Chapter 5. 소켓 프로그래밍

★목 차

5.1 소켓 옵션들

- TCP/IP 프로토콜 개발자들은 대부분의 애플리케이션들을 만족시 킬 수 있는 디폴트 동작에 대해 많은 시간 고려

- 소켓의 특정부분은 소켓 옵션에 연관되어 있음

- 연관된 소켓 옵션의 값을 변경함으로써 자신의 애플리키에션에 맞는 소켓의 수신 버퍼 크기 변경가능

- 소켓의 작용은 소켓레벨 옵션에 의해 제어된다. 이러한 옵션들은 <sys/socket.h> 에 정의 되어있다. Setsockopt(2) 와 getsockopt(2) 는 옵션을 설정하고 얻는데 사용된다.

[옵션을 변경하는 함수]

- int getsockopt(int socket, int level, int optName, void *optVal, unsigned int *optLen)
- int setsockopt(int socket, int level, int optName, const void *optVal, unsigned int optLen)

- 소켓의 옵션을 추출한다.
- 소켓 옵션들은 디폴트 소켓 동작을 변경하는데 사용된다.

socket	소켓
level	옵션 단계
optName	옵션 이름
optVal	옵션 값을 기록할 버퍼의 주소
optLen	옵션 값의 길이(바이트 수) - getsockopt()
	옵션 값 버퍼의 길이(바이트 수) - setsockopt()

- 오류가 발생하지 않으면 0을, 그렇지 않으면 -1을 반환한다.

- Level : 가능한 소켓 옵션들은 프로토콜 스택 계층(layer)에 대응되는 레벨(level)들로 나뉘는데, 두 번째 파라미터가 해당 옵션의 레벨을 나타낸다.

  SOL_SOCKET옵션 – 프로토콜에 무관하여 소켓 계층자체 에서 처리

  IPPROTO_TCP옵션 – 전송 프로토콜에 특화됨

  IPPROTO_IP옵션 – 인터네트워크 프로토콜에 의해서 처리
- optName : 옵션 그자체는 정수 optName에 의해서 표현됨
- optVal : 버퍼의 포인터

- Getsockopt()에서 옵션의 현재 값은 그 버퍼에 저장되고
- Setsockopt()에서 소켓 옵션은 그 버퍼에 있는 값으로 지정됨
- optLen : 버퍼의 길이를 지정하는데 이것은 해당 특정 옵션에 대해 정확해야 함

  Getsockopt()에서 optLen은 in-out 파라미터로서 초기에 버퍼의 크기를 포함하는 정수를 가리키며, 반환시 옵션 값의 크기를 포함하는 정수를 가리키게 됨.

[소켓 옵션]

optName	형태	값	설명
SOL_SOCKET단계			
SO_BROADCAST	Int	0,1	브로드 캐스트를 허용함
SO_KEEPALIVE	Int	0,1	킵얼라이브(“살아있음”)메세지들가능(프로토콜에 의해 구현된 경우)
SO_LINGER	Linger{}	시간	확인을 기다리며 close()반환을 지연시키기 위한 시간(6.4.2절 참조)
SO_RCVBUF	Int	바이트	소켓 수신 버퍼의 바이트들(73쪽 코드 및 6.1절 참조)
SO_RCVLOWAT	Int	바이트	Recv()의 반환을 유발하는 최소 사용 가능 바이트의 수
SO_REUSEADDR	Int	0.1	이미 사용 중인 주소나 포트(특정 조건하에 결합을 허용함 (6.4 및 6.5절 참조)
SO_SNDLOWAT	Int	바이트	보내는 최소 바이트들
SO_SNDBUF	int	바이트	소켓 송신 버퍼의 바이트들 (6.1절 참조)

optName	형태	값	설명
IPPROTO_TCP단계			
TCP_MAX	Int	초	킵얼라이브(“살아있음”)메세지들 사이의 초 단위 시간
TCP_NODELAY	Int	0,1	데이터 통합(merging)을 위한 지연 불허(Nagle의 알고리즘)
IPPROTO_IP단계			
IP_TTL	int	0-255	유니캐스트 IP 패킷들을 위한 time-to-live
IP_MULTICAST_TTL	Unsigned char	0-255	멀티캐스트 IP 패킷들을 위한 time-to-live(120쪽의 MulticastSender.c참조)
IP_MULTICAST_LOOP	int	0,1	멀티캐스트 소켓이 자신이 보낸 패킷들 수신 가능
IP_ADD_MEMBERSHIP	Ip_mreq{}	그룹 주소	특정 멀티캐스트 그룹으로 보내진 패킷들의 수신 가능(122쪽 MulticastReceiver.c참조)-지정하기만 함
IP_DROP_MEMBERSHIP	Ip_mreq()	그룹 주소	특정 멀티캐스트 그룹으로 보내진 패킷들의 수신 불가능 – 지정하기만 함

5.2 신호(Signals)

- 신호는 프로그램에게 어떤 사건들이 일어났다는 것을 알리는 체계를 제공한다.

  EX)

  1. 사용자가 “인터럽트” 문자를 입력한 경우
  2. 타이머가 만료된 경우

- 전달된 신호의 4가지 처리 유형

  1. 신호가 무시시킴
  2. 프로그램을 강제종료시킴
  3. 신호가 블록됨
  4. 신호처리 루틴을 수행함

[소켓에 주로 사용되는 신호들 5가지]

신호	유발 사건	디폴트 동작
SIGALRM	경보타이머의 만료	종료
SIGCHLD	자식 프로세스에서 빠져 나옴	무시
SIGINT	인터럽트 문자 입력	종료
SIGIO	소켓 입,출력 준비 완료	무시
SIGPIPE	닫힌 소켓에 쓰려고 시도	종료

[신호의 디폴트 동작 변경함수]

```
int sigaction( int whichSignal,
      const struct sigaction* newAction,
            struct sigaction* oldAction)
```

성공시 0, 실패시 -1을 반환

- sigaction( ) 인수

whichSignal : 동작이 변경되는 신호
newAction : 해당 신호 형태의 새로운 동작을 정의하는 sigaction 구조체를 가리킨다.
oldAction : 값이 널이 아니라면 해당 신호의 이전동작을 나타내는 sigaction구조체가 여기에 복사된다.

- sigaction() 구조체-1

```
struct sigaction {
        void ( *sa_handler ) (int);
        sigset_t sa_mask;

        int sa_flags;
 };
```

- sigaction() 구조체-2

  sa_handler : 신호가 전달 되었을 때 제어하는 함수를 가리키는 함수포인터

  sa_handler의 3가지

  1. SIG_IGN : 신호가 무시됨
  2. SIG_DFL : 디폴트 동작이 수행됨
  3. 함수주소 : 함수가 실행됨

- sigaction() 구조체-3

sa_mask 필드는 whichSignal을 처리하는 도중 블록된 신호들을 나타내는데, 이것은 sa_handler가 SIG_IGN이거나SIG_DFL일 때만 의미가 있다. 디폴트로 whichSignal은 sa_mask에 상관없이 항상 블록된다.

sa_flags필드는 whichSignal이 처리되는 방식을 추가적으로 제어한다.

- sigaction() 구조체-4

sa_mask-BOOL형태의 플래그들의 집합으로 구현됨.
한 플래그가 각 신호 형태에 해당되고 이들의 집합은 다음의 네 가지 함수로서 조작가능.

int sigemptyset(sigset_t * set) -다 0으로 셋팅
int sigfillset(sigset_t * set) -각플래그값안에 1로 셋팅
int sigaddset(sigset_t * set, int whichSignal) - 세부
int sigdelset(sigset_t * set , int whichSignal) -
네 가지 함수는 성공 시 0, 실패 시 -1 반환

- sigaction() 구조체-5

  int sigfillset(sigset_t * set)

  -> 주어진 집합의 모든 플래그들을 지정한다.

  int sigemptyset(sigset_t * set)

  -> 주어진 집합의 모든 플래그들을 해지한다.

  int sigaddset(sigset_t * set, int whichSignal)

  주어진 집합의 신호 번호에 의해 지정된 플래그들을 지정한다.

  int sigdelset(sigset_t * set , int whichSignal)

  주어진 집합의 신호 번호에 의해 지정된 플래그들을 해지한다.

예제 sigaction.c - 1

```
0 #include <stdio.h> /* for printf() */
1 #include <signal.h> /* for signal() */
2 #include <unistd.h> /* for pause() */
3
4 void DieWithError(char *errorMessage); /Error handling function*/
5 void InterruptSignalHandler(int signalType);/*Interrupt signal handling function*/
6
7 int main(int argc, char * argv[])
8 {
9     Struct sigaction handler; /* Signal handler specification structure */
10
11   /* Set InterruptSignalHandler as handler function*/
12   Handler.sa_handler = InterruptSignalHandler;
13   /*Create mask that masks all signals*/
14   If (sigfillset(&handler.sa_mask) < 0)
15   DieWithError(“sigfillset() failed”);
16   /* No flags */
17   Handler.sa_flags = 0;
```

```
18
19   /*Set signal handling for interrupt signals */
20   If (sigaction(SIGINT, &handler, 0) < 0)
21      DieWithError("sigaction() failed");
22
23   for(;;)
24         pause(); /* suspend program until signal received */
25
26   exit(0);
27  }
28
29 void InterruptSignalHandler(int signalType)
30 {
31         printf("Interrupt Received. Exiting program.\n");
32         exit(1);
33 }
```

예제 sigaction.c 소스 분석

1. 신호 처리기 함수 프로토타입: 5줄
2. 신호 처리기 설정: 9 – 21줄
   ◎신호 처리를 위한 함수 지정: 12줄
   ◎신호 마스크 채움: 14-15줄
   ◎SIGINT를 위한 신호 처리기 설정: 20-21줄
3. SIGINT까지 무한 루프:23 – 24줄
   pause()가 신호를 받을 때까지 프로세스를 중단
4. 신호 처리를 위한 함수:29 – 33줄
   InterruptSignalHandler()가 메시지를 인쇄하고
   프로그램에서 빠져 나온다.

[신호의 중첩]

신호의 중첩이란-

신호가 처리 중일 때 다른 신호가 전달된 경우 전달된 신호는 처리기가 일을 끝낼 때까지 연기된다. 이런 신호를 계류중 이라고 한다. 즉, 신호는 계류중 이거나 그렇지 않거나 이다.

같은 신호가 처리 중에 두 번 이상 전달되면 처리기는 원래의 실행을 마치고 오직 한번만 더 수행한다.

5.3 블로킹 넌블로킹

[유닉스의 I/O]

- Linux 는 다른 Unix 와 마찬가지로 일반적인 파일을 비롯해서 다른 모든 것들이 파일로 처리되므로(소켓, 각종 디바이스) I/O 란 곧 파일에 대한 입출력 을 말한다.
- 소켓에 대한 접근 또한 파일에 대한 입출력으로 접근한다.
- 유닉스의 I/O의 종류
  - Blocking I/O : 봉쇄 입출력이라고 한다.
  - Non-Blocking I/O : 비봉쇄 입출력이라고 한다.
  - I/O Multiplexing : 입출력 다중화 라고 한다.
  - Asynchronous : 비동기 입출력이라고 한다.

5.3.1 블로킹이란..?

- 일반적으로 우리가 함수를 호출하면 그 함수가 수행을 마치고 리턴할때까지 다른 아무일을 할수 없는 상태가 발생한다.

  ->이를 블록 상태라고 말한다.

- 소켓호출의 기본동작은 블록킹이다.
  - 예) 1. recv()함수는 적어도 하나의 메시지를 받을때까지 반환하지 않는다.(즉 블록된다)

    2. send()의 경우 전송할 데이터를 저장할 공간이 충분하지 못하면 블록된다.

    3. connect()의 경우도 연결이 설정될때까지는 블록된다.
- **블로킹의 문제점.**
  - 송신측은 필요할때 send()을 호출하여 데이터를 전송하면되지만 수신측은 언제 Recv()을 호출하여 전달된 데이터를 읽어들여야 할지 그 점이 명확하지않다 이때 수신측에서 블로킹함수를 사용한다면 송신측에서 데이터는 보내지않는한 다른작업을 수행할수가 없다

5.3.2 넌블록킹의 이용

- 블로킹의 해결책
  - 넌블록킹과 비동기화를 사용하면 된다.
- 넌블록킹이란?
  - 넌블록킹화 시키면 I/O 함수는 읽을 값이 없을 경우 블록하지 않고 바로 error를 리턴한다. 단, 표준입출력 장치의 error가 아닌 경우 error값을 EAGAIN으로 선택하여 리턴하므로써, 입출력장치의 error인지 데이터가 준비되지 않는 것인지 구분할수 있다.
  - 즉 EAGAIN은 넌-블록킹 I/O가 **O_NONBLOCK**을 사용하여 선택되어졌고 즉시 읽을 수 있는 데이터가 없다라는 error 메시지다.
- 넌블록킹의 사용방법
  - 함수의 기본동작이 블록킹으로 작성되었다 할지라도 fcntl과 같은 I/O제어 함수를 통해서 동작을 변경시켜줄수 있다.
- 소켓의 경우
  - 시스템은 errno를 connect()의 경우 EINPROGRESS로 반환
  - 그밖의 경우는 errno를 EWOULDBLOCK으로 반환한다.
    - # 여기서 EWOULDBLOCK은 EAGAIN과 같은 값이다.

5.3.3 fcntl()함수의 사용 –File Control

- fcntl 시스템호출은 이미 열려있는 파일의 특성 제어를 위해서 사용된다
- int fcntl(int fd, int cmd, long argumnet) :fd file descriptor
  - int fd : open이나 socket 등의 시스템 호출을 통해서 만들어진 파일 지정자이다
  - int cmd: fd에 대한 특성을 제어하는 값
  - long argumnet: cmd를 지정할때 사용되는 argment
    - 플래그를 읽어올때(F_GETFL) argment는 0
    - 플래그를 쓸때(F_SETFL) argment는 플래그 값이 된다.
      - 자세한 내용은 뒷편 (Flag_Setting)
    - SIGIO 시그널을 받을 프로세스를 설정할때(F_SETOWN) argment는 프로세스 ID나 프로세스 그룹 ID가 된다.

argument로 사용되는 플레그

Open()에서 사용되는 플래그들과 비슷하므로 open에 사용되는 플래그들을 나열하겠음.

- 반드시 하나만 정의되어야 하는 플래그
  - O_RDONLY : 읽기 전용
  - O_WRONLY : 쓰기 전용
  - O_RDWR : 읽기, 쓰기 가능
- 중복 지정이 가능한 플래그(보통 |연산으로 중복)
  - O_APPEND : 모든 쓰기 작업은 파일의 끝에서 수행된다.
  - O_CREAT : 파일이 없을 경우 파일을 생성한다. (세 번째 인자 필요)
  - O_EXCL : O_CREAT와 같이 쓰이며, 파일이 있는 경우에 error를 발생시킨다.
  - O_TRUNC : O_CREAT와 같이 쓰이며, 파일이 있는 경우에 기존 파일을 지운다.
  - O_NONBLOCK : blocking I/O를 nonblocking 모드로 바꾼다.
  - O_SYNC : 매 쓰기 연산마다 디스크 I/O가 발생하도록 설정한다.
  - O_ASYNC :비 동기적 입출력을 사용하기 위하도록 플래그를 설정한다. **FASYNC를 쓰기도 한다.**

- 명령의 종류 (cmd) – 붉은색이 자주쓰는것
  - F_DUPFD : 파일 디스크립터를 복사. 세 번째 인수 보다 크거나 같은 값 중, 가장 작은 미사용의 값을 리턴한다.
  - F_GETFD : 파일 디스크립터의 플래그를 반환 (FD_CLOEXEC)
  - F_SETFD : 파일 디스크립터의 플래그를 설정
  - F_GETFL : 파일 테이블에 저장되어 있는 파일 상태 플래그를 반환
  - F_SETFL : 파일 상태 플래그의 설정 (O_APPEND, O_NONBLOCK, O_SYNC 등을 지정)
  - F_GETOWN : SIGIO, SIGURG 시그널을 받는 프로세스 ID와 프로세스 그룹 ID를 반환
  - F_SETOWN : SIGIO, SIGURG 시그널을 받는 프로세스 ID와 프로세스 그룹 ID를 설정

5.3.4 errono의 정리

- 입출력 함수 사용시 리턴되는 값들(open()과 비슷하므로 open()함수의 return값(error)을 정리함.
  - **EINTR**
    - 어떤 데이터를 읽기도 전에 함수가 신호에 의해 인터럽트되었다.
  - **EAGAIN**
    - 넌-블록킹 I/O가 **O_NONBLOCK**을 사용하여 선택되어졌고 즉시 읽을 수 있는 데이터가 없다.
  - **EIO**
    - I/O 에러. 이것은 백그라운드 프로세스 그룹에 있는 프로세스가 제어되는 tty 단말기에서 읽기를 시도할때, 그리고 이것이 무시되거나 봉쇄되는 SIGTTIN이거나 또는 프로세스 그룹이 고아일때 일어난다. 또한 디스크나 테이프에서 읽는동안 저레벨(low-level) I/O 에러가 있을 때 일어난다.
  - **EISDIR**
    - *fd*가 디렉토리를 가리킨다.
  - **EBADF**
    - *fd*가 유효한 파일 기술자가 아니거나 읽기 위해 열려지지 않았다.
  - **EINVAL**
    - *fd*가 읽기에 적당하지 않은 객체와 연결되었다.
  - **EFAULT**
    - *buf*는 접근할 수 없는 주소공간을 가리키고 있다.

5.3.4 비동기적 입출력

- 넌블로킹의 어려운점
  - 주기적으로 성공할 때까지 폴링을 해야한다.
- 비동기적 입출력이란?
  - 운영체제가 소켓 호출이 성공적이었을때 프로그램에게 알려주도록 하는 방법
  - 임의 입출력과 관련 event가 소켓에서 발생하면 **SIGIO신호를 프로세스에 전달하는 방식을 사용**하고 있다.
- 비동기적 입출력 생성법
  - 1단계: SIGIO를 위한 신호핸들러 설정이 필요하다.
  - 2단계: 소켓에 관련된 신호들이 이 프로세스에게 전달되도록 설정한다.(즉 소켓의 소유자 프로세스설정)
  - 3단계: 비동기입출력이 가능하도록 설정한다. Fctnl, Asynchronous

[책의 예제를 통한 접근]

- sigaction(SIGIO, &handler, 0)
  - SIGIO 신호를 통해 소켓 입출력 준비 완료를 통보 받아 적합한 루틴을 실행하기 위해 설정
- fcntl(sock, F_SETOWN, getpid())
  - sock이 나타내는 소켓을
  - cmd: F_SETOWN → SIGIO, SIGURG 시그널을 받는 프로세스 ID와 프로세즈 그룹 ID를 설정하겠다.(같은 소켓을 여러 프로세스가 사용할수 있으므로)
  - Argment: getpid() → 현재 프로세스 아이디를 아규먼트 값으로 넣는다.
- fcntl(sock,F_SETEL,O_NONBLOCK | FASYNC)
  - cmd : F_SETEL→F_SETFL 에 의해서 파일지정자에 대한 값(특성)을 세팅한다.
  - Argment: O_NONBLOCK : blocking I/O를 noneblocking으로 바꿈
    FASYNC : FASYNC 상황 플래그를 설정하면 SIGIO 신호는 파일 기술자 상에서 입력이나 출력이 가능하게 될 때마다 보내어진다.
- Void SIGIOHandler(int signalType){…….}
  - 단지 SIGIO의 신호를 받으면 recvfrom()을 시도하며, 만약 recvfrom의 return값이 EWOULDBLOCK이 아니면 실패로 간주하고 프로세스를 마무리한다.

[fcntl()의 참고문서]

- Fcntl()에 대해 더 자세히 알고 싶을 경우 참고 문서를 덧 붙인다.
  - http://cantata.kaist.ac.kr/%7Ezerone/data/c/unix_c/8.htm#10
  - http://www.joinc.co.kr/modules/moniwiki/wiki.php/article_fcntl%C0%BB_%C0%CC%BF%EB%C7%D1_%C6%C4%C0%CF%C1%A6%BE%EE
- Signal에 대한 문서
  - http://www.joinc.co.kr/modules/moniwiki/wiki.php/article_signal%20%B4%D9%B7%E7%B1%E2%202

5.3.3 타임아웃

- 문제제기:
  - 만약 UDP 패킷이 유실된다면?
    - 클라이언트는 패킷이 유실됐는지 알지 못한다. 즉 무한이 기다리게 된다.
- 방법제시: 시간 제한을 두어 패킷의 유실여부를 파악
  - 블로킹 함수를 호출하기 전에 alarm()함수를 통해 경보를 설정한다.
  - 사용함수: unsigned int alarm(unsigned int secs)
  - alarm()은 타이머를 **구동시키며 지정된 시간이 지난후 만료될때** SIGALRM신호가 보내진다.
  - 이 신호를 받고 적적한 핸들러를 구동시킨다.
    - 책의 예제는 trie라는 값을 두어 recv()의 시도를 체크한다.
  - 만약 정해놓은 수만큼 시도를 하고 그 이후에도 패킷을 받지 못하는 경우 유실로 파악하고 종료를 한다.

[교제 예제를 통한 타임아웃이해]

- myAction.sa_handler = CatchAlarm;
- Sigfillset(&myAction.sa_mask)
  - 주어진 집합의 모든 플래그들을 지정한다.(즉 한 신호가 처리중일 경우 다른 신호들은 계류시킨다)
- If(sigaction(SIGALRM, &myAction,0)
  - Sigaction을 통해 SIGALRM 신호에 대해 myAction을 호출한다
- Sendto(……) : 패킷을 보낸다.
- While((recvfrom(……))<0) :받은 패킷이 0보다 작은 동안 반복해서 체크한다.
  - If(errno == EINTR) 즉 어떤 데이터를 읽기도 전에 함수가 신호에 의해 인터럽트되었다면(이때 인터럽트는 alarm에 의해 생성된다)
  - If(tries <MAXTRIES) 최대 읽기 시도 값보다 현재 시도한 값이 적다면
    - Sendto(……) 다시한번 보내 보고
    - Alarm(TIMEOUT_SECS); alarm함수를 다시 시작한다.
  - Else
    - DieWithError(.....) 최대 읽기 시도 값보다 현재 시도값이 같거나 클경우 종료를 한다.
- Void CatchAlarm(int ignored) { tries += 1}
  - Alarm이 SIGALRM을 보내면 받아 이 함수를 처리한다.(현재 시도한 값을 1올린다.)

5.4 Multitsaking
(한번에 여러 개의 클라이언트 다루기)