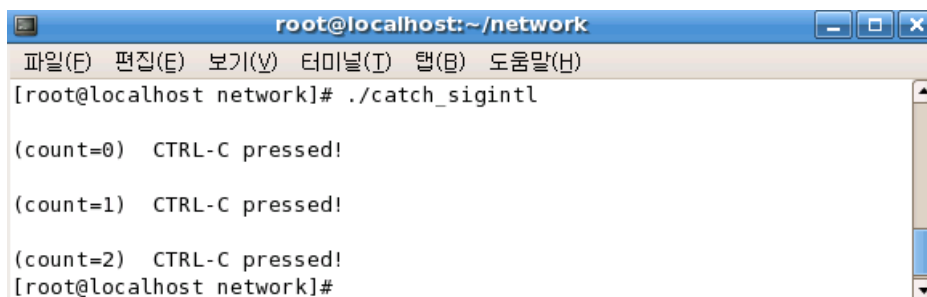
    // 시그널 핸들러 설치
    act.sa_handler = catch_sigint;

    // 시그널 핸들러가 실행되는 동안 모든 시그널을 블록함
    act.sa_mask = masksets;
    act.sa_flags = 0;
    sigaction(SIGINT, &act, NULL);

    for(count=0; count<3 ; count++ )
        read(0, buf, sizeof(buf));
    return 0;
}

```

실행 화면은 다음과 같다.



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./catch_sigint

(count=0) CTRL-C pressed!

(count=1) CTRL-C pressed!

(count=2) CTRL-C pressed!
[root@localhost network]#

```

CTRL-C를 눌러도 `catch_sigint()`가 SIGINT를 잡아서 처리하므로 종료가 되지 않는다. 시그널 핸들러를 수행하는 동안 도착한 모든 시그널은 블록된다.

SECTION

03 SIGCHLD와 프로세스의 종료

1. 프로세스의 종료

가. SIGCHLD 시그널

- 1) 프로세스 종료시 자신의 부모 프로세스에게 보내는 시그널
- 2) 부모 프로세스는 원하면 이 시그널을 받아 처리함

나. wait()

- 1) 부모 프로세스가 자식 프로세스의 종료 시점이나 종료 상태 값을 알기 위해 사용

```
pid_t pid;
int stat;
pid = wait(&stat);
```

- 2) 부모 프로세스가 자식 프로세스가 종료된 것을 확인하고 다른 작업을 할 경우

가) pid = wait(&stat); 의 작업 수행

- (1) 자식 프로세스가 종료될 때까지 블록 됨
- (2) 종료 상태 값은 stat 인자로 리턴 됨

나) 위의 작업 이후에 다른 작업 수행됨

- 3) 좀비 상태 발생

가) 자식 프로세스 종료 후에 부모 프로세스가 wait() 호출로 종료 상태를 읽지 않을 경우 발생

나) 부모 프로세스가 나중에라도 자식 프로세스의 종료 상태를 읽기 위해 좀비 상태를 만듦

다) 좀비 상태의 프로세스는 사용자 영역의 메모리는 모두 free임

라) 커널이 관리하는 메모리는 남아 있으므로 메모리 낭비가 됨

- 4) SIGCHLD를 잡아 처리하는 것으로 wait()를 대신할 수 있음

가) 인터럽트 방식으로 자식 프로세스의 종료 시점 파악

다. 기본동작으로서 SIGCHLD를 무시하는 경우

- 1) 부모 프로세스가 자식 프로세스의 종료 상태를 읽을 필요가 없을 때 단순히 무시

- 2) 부모 프로세스가 자식 프로세스의 종료 시점을 SIGCHLD를 통해 파악할 경우

가) 부모 프로세스에서 wait()를 호출한 이후에 자식 프로세스가 종료

- (1) 부모 프로세스는 wait()함수에서 대기
- (2) 자식 프로세스가 종료할 때 wait()함수가 리턴

나) 부모 프로세스가 wait()를 호출하기 이전에 자식 프로세스가 종료

- (1) 자식 프로세스가 좀비가 됨
- (2) 부모 프로세스는 필요시 나중에 wait()를 호출하여 자식의 종료 상태를 읽음

다) 부모 프로세스가 wait()를 호출하지 않고 종료

- (1) 시스템의 init 프로세스가 좀비 프로세스에 대해 부모 프로세스 역할을 함
- (2) 대신 wait()를 호출

라. SIG_IGN으로 SIGCHLD를 무시하는 경우

- 1) 시그널 핸들러에 SIG_IGN을 등록하여 SIGCHLD를 무시
- 2) 기본 동작으로 SIGCHLD를 무시하는 것과 거의 같은 동작을 함
 - 가) 부모 프로세스가 wait()호출 전에 자식 프로세스가 종료되어도 좀비가 발생하지 않음
 - 나) 이미 종료된 자식 프로세스의 상태를 얻지 못함
 - 다) wait()호출 시 자식 프로세스가 없으면 -1을 리턴

마. 시그널 핸들러를 등록한 경우

- 1) 부모 프로세스가 자신의 작업을 수행하다가 자식 프로세스의 종료 시점에 종료 상태를 알고자 할 때
 - 가) 미리 wait()를 호출하여 블록될 필요가 없음
 - 나) 시그널 핸들러가 실행되어도 자식 프로세스의 좀비 상태는 끝나지 않음
 - (1) wait()를 실행해야 함

2. 프로세스의 종료 처리

예제 38 wait() 함수 동작의 이해

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

// 사용자 시그널 핸들러 함수
void catch_sigchld(int signo) { puts("###( Parent ) catch SIGCHLD "); }

int chstat;          // 종료상태 값
int main(int argc, char *argv[]) {
    int i,n;
    struct sigaction sact;
    sact.sa_flags = 0;
    sigemptyset(&sact.sa_mask);
    sigaddset(&sact.sa_mask, SIGCHLD);

    // 시그널 핸들러 등록
    sact.sa_handler = SIG_DFL;          // 기본동작(무시)
    //sact.sa_handler = SIG_IGN;        // 시그널 무시 등록
    //sact.sa_handler = catch_sigchld; // 사용자 핸들러 등록
    sigaction(SIGCHLD, &sact, NULL);

    // 자식 프로세스 생성
```

```

for(i=0; i<5; i++) {
    if(fork()==0) {
        if(i>2) sleep(6); // 부모의 wait() 호출보다 늦게 종료
        printf("(%d번 Child),PID=%d,PPID=%d Exited\n",i,
                                                       getpid(),getppid());
        exit(13); // 종료값 13번
    }
}

sleep(3); // 0,1,2 번 자식이 종료하기를 기다림
puts("-----");
system("ps -a"); // 현재 시스템의 프로세스 상태확인
puts("-----");

puts("#( Parent ) wait 호출함");
for( ; ; ) {
    chstat = -1; // 초기화
    n = wait(&chstat);
    printf("# wait = %d(child stat=%d)\n",n, chstat);
    if( n == -1) {
        if(errno ==ECHILD) {
            perror("기다릴 자식프로세스가 존재하지 않음"); break;
        }
        else if(errno == EINTR) {
            perror("wait 시스템 콜이 인터럽트 됨"); continue;
        }
    }
}
puts("#( Parent ) 종료함");
return 0;
}

```

다음 실행 화면 5개의 프로세스 중 wait() 호출 이전에 종료한 0, 1, 2는 좀비가 됨을 보여준다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 랩(B) 도움말(H)
[root@localhost network]# ./wait_test
(0번 Child),PID=28128,PPID=28127 Exited
(1번 Child),PID=28129,PPID=28127 Exited
(2번 Child),PID=28130,PPID=28127 Exited
-----
  PID TTY          TIME CMD
 28127 pts/0        00:00:00 wait_test
 28128 pts/0        00:00:00 wait_test <defunct>
 28129 pts/0        00:00:00 wait_test <defunct>
 28130 pts/0        00:00:00 wait_test <defunct>
 28131 pts/0        00:00:00 wait_test
 28132 pts/0        00:00:00 wait_test
 28133 pts/0        00:00:00 ps
-----
#( Parent ) wait 호출함
# wait = 28128(child stat=3328)
# wait = 28129(child stat=3328)
# wait = 28130(child stat=3328)
(3번 Child),PID=28131,PPID=28127 Exited
(4번 Child),PID=28132,PPID=28127 Exited
# wait = 28131(child stat=3328)
# wait = 28132(child stat=3328)
# wait = -1(child stat=-1)
기다릴 자식프로세스가 존재하지 않음: No child processes
#( Parent ) 종료함
[root@localhost network]#

```

하지만 `sact.sa_handler = SIG_IGN;`으로 시그널 무시를 등록한 경우, 좀비 프로세스가 발생하지 않아 다음 실행 화면과 같이 0, 1, 2번 프로세스의 종료 상태는 영원히 알 수 없다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 랩(B) 도움말(H)
[root@localhost network]# ./wait_test
(0번 Child),PID=28145,PPID=28144 Exited
(1번 Child),PID=28146,PPID=28144 Exited
(2번 Child),PID=28147,PPID=28144 Exited
-----
  PID TTY          TIME CMD
 28144 pts/0        00:00:00 wait_test
 28148 pts/0        00:00:00 wait_test
 28149 pts/0        00:00:00 wait_test
 28150 pts/0        00:00:00 ps
-----
#( Parent ) wait 호출함
(3번 Child),PID=28148,PPID=28144 Exited
(4번 Child),PID=28149,PPID=28144 Exited
# wait = -1(child stat=-1)
기다릴 자식프로세스가 존재하지 않음: No child processes
#( Parent ) 종료함
[root@localhost network]#

```

또한 `sact.sa_handler = catch_sigchld;`로 SIGCHLD 시그널 핸들러를 등록한 경우, 0, 1, 2번 프로세스가 종료하면서 발생한 SIGCHLD에 의해 시그널 핸들러 `catch_sigchld()` 함수가 호출된다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./wait_test
(0번 Child),PID=28159,PPID=28158 Exited
###( Parent ) catch SIGCHLD
(1번 Child),PID=28160,PPID=28158 Exited
###( Parent ) catch SIGCHLD
(2번 Child),PID=28161,PPID=28158 Exited
###( Parent ) catch SIGCHLD
-----
  PID TTY          TIME CMD
28158 pts/0    00:00:00 wait_test
28159 pts/0    00:00:00 wait_test <defunct>
28160 pts/0    00:00:00 wait_test <defunct>
28161 pts/0    00:00:00 wait_test <defunct>
28162 pts/0    00:00:00 wait_test
28163 pts/0    00:00:00 wait_test
28164 pts/0    00:00:00 ps
###( Parent ) catch SIGCHLD
-----
#( Parent ) wait 호출 함
# wait = 28159(child stat=3328)
# wait = 28160(child stat=3328)
# wait = 28161(child stat=3328)
(3번 Child),PID=28162,PPID=28158 Exited
(4번 Child),PID=28163,PPID=28158 Exited
###( Parent ) catch SIGCHLD
# wait = 28162(child stat=3328)
# wait = 28163(child stat=3328)
# wait = -1(child stat=-1)
기다릴 자식프로세스가 존재 하지 않음: No child processes
#( Parent ) 종료함
[root@localhost network]#

```

부모 프로세스에게 아직 종료하지 않은 자식 프로세스가 있으므로 이후 wait()로 대기하다가 자식 프로세스가 종료하면 부모 프로세스에게 SIGCHLD 시그널이 전달된다.

가. 시스템 콜 인터럽트 경우

- 1) 부모 프로세스가 다른 시스템 콜을 호출한 상태에서 SIGCHLD 시그널 도착한 경우
- 2) SIGCHLD의 시그널 핸들러 함수를 등록했다면 SIGCHLD 시스템 콜을 인터럽트 시킴
- 3) 시그널 처리가 기본 동작이나 시그널 무시 상태이면 시스템 콜은 아무 영향 받지 않음

나. waitpid()

- 1) 특정 PID를 갖는 자식 프로세스의 종료만을 기다림
- 2) 만약 앞으로 종료할 자식 프로세스가 없을 경우
 - 가) 부모 프로세스는 영원히 기다리게 됨
 - 나) flags에서 WNOHANG 옵션을 사용해서 문제 해결
 - (1) waitpid() 호출 시에 실행중인 자식 프로세스가 있을 때에만 블록
 - (2) 실행중이거나 종료된 자식 프로세스가 없으면 블록되지 않음

3. 연습문제 풀이

가. 비동기형 채팅 서버 프로그램에서 SIGINT 시그널을 3초 이내에 연속적으로 두 번 받으면 프로그램을 종료하도록 수정하시오.

(힌트: SIGINT 시그널이 발생했을 때 시각을 저장하고 또 한 번의 SIGINT 시그널이 발생하면 이전에 저장한 시각과의 차이를 구해서 3초보다 적으면 종료하도록 한다. 3초보다 크면 현재 시각을 저장하고 다음 SIGINT 시그널이 발생하기를 기다렸다가 같은 작업을 반복한다. SIGINT 시그널 핸들러 동작 중에는 모든 시그널이 블록되도록 하고 sigaction 구조체의 sa_flags에는 SA_RESTART를 설정한다.)

예제 39 서버 프로그램 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

#define MAXLINE 511
#define MAX_SOCKET 1024

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server \n";

int maxfdp1;
int num_chat = 0;
int clisock_list[MAX_SOCKET];
int listen_sock;

time_t oldTime = 0;
time_t curTime = 0;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

// signal handler
void catch_sigint( int signum )
{
    time( &curTime );

    if( oldTime == 0 )
    {
        oldTime = curTime;
        printf( "\nFirst CTRL-C Pressed!\n" );
    }
}
```



```

    }
    else if( curTime - oldTime > 3 )
    {
        oldTime = curTime;
        printf( "\nFirst CTRL-C Pressed!\n" );
    }
    else
    {
        printf( "\nSecond CTRL-C Pressed! Program will be exit!\n" );
        exit( 0 );
    }

    return;
}

int main(int argc, char *argv[]) {
    struct sockaddr_in cliaddr;
    char buf[MAXLINE+1];
    int i, j, nbyte, accp_sock, addrlen = sizeof(struct
        sockaddr_in);
    fd_set read_fds;
    int n;

    struct sigaction act;
    sigset_t masksets;
    sigfillset( &masksets );

    act.sa_handler = catch_sigint;
    act.sa_mask = masksets;
    act.sa_flags = SA_RESTART;
    sigaction( SIGINT, &act, NULL );

    if(argc != 2) {
        printf("Usage :%s port\n", argv[0]);
        exit(0);
    }

    listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
    while(1) {
        FD_ZERO(&read_fds);
        FD_SET(listen_sock, &read_fds);
        for(i=0; i<num_chat; i++)
            FD_SET(clisock_list[i], &read_fds);
        maxfdp1 = getmax() + 1;
        puts("wait for client");
        if( select( maxfdp1, &read_fds, NULL, NULL, NULL ) < 0 )
            errquit("select fail");

        if(FD_ISSET(listen_sock, &read_fds)) {
            accp_sock=accept(listen_sock, (struct sockaddr
                *)&cliaddr, &addrlen);
            if(accp_sock == -1)
                errquit("accept fail");
            addClient(accp_sock,&cliaddr);

```

```

        send(accp_sock, START_STRING, strlen(START_STRING), 0);
        printf("Added %dth user.\n", num_chat);
    }

    for(i = 0; i < num_chat; i++) {
        if(FD_ISSET(clisock_list[i], &read_fds)) {
            nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
            if(nbyte <= 0) {
                removeClient(i);
                continue;
            }
            buf[nbyte] = 0;

            if(strstr(buf, EXIT_STRING) != NULL) {
                removeClient(i);
                continue;
            }
            for (j = 0; j < num_chat; j++)
                send(clisock_list[j], buf, nbyte, 0);
            printf("%s\n", buf);
        }
    }
}
return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET, &newcliaddr->sin_addr, buf, sizeof(buf));
    printf("new client: %s\n", buf);

    clisock_list[num_chat] = s;
    num_chat++;
}

void removeClient(int s) {
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
    num_chat--;
    printf("Leave a user. Current users = %d\n", num_chat);
}

int getmax() {
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

int tcp_listen(int host, int port, int backlog) {
    int sd;

```

```

    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }

    listen(sd, backlog);
    return sd;
}

```

```

s200100891@db:~/NetProg/prac7
[s200100891@db prac7]$ ./q1 6660
wait for client

First CTRL-C Pressed!
select fail: Interrupted system call
[s200100891@db prac7]$

```

sa_flags를 SA_RESTART로 설정하였음에도 select()함수에서 인터럽트를 발생하며 멈췄다. 교재 190페이지의 내용처럼 DB서버(Redhat Linux)에서는 select()함수가 시그널 이후 다시 시작되지 않는 예외적인 함수임을 볼 수 있다.

나. 위의 문제에서 SA_RESTART 플래그를 해제하면 결과가 어떻게 달라지는지 확인하시오. 에러가 발생한다면 에러의 이유를 알아보고 SA_RESTART플래그를 설정하지 않고도 정상적으로 동작할 수 있도록 프로그램을 수정하시오.

(힌트: select()가 음수를 리턴하면 에러가 발생한 것이며 이때 errno 변수에 에러 원인이 저장된다. 시그널에 의해 select()가 실패한 경우에는 errno는 EINTR로 설정되므로 errno가 EINTR이면 다시 한 번 select() 함수를 호출하도록 수정하면 된다. recv() 함수에 대해서도 마찬가지다.)

예제 40 서버 프로그램 소스

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <netinet/in.h>

```

```

#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

#define MAXLINE 511
#define MAX_SOCK 1024

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server \n";

int maxfdp1;
int num_chat = 0;
int clisock_list[MAX_SOCK];
int listen_sock;

time_t oldTime = 0;
time_t curTime = 0;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

// signal handler
void catch_sigint( int signum )
{
    time( &curTime );

    if( oldTime == 0 )
    {
        oldTime = curTime;
        printf( "\nFirst CTRL-C Pressed!\n" );
    }
    else if( curTime - oldTime > 3 )
    {
        oldTime = curTime;
        printf( "\nFirst CTRL-C Pressed!\n" );
    }
    else
    {
        printf( "\nSecond CTRL-C Pressed! Program will be exit!\n" );
        exit( 0 );
    }

    return;
}

int main(int argc, char *argv[]) {
    struct sockaddr_in cliaddr;
    char buf[MAXLINE+1];
    int i, j, nbyte, accp_sock, addrlen = sizeof(struct
        sockaddr_in);

```

```

fd_set read_fds;
int n;

struct sigaction act;
sigset_t masksets;
sigfillset( &masksets );

act.sa_handler = catch_sigint;
act.sa_mask = masksets;
act.sa_flags = 0;
sigaction( SIGINT, &act, NULL );

if(argc != 2) {
    printf("Usage :%s port\n", argv[0]);
    exit(0);
}

listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
while(1) {
    FD_ZERO(&read_fds);
    FD_SET(listen_sock, &read_fds);
    for(i=0; i<num_chat; i++)
        FD_SET(clisock_list[i], &read_fds);
    maxfdp1 = getmax() + 1;
    puts("wait for client");
    n = select( maxfdp1, &read_fds, NULL, NULL, NULL );
    if( n == -1 && errno == EINTR )
    {
        printf( "select function Interrupted\n" );
        continue;
    }
    else if( n < 0 )
        errquit("select fail");

    if(FD_ISSET(listen_sock, &read_fds)) {
        accp_sock=accept(listen_sock, (struct sockaddr
        *)&cliaddr, &addrlen);
        if(accp_sock == -1)
            errquit("accept fail");
        addClient(accp_sock,&cliaddr);
        send(accp_sock, START_STRING, strlen(START_STRING), 0);
        printf("Added %dth user.\n", num_chat);
    }

    for(i = 0; i < num_chat; i++) {
        if(FD_ISSET(clisock_list[i], &read_fds)) {
            nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
            if(nbyte<= 0) {
                removeClient(i);
                continue;
            }
            buf[nbyte] = 0;

            if(strstr(buf, EXIT_STRING) != NULL) {

```

```

        removeClient(i);
        continue;
    }
    for (j = 0; j < num_chat; j++)
        send(clisock_list[j], buf, nbyte, 0);
    printf("%s\n", buf);
}
}
}
return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET, &newcliaddr->sin_addr, buf, sizeof(buf));
    printf("new client: %s\n", buf);

    clisock_list[num_chat] = s;
    num_chat++;
}

void removeClient(int s) {
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
    num_chat--;
    printf("Leave a user. Current users = %d\n", num_chat);
}

int getmax() {
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }
}

```

```

    }

    listen(sd, backlog);
    return sd;
}

```

```

s200100891@db:~/NetProg/prac7
[s200100891@db:~/NetProg/prac7]$ ./q2 6660
wait for client

First CTRL-C Pressed!
select function Interrupted
wait for client

Second CTRL-C Pressed! Program will be exit!
[s200100891@db:~/NetProg/prac7]$

```

다. 채팅 서버 프로그램을 수정하여 SIGUSR1 시그널을 받으면 모든 클라이언트에게 서버에 처음 접속한 시간을 보내고, 서버 화면에는 총 접속자 수와 각 클라이언트의 접속한 시간, 클라이언트의 IP 주소를 출력하는 프로그램을 작성하시오.

예제 41 서버 프로그램 소스

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>

#define MAXLINE 511
#define MAX SOCK 1024

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server \n";

int maxfdp1;
int num_chat = 0;

int clisock_list[MAX SOCK];
time_t clitime_list[MAX SOCK];
char cliip_list[MAX SOCK][20];
int listen_sock;

struct sockaddr_in cliaddr;

```

```

char buf[MAXLINE+1];
char buf2[MAXLINE+1];
int i, j, nbyte, accp_sock, addrlen = sizeof(struct sockaddr_in);
fd_set read_fds;
int n;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

// signal handler
void catch_sigusr1( int signum )
{
    printf( "\nCurrent users : %d\n", num_chat );
    for( i = 0; i < num_chat; i++ )
    {
        printf( "%dth user's connected time : %d, IP addr : %s\n",
                i+1, clitime_list[i], cliip_list[i] );
    }

    for(i = 0; i < num_chat; i++) {
        for (j = 0; j < num_chat; j++)
        {
            sprintf( buf2, "Server Conntected Time is %d", clitime_list[j] );
            send( clisock_list[j], buf2, strlen(buf2), 0 );
        }
    }

    return;
}

int main(int argc, char *argv[]) {
    struct sigaction act;
    sigset_t masksets;
    sigfillset( &masksets );

    act.sa_handler = catch_sigusr1;
    act.sa_mask = masksets;
    act.sa_flags = 0;
    sigaction( SIGUSR1, &act, NULL );

    if(argc != 2) {
        printf("Usage :%s port\n", argv[0]);
        exit(0);
    }

    listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
    while(1) {
        FD_ZERO(&read_fds);
        FD_SET(listen_sock, &read_fds);
        for(i=0; i<num_chat; i++)
            FD_SET(clisock_list[i], &read_fds);

```



```

    maxfdp1 = getmax() + 1;
    puts("wait for client");
    n = select( maxfdp1, &read_fds, NULL, NULL, NULL );
    if( n == -1 && errno == EINTR )
    {
        printf( "select function Interrupted\n" );
        continue;
    }
    else if( n < 0 )
        errquit("select fail");

    if(FD_ISSET(listen_sock, &read_fds)) {
        accp_sock=accept(listen_sock, (struct sockaddr
            *)&cliaddr, &addrlen);
        if(accp_sock == -1)
            errquit("accept fail");
        addClient(accp_sock,&cliaddr);
        send(accp_sock, START_STRING, strlen(START_STRING), 0);
        printf("Added %dth user.\n", num_chat);
    }

    for(i = 0; i < num_chat; i++) {
        if(FD_ISSET(clisock_list[i], &read_fds)) {
            nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
            if(nbyte<= 0) {
                removeClient(i);
                continue;
            }
            buf[nbyte] = 0;

            if(strstr(buf, EXIT_STRING) != NULL) {
                removeClient(i);
                continue;
            }
            for (j = 0; j < num_chat; j++)
            {
                send(clisock_list[j], buf, nbyte, 0);
            }
            printf("%s\n", buf);
        }
    }
}
return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET,&newcliaddr->sin_addr,buf,sizeof(buf));
    strcpy( cliip_list[i], buf );
    printf("new client: %s\n",buf);

    clisock_list[num_chat] = s;
    time( &clitime_list[num_chat] );
}

```

```

    num_chat++;
}

void removeClient(int s) {
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
    num_chat--;
    printf("Leave a user. Current users = %d\n", num_chat);
}

int getmax() {
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

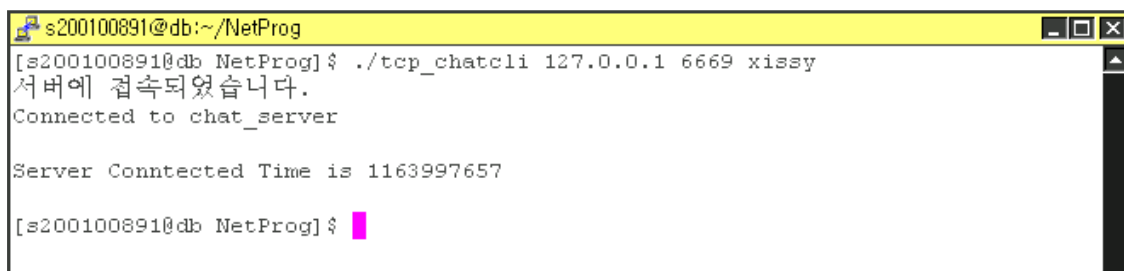
int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }

    listen(sd, backlog);
    return sd;
}

```



```

s200100891@db:~/NetProg
[s200100891@db NetProg]$ ./tcp_chatcli 127.0.0.1 6669 xissy
서버에 접속되었습니다.
Connected to chat_server

Server Connctected Time is 1163997657

[s200100891@db NetProg]$

```

```
s200100891@db:~/NetProg
[s200100891@db NetProg]$ ./tcp_chatcli 127.0.0.1 6669 taeho
서버에 접속되었습니다.
Connected to chat_server

Server Connected Time is 1163997663

[s200100891@db NetProg]$
```

```
s200100891@db:~/NetProg/prac7
[s200100891@db prac7]$ ./q3 6669 &
[1] 1421
wait for client
[s200100891@db prac7]$ new client: 127.0.0.1
Added 1th user.
wait for client
new client: 127.0.0.1
Added 2th user.
wait for client
ps
  PID TTY          TIME CMD
 32520 pts/10    00:00:00 bash
   1421 pts/10    00:00:00 q3
   1426 pts/10    00:00:00 ps
[s200100891@db prac7]$ kill -USR1 1421

Current users : 2
1th user's connected time : 1163997657, IP addr : 127.0.0.1
2th user's connected time : 1163997663, IP addr : 127.0.0.1
select function Interrupted
wait for client
[s200100891@db prac7]$ Leave a user. Current users = 1
wait for client
Leave a user. Current users = 0
wait for client
[s200100891@db prac7]$
```

Chapter 10. 프로세스간 통신

SECTION

01 파이프

운영체제가 제공하는 프로세스간 통신 채널로서 특별한 타입의 파일

1. 파이프의 정의

가. 운영체제가 관리하는 임시 파일

- 1) 일반 파일과 달리 메모리에 저장되지 않음
- 2) 데이터 저장용이 아님
- 3) 프로세스간 데이터 전달용으로 사용

나. 통신 기능

- 1) 송신 프로세스에서 파이프에 데이터를 쓰고 수신 프로세스에서 파이프에서 데이터를 읽음
- 2) 스트림 채널을 제공하여 송신된 데이터는 바이트 순서가 유지됨
- 3) 같은 컴퓨터 내의 프로세스 간에 스트림 채널을 제공

2. pipe()

가. 파이프를 열기 위한 명령

```
#include <unistd.h>
int pipe(int fd[2]);
```

나. 두 개의 파일 디스크립터가 생성됨

- 1) 0번 : 파이프로부터 데이터를 읽음
- 2) 1번 : 파이프에 데이터를 씀

3. fork() 호출

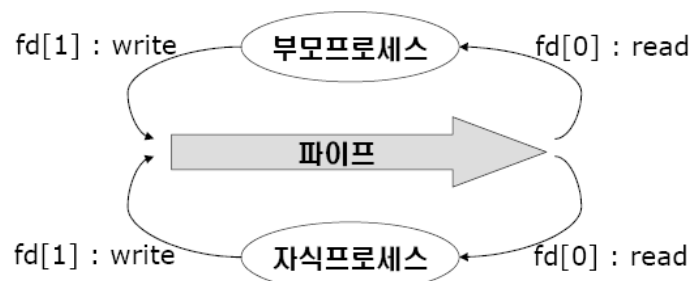


그림 165 파이프 생성 후 fork()를 호출

가. 파일 디스크립터 fd[0], fd[1]

- 1) 부모와 자식 프로세스는 모두 파일 디스크립터 fd[0]로부터 데이터를 읽음
- 2) 부모와 자식 프로세스는 모두 파일 디스크립터 fd[1]로 데이터를 씀

4. 불필요한 파일 디스크립터 해제

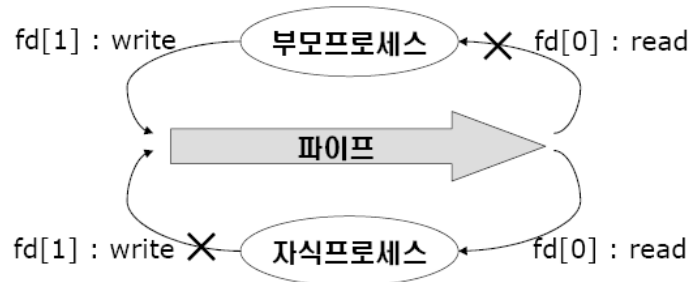


그림 166 사용하지 않는 파일 디스크립터를 닫음

가. 자식 프로세스가 부모 프로세스로 데이터를 보내는 경우

- 1) 자식 프로세스는 fd[1]로 데이터를 쓰고 부모 프로세스는 fd[0]으로 읽음
- 2) 자식 프로세스의 fd[0]과 부모 프로세스의 fd[1]은 사용하지 않으므로 닫음

5. 파이프를 이용한 에코 서버 프로그램

예제 42 파이프를 통해 에코 메시지를 전달하는 UDP 에코 서버

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>

#define MAX_BUFSZ 512

// 파이프에 쓰는 데이터 구조
typedef struct mesg {
    struct sockaddr_in addr;    // 클라이언트 주소
    char data[MAX_BUFSZ];      // 에코할 데이터
} mesg_t;

void child_start(int sock, int pipefd[]);    // 자식 프로세스
void parent_start(int sock, int pipefd[]);    // 부모 프로세스
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char **argv) {
```

```

struct sockaddr_in servaddr;
pid_t pid;
int sock,
    pipefd[2],
    port,
    len = sizeof(struct sockaddr);

if(argc!=2) {
    printf("\n Usage : %s port\n",argv[0]);
    exit(EXIT_FAILURE);
}

// 포트번호
port = atoi(argv[1]);
sock = socket(AF_INET,SOCK_DGRAM,0);
if(sock<0) {
    perror("socket failed");
    exit(EXIT_FAILURE);
}

bzero(&servaddr,len);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_family=AF_INET;
servaddr.sin_port = ntohs(port);
bind(sock, (struct sockaddr*)&servaddr, len);

// 파이프 생성
if(pipe(pipefd) == -1)
    errquit("pipe fail ");

pid = fork();
if (pid < 0) errquit("fork fail");
else if(pid > 0) parent_start(sock, pipefd);
else if(pid ==0) child_start(sock, pipefd);
return 0;
}

// 자식 프로세스
void child_start(int sock,int pipefd[]) {
    mesg_t pmsg;
    int nbytes=0,
        len = sizeof(struct sockaddr);
    close(pipefd[1]);

    while(1) {
        // 부터 읽기 대기
        nbytes = read(pipefd[0], (char *)&pmsg, sizeof(mesg_t));
        if(nbytes < 0)
            errquit("read failed ");
        printf("Child : read from pipe\n",nbytes);
    }
}

```

```

        // 파이프로부터 읽은 데이터를 에코
        nbytes = sendto(sock, &pmsg.data, strlen(pmsg.data), 0,
            (struct sockaddr*)&pmsg.addr, len);
        printf("Child : %d bytes echo response\n",nbytes);
        printf("-----\n");
    }
}

// 부모 프로세스
void parent_start(int sock, int pipefd[]) {
    mesg_t pmsg;
    int nbytes,
        len = sizeof(struct sockaddr);

    // 읽기 파이프 닫음
    close( pipefd[0] );

    printf("my echo server wait...\n");
    while(1) {
        // 소켓으로부터 읽기
        nbytes = recvfrom(sock, (void*)&pmsg.data,
            MAX_BUFSZ, 0, (struct sockaddr*)&pmsg.addr, &len);
        if(nbytes < 0) errquit("recvfrom failed ");
        printf("Parent : %d bytes recv from socket\n",nbytes);
        pmsg.data[nbytes]=0;

        // 소켓으로부터 읽은 데이터를 파이프에 쓰기
        if( write(pipefd[1], (char*)&pmsg, sizeof(pmsg))<0 )
            perror("write fail ");
        printf("Parent : write to pipe\n",nbytes);
    }
}
}

```

예제 43 udp 에코 클라이언트

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

int main(int argc, char *argv[]) {
    struct sockaddr_in peer;
    int sock;
    int nbytes;
    char buf[512];

```



```

if(argc != 3) {
    printf("Usage : %s ip port\n",argv[0]);
    exit(1);
}

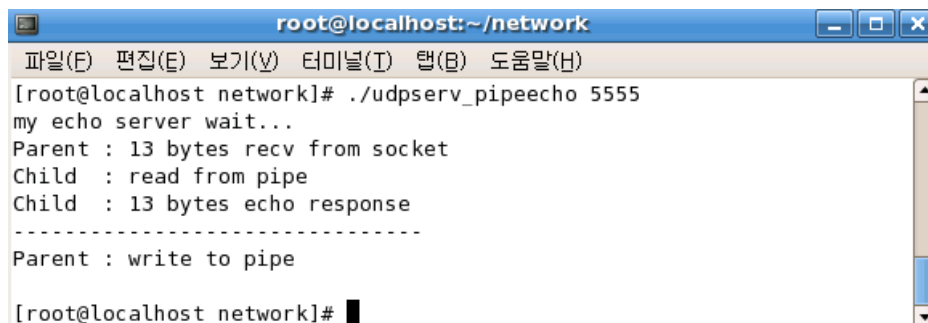
// UDP 소켓 생성 및 바인딩
sock = socket(AF_INET,SOCK_DGRAM,0);
bzero(&peer,sizeof(struct sockaddr));
peer.sin_family=AF_INET;
peer.sin_port = htons(atoi(argv[2]));
peer.sin_addr.s_addr = inet_addr(argv[1]);

// 키보드 입력을 받고 서버로 보내는 부분
while( fgets(buf, sizeof(buf), stdin) != NULL) {
    nbytes = sendto(sock, buf, strlen(buf), 0,
        (struct sockaddr*)&peer, sizeof(peer));
    if(nbytes<0) { perror("sendto fail "); exit(0); }
    nbytes = recvfrom(sock, buf, sizeof(buf)-1, 0,0,0);
    if(nbytes<0) { perror("recvfrom fail "); exit(0); }
    buf[nbytes] = 0;
    fputs(buf, stdout);
}

close(sock);
exit(0);
}

```

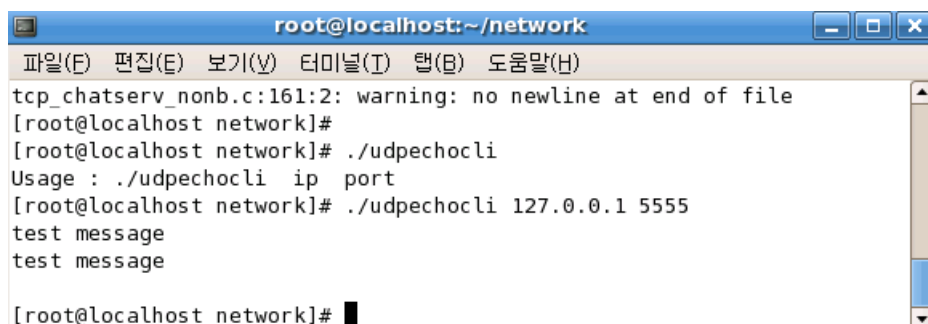
실행 화면은 다음과 같다.



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./udpserve_pipeecho 5555
my echo server wait...
Parent : 13 bytes recv from socket
Child  : read from pipe
Child  : 13 bytes echo response
-----
Parent : write to pipe
[root@localhost network]#

```



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
tcp_chatserv_nonb.c:161:2: warning: no newline at end of file
[root@localhost network]#
[root@localhost network]# ./udpechocli
Usage : ./udpechocli ip port
[root@localhost network]# ./udpechocli 127.0.0.1 5555
test message
test message
[root@localhost network]#

```

가. 동작설명

- 1) 'test message'라는 빈 칸 포함 12바이트에 문장 끝 'NULL' 문자까지 13바이트가 쓰임
- 2) 부모 프로세스는 클라이언트로부터 받은 메시지를 파이프에 씀
- 3) 자식 프로세스는 파이프로부터 데이터를 읽어서 클라이언트로 에코 해줌

SECTION

02 FIFO(First In First Out)

임의의 프로세스에서 파이프에 접근하는 named pipe

1. FIFO의 정의

가. 파이프의 한계 극복

- 1) 파이프는 자식 · 부모 프로세스간의 통신에만 사용됨
- 2) 파이프에 이름을 지정해 가족관계가 아닌 임의의 다른 프로세스에서 접근

2. mkfifo()

```
int mkfifo(const char *pathname, mode_t mode);
```

가. pathname 인자

- 1) 파이프의 이름 지정
- 2) 경로명 없이 파이프의 이름만 인자로 주면 현재 디렉토리에 FIFO 생성

나. mode

- 1) 생성되는 FIFO의 파일 접근 권한 설정
- 2) 일반 파일 생성과 마찬가지로 최종적으로 umask 값에 영향을 받음

3. FIFO를 이용한 프로세스간 통신

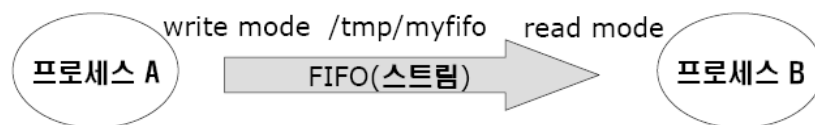


그림 169 FIFO를 이용한 프로세스간 통신

가. FIFO도 파이프의 일종으로 프로세스간에 스트림 채널을 형성

나. FIFO는 파이프와 달리 파일 이름으로 액세스함

4. FIFO를 이용한 에코 서버 프로그램

예제 44 FIFO를 이용하여 에코 메시지를 전달

```
#include <stdio.h>
#include <stdlib.h>
```

```

#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

#define MAX_BUFSZ 512
#define FIFONAME "myfifo"

typedef struct mesg {
    struct sockaddr_in addr;      // 클라이언트 주소
    char data[ MAX_BUFSZ ];      // 읽은 데이터
} mesg_t;

void child_start(int sock);      // 자식 프로세스
void parent_start(int sock);    // 부모 프로세스
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char **argv) {
    struct sockaddr_in servaddr;
    pid_t pid;
    int sock, port, len = sizeof(struct sockaddr);

    if(argc!=2) {
        printf("\n Usage : %s port\n",argv[0]);
        exit(EXIT_FAILURE);
    }

    port = atoi(argv[1]);
    sock = socket(AF_INET,SOCK_DGRAM,0);
    if(sock<0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    bzero(&servaddr,len);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = ntohs(port);
    bind(sock, (struct sockaddr*)&servaddr, len);

    // FIFO 생성
    if(mkfifo(FIFONAME,0660) ==-1 && errno!=EEXIST)
        errquit("mkfifo fail ");
}

```

```

    pid = fork();
    if (pid < 0 ) errquit("fork fail");
    else if(pid > 0) parent_start(sock);
    else if(pid ==0) child_start(sock);

    return 0;
}

// 자식 프로세스
void child_start(int sock) {
    mesg_t pmsg;
    int nbytes,
        fiford, // 읽기 모드의 FIFO
        len = sizeof(struct sockaddr);

    // 읽기 모드로 FIFO open
    fiford = open(FIFONAME,O_RDONLY);
    if(fiford== -1)
        errquit("fifo open fail ");

    while(1) {
        // 파이프로부터 읽기 대기
        nbytes = read(fiford, (char *)&pmsg,sizeof(pmsg));
        if(nbytes<0)
            errquit("read failed ");

        printf("Child  : read from fifo\n",nbytes);
        // 파이프로부터 읽은 데이터를 클라이언트로 전송
        nbytes = sendto(sock, &pmsg.data, strlen(pmsg.data),0
            ,(struct sockaddr*)&pmsg.addr, len);
        printf("Child  : %d bytes echo response\n",nbytes);
        printf("-----\n");
    }
}

// 부모 프로세스
void parent_start(int sock) {
    mesg_t pmsg;
    int nbytes,
        fifowd, // 쓰기 모드의 FIFO
        len = sizeof(struct sockaddr);

    // 쓰기 모드로 FIFO open
    fifowd = open(FIFONAME,O_WRONLY);
    if(fifowd== -1)
        errquit("fifo open fail ");
    printf("my echo server wait...\n");
    while(1) {
        // 소켓으로부터 읽기 대기
        nbytes = recvfrom(sock, (void*)&pmsg.data,

```

```

        MAX_BUFSZ, 0, (struct sockaddr*)&pmsg.addr, &len);
    if(nbytes < 0) errquit("recvfrom failed ");
    pmsg.data[ nbytes ] = 0;
    printf("Parent : %d bytes recv from socket\n", nbytes);

    // 소켓으로부터 읽은 데이터를 FIFO에 쓰기
    if( write(fifowd, &pmsg, sizeof(pmsg)) < 0)
        perror("write fail ");
    printf("Parent : write to fifo\n", nbytes);
}
}

```

서버 프로그램의 실행 화면

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./udpserv_fifoecho 5555
my echo server wait...
Parent : 13 bytes recv from socket
Parent : write to fifo
Child : read from fifo
Child : 13 bytes echo response
-----
[root@localhost network]#

```

클라이언트 프로그램의 실행 화면

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./udpchocli 127.0.0.1 5555
test message
test message
[root@localhost network]#

```

fifo 파일이 생성되었음을 볼 수 있다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ls myfifo
myfifo
[root@localhost network]#

```

가. 동작설명

- 1) 부모 프로세스는 클라이언트로부터 받은 메시지를 FIFO에 씀
- 2) 자식 프로세스는 FIFO로부터 데이터를 읽어서 클라이언트로 에코해 줌
- 3) 디렉토리에 서버 프로그램에서 정한대로 FIFO 파일이 생성됨
 - 가) 여기서는 'myfifo'로 생성됨

SECTION

03 메시지큐

메시지 단위의 송수신용 메시지 큐

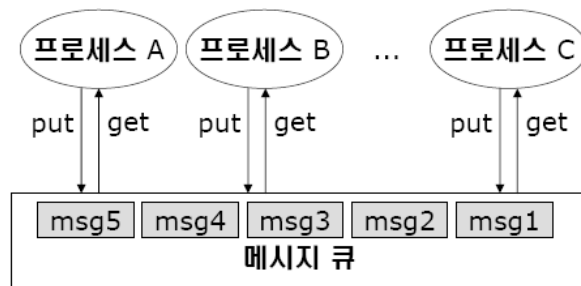


그림 173 메시지큐를 이용한 프로세스간 통신

1. 메시지큐의 정의

가. 특정 프로세스를 대상으로 메시지를 전송할 수 있음

나. 메시지 전송에 우선순위를 줄 수 있음

다. 메시지큐의 기능

1) 임의의 프로세스는 메시지큐에 메시지를 쓰거나(put), 읽을 수(get)있음

라. 메시지큐는 일차원적인 큐를 제공하지 않음

1) 여러 개의 큐를 동시에 제공하는 2차원 큐로 제공됨

2. 타입이 있는 메시지큐

가. 동작방식

1) 각 타입의 메시지큐에 메시지가 있을 때

2) 메시지 송수신 시 프로세스들은 타입을 지정하여야함

가) 메시지큐에 메시지를 쓰거나 읽을 때 타입을 지정

3) 메시지를 쓸 때에 수신할 프로세스의 PID 값을 타입 값으로 지정

4) 해당 프로세스에서는 자신의 PID를 타입 값으로 하여 메시지를 읽음

5) 지정된 타입에서 꺼내올 메시지가 없을 경우

가) 그 보다 작은 타입의 큐에서 메시지를 꺼내옴

나. 메시지큐의 우선순위

1) 타입 값을 우선순위로 사용하면 우선순위가 높은 타입 값이 큰 메시지를 먼저 수신

2) 우선순위가 높은 메시지가 없는 때에만 우선순위가 낮은 메시지를 처리

3) 메시지를 쓰인 순서나 우선순위 순으로 읽을 수 있음

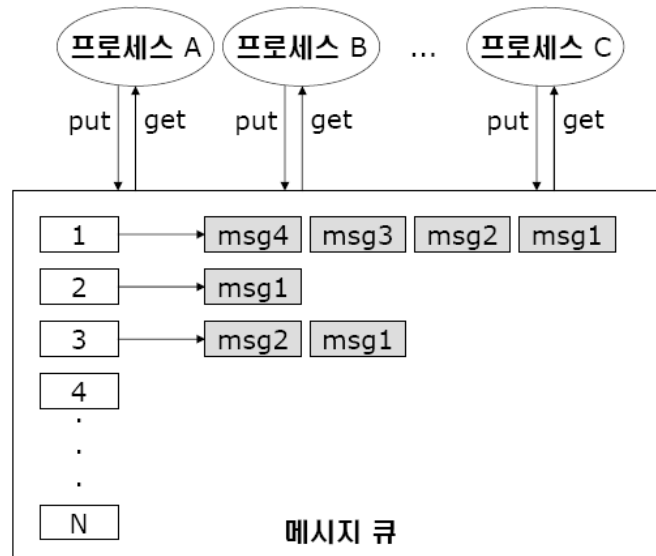


그림 174 타입이 있는 메시지큐

3. 메시지큐 생성

가. msgget()

```
int msgget(key_t key, int msgflg);
```

1) key

- 가) 메시지큐를 구분하기 위한 고유 키
- 나) 다른 프로세스에서 이 메시지큐에 접근하기 위해서는 key 값을 알아야함

2) msgflg 인자

- 가) 메시지큐의 생성시 옵션을 지정
- 나) bitmask 형태의 인자를 취함
- 다) IPC_CREAT, IPC_EXCL 등의 상수와 유닉스 파일 접근 권한도 bitmask 형태로 추가 지정 가능

나. msgflg 설정

```
int mode = 0660;
int msgflag = IPC_CREAT | mode;
int msgid = msgget(key, size, msgflag);
```

- 1) msgget()을 호출시 똑같은 key 값을 사용하는 메시지큐 존재 시 그 객체에 대한 ID를 리턴
- 2) key 값에 대한 메시지큐 객체가 존재하지 않으면 새로운 메시지큐 객체를 생성하고 ID를 리턴

다. msgflg 추가 설정


```
int mode = 0660;
int msgflag = IPC_CREAT | IPC_EXCL | mode;
int msgid = msgget(key, size, msgflag);
```

- 1) IPC_EXCL을 추가 설정하여 key 값을 사용하는 메시지큐 존재 시 msgget() 실패 후 -1 리턴
- 2) IPC_EXCL은 IPC_CREAT와 같이 사용하여야 함
- 3) key 값에 IPC_PRIVATE를 넣을 경우 key가 없는 메시지큐 생성
 - 가) 메시지큐 ID를 공유하는 부모 · 자식 프로세스간은 통신하나 외부 프로세스는 접근 불가

4. 메시지 송수신

가. msgsnd()

```
int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);
```

- 1) 메시지큐에 메시지를 넣는 함수
- 2) msqid 인자
 - 가) 메시지큐 객체의 ID
- 3) msgp 인자
 - 가) msgbuf 인자의 첫 4바이트는 반드시 long 타입이어야 함
 - 나) mtype는 메시지 타입
 - (1) 반드시 1 이상이어야 함
 - 다) mtext
 - (1) 메시지 데이터
 - (2) 문자열일 필요는 없으며 binary등 임의의 데이터를 가리킬 수 있음
- 4) msgsz 인자
 - 가) 메시지 데이터의 길이를 나타내며 mtext의 크기만을 나타냄
- 5) msgflg 인자
 - 가) 0으로 한 경우
 - (1) msgsnd() 호출시 메시지큐 공간이 부족하면 블록됨
 - 나) IPC_NOWAIT로 한 경우
 - (1) 메시지큐 공간이 부족한 경우 블록되지 않고 EAGAIN 에러코드와 -1을 리턴

나. msgrcv()

```
ssize_t msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz, long msgtype, int msgflg);
```

- 1) 메시지큐로부터 메시지를 읽는 함수
- 2) msqid 인자
 - 가) 메시지큐 객체의 ID
- 3) msgp 인자

- 가) 메시지큐로부터 읽은 메시지를 저장하는 수신 공간
- 나) msgbuf 타입의 구조체를 가리킴
- 4) msgsz 인자
 - 가) 수신 공간의 크기
- 5) msgtype 인자
 - 가) 읽을 메시지의 타입을 지정
 - 나) 0이면 타입의 구분 없이 메시지큐에 입력된 순서대로 메시지를 읽음
 - 다) 음수 값을 넣으면 특별한 동작을 함
 - (1) 예로 -10이면 타입이 10보다 같거나 작은 메시지를 읽는데 10인 메시지부터 우선순위
 - (2) 우선순위 큐의 구현에 사용할 수 있음
- 6) msgflag 인자
 - 가) 메시지큐에 메시지가 없는 경우 취할 동작을 지정
 - 나) 0이면 데이터가 없을 때 msgrcv() 함수는 대기
 - 다) IPC_NOWAIT이면 EAGAIN 에러코드와 -1 리턴
 - 라) 읽을 메시지가 수신 공간의 크기보다 클 경우
 - (1) E2BIG 에러 발생
 - 마) MSG_NOERROR로 설정 시 msgsz 크기만큼만 읽고 뒷부분은 잘려짐

5. 메시지큐 제어

가. msgctl()

- 1) 메시지큐를 제어
 - 가) 정보 읽기, 동작 허가 권한 변경, 메시지큐 삭제 등
- 2) 사용 문법

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

가) msqid

- (1) 메시지큐 객체 ID

나) cmd

- (1) 제어 명령 구분
- (2) IPC_STAT로 메시지큐 객체를 얻은 후 메시지큐 객체를 변경 후 IPC_SET 호출
- (3) IPC_STAT : 메시지큐 객체에 대한 정보를 얻어오는 명령
- (4) IPC_SET : r/w 권한, euid, egid, msg_qbytes를 변경하는 명령
- (5) IPC_RMID : 메시지큐를 삭제하는 명령

6. 메시지큐 이용 예

예제 45 에코 메시지를 수신하여 메시지큐에 넣어줌

```
#include <stdio.h>
```

```

#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_BUFSZ 512
#define RES_SEND_PROC "./res_send"

// 메시지 구조 정의
typedef struct _msg {
    long msg_type;
    struct sockaddr_in addr;
    char msg_text[ MAX_BUFSZ ];
} msg_t ;

void fork_and_exec(char *key, char *port);
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char *argv[]) {
    struct sockaddr_in servaddr;
    msg_t pmsg;
    key_t key;
    int msqid, size,
        nbytes, sock, port,
        len = sizeof(struct sockaddr);

    if(argc != 3) {
        printf("Usage: %s msgq_key port\n",argv[0]);
        exit(1);
    }
    key = atoi(argv[1]);
    port = atoi(argv[2]);

    // 메시지큐 생성
    msqid = msgget(key, IPC_CREAT | 0600);
    if(msqid == -1) errquit("msgget fail ");
    // 소켓 생성 및 바인딩
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if(sock<0) errquit("socket fail ");

    bzero(&servaddr,len);
    servaddr.sin_port = htons(port);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_family=AF_INET;

```

```

if(bind(sock, (struct sockaddr*)&servaddr, len) < 0)
    errquit("bind fail ");
fork_and_exec(argv[1], argv[2]);
fork_and_exec(argv[1], argv[2]);
fork_and_exec(argv[1], argv[2]);

pmsg.msg_type = 1;
size = sizeof(msg_t) - sizeof(long);
puts("Server starting...");
while(1) {
    nbytes = recvfrom(sock, pmsg.msg_text, MAX_BUFSZ, 0,
                     (struct sockaddr *)&pmsg.addr, &len);
    if(nbytes < 0) { perror("recvfrom fail "); continue; }
    pmsg.msg_text[ nbytes ] = 0;

    // 메시지큐에 쓰기
    if( msgsnd(msqid, &pmsg, size, 0)==-1 )
        errquit("msgsnd fail ");
}
return 0;
}

void fork_and_exec(char *key, char *port) {
    pid_t pid = fork();
    if (pid < 0) errquit(" fork fail");
    else if(pid > 0)
        return;
    execlp(RES_SEND_PROC, RES_SEND_PROC, key, port, 0);
    perror("execlp fail ");
}

```

예제 46 메시지큐에서 에코 메시지를 읽어서 응답해줌 (res_send)

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_BUFSZ 512

typedef struct _msg {

```

```

    long msg_type;
    struct sockaddr_in addr;
    char msg_text[MAX_BUFSZ];
} msg_t ;

// 메시지큐에서 읽어서 응답하기
int qread_and_echoreply(int msqid, int sock);
void errquit(char *mesg) { perror(mesg); exit(1); }

int main(int argc, char **argv) {
    struct sockaddr_in servaddr;
    key_t key;
    int sock, port, msqid,
        len = sizeof(struct sockaddr);

    key = atoi(argv[1]);
    port = atoi(argv[2]);
    if((msqid=msgget(key,0))==-1)
        errquit("rep msgget fail ");

    // 소켓 생성 및 바인딩
    sock = socket(AF_INET,SOCK_DGRAM, 0);
    bzero(&servaddr,len);
    servaddr.sin_family=AF_INET;
    servaddr.sin_port = htons(port);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(sock, (struct sockaddr*)&servaddr, len);

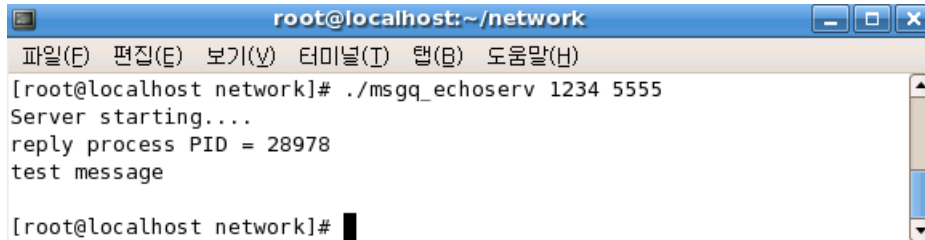
    // msg queue에서 읽어서 reply
    qread_and_echoreply( msqid , sock);
    return 0;
}
// 메시지큐에서 읽어서 응답하기
int qread_and_echoreply(int msqid, int sock) {
    int size,
        len = sizeof(struct sockaddr);
    msg_t pmsg;
    size = sizeof(msg_t) - sizeof(long);
    pmsg.msg_type = 0;
    for( ; ; ) {
        if(msgrcv( msqid,(void *)&pmsg, size, 0, 0) < 0 )
            errquit("msgrcv fail ");
        printf("reply process PID = %d\n", getpid());
        printf(pmsg.msg_text);
        if(sendto(sock, pmsg.msg_text, strlen(pmsg.msg_text), 0,
            (struct sockaddr*)&pmsg.addr,len) < 0)
            errquit("serv2 sendto fail ");
        pmsg.msg_text[0] = 0;
    }
    return 0;
}

```

```
}

```

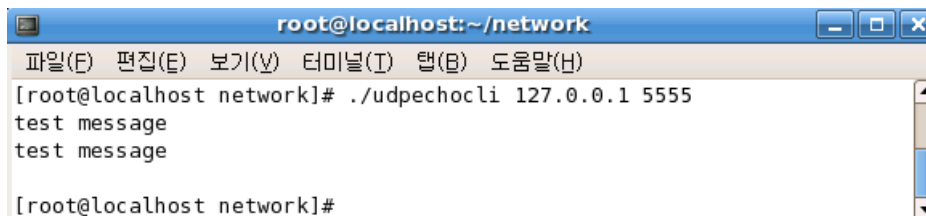
실행 화면은 다음과 같다.



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./msgq_echoserv 1234 5555
Server starting...
reply process PID = 28978
test message
[root@localhost network]#

```



```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./udpechocli 127.0.0.1 5555
test message
test message
[root@localhost network]#

```

가. 동작설명

- 1) res_send.c는 exec가 호출될 때 인자로 넘어온 msgq key와 port 값으로부터 해당 메시지큐의 키와 포트번호를 얻음
- 2) 메시지큐에서 읽기를 대기하다가 메시지가 도착하면 클라이언트에게 응답 메시지 전송

SECTION

04 공유메모리

프로세스들이 공통으로 사용할 수 있는 메모리 영역

1. 공유메모리의 정의

가. 프로세스들이 공통으로 사용하는 메모리 영역

- 1) 어떤 프로세스에서 사용 중인 메모리는 그 프로세스만 접근가능
- 2) 공유메모리는 특정 메모리 영역을 다른 프로세스와 공유
 - 가) 프로세스간 통신 가능
- 3) 통신에서 데이터를 한 번 읽어도 데이터가 계속 남아 있음
- 4) 같은 데이터를 여러 프로세스가 중복하여 읽을 경우 효과적

2. 공유메모리 생성

가. shmget()

```
int shmget(key_t key, int, size, int msgflg);
```

- 1) key
 - 가) 공유메모리를 구분하기 위한 고유 키
 - 나) 다른 프로세스에서 이 공유메모리에 접근하기 위해서는 key 값을 알아야함
- 2) shmflg 인자
 - 가) 공유메모리의 생성시 옵션을 지정
 - 나) bitmask 형태의 인자를 취함
 - 다) IPC_CREAT, IPC_EXCL등의 상수와 유닉스 파일 접근 권한도 bitmask 형태로 추가 지정 가능

나. shmflg 설정

```
int mode = 0660;
int shmflag = IPC_CREAT | mode;
int shmid = shmget(key, size, shmflag);
```

- 1) shmget()을 호출시 똑같은 key 값을 사용하는 공유메모리 존재 시 그 객체에 대한 ID를 리턴
- 2) key 값에 대한 공유메모리 객체가 존재하지 않으면 새로운 공유메모리 객체를 생성하고 ID를 리턴

다. shmflg 추가 설정

```
int mode = 0660;
int shmflag = IPC_CREAT | IPC_EXCL | mode;
int shmid = shmget(key, size, shmflag);
```

- 1) IPC_EXCL을 추가 설정하여 key 값을 사용하는 공유메모리 존재 시 shmget() 실패 후 -1 리턴
- 2) IPC_EXCL은 IPC_CREAT와 같이 사용하여야 함
- 3) key 값에 IPC_PRIVATE를 넣을 경우 key가 없는 공유메모리 생성
- 4) 공유메모리 ID를 공유하는 부모 · 자식 프로세스간은 통신하나 외부 프로세스는 접근 불가

3. 공유메모리 첨부

가. 공유메모리의 물리적 주소를 자신의 프로세스의 가상메모리 주소로 매핑

나. **shmat()**

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- 1) shmid인자 : 공유메모리 객체 ID
 - 가) shmaddr : 프로세스의 메모리 주소 중 첨부시킬 주소
 - (1) 0을 넣으면 커널이 자동으로 빈 공간을 찾아서 처리함
 - 나) shmflg
 - (1) SHM_RDONLY : 공유메모리를 읽기 전용으로 첨부
 - (2) 0 : 공유메모리에 읽기, 쓰기 가능
 - 다) 정상 작동시 프로세스 내의 첨부된 주소 리턴
 - 라) 에러 발생시 NULL 포인터 리턴

4. 공유메모리의 분리

가. 공유메모리 사용 종료 후 자신이 사용하던 메모리 영역에서 분리

나. **shmdt()**

```
int shmdt(const void *shmaddr);
```

- 1) shmaddr 인자
 - 가) shmdt()가 리턴했던 주소
 - 나) 현재 프로세스에 첨부된 공유메모리의 시작주소

다. 공유메모리가 메모리에서 삭제된 것이 아님

- 1) 현재 프로세스에서만 사용하지 못함
- 2) 다른 프로세스에서는 계속 사용 가능

라. 공유메모리 객체 **shmid_ds** 구조체의 **shm_nattch** 멤버 변수

- 1) 현재 공유메모리를 첨부하고 있는 프로세스의 수를 나타냄
- 2) shmat() 호출시 1증가
- 3) shmdt() 호출시 1감소

5. 공유메모리 제어

가. shmctl()

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- 1) 공유메모리를 제어
 - 가) 정보 읽기, 동작 허가 권한 변경, 공유메모리 삭제 등
- 2) 사용 문법
 - 가) shmid
 - (1) 공유메모리 객체 ID
 - 나) cmd
 - (1) 제어 명령 구분
 - (2) IPC_STAT : 공유메모리 객체에 대한 정보를 얻어오는 명령
 - (3) struct shmid_ds *buf 인자에 공유메모리 객체 복사
 - (4) IPC_SET : r/w 권한, euid, egid, msg_qbytes를 변경하는 명령
 - (5) IPC_RMID : 공유메모리를 삭제하는 명령
 - 다) buf
 - (1) cmd에 따라 의미가 바뀌는 인자
 - (2) 공유메모리 객체 정보를 얻어오는 명령어에서는 얻어온 객체를 buf에 저장
 - (3) 동작 허가 권한을 변경하는 명령에서는 변경할 내용을 buf에 저장

6. 공유메모리 정보 얻기

가. 공유메모리 객체 정보 얻기

```
struct shmid_ds shmids;
shmctl(shmid, IPC_STAT, buf);
```

- 1) IPC_STAT 명령으로 공유메모리 객체 정보 얻음
- 2) buf 인자에 저장

나. 공유메모리 객체의 동작허가 권한 변경

```
shmctl(shmid, IPC_SET, &shmids);
```

- 1) 기존의 공유메모리 객체의 shmids.shm_perm.uid, shmids.shm_perm.gid, shmids.shm_perm.mode 값만 변경됨
- 2) 위의 값들이 변경되면 shmids의 shm_ctime 값이 변경된 시각으로 바뀜

다. 공유메모리 객체의 접근 권한 변경

```

struct shmid_ds shmds;
shmctl(shmid, IPC_STAT, &shmds);
shmds.shm_perm.uid = changeuid;
shmds.shm_perm.gid = changegid;
shmds.shm_perm.mode = changemode;
shmctl(shmid, IPC_SET, &shmds);

```

- 1) struct shmid_ds shmds; : 공유메모리 객체 선언
- 2) shmctl(shmid, IPC_STAT, &shmds); : 공유메모리 객체 얻어옴
- 3) shmds.shm_perm.uid = changeuid; : 실행 권한 uid 값을 변경
- 4) shmds.shm_perm.gid = changegid; : 실행 권한 gid 값을 변경
- 5) shmds.shm_perm.mode = changemode; : 접근 권한 변경
- 6) shmctl(shmid, IPC_SET, &shmds); : 공유메모리 객체 내용 변경

7. 공유메모리 삭제

가. shmctl()의 IPC_RMID 명령을 이용

```
shmctl(shmid, IPC_RMID, 0);
```

나. 공유메모리 삭제 요청시 하나 이상의 다른 프로세스가 이 공유메모리를 사용시 공유메모리는 삭제되지 않음

- 1) shm_nattach 값이 0이 될 때까지 기다린 후 삭제됨

8. 공유메모리의 동기화문제

가. 동기화 문제

- 1) 하나의 공유데이터를 둘 이상의 프로세스가 동시에 접근하는 문제

예제 47 공유메모리에서 동기화문제를 보임

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

void errquit(char *msg){perror(msg); exit(1);}
// 자식 프로세스를 생성하여 busy() 함수를 호출하도록 함
void fork_and_run();
// 각 프로세스가 공유메모리에 경쟁적으로 접근하는 함수
void busy();
// 공유메모리에 접근하는 함수

```

```

void access_shm(int count);
char *shm_data;          // 공유메모리 포인터
int shmidx;              // 공유메모리 ID

int main(int argc, char *argv[]) {
    key_t  shmkey;        // 공유메모리 키

    if(argc<2) {
        printf("Usage : %s shmkey\n",argv[0]);
        exit(1);
    }
    shmkey = atoi(argv[1]);
    shmidx = shmget(shmkey, 128, IPC_CREAT|0660);
    if(shmidx<0)
        errquit("shmget fail ");
    shm_data = (char *)shmat(shmidx, (void*)0, 0);
    if(shm_data == (char*)-1)
        errquit("shmat fail");
    fork_and_run();
    fork_and_run();
    busy();
    wait(NULL);
    shmctl(shmidx,IPC_RMID,0); // 공유메모리 제거
    return 0;
}

// 자식 프로세스 생성 및 busy() 수행
void fork_and_run() {
    pid_t pid = fork();
    if (pid < 0)
        errquit("fork fail");
    else if(pid == 0) {
        busy();
        exit(0);
    }
    return ;
}

// 자식 프로세스가 수행하는 함수
void busy() {
    int i=0;
    for( i=0;i<500000 ;i++ )
        access_shm(i);
    shmdt(shm_data);      // 공유메모리 분리
}

// 공유메모리에 접근을 하는 부분
void access_shm(int count) {
    int i;
    pid_t pid;

```

```

// 공유메모리에 자신의 PID 기록
sprintf(shm_data, "%d", getpid());
// 공유메모리 접근 시간에 포함
for(i=0; i<1000; i++);
pid = atoi(shm_data);
// 공유메모리에 기록한 PID가 자신의 PID가 아니면 Error
if(pid != getpid())
    printf("Error(count=%d) : 다른 프로세스도 동시에 공유메모리 접근함\n",count);
else {
    // 정상이며 아무 출력을 하지 않음
}
return;
}
}

```

실행 화면은 다음과 같다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./shmbusyaccess 1234
Error(count=2934) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=1256) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=1443) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=18816) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=38561) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=18291) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=77149) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=91741) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=52523) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=113473) : 다른 프로세스도 동시에 공유메모리 접근함
Error(count=128037) : 다른 프로세스도 동시에 공유메모리 접근함

```

나. 동작설명

- 1) 공유메모리를 생성하고 fork()를 두 번 호출하여 자식 프로세스를 두 개 생성함
- 2) 부모 프로세스를 포함한 3개의 프로세스가 busy()를 통해 공유메모리에 경쟁적으로 접근함

SECTION

05 세마포어

한 순간에 공유데이터를 액세스하는 프로세스 수를 하나로 제한

1. 세마포어의 정의

가. 어떤 공유데이터에 대해 현재 사용 가능한 데이터의 수

나. 크리티컬 영역

- 1) 동기화문제가 발생될 수 있는 코드 블록

2. 세마포어 생성

가. `semget()`

```
int semget(key_t key, int nsems, int semflg);
```

- 1) `key` : 세마포어를 구분하기 위해 지정해 주는 키
- 2) `nsems` : 세마포어 집합을 구성하는 멤버의 수
- 3) `semflg` : 세마포어 생성에 관한 옵션을 설정하거나 세마포어 객체에 대한 접근 권한 설정
- 4) 성공적으로 수행시 세마포어 집합에 관한 정보를 담고 있는 세마포어 객체 생성
- 5) 세마포어 객체 ID를 리턴

3. 플래그 사용 예

가. `IPC_CREAT`

```
semget(key, nsems, IPC_CREAT | mode);
```

- 1) `key`에 해당하는 세마포어 객체가 존재하면 기존의 세마포어 ID를 리턴
- 2) `key`에 해당하는 세마포어 객체가 없으면 새로 생성하고 세마포어 객체 ID 리턴
- 3) `semget(key, 0, 0)`
 - 가) `key`값을 갖는 세마포어의 ID를 단순히 알기위해 사용

나. `IPC_EXCL`

```
mode = 0660;
semget(key, nsems, IPC_CREAT | IPC_EXCL | mode);
```

- 1) `key`값에 `IPC_PRIVATE`를 사용하면 `key`가 없는 세마포어 객체 생성
 - 가) 키가 없는 세마포어 이므로 다른 프로세스에서 접근 못함
- 2) 세마포어 생성 후 `fork()`를 호출하면 자식 프로세스는 세마포어 객체의 ID를 상속받음
 - 가) `semget()` 함수를 사용하지 않고 세마포어에 접근

4. 세마포어 객체

```
struct semid_ds {
    struct ipc_perm sem_perm;
    time_t sem_otime;
    time_t sem_ctime;
    struct sem *sem_base;
    struct wait_queue *eventn;
    struct wait_queue *eventz;
    struct sem_undo *undo;
    ushort sem_nsems;
};
```

가. struct ipc_perm sem_perm;

- 1) 접근 허가 내용

나. time_t sem_otime;

- 1) 최근 세마포어 조작 시간

다. time_t sem_ctime;

- 1) 최근 변경 시각

라. struct sem *sem_base;

- 1) 첫 세마포어 포인터

마. ushort sem_nsems;

- 1) 세마포어 멤버 수

5. 세마포어 연산

가. 세마포어의 값을 증가 또는 감소하는 것

나. semop()

```
int semop(int semid, struct sembuf *operations, unsigned nsops);

struct sembuf {
    short sem_num;
    short sem_op;
    short sem_flg;
};
```

- 1) semid : 세마포어 ID

- 2) operations : sembuf 타입의 구조체를 가짐

가) short sem_num; - 멤버 세마포어 번호(첫 번째 멤버 세마포어는 0)

나) short sem_op; - 세마포어 연산 내용

다) short sem_flg;

(1) 조작 플래그, 0, IPC_NOWAIT, SEM_UNDO등의 값을 bitmask 형태로 가질 수 있음

- (2) IPC_NOWAIT시 세마포어 값이 부족해도 블록되지 않음
- (3) SEM_UNDO시 프로세스의 종료 시 커널은 해당 세마포어 연산 취소
- 3) nsops : 두 번째 인자 operations 구조체가 몇 개의 리스트를 가지고 있는지 나타냄

6. 세마포어 제어

가. semctl()

```
int semctl(int semid, int member_index, int cmd, union semun semarg);
```

- 1) 세마포어의 사용종료, 세마포어 값 읽기 및 설정, 특정 멤버 세마포어를 기다리는 프로세스의 수 알기에 사용
- 2) semid
 - 가) 제어할 대상의 세마포어 ID
- 3) member_index
 - 가) 세마포어 멤버 번호
- 4) cmd
 - 가) 수행할 동작
- 5) semum 타입의 semarg 인자
 - 가) cmd의 종류에 따라 각각 다르게 사용될 수 있는 인자

나. semum 공용체

```
union semum {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *__buf;
    void *__pad;
};
```

- 1) int val;
 - 가) SETVAL을 위한 값
- 2) struct semid_ds *buf;
 - 가) IPC_STAT, IPC_SET을 위한 버퍼
- 3) unsigned short int *array;
 - 가) GETALL, SETALL을 위한 배열
- 4) struct seminfo *__buf;
 - 가) IPC_INFO를 위한 버퍼
- 5) void *__pad;
 - 가) dummy

다. IPC_STAT

```
struct semid_ds semobj;
union semun semarg;
semarg.buf = &semobj;
semctl(semid, 0, IPC_STAT, semarg);
```

- 1) semid_ds타입의 세마포어 객체를 얻어오는 명령
- 2) union semun semarg인자에 세마포어 객체가 리턴
- 3) semarg.buf 포인터를 통해 세마포어 객체를 접근
- 4) IPC_STAT 명령을 사용할 때엔 semctl()하수의 member_index 인자는 사용되지 않음(0으로 설정)

라. SETVAL

```
union semun semarg;
unsigned short semvalue = 5;
semarg.val = semvalue;
semctl(semid, 1, SETVAL, semarg);
```

- 1) 세마포어를 생성한 후에 공유데이터의 수를 설정해 주는 작업
- 2) 세마포어 객체의 초기화를 하는 함수
- 3) union semun semarg 인자에는 설정하려는 세마포어의 값을 지정
- 4) semarg인자의 semarg.val 변수에 원하는 초기 값을 입력
- 5) 위의 예에서는 1번 멤버 세마포어의 값을 5로 설정

마. SETALL

```
unsigned short values = {1, 2, 3, 4};
union semun semarg;
semarg.array = values;
semctl(semid, 0, SETALL, semarg);
```

- 1) 세마포어 집합 내의 모든 세마포어의 값을 초기화하는 명령
- 2) union semun semarg 인자의 semarg.array인자에 초기 값을 배열 형태로 넣어줌
- 3) member_index는 사용되지 않음
- 4) 위의 예제에서는 세마포어 집합의 멤버 세마포어 수는 4이며 1, 2, 3, 4로 초기화

바. GETVAL

```
int n semctl(semid, 1, GETVAL, 0);
```

- 1) 특정 멤버 세마포어의 현재 값을 얻어오는 명령
- 2) 위의 예제에서는 1번 멤버 세마포어 값을 얻어옴

사. GETALL


```
union semun semarg;
unsigned short semvalues[4];
semarg.array = semvalues;
semctl(semid, 0, GETALL, semarg);
```

- 1) 모든 멤버 세마포어의 현재 값을 읽음
- 2) 위의 예제에서는 세마포어 집합의 멤버 수가 4인 경우 각 멤버 세마포어 값을 읽음

아. GETNCNT

```
semctl(semid, member_index, GETNCNT, 0);
```

- 1) 세마포어 값이 원하는 값 이상으로 증가되기를 기다리는 프로세스의 수를 얻음
가) 특정 멤버 세마포어를 사용하기 위해 블록되어 있는 프로세스의 수
- 2) semctl() 함수의 리턴 값으로 세마포어를 기다리는 프로세스의 수를 얻음
- 3) union semun semarg 인자는 사용되지 않음

자. GETPID

```
int pid = semctl(semid, member_index, GETPID, 0);
```

- 1) 특정 멤버 세마포어에 대해 마지막으로 semop() 함수를 수행한 프로세스 PID를 얻음

차. IPC_RMID

```
semctl(semid, 0, IPC_RMID, 0);
```

- 1) 세마포어를 삭제

7. 세마포어 이용 예

예제 48 공유데이터의 경쟁적 접근을 세마포어로 제어

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/sem.h>

void errquit(char *mesg) {
    perror(mesg);
    exit(0);
}
```

```

}

#define PEN 0
#define NOTE 1

// 세마포어 조작
struct sembuf increase[] = {
    {0, +1, SEM_UNDO}, {1, +1, SEM_UNDO} };
struct sembuf decrease[] = {
    {0, -1, SEM_UNDO}, {1, -1, SEM_UNDO} };

// 세마포어 초기값, 연필 한 자루와 노트 두권
unsigned short seminitval[] = { 1, 2 };

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *__buf;
} semarg;

int semid;           // 세마포어 ID
void do_work();      // 각 프로세스가 수행할 작업

int main(int argc, char *argv[]) {
    semid = semget(0x1234, 2, IPC_CREAT | 0600);
    if(semid == -1)
        semid=semget(0x1234, 0, 0);

    // 세마포어 값 초기화
    semarg.array = seminitval;
    if(semctl(semid, 0, SETALL, semarg) == -1)
        errquit("semctl");

    // 표준 출력 non-buffering
    setvbuf(stdout, NULL, _IONBF, 0);

    // 총 4개의 프로세스를 만듦
    fork(); fork();
    do_work();
    semctl(semid, 0, IPC_RMID, 0); // 세마포어의 삭제
    return 0;
}

void do_work() {
    int count=0;
#define Semop(val) if((semop val)==-1) errquit("semop")
    while(count<3) {
        Semop((semid, &decrease[PEN], 1));
        printf("[pid:%5d]연필을 들고\n", getpid());

```

```

    Semop({semid, &decrease[NOTE], 1});
    printf("\t[pid:%5d] 노트를 들고\n", getpid());

    printf("\t[pid:%5d] 공부를 함\n", getpid());

    Semop({semid, &increase[PEN], 1});
    printf("\t[pid:%5d] 연필을 내려놓음\n", getpid());
    Semop({semid, &increase[NOTE], 1});
    printf("\t[pid:%5d] 노트를 내려놓음\n", getpid());

    sleep(1);
    count++;
}
}

```

실행 화면은 다음과 같다.

```

root@localhost:~/network
[root@localhost network]# ./pen_and_note
[pid:29098]연필을 들고
    [pid:29098] 노트를 들고
    [pid:29098] 공부를 함
    [pid:29098] 연필을 내려놓음
    [pid:29098] 노트를 내려놓음
[pid:29100]연필을 들고
    [pid:29100] 노트를 들고
    [pid:29100] 공부를 함
    [pid:29100] 연필을 내려놓음
    [pid:29100] 노트를 내려놓음
[pid:29097]연필을 들고
    [pid:29097] 노트를 들고
    [pid:29097] 공부를 함
    [pid:29097] 연필을 내려놓음
    [pid:29097] 노트를 내려놓음
[pid:29099]연필을 들고
    [pid:29099] 노트를 들고

```

가. 동작설명

- 1) 이 프로그램은 세마포어의 이용 예를 보이는 최소한의 코드임
- 2) IPC_RMID에 의해 세마포어를 삭제하는 코드는 4개의 프로세스에 의해 4번 실행
가) 실제로 3번은 실패
- 3) do_work()실행 동안 다른 프로세스에 의해 세마포어가 삭제될 수 있음

8. 공유메모리의 동기화문제 처리

예제 49 세마포어를 이용한 공유메모리 동기화문제 해결

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

```

```

#include <errno.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/types.h>

// 세마포어 값을 조절하는 sembuf
struct sembuf waitsem[] = { {0, -1, 0} };
struct sembuf notifysem[] = { {0, +1, 0} };

// semop 함수 wrapper 매크로
// #define Semop(val) \
// { if(semop val== -1) \
//     errquit("semop fail"); \
// }
#define Semop(val) { if(semop val== -1) errquit("semop fail"); }
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *__buf;
} semarg;

void errquit(char *msg){perror(msg); exit(1);}

// 자식 프로세스를 생성하여 busy() 함수를 호출
void fork_and_run();

void busy();           // 각 프로세스가 공유메모리에 접근하는 함수
void access_shm();     // 공유메모리에 접근하는 함수
char *shm_data;        // 공유메모리에 대한 포인터
int shmid, semid;       // 공유메모리 ID, 세마포어 ID

int main(int argc, char *argv[]) {
    key_t shmkey, semkey;      // 공유메모리, 세마포어 키
    unsigned short initsemval[1]; // 세마포 초기값
    if(argc<2) {
        printf("Usage : %s shmkey semkey\n", argv[0]);
        exit(1);
    }
    shmkey = atoi(argv[1]);
    semkey = atoi(argv[2]);

    // 공유메모리 생성
    shmid = shmget(shmkey, 128, IPC_CREAT|0660);
    if(shmid<0)
        errquit("shmget fail ");
    // 공유메모리 첨부
    shm_data = (char *)shmat(shmid, (void*)0, 0);
    if(shm_data == (char*)-1)

```

```

    errquit("shmat fail");
// 세마포어 생성
semid = semget(semkey, 1, IPC_CREAT|0660);
if(semid == -1)
    errquit("semget fail ");

// 세마포어의 초기값을 1로 설정
// 공유메모리에 한 프로세스만이 접근가능함
initsemval[0] = 1;
semarg.array = initsemval;
if( semctl(semid,0,SETALL, semarg)==-1)
    errquit("semctl ");

fork_and_run(); // 자식 프로세스 생성
fork_and_run(); // 자식 프로세스 생성
busy();        // 부모 프로세스도 공유메모리 접근
wait(NULL);    // 자식 프로세스가 끝나기를 기다림
wait(NULL);    // 자식 프로세스가 끝나기를 기다림
shmctl(shmid, IPC_RMID, 0); // 공유메모리 삭제
semctl(semid, 0, IPC_RMID, 0); // 세마포어 삭제
return 0;
}

// 자식 프로세스 생성 및 busy() 수행
void fork_and_run() {
    pid_t pid = fork();
    if (pid < 0)
        errquit("fork fail");
    else if(pid == 0) {
        busy(); // child race
        exit(0);
    }
    return;
}

// 자식 프로세스가 수행하는 함수
void busy() {
    int i=0;
    for( i=0; i<100; i++ ) { // 100번 접근
        Semop((semid, &waitsem[0], 1));
        access_shm();
        Semop((semid, &notifysem[0], 1));
    }
    shmdt(shm_data); // 공유메모리 분리
}

// 공유메모리에 접근을 하는 부분
void access_shm() {
    int i;
    struct shmid_ds *buf;

```

```

pid_t pid;
struct timespec ts;
ts.tv_sec = 0;
ts.tv_nsec = 100000000; // 0.1 초
// 공유메모리에 자신의 pid를 기록
sprintf(shm_data, "%d", getpid());
// 공유메모리 접근 시간
for(i=0; i<1000; i++) ;
pid = atoi(shm_data);
// 공유메모리에 기록한 pid가 자신의 pid가 아니면 Error
if(pid != getpid())
    puts("Error : 다른 프로세스도 동시에 공유메모리 접근함\n");
else {
    putchar('.'); // ok
}
fflush(stdout);
//nanosleep(&ts, NULL); // sleep
sleep(0.5); // sleep
return ;
}

```

실행 화면은 다음과 같다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./shmcontrol 1234 5555
.....
.....
.....
.....
[root@localhost network]#

```

9. 연습문제 풀이

가. FIFO를 이용한 파일 서버와 클라이언트를 구현하시오. 클라이언트는 서버에 자신의 PID와 파일 경로명을 fifo.serv(FIFO)에 전달하면 서버는 fifo.serv로부터 경로명을 읽어서 fifo.1231(클라이언트의 PID가 1231일 때 클라이언트가 생성한 FIFO)에 파일의 내용을 전달해 주는 프로그램을 작성하시오.

(힌트: 서버는 먼저 fifo.serv라는 FIFO를 생성하고 클라이언트는 fifo.1231이라는 FIFO를 생성한다.)

예제 50 서버 프로그램 소스

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>

```

```

#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

#define MAX_BUFSZ 512
#define FIFONAME "fifo.serv"

// 클라이언트가 서버에 FIFO로 전달할 자료의 구조
typedef struct mesg
{
    pid_t pid;
    char path[MAX_BUFSZ];
} mesg_t;

void errquit( char* mesg ) { perror(mesg); exit(1); }

// 파일의 내용을 읽어 반환하는 함수
int readsrcfile( char* filename, char* filedata )
{
    FILE* fd;

    fd = fopen( filename, "r" );
    return ( fread( filedata, sizeof(char), MAX_BUFSZ-1, fd ) );
}

int main( int argc, char** argv )
{
    mesg_t pmsg;
    int fiford, fifowd;
    int nbytes;
    char clifile[MAX_BUFSZ];
    char filedata[MAX_BUFSZ];

    // "fifo.serv" 파일 생성
    if( mkfifo( FIFONAME, 0660 ) == -1 && errno != EEXIST )
        errquit( "mkfifo fail " );

    // 읽기전용으로 연다
    fiford = open( FIFONAME, O_RDONLY );
    if( fiford == -1 )
        errquit( "fifo open fail " );

    printf( "Server is waiting client...\n" );
    // 내용이 들어오면 읽는다
    nbytes = read( fiford, (char*)&pmsg, sizeof(pmsg) );
    if( nbytes < 0 )
        errquit( "read failed " );

    printf( "Server reads from fifo\n", nbytes );
    // FIFO를 통해 클라이언트에서 넘어온 내용 확인
    printf( "pid : %d, path : %s\n", pmsg.pid, pmsg.path );
}

```

```

// 지정된 파일 내용을 읽어온다
nbytes = readsrcfile( pmsg.path, filedata );

sprintf( clifile, "fifo.%d", pmsg.pid );
// 클라이언트측 FIFO를 쓰기전용으로 연다
fifowd = open( clifile, O_WRONLY );
if( fifowd == -1 )
    errquit( "fifo open fail " );

filedata[nbytes] = 0;
printf( "File Data - \n" );
printf( "%s\n", filedata );

// 파일의 내용을 보낸다
if( write( fifowd, filedata, nbytes ) < 0 )
    perror( "write fail " );
printf( "Server writes to fifo\n" );

return 0;
}

```

예제 51 클라이언트 프로그램 소스

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

#define MAX_BUFSZ 512

// 클라이언트가 서버에 FIFO로 전달할 자료의 구조
typedef struct mesg
{
    pid_t pid;
    char path[MAX_BUFSZ];
} mesg_t;

void errquit( char* mesg ) { perror(mesg); exit(1); }

int main( int argc, char** argv )
{
    char fifoname[MAX_BUFSZ];
    int fiford, fifowd;
    int nbytes;
    char filedata[MAX_BUFSZ];
    pid_t pid;
    mesg_t pmsg;

```



```

// pid를 얻어 파일명을 만든다
pid = getpid();
sprintf( fifoname, "fifo.%d", pid );

// FIFO를 생성
if( mkfifo( fifoname, 0660 ) == -1 && errno != EEXIST )
    errquit( "mkfifo fail " );

// 서버측 FIFO를 쓰기전용으로 연다
fifowd = open( "fifo.serv", O_WRONLY );
if( fifowd == -1 )
    errquit( "fifo open fail " );

// 서버에 전달한 내용을 채운다
pmsg.pid = pid;
strcpy( pmsg.path, argv[1] );

// 서버측 FIFO에 쓴다
if( write( fifowd, &pmsg, sizeof(pmsg) ) < 0 )
    perror( "write fail " );

// 클라이언트측 FIFO를 읽기전용으로 연다
fiford = open( fifoname, O_RDONLY );
// 내용이 생기면 읽어온다
nbytes = read( fiford, (char*)&filedata, MAX_BUFSZ-1 );
if( nbytes < 0 )
    errquit( "read failed " );
filedata[nbytes] = 0;

// 서버에서 넘어온 파일의 내용을 출력
printf( "File Data - \n" );
printf( "%s\n", filedata );

return 0;
}

```



```

s200100891@db:~/NetProg/prac8
[s200100891@db prac8]$ ./q1serv
Server is waiting client...
Server reads from fifo
pid : 5003, path : testfile.txt
File Data -
This file has test data!
1 - TaeHo Kim
2 - 200100891
3 - Computer Science Engineering
4 - Network Programming
5 - End of file.

Server writes to fifo
[s200100891@db prac8]$

```

```
s200100891@db:~/NetProg/prac8
[s200100891@db prac8]$ ./q1cli testfile.txt
File Data -
This file has test data!
1 - TaeHo Kim
2 - 200100891
3 - Computer Science Engineering
4 - Network Programming
5 - End of file.

[s200100891@db prac8]$
```

A screenshot of a Notepad++ application window. The title bar at the top reads "s200100891@db:~/NetProg/prac8". The main editing area contains the following text:

This file has test data!
1 - TaeHo Kim
2 - 200100891
3 - Computer Science Engineering
4 - Network Programming
5 - End of file

The text is followed by several blank lines. At the bottom status bar, it displays '"testfile.txt" 6L, 127C' on the left, '6,16' in the center, and 'All' on the right.

나. TCP 채팅 서버에서 클라이언트 로그에 대한 정보를 메시지큐에 남겨두면 이를 읽어 파일에 기록하는 로그 관리 프로세스를 fork()를 이용하여 작성하시오, 예를 들어 클라이언트가 접속할 때마다 아래와 같은 로그를 남길도록 하시오.

```
new client from (210.115.33.6) - Mon Nov 24 15:29:50 2003
```

예제 52 서버 프로그램 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
```

```

#include <sys/file.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <time.h>

#define MAXLINE 511
#define MAX_BUFSZ 512
#define MAX_SOCK 1024

// 메시지큐에서 사용할 자료의 구조
typedef struct _msg
{
    long msg_type;
    struct sockaddr_in addr;
    char msg_text[ MAX_BUFSZ ];
} msg_t;

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server \n";

int maxfdp1;
int num_chat = 0;
int clisock_list[MAX_SOCK];
int listen_sock;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

// 로그 관리 프로세스를 생성하고 지정된 파일에 로그 내용을 쓰는 함수
void fork_and_log( int msqid, char* filename )
{
    pid_t pid;
    FILE* fd;
    int size;
    msg_t pmsg;

    pid = fork();
    if( pid < 0 )
        errquit( "Fork fail " );
    else if( pid > 0 )
        return;

    size = sizeof(msg_t) - sizeof(long);
    pmsg.msg_type = 1;

```

```

while( 1 )
{
    // 메시지큐에서 로그내용이 있는 메시지를 가져온다
    if( msgrcv( msqid, (void*)&pmsg, size, 0, 0 ) < 0 )
        errquit( "msgrcv fail " );

    // 파일을 열고 로그내용을 추가한다
    fd = fopen( filename, "a" );
    fwrite( pmsg.msg_text, sizeof(char), strlen(pmsg.msg_text), fd );
    fclose( fd );
}
}

int main(int argc, char *argv[]) {
    struct sockaddr_in cliaddr;
    msg_t pmsg;
    key_t key;
    int msqid, size;
    time_t contime;
    char buf[MAXLINE+1];
    int i, j, nbyte, accp_sock, addrlen = sizeof(struct
        sockaddr_in);
    fd_set read_fds;
    int n;

    if(argc != 3) {
        printf("Usage :%s port msgq_key\n", argv[0]);
        exit(0);
    }

    key = atoi( argv[2] );
    // key값에 따라 메시지큐 생성
    msqid = msgget( key, IPC_CREAT | 0600 );
    if( msqid == -1 )
        errquit( "socket fail " );

    // 로그 관리 프로세스를 생성하고 지정된 파일에 로그 내용을 쓰는 함수 호출
    fork_and_log( msqid, "q2.log" );

    listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
    while(1) {
        FD_ZERO(&read_fds);
        FD_SET(listen_sock, &read_fds);
        for(i=0; i<num_chat; i++)
            FD_SET(clisock_list[i], &read_fds);
        maxfdp1 = getmax() + 1;
        puts("wait for client");
        if( select( maxfdp1, &read_fds, NULL, NULL, NULL ) < 0 )
            errquit("select fail");

        if(FD_ISSET(listen_sock, &read_fds)) {
            accp_sock=accept(listen_sock, (struct sockaddr

```

```

        *)&cliaddr, &addrlen);
    if(accp_sock == -1)
        errquit("accept fail");
    addClient(accp_sock,&cliaddr);
    send(accp_sock, START_STRING, strlen(START_STRING), 0);
    printf("Added %dth user.\n", num_chat);

    pmsg.msg_type = 1;
    size = sizeof(msg_t) - sizeof(long);
    // IP주소와 접속시간을 얻는다
    inet_ntop( AF_INET, &cliaddr.sin_addr, buf, sizeof(buf) );
    time( &contime );
    sprintf( pmsg.msg_text, "new client from (%s) - %s", buf, ctime(&contime) );
    // IP주소와 접속시간을 메시지큐로 보낸다
    if( msgsnd( msqid, &pmsg, size, 0 ) == -1 )
        errquit( "msgsnd fail " );
}

for(i = 0; i < num_chat; i++) {
    if(FD_ISSET(clisock_list[i], &read_fds)) {
        nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
        if(nbyte<= 0) {
            removeClient(i);
            continue;
        }
        buf[nbyte] = 0;

        if(strstr(buf, EXIT_STRING) != NULL) {
            removeClient(i);
            continue;
        }
        for (j = 0; j < num_chat; j++)
            send(clisock_list[j], buf, nbyte, 0);
        printf("%s\n", buf);
    }
}
return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET,&newcliaddr->sin_addr,buf,sizeof(buf));
    printf("new client: %s\n",buf);

    clisock_list[num_chat] = s;
    num_chat++;
}

void removeClient(int s) {
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
}

```

```

    num_chat--;
    printf("Leave a user. Current users = %d\n", num_chat);
}

int getmax() {
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }

    listen(sd, backlog);
    return sd;
}

```

```

s200100891@db:~/NetProg/prac8
[s200100891@db prac8]$ ./q2serv 6666 1
wait for client

[s200100891@db prac8]$ ./q2serv 6667 1
wait for client
new client: 127.0.0.1
Added 1th user.
wait for client
new client: 127.0.0.1
Added 2th user.
wait for client
Leave a user. Current users = 1
wait for client
Leave a user. Current users = 0
wait for client

[s200100891@db prac8]$

```



```

int maxfdp1, num_chat = 0;
int clisock_list[MAX SOCK];
char cliip_list[MAX SOCK][32];
int listen_sock;

// 공유메모리를 위한 전역변수 선언
char* shm_data;
int shmid;

// 세마포어를 위한 전역변수 선언
struct sembuf increase = { 0, +1, SEM_UNDO };
struct sembuf decrease = { 0, -1, SEM_UNDO };
unsigned short seminitval[1];
int semid;

union semun
{
    int val;
    struct semid_ds* buf;
    unsigned short int* array;
    struct seminfo* __buf;
} semarg;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

// 공유메모리 내용을 3초 주기로 읽어서 출력하는 프로세스 생성 함수
void fork_and_print()
{
    pid_t pid;

    pid = fork();
    if( pid < 0 )
        errquit( "fork fail " );
    else if( pid > 0 )
        return;

    while( 1 )
    {
        // 세마포어 decrease
        semop( semid, &decrease, 1 );
        // 공유메모리 내용 출력
        printf( "Shared Memory - \n%s\n", shm_data );
        // 세마포어 increase
        semop( semid, &increase, 1 );

        // 3초 대기
        sleep( 3 );
    }
}

```



```

    }
}

int main(int argc, char *argv[]) {
    struct sockaddr_in cliaddr;
    key_t shmkey;
    int msqid, size;
    time_t contime;
    char buf[MAXLINE+1];
    int i, j, nbyte, accp_sock, addrlen = sizeof(struct
        sockaddr_in);
    fd_set read_fds;
    int n;

    if(argc != 3) {
        printf("Usage :%s port shmkey\n", argv[0]);
        exit(0);
    }

    // 세마포어 생성
    semid = semget( 0x1234, 1, IPC_CREAT | 0600 );
    if( semid == -1 )
        semid = semget( 0x1234, 0, 0 );
    seminitval[0] = 1;
    semarg.array = seminitval;
    if( semctl( semid, 0, SETALL, semarg ) == -1 )
        errquit( "semctl " );

    shmkey = atoi( argv[2] );

    // key에 따라 공유메모리 생성
    shmid = shmget( shmkey, 128, IPC_CREAT | 0660 );
    if( shmid < 0 )
        errquit( "shmget fail " );
    shm_data = (char*)shmat( shmid, (void*)0, 0 );
    if( shm_data == (char*)-1 )
        errquit( "shmat fail " );

    // 보고용 프로세스 생성
    fork_and_print();

    listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
    while(1) {
        FD_ZERO(&read_fds);
        FD_SET(listen_sock, &read_fds);
        for(i=0; i<num_chat; i++)
            FD_SET(clisock_list[i], &read_fds);
        maxfdp1 = getmax() + 1;
        puts("wait for client");
        if( select( maxfdp1, &read_fds, NULL, NULL, NULL ) < 0 )
            errquit("select fail");

        if(FD_ISSET(listen_sock, &read_fds)) {

```

```

    accp_sock=accept(listen_sock, (struct sockaddr
        *)&cliaddr, &addrlen);
    if(accp_sock == -1)
        errquit("accept fail");
    addClient(accp_sock,&cliaddr);
    send(accp_sock, START_STRING, strlen(START_STRING), 0);
    printf("Added %dth user.\n", num_chat);
}

for(i = 0; i < num_chat; i++) {
    if(FD_ISSET(clisock_list[i], &read_fds)) {
        nbyte = recv(clisock_list[i], buf, MAXLINE, 0);
        if(nbyte<= 0) {
            removeClient(i);
            continue;
        }
        buf[nbyte] = 0;

        if(strstr(buf, EXIT_STRING) != NULL) {
            removeClient(i);
            continue;
        }
        for (j = 0; j < num_chat; j++)
            send(clisock_list[j], buf, nbyte, 0);
        printf("%s\n", buf);
    }
}

// 세미포어 decrease
semop( semid, &decrease, 1 );
// 접속자 수와 IP주소를 공유메모리에 써넣음
sprintf( shm_data, "Current clients : %d\n", num_chat );
for( j = 0; j < num_chat; j++ )
{
    sprintf( buf, "%dth client's IP : %s\n", j+1, cliip_list[j] );
    strcat( shm_data, buf );
}
// 세마포어 increase
semop( semid, &increase, 1 );
}

// 세마포어 삭제
semctl( semid, 0, IPC_RMID, 0 );
return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET,&newcliaddr->sin_addr,buf,sizeof(buf));
    printf("new client: %s\n",buf);

    clisock_list[num_chat] = s;
    strcpy( cliip_list[num_chat], buf );
}

```

```

    num_chat++;
}

void removeClient(int s) {
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
    num_chat--;
    printf("Leave a user. Current users = %d\n", num_chat);
}

int getmax() {
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);
    if (bind(sd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }

    listen(sd, backlog);
    return sd;
}

```

```

s200100891@db:~/NetProg/prac8
[s200100891@db prac8]$ ./q3serv 6666 3
Shared Memory -
Current clients : 0

wait for client
new client: 127.0.0.1
Added 1th user.
wait for client
Shared Memory -
Current clients : 1
1th client's IP : 127.0.0.1

new client: 127.0.0.1
Added 2th user.
wait for client
Shared Memory -
Current clients : 2
1th client's IP : 127.0.0.1
2th client's IP : 127.0.0.1

Shared Memory -
Current clients : 2
1th client's IP : 127.0.0.1
2th client's IP : 127.0.0.1

Leave a user. Current users = 1
wait for client
Leave a user. Current users = 0
wait for client
Shared Memory -
Current clients : 0

Shared Memory -
Current clients : 0

[s200100891@db prac8]$

```

Chapter 11. 스레드 프로그래밍

SECTION

01 스레드의 생성과 종료

1. 스레드의 정의

가. 프로세스처럼 독립적으로 수행되는 프로그램 코드

나. 경량 프로세스

1) 프로세스 내에서 독립적으로 수행되는 제어의 흐름

다. 한 프로세스 내에서 생성된 스레드들은 서로 전역 변수를 공유

1) 스레드간에 데이터를 편리하게 공유

2) 멀티프로세스 프로그램과 달리 IPC 기능을 사용하지 않아도 스레드간 데이터 공유

2. 스레드 생성과 종료

가. `pthread_create()`

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);
```

1) `pthread_t *thread`

가) 생성된 스레드 ID 리턴하며 이 ID를 통해 스레드에 접근

2) `pthread_attr_t *attr`

가) 스레드 속성을 지정하는 인자

나) 디폴트로 사용하려면 NULL 지정

3) `void *(*start_routine)(void *)`

가) 스레드 시작 함수(start routine)

4) `void *arg`

가) 스레드 start routine 함수의 인자

5) `pthread_create()` 호출시

가) start_routine 인자로 지정한 함수 실행

나) 스레드 시작 함수는 void* 타입의 인자를 받거나 임의의 타입을 가리키는 포인터를 리턴

다) 성공시 0, 실패시 에러코드를 정수 값으로 리턴

(1) 시스템 자원부족, 최대 생성 스레드 수(PTHREAD_THREADS_MAX)를 넘을 때 EAGAIN 리턴

나. `pthread_self()`

```
pthread_t pthread_self(void);
```

1) 스레드 자신의 ID를 얻음

다. `pthread_exit()`

```
void pthread_exit(void *retval);
```

- 1) 스레드가 스스로 종료
- 2) 스레드 종료 방법
 - 가) 스레드 시작 함수 내에서 return을 만남
 - 나) 임의의 위치에서 pthread_exit()호출
- 3) main()함수
 - 가) 일종의 스레드로 main 스레드 또는 initial 스레드라 함
 - 나) main()에서의 return은 다른 스레드에서의 return과 달리 프로세스를 종료시킴
 - 다) main()에서 여러 스레드를 생성한 후 return하면 모두 종료됨
 - 라) main()에서 exit()를 호출하거나 return하는 대신 pthread_exit()를 호출하는 경우
 - (1) main 스레드만 종료하며 생성된 스레드들은 종료되지 않음
 - (2) 생성된 모든 스레드가 종료될 때까지 프로세스는 종료되지 않고 기다림

라. pthread_join()

```
int pthread_join(pthread_t thrd, void **thread_return);
```

- 1) 자신이 생성한 자식 스레드가 종료할 때까지 기다려야 하는 경우에 사용
- 2) thrd 인자
 - 가) 종료를 기다리는 스레드의 ID
 - 나) 지정한 스레드가 종료할 때까지 또는 취소될 때까지 대기
- 3) thread_return 인자
 - 가) 자식 스레드의 종료 상태가 저장
 - 나) NULL로 지정할 경우 스레드 종료 상태 값을 받지 않음
 - 다) 자식 스레드의 종료 값은 자식 스레드에서 스레드 종료 시에 pthread_exit()의 인자 또는 return 값으로 남길수 있음
 - 라) 자식 스레드가 다른 스레드에 의해 취소된 경우
 - (1) 스레드 종료 값은 PTHREAD_CANCELED
- 4) 자식 스레드가 종료되기 전에 부모 스레드가 먼저 종료 되지 않도록 할 경우에 사용
 - 가) 자식 스레드의 종료 시점을 정확히 파악하여 다른 작업을 할 경우에 사용
 - 나) 자식 스레드의 종료 상태 값을 얻기 위해 사용
- 5) main 스레드에서 주로 사용하게 됨
- 6) main에서 생성된 스레드들이 종료하기 전에 main이 종료되면 생성된 스레드가 모두 종료됨

3. 스레드의 사용 예

예제 54 스레드의 생성과 스레드 ID와 pid를 확인

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```

#include <unistd.h>
#include <pthread.h>

void *thrfunc(void *arg) ; // 스레드 시작 함수

char who[10];
int main(int argc, char **argv) {
    int status;
    pthread_t tid;
    pid_t pid;

    // 자식 프로세스 생성
    pid = fork();
    if(pid==0)
        sprintf(who,"child");
    else
        sprintf(who,"parent");

    // 프로세스 ID와 초기 스레드의 ID 확인
    printf("(%s's main) Process ID = %d\n", who, getpid());
    printf("(%s's main) Init thread ID = %d\n", who, pthread_self());

    // 에러 발생시 에러코드를 리턴
    if( (status=pthread_create(&tid, NULL, &thrfunc, NULL))!=0) {
        printf("thread create error: %s\n", strerror(status));
        exit(0);
    }

    // 인자로 지정한 스레드 id가 종료하기를 기다림
    pthread_join(tid, NULL);
    printf("\n(%s)[%d] 스레드가 종료했습니다\n", who,tid);
    return 0;
}

void *thrfunc(void *arg) {
    printf("(%s' thread routine) Process ID = %d\n",who,getpid());
    printf("(%s' thread routine) Thread ID = %d\n",who,pthread_self());
}

```

실행 화면은 다음과 같다.


```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./thread_basic
(child's main) Process ID = 29171
(child's main) Init thread ID = -1208486208
(parent's main) Process ID = 29170
(parent's main) Init thread ID = -1208486208
(child' thread routine) Process ID = 29171
(child' thread routine) Thread ID = -1208489040

(child)[-1208489040] 스레드가 종료했습니다
(parent' thread routine) Process ID = 29170
(parent' thread routine) Thread ID = -1208489040

(parent)[-1208489040] 스레드가 종료했습니다
[root@localhost network]#

```

가. 동작설명

- 1) pthread_create()와 pthread_self()함수를 이용해 스레드를 생성하고 스레드 ID를 얻음
- 2) 컴파일시 "-lpthread"로 pthread 라이브러리를 링크

4. 스레드의 상태

가. 준비, 실행, 블록, 종료 중 하나를 가지게 됨

나. 준비

- 1) 처음 생성 시 가지게 되는 상태
- 2) 스레드가 실행될 수 있는 상태

다. 실행

- 1) 운영체제의 스케줄링에 의해 이동
- 2) CPU의 서비스를 받고 있는 상태

라. 블록

- 1) 즉시 처리할 수 없는 작업을 만날 경우
- 2) sleep(), read(), 세마포어 연산 등으로 기다리는 상태

마. 종료

- 1) 스레드 시작 함수에서 return하거나 pthread_exit() 호출, 다른 스레드에 의해 취소될 경우
- 2) 스레드가 종료 또는 취소된 상태

5. 스레드의 분류

가. joinable 스레드와 분리된 스레드로 나뉨

나. 스레드에서 pthread_detach()를 호출 시

- 1) 데몬 프로세스처럼 부모 스레드와 분리된 스레드로서 실행됨
- 2) 분리된 스레드는 pthread_join()을 호출할 수 없고 종료시 종료 상태에 남아 있지 않음
 - 가) 메모리가 모두 반환됨

다. 모든 스레드는 디폴트로 joinable 스레드

- 1) 부모 스레드는 pthread_join()을 호출하여 이 스레드의 종료를 기다림

SECTION

02 스레드 동기화

1. 동기화 문제

가. 한 스레드가 공유 데이터를 액세스하는 도중에 다른 스레드가 이 공유 데이터 액세스 시

- 1) 데이터 값을 두 스레드가 정확히 예측할 수 없음

나. 해결 방법

- 1) 스레드들이 공유데이터에 접근할 때 서로 배타적으로 접근
- 2) 한 번에 한 스레드만 공유데이터에 접근
- 3) 뮤텝스(mutex = mutual + exclusion) 사용

2. 동기화 문제 예

예제 55 스레드 동기화문제(경쟁 조건)의 발생

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

#define MAX_THR 2
void *thrfunc(void *arg);      // 스레드 시작 함수
void prn_data(long who);      // 스레드 ID 출력
int who_run = -1;             // prn_data() 수행중인 스레드 ID
                                // 초기값은 -1

int main(int argc, char **argv) {
    pthread_t tid[MAX_THR];
    int i, status;
    for(i=0; i<MAX_THR; i++) {
        if( (status=pthread_create(&tid[i], NULL, &thrfunc, NULL))!=0) {
            printf("thread create error: %s\n",strerror(status));
            exit(0);
        }
    }
    pthread_join(tid[0],NULL);
    return 0;
}

void *thrfunc(void *arg) {
    while(1) {
        prn_data(pthread_self());
    }
}
```

```

    return NULL;
}

void prn_data(long me) {
    who_run = me;
    if(who_run != pthread_self()) {
        printf("Error : %d스레드 실행중 who_run=%d\n", me, who_run);
    }
    who_run = -1;    // 초기값으로 환원
}

```

실행 화면은 다음과 같다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./race
Error : -1218950224스레드 실행중 who_run=-1
Error : -1208460368스레드 실행중 who_run=-1
Error : -1218950224스레드 실행중 who_run=-1
Error : -1218950224스레드 실행중 who_run=-1
Error : -1218950224스레드 실행중 who_run=-1
Error : -1208460368스레드 실행중 who_run=-1
[root@localhost network]#

```

가. 동작설명

- 1) 스레드들이 스레드 ID를 저장하는 run 변수를 공유
- 2) 한 스레드가 prn_data()를 호출하는 동안 다른 스레드가 prn_data() 호출시 run 값 변경
- 3) 중간에 run값이 변경될 경우 에러 메시지를 출력함

3. 플래그 사용 예

```

void *thrfunc(void *arg)
{
    while(1)
    {
        if(run == -1)
        {
            prn_data(pthread_self());
        }
    }
    return NULL;
}

```

가. prn_data() 함수에서 동기화 문제 발생

- 1) 위의 thread_syn.c 프로그램에서 prn_data() 함수가 동시에 호출되므로 동기화 문제 발생

나. 스레드 시작 함수 thrfunc()를 수정

- 1) 플래그를 하나 정의하여 사용함으로써 한 스레드만 prn_data()를 실행하도록 수정
- 2) run == -1 인 경우, 즉 아무 스레드도 prn_data()를 호출하지 않을 때만 prn_data() 호출
- 3) 플래그를 조사한 직후에 스레드 스케줄링이 일어날 경우
 - 가) 동기화 문제 발생

4. 뮤텝스

스레드의 동기화 문제를 해결하는 커널이 제공하는 일종의 플래그

가. 뮤텝스 사용 방법

```
pthread_mutex_t mutex;
pthread_mutex_lock(mutex);
pthread_mutex_unlock(mutex);
```

- 1) 뮤텝스 잠금
 - 가) 스레드가 공유데이터를 사용하기 전, 즉 크리티컬 영역에 들어가기 전에 잠금
 - 나) 뮤텝스 잠금을 한 스레드만 공유데이터에 접근
- 2) 뮤텝스 해제
 - 가) 스레드가 크리티컬 영역을 나을 때 뮤텝스 잠금을 해제
 - 나) 다른 스레드는 뮤텝스 잠금이 해제될 때까지 대기
- 3) pthread_mutex_t mutex;
 - 가) 뮤텝스 선언
- 4) pthread_mutex_lock(mutex);
 - 가) 뮤텝스 잠금
- 5) pthread_mutex_unlock(mutex);
 - 가) 뮤텝스 해제

나. 뮤텝스 사용

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
void *thrfunc(void *arg)
{
    while(1)
    {
        pthread_mutex_lock(&lock);
        prn_data(pthread_self());
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}
```

- 1) pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
 - 가) 뮤텝스 선언 및 초기화

- 2) pthread_mutex_lock(&lock);
 - 가) 뮤텍스 잠금
- 3) prn_data(pthread_self());
 - 가) 공유데이터 액세스
- 4) pthread_mutex_unlock(&lock);
 - 가) 뮤텍스 해제

다. 뮤텍스 선언 및 초기화

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t *attr);
```

- 1) 뮤텍스 선언
 - 가) 뮤텍스의 변수 타입 이름은 pthread_mutex_t
 - 나) 여러 스레드들이 사용하므로 전역 변수로 선언
- 2) 뮤텍스 초기화
 - 가) pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
 - (1) 매크로를 이용한 초기화 방법으로 기본 속성만 가짐
 - 나) pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t *attr);
 - (1) 함수를 이용한 초기화 방법으로 attr 인자를 이용해 속성을 지정함
 - (2) attr 인자가 NULL 일 경우 기본 속성이 적용됨
- 3) pthread_mutexattr_settype()
 - 가) attr 인자의 속성을 지정
 - 나) 인자로 &attr 과 type가 있음
 - 다) type은 뮤텍스의 타입을 지정하기 위한 상수
 - (1) PTHREAD_MUTEX_TIMED_NP : timed 타입
 - (2) PTHREAD_MUTEX_RECURSIVE_NP : recursive 타입
 - (3) PTHREAD_MUTEX_ERRORCHECK_NP : error checking 타입

라. 기본 뮤텍스

- 1) 특별한 타입을 지정하지 않은 뮤텍스
- 2) 단 한 번의 잠금만 허용
 - 가) 1번 스레드가 뮤텍스 잠금을 한 상황에서 2번 스레드가 잠금 시도시
 - (1) 2번 스레드는 1번 스레드의 잠금이 해제될 때까지 대기
 - 나) 1번 스레드에 의해 잠긴 뮤텍스에 대해 1번 스레드가 또 잠금 시도시
 - (1) 데드락이 발생하여 1번 스레드는 영원히 블록 상태가 됨
 - (2) pthread_mutex_lock() 대신 pthread_mutex_trylock()를 사용하여 블록 현상 피함
 - (3) 뮤텍스를 얻지 못하는 경우 스레드는 블록되지 않고 리턴되며 EBUSY 에러 발생
- 3) 다른 스레드에서 해제 가능
 - 가) 1번 스레드에 의해 잠금된 뮤텍스를 2번 스레드에서 해제 가능

마. Timed 타입 뮤텍스

- 1) 지정 시간 동안만 블록
- 2) pthread_mutex_timedlock()
 - 가) pthread_mutex_lock()는 뮤텝스를 얻을 때까지 무한 대기
 - 나) timed 타입 뮤텝스로 지정된 시간 동안만 블록
 - 다) 리눅스에서 기본 뮤텝스가 timed 타입 뮤텝스로 동작함

바. Recursive 타입 뮤텝스

- 1) 한 스레드가 한 번 이상 잠금
- 2) pthread_mutex_lock()을 호출한 횟수만큼 pthread_mutex_unlock()를 호출
 - 가) 한 스레드가 한 번 이상의 잠금을 하므로 호출 횟수만큼 해제해야 잠금이 해제됨
 - 나) 한 스레드가 잠금을 한 상태에서 다른 스레드가 잠금 해제를 시도 시 에러 발생

사. Error Check 타입 뮤텝스

- 1) 두 번 잠금 시도 시 에러 발생 후 프로세스 종료
- 2) 기본 뮤텝스에서 한 스레드가 뮤텝스 잠금을 두 번 시도 시
 - 가) 무한 블록 상태
- 3) error check 타입 뮤텝스에서 두 번 잠금 시도 시
 - 가) 에러 발생 후 프로세스 종료
- 4) error check 타입 뮤텝스에서 잠겨 있지 않은 뮤텝스에 잠금 해제 시도 시
 - 가) 에러 발생 후 프로세스 종료
 - 나) 기본 뮤텝스에서는 잠겨있지 않은 뮤텝스에 잠금 해제 시도 시 에러 없음
- 5) 자체적으로 에러 검사를 하므로 기본 뮤텝스보다 처리 속도가 느림

아. 뮤텝스 삭제

- 1) 뮤텝스를 더 이상 사용하지 않을 때 사용

```
pthread_mutex_unlock(&mutex);
pthread_mutex_destroy(&mutex);
```

- 2) 뮤텝스가 어떤 스레드에 의해 잠금된 상태에서 제거 시도 시
 - 가) EBUSY 에러 발생
 - 나) 뮤텝스의 현재 상태를 알아보는 방법이 없음
- 3) 안전한 뮤텝스 제거를 위해 무조건 뮤텝스 해제 후 제거

자. 데드락

- 1) 두 개의 스레드가 뮤텝스 해제를 서로 기다리는 현상
- 2) 두 개 이상의 스레드가 두 개 이상의 뮤텝스를 사용하는 경우에 발생
 - 가) 각 스레드가 뮤텝스를 하나씩 잠그고 있는 상태에서 상대방의 뮤텝스 해제를 서로 기다림
- 3) 데드락 발생시 프로그램은 영원히 블록됨
- 4) 서로 연관이 있는 작업에서는 다수의 뮤텝스 사용을 피함
 - 가) 작업 처리의 순서를 정하여 해결
- 5) 뮤텝스를 많이 사용하면 프로그램 성능이 저하됨

- 가) 뮤텍스를 얻은 상태에서 처리하는 작업량을 최소화하여 해결
- 나) 뮤텍스가 잠긴 동안 다른 스레드가 블록될 확률이 높아서 크리티컬 영역이 길수록 전체 성능이 저하됨

5. 뮤텍스 사용 예

예제 56 1초 간격으로 두개의 스레드가 돌아가면서 count 출력

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

// 스레드 시작 함수
void *thrfunc(void *arg) ;
int counting = 0;           // 공유데이터
pthread_mutex_t count_lock; // 뮤텍스 초기화
pthread_mutexattr_t mutex_attr; // 뮤텍스 속성 초기화

int main(int argc, char **argv) {
    pthread_t tid[2];
    int i, status;
    pthread_mutexattr_init(&mutex_attr); // 뮤텍스
    pthread_mutex_init(&count_lock, &mutex_attr); // 뮤텍스 속성변수
    for(i=0; i <2; i++) {
        // 에러 발생시 non-zero 값 리턴
        if((status=pthread_create(&tid[i], NULL, &thrfunc, NULL))!=0) {
            printf("pthread_create fail: %s",strerror(status));
            exit(0);
        }
    }
    for(i=0; i<2; i++) pthread_join(tid[i], NULL);
    return 0;
}

// 스레드 시작 함수
void *thrfunc(void *arg) {
    while(1) {
        pthread_mutex_lock( &count_lock );
        printf("\n[%ld 스레드] 뮤텍스 잠금\n", pthread_self());
        printf("[%ld 스레드] counting = %d\n", pthread_self(), counting);
        counting++;
        sleep(1);
        printf("[%ld 스레드] 뮤텍스 해제\n", pthread_self());
        pthread_mutex_unlock(&count_lock);
    }
    return NULL;
}
```

실행 화면은 다음과 같다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./mutextest

[-1209074768 스레드] 뮉텍스 잠금
[-1209074768 스레드] counting = 0
[-1209074768 스레드] 뮉텍스 해제

[-1209074768 스레드] 뮉텍스 잠금
[-1209074768 스레드] counting = 1
[-1209074768 스레드] 뮉텍스 해제

[-1209074768 스레드] 뮉텍스 잠금
[-1209074768 스레드] counting = 2
[-1209074768 스레드] 뮉텍스 해제

```

가. 동작설명

- 1) 한 스레드가 count 변수에 접근하는 동안 다른 스레드의 접근을 못하게 함
- 2) main()에서 두 개의 스레드를 만들고 스레드 시작 함수에서 뮉텍스를 이용해 1초 간격으로 count 값을 출력

SECTION

03 스레드간 통신

1. 조건변수 사용 방법

가. 조건변수의 정의

- 1) 스레드간에 공유데이터의 상태에 대한 정보를 주고받는 변수
- 2) 조건변수의 사용 예
 - 가) 생산자 스레드는 큐에 메시지를 쓰고 소비자 스레드는 큐의 메시지를 꺼내서 출력
 - (1) 큐는 스레드간 공유데이터
 - 나) 동기화 문제를 피하기 위해 뮤텁스를 사용
 - 다) 운영체제의 스케줄러가 생산자와 소비자의 두 스레드가 교대로 실행함을 보장하지 않음
 - (1) 어느 한 스레드가 연속으로 두 번 실행될 수 있음
 - 라) 조건변수를 이용해 큐에 메시지가 쓰였음을 소비자 스레드에 알려줌

나. 조건알림

- 1) 조건변수를 통해 어떤 조건이 만족됨을 다른 스레드에 알려주는 기능

2. 조건변수의 동작

가. 뮤텁스를 정의한 후 뮤텁스와 연계하여 조건변수를 정의

나. 반드시 뮤텁스와 함께 사용

```
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
```

다. pthread_cond_wait()

- 1) 소비자 스레드에서 조건변수를 기다리는 함수
- 2) 조건변수 cond 인자
 - 가) 조건알림이 발생할 때까지 대기
- 3) 뮤텁스 타입의 mutex 인자
 - 가) 조건변수를 제어하는 인자

라. pthread_cond_wait() 호출시 내부 동작

- 1) 뮤텁스해제 → 블록 상태 → 조건알림을 받음 → 깨어나면서 뮤텁스를 얻음

마. pthread_cond_timedwait()

- 1) 지정한 시간동안만 블록 상태에서 조건알림을 기다림
 - 가) 조건알림을 무한히 기다리는 것을 방지

바. pthread_cond_signal()

```
int pthread_cond_signal(pthread_cond_t *cond);
```

- 1) 조건알림을 기다리는 스레드에게 조건알림을 보냄
- 2) 조건알림을 기다리는 스레드를 블록 상태에서 깨어나게 함

사. pthread_cond_broadcast()

- 1) 여러 스레드를 동시에 깨움
- 2) 생산자 스레드보다 소비자 스레드가 많을 경우
 - 가) pthread_cond_signal()은 한 번에 하나의 스레드만 깨움
 - 나) 조건알림을 기다리는 스레드가 둘 이상일 때 모든 wait 함수를 깨우기 위해 사용
- 3) 여러 스레드를 동시에 깨우지만 공유데이터 접근을 위해서는 뮤텝스를 얻어야 함

3. 조건변수의 생성과 삭제

가. 조건변수 초기화

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

pthread_cond_t cond;
pthread_cond_init(&cond, NULL);
```

- 1) pthread_cond_init()를 이용해 사용전 초기화
- 2) pthread_cond_t cond = PTHREAD_COND_INITIALIZER
 - 가) 매크로를 이용한 초기화 방법
- 3) 함수를 이용한 초기화 방법으로 3번줄에서 조건변수 선언
- 4) &cond는 초기화 시킬 조건변수
- 5) 두 번째 인자는 조건변수의 속성으로 NULL 일 경우 기본 속성으로 초기화

나. 조건변수 삭제

```
pthread_cond_destroy(&cond);
```

- 1) pthread_cond_destroy()를 이용해 삭제

4. 조건변수 사용 예

예제 57 조건변수를 통해서 두 스레드간에 통신하는 예

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>
```

```

// 뮤텍스, 조건변수 및 공유데이터
typedef struct _complex {
    pthread_mutex_t mutex;    // 뮤텍스
    pthread_cond_t cond ;    // 조건변수
    int value;                // 공유데이터
} thread_control_t;

// 뮤텍스, 조건변수, 공유데이터의 초기화
thread_control_t data = {
    PTHREAD_MUTEX_INITIALIZER, PTHREAD_COND_INITIALIZER, 0};

// 에러 출력 및 스레드 종료
void thr_errquit(char *msg,int errcode) {
    printf("%s: %s\n",msg,strerror(errcode));
    pthread_exit(NULL);
}

void *wait_thread(void *arg);    // 스레드 시작 함수
int sleep_time ;                // 자식 스레드가 처음 잠자는 시간

int main(int argc, char **argv) {
    int status;
    pthread_t wait_thr;
    struct timespec timeout;
    if (argc != 2 ) {
        printf("사용법 : cond 5 (자식 스레드가 sleep할 시간)\n");
        exit(0);
    }
    sleep_time = atoi(argv[1]);
    if( (status=pthread_create(&wait_thr, NULL,
                             wait_thread, NULL))!=0) {
        printf("pthread_create fail : %s\n",strerror(status));
        exit(1);
    }

    timeout.tv_sec = time(NULL)+3;    // 현재의 시간 + 3초
    timeout.tv_nsec = 0;

    // 뮤텍스 잠금
    if( (status=pthread_mutex_lock(&data.mutex)) != 0) {
        printf("pthread_mutex_lock fail : %s\n",strerror(status));
        exit(1);
    }

    if (data.value == 0) {
        status = pthread_cond_timedwait(&data.cond,
                                         &data.mutex, &timeout);
        if(status == ETIMEDOUT) {
            printf("Condition wait time out\n");

```

```

    }
    else {
        printf("Wait on Condition.....\n");
    }
}

// data.value 값이 자식 스레드에서 1로 바뀜
if(data.value == 1)
    printf("Condition was signaled.\n");
if( (status=pthread_mutex_unlock(&data.mutex)) != 0) {
    printf("pthread_mutex_unlock fail : %s\n",strerror(status));
    exit(0);
}

// 자식 스레드가 종료하기를 기다림
if( (status=pthread_join(wait_thr, NULL)) != 0) {
    printf("pthread_join: %s\n",status);
    exit(0);
}
return 0;
}

// 스레드 시작 함수
void *wait_thread(void *arg) {
    int status;
    sleep(sleep_time);
    if((status=pthread_mutex_lock(&data.mutex)) != 0)
        thr_errquit("pthread_mutex_lock failure",status);
    data.value = 1;
    if((status=pthread_cond_signal(&data.cond)) != 0)
        thr_errquit("pthread_cond_signalure",status);
    if((status=pthread_mutex_unlock(&data.mutex)) != 0)
        thr_errquit("pthread_mutex_unlock failure",status);
    return NULL;
}

```

실행 화면은 다음과 같다.

```

root@localhost:~/network
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
[root@localhost network]# ./cond 1
Wait on Condition.....
Condition was signaled.
[root@localhost network]# ./cond 5
Condition wait time out
[root@localhost network]#

```

가. 동작설명

- 1) 공유데이터인 data.value의 초기 값은 0이나 부모 스레드는 지정된 간동안 이 값을 자식 스레드가 1로 설정해주기를 기다림

- 2) 자식 스레드는 시작 직후 인자로 넣어준 시간동안 sleep함
- 3) 부모 스레드는 pthread_cond_timedwait()를 호출하여 5초동안만 조건 알림을 기다림
- 4) 자식 스레드는 sleep()에서 깨어난 후 공유데이터 값을 1로 설정
- 5) 조건변수 data.cond를 통해 대기 상태인 부모 스레드에게 조건알림을 함
- 6) 자식 스레드가 5초이상 sleep하면 부모 스레드는 ETIMEDOUT 에러를 발생시킴



SECTION

04 멀티스레드 에코 서버 프로그램

1. 멀티스레드 에코 서버 프로그램

예제 58 멀티스레드 에코 서버

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_BUFSZ  1024

typedef struct _msg {
    long msg_type;
    struct sockaddr_in addr;
    char msg_text[ MAX_BUFSZ ];
} msg_t ;
struct sockaddr_in servaddr;
int sock;      // 서버 소켓
int msqid;     // 메시지큐 ID

void *echo_recv(void *arg);  // 에코 수신 스레드
void *echo_resp(void *arg);  // 에코 송신 스레드
void errquit(char *msg)
{ perror(msg); exit(-1);}

void thr_errquit(char *msg,int errcode)
{ printf("%s:%s\n",msg,strerror(errcode)); pthread_exit(NULL);}

int main(int argc, char *argv[]) {
    pthread_t tid[6];
    struct sockaddr_in cliaddr;
    int port, status, i, len = sizeof(struct sockaddr);
    key_t msqkey;

    if(argc!=3) {

```

```

    printf("Usage: %s msgkey port\n", argv[0]);
    exit(1);
}
msgkey = atoi(argv[1]);
port = atoi(argv[2]);

// 메시지큐 생성
if( (msqid = msgget(msgkey, IPC_CREAT | 0660)) < 0)
    errquit("msgget fail");

// 소켓 생성
if( (sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    errquit("socket fail ");

bzero(&servaddr, len);
servaddr.sin_port = htons(port);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_family = AF_INET;
bind(sock, (struct sockaddr*)&servaddr, len);

// 스레드 생성
for(i=0; i<5; i++)
    if((status=pthread_create(&tid[i], NULL, echo_recv, NULL))!=0)
        thr_errquit("pthread_create", status);

if((status=pthread_create(&tid[5], NULL, echo_resp, NULL))!=0)
    thr_errquit("pthread_create", status);
for(i=0; i<6; i++)
    pthread_join(tid[i], NULL);

// 메시지큐 삭제
msgctl(msqid, IPC_RMID, 0);
return 0;
}

// 에코 수신 스레드
void *echo_recv(void *arg) {
    int nbytes, status, len = sizeof(struct sockaddr);
    msg_t pmsg;
    int size;
    size = sizeof(pmsg) - sizeof(long);
    pmsg.msg_type = pthread_self();
    while(1) {
        // 에코 메시지 수신
        nbytes = recvfrom(sock, pmsg.msg_text, MAX_BUFSZ, 0,
                        (struct sockaddr *)&pmsg.addr, &len);
        if(nbytes < 0)
            thr_errquit("recvfrom fail", errno);
        pmsg.msg_text[nbytes] = 0;
        printf("recv thread = %ld\n", pthread_self());
    }
}

```

```

        // 메시지큐로 전송
        if(msgsnd(msqid, &pmsg, size, 0)==-1)
            thr_errquit("msgsnd fail",errno);
    }
}

// 에코 송신 스레드
void *echo_resp(void *arg) {
    msg_t pmsg;
    int nbytes, len = sizeof(struct sockaddr);
    int size;
    size = sizeof(pmsg) - sizeof(long);
    pmsg.msg_type = 0;
    while(1) {
        // 메시지큐에서 읽음
        if(msgrcv( msqid,(void *)&pmsg, size, 0, 0) < 0) {
            perror("msgrcv fail"); exit(0);
        }
        // 에코 응답
        nbytes = sendto(sock, pmsg.msg_text, strlen(pmsg.msg_text), 0,
                        (struct sockaddr*)&pmsg.addr, len);

        if(nbytes < 0)
            thr_errquit("send to fail",errno);

        printf("response thread = %ld\n\n", pthread_self());
        pmsg.msg_text[0] = 0;
    }
}

```

2. 연습문제 풀이

가. 멀티스레드를 이용하여 채팅 서버를 구현하시오. select() 함수를 사용하지 말고 새로운 클라이언트가 접속할 때마다 새로운 스레드를 생성하여 한 스레드가 하나의 클라이언트와의 통신을 담당하도록 하시오.

(힌트: accept() 함수가 리턴하면 pthread_create() 함수를 호출한다. 클라이언트로부터 메시지를 받으면 다른 클라이언트에게 메시지를 전달해야 하므로 모든 스레드가 클라이언트 리스트를 공유해야 한다. 따라서 클라이언트 리스트는 전역변수로 정한다. 전역변수를 려어 스레드가 동시에 접근하게 되므로 뮤텁스를 사용하여 동기화를 처리해야 한다.)

예제 59 멀티스레드 채팅 서버

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <netinet/in.h>

```



```

#include <arpa/inet.h>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>

#define MAXLINE 511
#define MAX_SOCKET 1024

char *EXIT_STRING = "exit";
char *START_STRING = "Connected to chat_server \n";

int maxfdp1;
int num_chat = 0;
int clisock_list[MAX_SOCKET];
int listen_sock;

// 스레드 사용을 위한 전역변수 선언
pthread_t thread;
// 뮤텍스 사용을 위한 전역변수 선언
pthread_mutex_t mutex;

void addClient(int s, struct sockaddr_in *newcliaddr);
int getmax();
void removeClient(int s);
int tcp_listen(int host, int port, int backlog);
void errquit(char *mesg) { perror(mesg); exit(1); }

// 각 클라이언트 접속마다 스레드를 생성해 실행한 함수
void* thread_func( void* arg );
// 모든 클라이언트에 메시지를 전달하는 함수
void send_message( char* msg, int len );

int main(int argc, char *argv[]) {
    struct sockaddr_in cliaddr;
    char buf[MAXLINE+1];
    int i, j, nbyte, accp_sock, addrlen = sizeof(struct
        sockaddr_in);
    fd_set read_fds;
    int n;

    if(argc != 2) {
        printf("Usage :%s port\n", argv[0]);
        exit(0);
    }

    // 뮤텍스 초기화
    if( pthread_mutex_init( &mutex, NULL ) )
        errquit( "pthread_mutex_init fail" );

    listen_sock = tcp_listen(INADDR_ANY, atoi(argv[1]), 5);
    while(1) {
        puts("wait for client");

```

```

    accp_sock = accept( listen_sock, (struct sockaddr*)&cliaddr, &addrlen );
    if( accp_sock == -1 )
        errquit( "accept fail" );

    // 클라이언트 등록
    addClient( accp_sock, &cliaddr );

    send( accp_sock, START_STRING, strlen(START_STRING), 0 );
    printf( "Added %dth user.\n", num_chat );

    // 스레드 생성
    pthread_create( &thread, NULL, thread_func, (void*)accp_sock );
}

return 0;
}

void addClient(int s, struct sockaddr_in *newcliaddr) {
    char buf[20];
    inet_ntop(AF_INET,&newcliaddr->sin_addr,buf,sizeof(buf));
    printf("new client: %s\n",buf);

    // 뮤텁스를 얻은 후 클라이언트를 추가
    pthread_mutex_lock( &mutex );
    clisock_list[num_chat] = s;
    num_chat++;
    pthread_mutex_unlock( &mutex );
}

void* thread_func( void* arg )
{
    int accp_sock = (int)arg;
    int len = 0;
    char buf[ MAXLINE+1 ];
    int i;

    // 클라이언트에서 입력을 받는다
    while( ( len = read( accp_sock, buf, sizeof(buf) ) ) != 0 )
    {
        if( strstr( buf, EXIT_STRING ) != NULL )
            break;
        else
        {
            buf[len] = 0;
            // 입력된 메시지를 모든 클라이언트로 전송
            send_message( buf, len );
            printf( "%s\n", buf );
        }
    }

    // 연결을 끊은 클라이언트의 번호를 찾아서
    for( i = 0; i < num_chat; i++ )

```

```

        if( accp_sock == clisock_list[ i ] )
        {
            // 삭제한다
            removeClient( i );
            break;
        }
    }

void removeClient(int s) {
    // 뮤텍스를 얻은 후 클라이언트를 삭제한다
    pthread_mutex_lock( &mutex );
    close(clisock_list[s]);
    if(s != num_chat-1)
        clisock_list[s] = clisock_list[num_chat-1];
    num_chat--;
    pthread_mutex_unlock( &mutex );
    printf("Leave a user. Current users = %d\n", num_chat);
}

void send_message( char* msg, int len )
{
    int i;
    // 뮤텍스를 얻은 후 모든 클라이언트에게 메시지를 보낸다
    pthread_mutex_lock( &mutex );
    for( i = 0; i < num_chat; i++ )
        write( clisock_list[i], msg, len );
    pthread_mutex_unlock( &mutex );
}

int getmax() {
    int max = listen_sock;
    int i;
    for (i=0; i < num_chat; i++)
        if (clisock_list[i] > max )
            max = clisock_list[i];
    return max;
}

int tcp_listen(int host, int port, int backlog) {
    int sd;
    struct sockaddr_in servaddr;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd == -1) {
        perror("socket fail");
        exit(1);
    }

    bzero((char *)&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(host);
    servaddr.sin_port = htons(port);

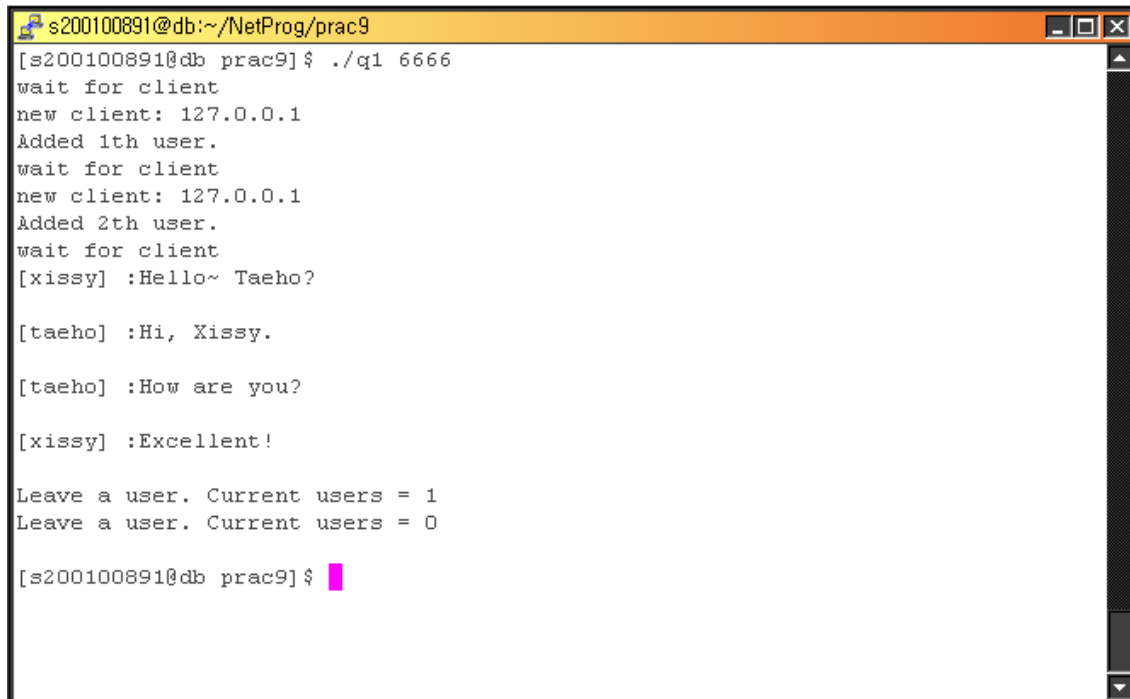
```

```

    if (bind(sd , (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind fail"); exit(1);
    }

    listen(sd, backlog);
    return sd;
}

```



```

s200100891@db:~/NetProg/prac9
[s200100891@db prac9]$ ./q1 6666
wait for client
new client: 127.0.0.1
Added 1th user.
wait for client
new client: 127.0.0.1
Added 2th user.
wait for client
[xissy] :Hello~ Taeho?

[taeho] :Hi, Xissy.

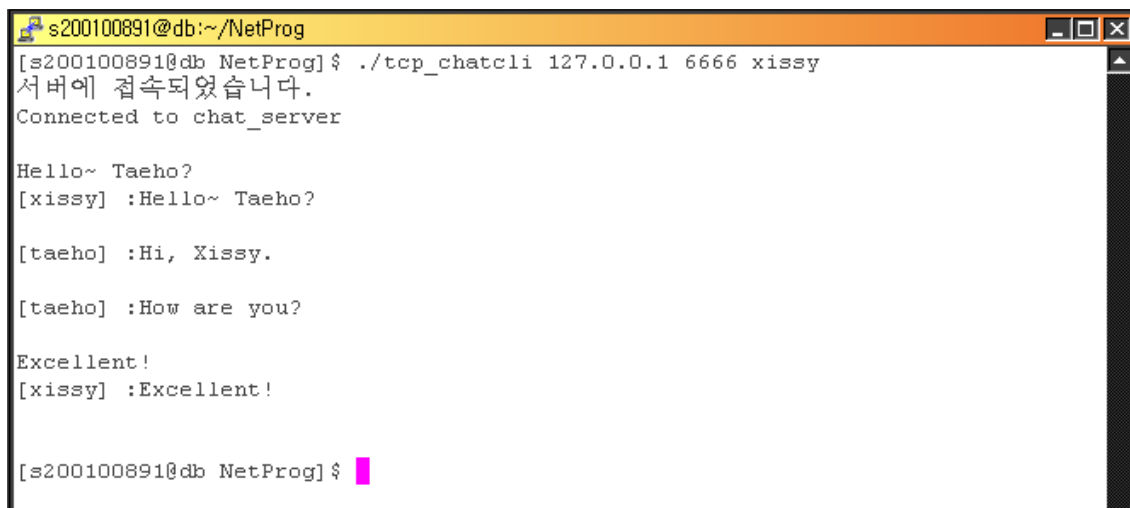
[taeho] :How are you?

[xissy] :Excellent!

Leave a user. Current users = 1
Leave a user. Current users = 0

[s200100891@db prac9]$

```



```

s200100891@db:~/NetProg
[s200100891@db NetProg]$ ./tcp_chatcli 127.0.0.1 6666 xissy
서버에 접속되었습니다.
Connected to chat_server

Hello~ Taeho?
[xissy] :Hello~ Taeho?

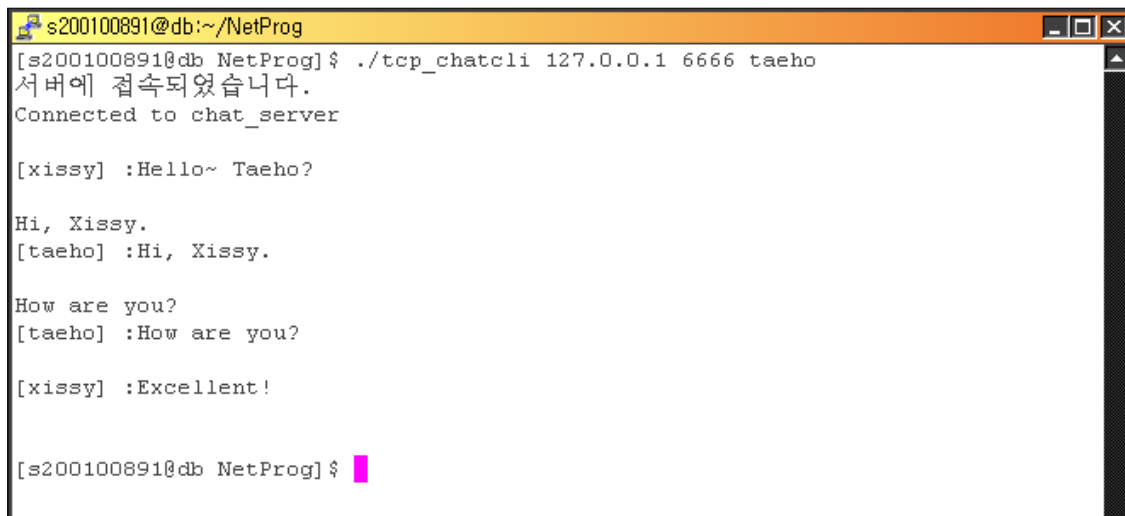
[taeho] :Hi, Xissy.

[taeho] :How are you?

Excellent!
[xissy] :Excellent!

[s200100891@db NetProg]$

```



```
s200100891@db:~/NetProg
[s200100891@db NetProg]$ ./tcp_chatcli 127.0.0.1 6666 taeho
서버에 접속되었습니다.
Connected to chat_server

[xissy] :Hello~ Taeho?

Hi, Xissy.
[taeho] :Hi, Xissy.

How are you?
[taeho] :How are you?

[xissy] :Excellent!

[s200100891@db NetProg]$
```