

## 4. 고급 소켓 프로그래밍

## 4.1 다중처리 기술

# 멀티태스킹

- 여러 작업을 병행하여 처리하는 기법
  - 멀티프로세스 또는 스레드를 사용

# 멀티프로세스

- 독립적으로 처리해야 할 작업의 수만큼 프로세스를 생성
- 장점
  - 간편한 구현
- 단점
  - 프로세스 증가로 인한 프로그램의 성능 저하
  - 프로세스간 데이터 공유가 불편함
  - 프로세스간 데이터 공유를 위해 IPC를 사용해야 함
  - IPC를 사용할 경우 프로그램의 복잡성 증가

# 멀티스레드

- 프로세스 내에서 독립적으로 실행되는 작업 단위
- 프로세스 내에서 여러 스레드를 실행시켰을 경우 외부에서는 하나의 프로세스처럼 취급
- 스레드는 프로세스의 이미지를 복사하여 사용하지 않고 원래 프로세스의 이미지를 같이 사용
  - 원래의 이미지를 공유
  - 생성된 스레드용 스택 영역은 별도로 배정
    - 스레드간에 스택은 공유되지 않음
- 데이터 공유
  - 프로세스 이미지를 공유하므로 전역변수를 같이 사용
    - 동기화 문제 발생

# 다중처리 기술의 선택

- **멀티프로세스**
  - 다중처리 작업들이 독립적으로 진행되어야 하는 경우
- **멀티스레드**
  - 다중처리 작업들이 밀접하게 연관되어 데이터 공유가 많이 필요한 경우

# 다중화

- **블록**
  - 예) read() 시스템 콜
    - 수신한 데이터를 읽기위해 read()를 호출한 경우 수신된 데이터가 있으면 리턴
    - 수신한 데이터가 없으면 데이터가 도착할 때까지 기다림(블록)
- **다수의 입출력 처리시 블록될 수 있는 입출력 함수 사용은 지양**
  - 한 곳에서 입출력이 블록되면 프로그램 전체가 블록됨
  - 멀티스레드, 멀티프로세스의 경우는 무관
- **다중화**
  - 한 프로세스(스레드) 내에서 이루어지는 다중 처리 방법
  - 폴링(polling)
  - 셀렉팅(selecting)
  - 인터럽트(interrupt)

# 폴링

- 처리해야 할 작업들을 순차적으로 돌아가면서 처리하는 방법
  - 서버가 각 클라이언트로부터의 데이터 수신을 순차적으로 처리
- 입출력 함수가 블록되지 않아야 하므로 소켓을 년블록 모드로 설정
  - 년블록 모드
    - 즉시 처리할 수 있으며 결과를 리턴하고
    - 처리할 수 없는 경우라도 리턴됨
- 여러 클라이언트가 고르게 트래픽을 발생 시키는 경우에 적합



# 선택팅

- 폴링과의 반대 개념으로 동작함
- 데이터가 도착하면 해당 클라이언트와의 입출력을 처리
- 유닉스의 select() 함수와 함께 소켓을 비동기(asynchronous)모드로 변경하여 사용

# 인터럽트

- 프로세스가 어떤 작업을 처리하는 중에 특정 이벤트가 발생하면 해당 이벤트를 처리하는 방식
- 유닉스에서 프로세스 사이에 이벤트 전달은 시그널을 사용

# 다중처리의 예 (채팅 프로그램)

- **멀티프로세스형 서버 프로그램**
  - 새로운 클라이언트 접속시 해당 클라이언트와의 통신을 담당하는 프로세스를 생성
  - 프로그램 작성이 편리하지만 수백명 이상의 클라이언트에서는 프로세스 수가 많아지는 문제가 있음
- **폴링형 서버 프로그램**
  - 소켓을 년블록 모드로 설정
  - 서버 프로그램은 접속된 클라이언트의 입출력 상태는 확인
- **선택형 서버 프로그램**
  - 소켓은 비동기 모드로 설정
  - 비동기형 방식이라고도 함
- **인터럽트형 서버 프로그램**
  - 원하는 I/O 이벤트가 발생 하였을때 서버 프로세스에게 시그널로 알리고
  - 서버가 시그널 처리 루틴에서 통신 서비스를 수행

## 4.2 소켓의 동작 모드

# 블록(blocking) 모드

- 소켓의 기본 모드
- 소켓에 대해 시스템 콜을 호출했을 경우 시스템이 동작을 완료할 때까지 프로세스가 멈추어 있는 모드
- 블록될 수 있는 소켓 관련 시스템 콜
  - `listen()`, `connect()`, `accept()`, `recv()`, `send()`, `read()`, `write()`
  - `recvfrom()`, `sendto()`, `close()`

## 년블록(non-blocking) 모드

- 시스템이 즉시 처리할 수 있으면 결과를 리턴하고 처리할 수 없는 경우에도 바로 리턴
- 다중화를 위해서 시스템 콜의 성공 여부를 확인하기 위해 폴링을 주로 사용

# 비동기(asynchronous) 모드

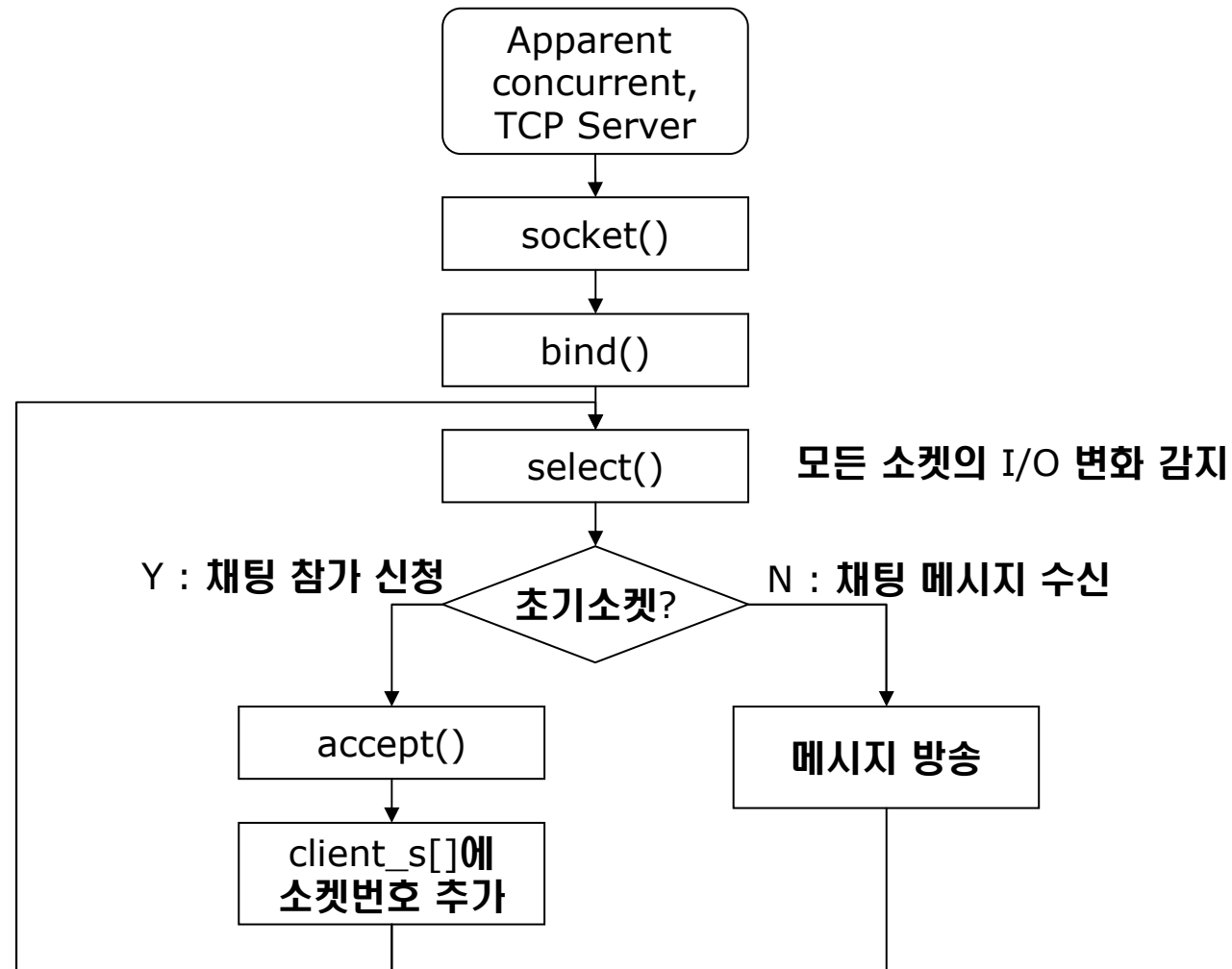
- 소켓에서 어떤 I/O 변화가 발생하면 이를 응용 프로그램에게 알려 원하는 동작을 실행시키는 모드
- 소켓을 비동기 모드로 변환해야 함
  - select(), fcntl()을 이용하여 signal-driven I/O 모드로 변환
- select()를 이용하는 방법
  - I/O 변화가 발생할 수 있는 소켓을 대상으로 select()를 호출해 두면 대상 소켓에서 I/O 변화가 발생하였을때 select()가 리턴되고 이때 해당 소켓에 대해 원하는 작업을 수행
- signal-driven I/O 방법
  - 특정 소켓에서 I/O 변화가 발생했을 때 SIGIO 시그널을 발생시키고 이 시그널을 받은 응용 프로그램에서 필요한 작업을 수행

## 4.3 비동기형 채팅 프로그램

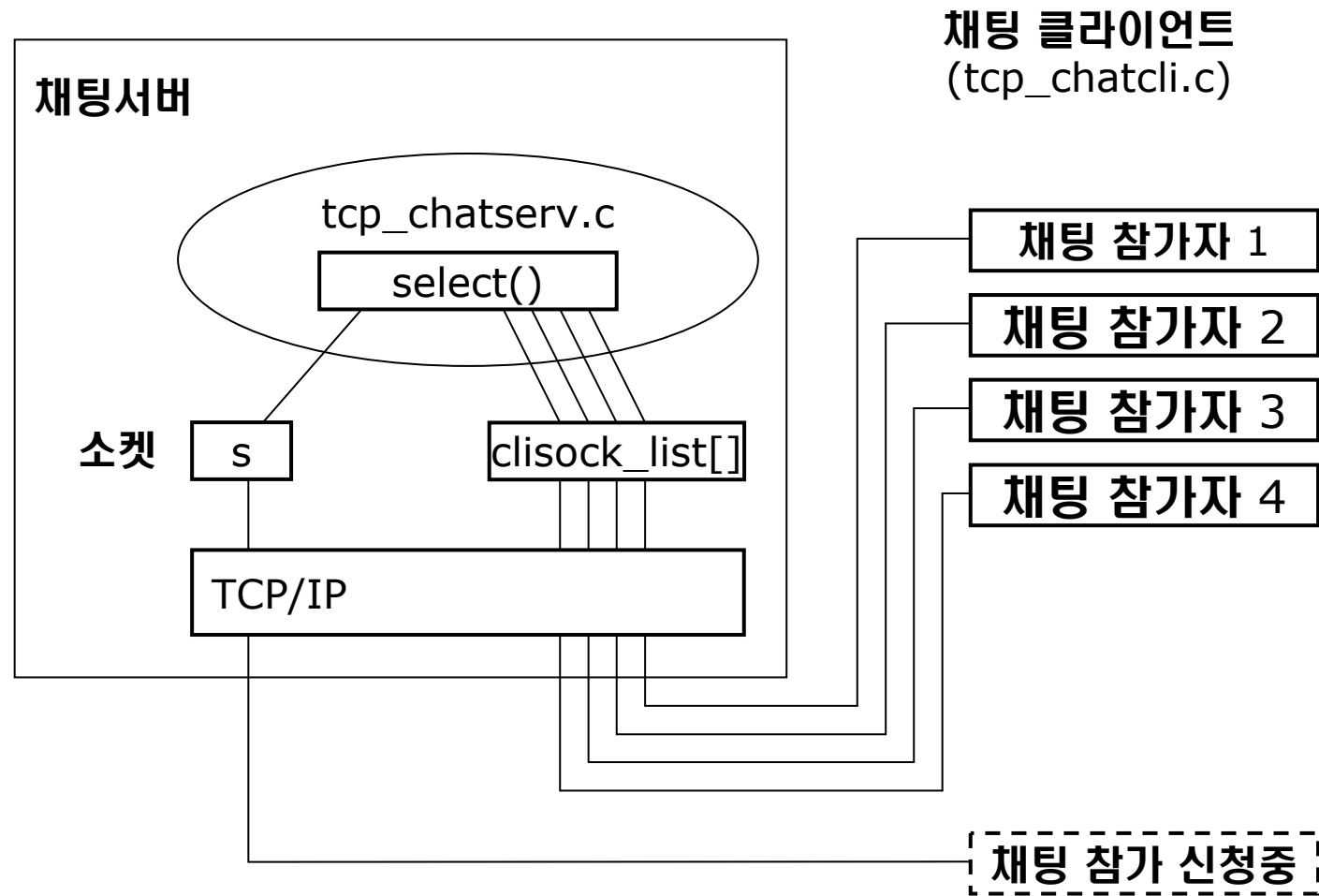


# 채팅 서버 프로그램 구조

- select()를 이용한 비동기형 채팅 서버



# 채팅 서버와 클라이언트의 연결 관계



# select()

- `int select(int maxfdp1, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *tvptr);`
  - Maxfdp1 : I/O 변화를 감지할 총 소켓의 개수 + 1
  - readfds : 읽기 I/O 변화를 감지할 소켓
  - writefds : 쓰기 I/O 변화를 감지할 소켓
  - exceptfds : 예외 상황 I/O 변화를 감지할 소켓
  - tvptr : select() 시스템 콜이 I/O 변화를 기다리는 시간
    - NULL : I/O 변화가 감지할 때까지 대기
    - 0 : I/O 변화를 기다리지 않고 바로 리턴
    - > 0 : 지정된 시간만큼 기다린 후 리턴 (도중에 변화가 감지되면 바로 리턴)

# select() 시스템 콜의 동작

readfds	1	0	0	1	...	0
writefds	0	1	0	1	...	0
exceptfds	0	0	0	0	...	0

maxfdp1-1

- fd\_set 타입 구조체에 I/O 변화를 감지할 소켓(파일)을 1로 세트하며 select()를 호출해 두면 해당 조건이 만족되는 순간 select()문이 리턴
- readfds
  - 0, 3번이 세트되어 있으므로 키보드(표준입력 0), 소켓번호 3에서 어떤 데이터가 입력되어 프로그램이 이를 읽을 수 있는 상태가 되면 select()문이 리턴
- writefds
  - 1, 3번이 세트되어 있으므로 파일기술자(표준출력 1)나 소켓번호 3번이 write를 할 수 있는 상태로 변하면 select()문이 리턴

# fd\_set 구조체의 값을 지정하기 위한 매크로

- 매크로

- FD\_ZERO(fd\_set \*fdset);
  - fdset의 모든 비트를 지움(삭제)
- FD\_SET(int fd, fd\_set \*fdset);
  - fdset 중 소켓 fd에 해당하는 비트를 1로 지정
- FD\_CLR(int fd, fd\_set \*fdset);
  - fdset 중 소켓 fd에 해당하는 비트를 0로 지정
- FD\_ISSET(int fd, fd\_set \*fdset);
  - fdset 중 소켓 fd에 해당하는 비트가 세트되어 있으면 양수값을 리턴

- fd\_set 지정

```
fd_set read_fds;
FD_ZERO(&read_fds);      // 모든 비트를 지움
FD_SET(s, &read_fds);     // 초기소켓(클라이언트 참가용)을 선택
for(i=0; i<num_chat, i++) // 참가중인 모든 클라이언트의 소켓을 선택
{
    FD_SET(client_s[i], &read_fds);
}
```

# fd\_set에서의 변화 확인

```
select(maxnfdsp1, &read_fds, (fd_set *)0, (fd_set *)0, (struct timeval *)0);
```

- 지정된 fd\_set에 어떤 소켓에서 변화가 있었는지 확인

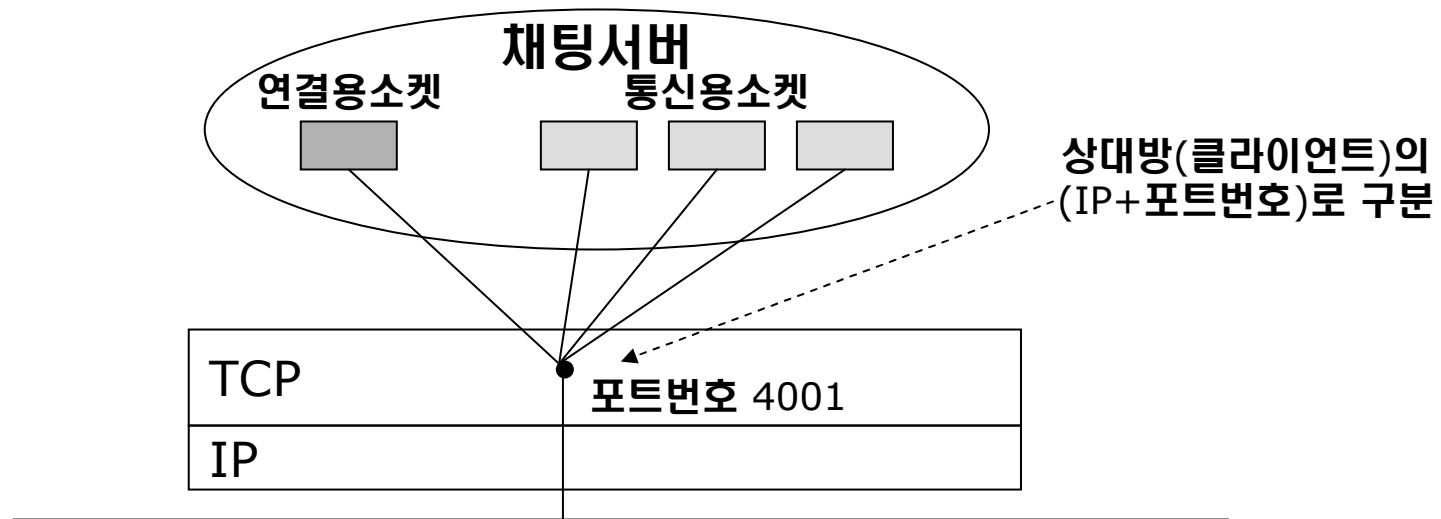
```
FD_ISSET(s, &read_fds);

if (FD_ISSET(s, &read_fds)) {
    // 초기소켓 s에서 입력 발생
    // 채팅 참가 신청 처리
}

for (i=0; i<num_chat, i++) {
    if (FD_ISSET(clisock_list[i], &read_fds)) {
        // 통신용 소켓 clisock_list[i]에서 채팅 메시지 수신
        // 모든 참가자에게 채팅 메시지 방송
    }
}
```

# 통신용 소켓 구분

- 서버는 클라이언트와의 통신에 사용하기 위한 다수의 통신용 소켓을 개설
  - 이 소켓들은 모두 같은 포트번호를 사용
  - 모든 통신용 소켓은 새로운 포트번호를 배정받는 것이 아니라 같은 4001번 포트를 사용
- 한 개의 포트번호를 이용한 각 클라이언트의 구분
  - 서버는 각 연결을 구분하기 위해서 클라이언트의 IP주소와 포트번호를 내부적인 키로 사용



# 채팅 클라이언트 프로그램

- 기능
  - 키보드 입력 메시지를 서버로 전송
  - 서버가 전송한 메시지를 수신하여 화면에 출력
- select()
  - 사용자의 키보드 입력 처리와 수신 메시지 출력을 처리
  - 소켓은 읽기에 대한 I/O만 처리하면 되므로 fd\_set 구조체 read\_fds 만 사용
  - 키보드 입력을 위해 파일 디스크립터 0(표준 입력)을 read\_fds에 사용



## 4.4 폴링형 채팅 프로그램

# fcntl()

- **소켓을 년블록 모드나 비동기 모드로 변환**
  - 소켓을 년블록 모드로 설정
  - 소켓을 비동기 모드로 설정
  - 소켓의 소유자 설정 또는 현재 소유자를 얻음
- **년블로 모드 설정**

```
#include <unistd.h>
#include <fcntl.h>
```

```
int fcntl(int fd, int cmd, long flag);
```

- **fd : 소켓 디스크립터**
- **cmd**
  - **F\_SETFL : 플래그 세트**
  - **F\_GETFL : 플래그 읽기**
  - **F\_SETOWN : 소켓의 소유자 설정**
  - **F\_GETOWN : 소켓의 소유자 얻기**
- **flag**
  - **O\_NONBLOCK : 년블록 모드로 설정**
  - **O\_ASYNC : SIGIO 시그널에 의해 구동되도록 비동기 모드로 설정**

# fcntl()

- **소켓을 년블록 모드로 설정하는 코드**
  - 기존의 플래그 값을 유지하기 위하여 F\_GETFL 명령으로 얻은 후 이를 O\_NONBLOCK와 OR 연산

```
int val;  
  
if ((val = fcntl(sock_fd, F_GETFL, 0)) < 0)  
    exit(1);  
  
val |= O_NONBLOCK;  
  
if ((fcntl, F_SETFL, val) < 0)  
    exit(1);
```

# 비동기 모드

- **select() 함수는 블록형 함수**
  - 인터럽트형 다중화에서는 SIGIO 등의 시그널이 발생할 때 입출력 처리가 가능
  - 이를 위해 소켓을 비동기 모드로 설정해야 함

```
int flag;  
  
if ((flag = fcntl(fd, F_GETFL, 0)) < 0)  
    exit(1);  
  
Flag |= O_ASYNC;  
  
If ((fcntl, F_SETFL, flag) < 0)  
    exit(1);
```

- 소켓을 처음 생성하면 소유자가 없음
- 시그널 수신을 위해서는 소유자가 필요하므로 소유자 설정이 필요함

```
fcntl(fd, F_SETOWN, getpid());  
Fcntl(fd, F_GETOWN, &pid);
```

# 폴링형 채팅 서버

- **넢블록 모드**

- **fcntl()을 이용하여 넢블록 모드로 설정한 후 무한 루프를 돌면서 입출력을 폴링**
- **넢블록 모드의 소켓에 대한 read(), write()는 바로 리턴**
  - 원하는 작업의 실행여부를 확인해야 함
  - 정상인 경우 0, 에러이면 -1을 반환
  - 에러일 경우 errno의 값으로 확인 가능
  - 넢블록 모드의 소켓으로 즉시 리턴된 것이면 EWOULDBLOCK의 errno 값을 가짐

```
n = recv(s, buf, length, 0);  
if (n>0) {  
    // 정상적으로 읽은 데이터 처리  
}  
else if (n==-1 && errno == EWOULDBLOCK)  
    //에러가 아니므로 다음 작업으로 진행  
else if (n==-1 && errno != EWOULDBLOCK)  
    // 시스템 콜 자체에서 에러가 발생함  
    // 에러 처리
```

# 폴링형 채팅 서버

- **recv()를 호출했을 때 EWOULDBLOCK 이외의 에러**
  - 클라이언트와의 연결 종료 또는 클라이언트가 리셋을 보낸 경우이므로 클라이언트를 채팅 목록에서 제거하는 등의 에러처리를 해야 함
- **accept()**
  - **년블록** 모드의 소켓에 accept()를 호출한 경우 accept()가 리턴한 소켓은 **블록** 모드이므로 필요한 경우 명시적으로 **년블록** 모드로 변환해야 함
- **소켓의 모드 확인**
  - **fcntl() 함수를 사용**

```
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;
}
```