12 저수준 터미널 인터페이스

장길석 2007.11.28 20:13:39

12 저수준 터미널 인터페이스

이 장은 터미널 디바이스의 여러 가지를 정하는 함수를 설명하고 있다. 당신은 이 함수들을 통해서 입력의 반향(echoing)을 없애는 것과 같은 일을 할 수 있고; 속도와 흐름제어와 같은 직렬통신 문자들을 설정할 수도 있고; 파일의 끝, 코멘트-라인 편집, 신호 보내기, 그리고 유사한 신호 함수들을 위해 사용된 문자들을 변경할 수도 있다.

이 장에 있는 함수들의 대부분은 파일 기술자 상에서 동작한다. 8장[Low-Level I/O] 에 파일 기술자가 무엇이고, 터미널 디바이스를 위해 파일 기술자를 어떻게 개방하는지에 대한 자세한 정보가 있다.

---

12. 1 터미널 확인하기

이 장에서 설명하고 있는 함수들은 터미널 디바이스에 해당하는 파일에서만 동작한다. 당신은 isatty 함수를 사용해서 터미널과 연관된 파일기술자인지 어떤지를 알아낼 수 있다. isatty 와 ttyname 함수들을 위한 프로토타입은 헤더파일 'unistd.h'에 선언되어 있다.

함수 : int isatty (int filedes)

> 이 함수는 만일 filedes가 터미널 디바이스와 연관된 파일 기술자이면 1을 반환하고, 그렇지 않으면 0을 반환한다. 만일 파일기술자가 터미널과 연관되어 있다면, 당신은 ttyname함수를 사용해서 그 연관된 파일 이름을 얻을 수 있다. 또한 ctermid함수를 참고하라, 그것은 24. 7. 1절 [Identifying the Termina.] 에 있다.

함수 : char *ttyname(int filedes)

> 만일 파일 기술자 filedes 가 터미널 디바이스와 연관되어 있으면, ttynaem함수는 정적으로 할당되고, 널 문자로 끝나는 문자열에 터미널 파일의 파일이름을 저장하여 그것을 가리키는 포인터를 반환한다. 만일 파일 기술자가 터미널과 연관되지 않거나, 또는 그 파일이름을 알아낼 수 없다면, 널 포인터를 반환한다.

---

12. 2 입/출력 큐

이 절에 있는 많은 함수들은 터미널 디바이스의 입력과 출력 큐(queues)를 조회한다. 이들 큐는 입/출력 스트림들에 의해 실행된 버퍼링의 커널안에 있는 버퍼링의 한 형식이다. (7장 [I/O on Streams] 참조.)

터미널 입력 큐는 또한 선행입력(typeahead) 버퍼로써 사용되어진다. 그 큐는 터미널로부터 받아들여졌지만, 아직 어느 프로세스에 의해서도 읽혀지지 않은 문자들은 저장한다.

-- 역자주: typeahead : 어떠한 이유로 인하여 입력의 속도가 프로그램의 작업처리 속도보다 빠를 경우 아직 처리되지 못한 입력들은 잠시 내부에 있는 기억장치에 저장해두고 나중에 처리하는 방법

터미널의 입력 큐의 크기는 _POSIX_MAX_INPUT 과 MAX_INPUT 파라미터로 표현된다; 27. 6절 [Limits for Files] 참조. 만일 IXOFF 입력 모드 비트가 설정되어서 입력 흐름제어가 가능하다면(12. 4. 4절 [Input Modes] 참조), 터미널 드라이버는 큐가 오버플로우가 나는 것을 방지하기 위해, 필요할 때 터미널로 STOP 와 START 문자들을 전송한다. 그렇지 않다면, 터미널로부터 너무 많은 입력이 쇄도할 경우 그 입력을 잃어버릴지도 모른다. (이런 상황은 손으로 타이핑을 통해 입력하는 것으로는 불가능하다.)

터미널 출력 큐는 입력큐와 같지만, 출력을 위해 쓰인다. 출력큐는 프로세스에 의해 출력되어 졌지만, 아직 터미널로 전송되지 않은 문자들을 저장하고 있다. 만일 IXON 입력 모드 비트(12. 4. 4절 [Input Modes] 참조)가 설정되어서 출력 흐름

제어가 가능하다면, 터미널 드라이버는 멈춤을 지시하기 위해 터미널에서 보낸 STOP 문자들과 출력의 재전송에 따른다.

터미널의 입력큐의 소거(Clearing)란 받아들이기는 했지만 아직 읽혀지지 않은 문자들을 버리는 것을 의미한다. 유사하게, 터미널 출력큐를 소거하기란 출력됐지만, 아직 전송되지 않은 문자들을 버리는 것을 의미한다.

---

12. 3 입력의 두 가지 스타일: Canonical 또는 Not

POSIX시스템은 입력의 두 가지 기본 모드를 제공한다: 정규와 비정규(canonical and noncanonical)

정규(canonical) 입력 프로세싱 모드에서, 터미널 입력은 새줄문자('\n'), EOF, 또는 EOL 문자들로 종료되는 한 라인으로 처리된다. 어떤 입력도 사용자에 의해 한 라인 전체의 입력이 종료되기 전에 읽혀질 수 없고, read 함수는(8. 2절 [I/O Primitives] 참조), 얼마나 많은 바이트가 요청되었는지에 상관없이, 많아야 오직 한 줄의 입력을 반환할 뿐이다.

정규입력 모드에서, 운영체제는 입력 편집 도구를 제공한다: ERASE 와 KILL 문자들은 텍스트의 현재의 줄에서 편집 명령을 수행하기 위해 특별하게 해석되어진다.

상수 _POSIX_MAX_CANON 과 MAX_CANON는 정규 입력의 한 줄에 나타낼 수 있는 최대 바이트 수를 한정한다. 27. 6절 [Limits for Files] 참조.

비정규입력(noncanonical input) 프로세싱 모드에서, 문자들은 라인들로 묶여지지 않고, ERASE 와 KILL 프로세싱은 수행되지 않는다. 비정규입력에서 읽혀진 바이트들은 MIN 과 TIME을 설정함으로 인해서 제어된다. 12. 4. 10절 [Noncanonical Input] 참조.

대부분의 프로그램들이 정규입력을 사용하는 것은, 정규입력의 방법이 사용자에게 라인으로 입력을 편집할 수 있는 방법을 제공하기 때문이다. 비정규입력을 사용하는 보통의 이유는 프로그램이 단일-문자 명령들을 받아들이거나 또는 프로그램 자체가 편집도구를 제공할 때 사용된다.

정규 혹은 비정규의 선택은 구조체 struct termios의 멤버인 c_lflag에 ICANON 플로그에 의해 제어된다. 12. 4. 7절 [Local Modes] 참조.

---

12. 4 터미널 모드

이 절은 어떻게 입력과 출력이 수행되어지는지를 제어하는 다양한 터미널 속성들은 설명하고 있다. 이곳에서 설명한 함수, 자료구조, 그리고 기호 상수들은 모두 헤더파일 'termios. h'에 선언되어 있다.

12. 4. 1 터미널 모드 데이터 타입들

터미널 속성의 전부는 구조체 struct termios 에 저장되어 있다. 이 구조체는 속성들을 읽고, 설정하기 위한 함수 tcgetattr 과 tcsetattr에서 사용된다.

데이터 타입 : struct termios

터미널의 모든 입출력 속성을 기록하는 구조체. 이 구조체는 적어도 다음과 같은 멤버들을 포함하고 있다.

tcflag_t c_iflag

입력 모드를 위한 플래그들을 정하는 비트마스크; 12. 4. 4절 [Input Modes] 참조.

tcflag_t c_oflag

출력모드를 위해 플래그들을 정하는 비트마스크; 12. 4. 5절 [Output Modes] 참조.

tcflag_t c_cflag

제어모드를 위해 플래그들을 정하는 비트마스크; 12. 4. 6절 [Control Modes] 참조.

tcflag_t c_lflag

로컬모드를 위해 플래그들은 정하는 비트마스크; 12. 4. 7절 [Local Modes] 참조.

cc_t c_cc[NCCS]

다양한 제어 함수들과 연관된 문자들을 정하는 배열; 12. 4. 9절 [Special Characters] 참조.

구조체 struct termios 는 또한 입력과 출력 전송 속도를 부호화(encode)한 멤버들을 갖고 있지만, 아직 설명하지 않았다. 12. 4. 8절 [Line Speed]를 참조로 해서 어떻게 속도 값을 시험하고 저장하는지를 살펴보라.

다음절에서 구조체 struct termios 의 멤버들에 대한 자세한 설명을 할 것이다.

데이터 타입 : tcflag__t

unsigned integer 형으로, 터미널플래그들을 위한 다양한 비트마스크를 나타내는데 사용한다.

데이터 타입 : cc__t

unsigned integer 형으로 다양한 터미널 제어 함수들과 연관된 문자들을 나타내기 위해 사용한다.

매크로 : int NCCS

이 매크로의 값은 c_cc 배열에 있는 요소들의 개수이다.

12. 4. 2 터미널 모드 함수들

함수 : int tcgetattr(int filedes, struct termios *termios_p)

이 함수는 파일기술자 filedes와 연관된 터미널 디바이스의 속성을 시험하는데 사용된다. 그 속성은 구조체 termios_p가 가리키는 곳으로 반환된다.
만일 성공하면, tcgetattr 은 0을 반환하고, 실패하면 -1을 반환한다.
다음의 errno는 이 함수를 위해 정의된 에러상황이다.

EBADF : filedes 인수가 유용한 파일기술자가 아니다.

ENOTTY : filedes 가 터미널과 연관이 없다.

함수 : int tcsetattr(int filedes, int when, const struct termios *termios_p)

이 함수는 파일기술자 filedes와 연관된 터미널 디바이스의 속성을 설정한다. 새로운 속성들은 termios_p가 가리키고 있는 구조체로부터 가져온다.
when 인수는 이미 큐된(큐에 저장되어 있는) 입력과 출력을 어떻게 취급할 것인지를 정하는 것으로 다음 값들중 하나를 사용할 수 있다.

TCSANOW : 즉시 속성을 변경시켜라.

TCSADRAIN

큐에 저장된 출력이 쓰여질 때까지 기다린 후에 속성을 변경하라. 당신은 변경하는 파라미터가 출력에 영향을 미칠 때 이 옵션을 사용한다.

TCSAFLUSH : 이것은 TCSADRAIN과 같지만, 큐에 저장된 입력을 버린다.

TCSASOFT

이것은 위에 있는 어떤 것과도 덧붙여 사용할 수 있는 플래그 비트이다. 이것은 터미널 하드웨어에 대한 상황의 변경을 금지하기 위한 것이다. 이것은 BSD 확장이다; BSD가 아닌 시스템에서는 아무런 영향을 받지 않는다.

만일 이 함수가 터미널을 제어하고 있는 배경 프로세스로부터 호출된다면, 보통 프로세스 그룹 안에 있는 모든 프로세스들은 터미널에 쓰기를 시도하는 프로세스가 있을 때와 같은 방법으로, SIGTTOU 시그널을 보낸다. 만일 함수를 호출한 프로세스 자신이 SIGTTOU 신호를 무시하거나 블록하고 있다면 이 명령은 수행되어 지고, 아무런 신호도 보내지 않는다. 24장 [Job control] 참조. 만일 성공하면,
tcsetattr 은 0을 반환하고, 실패하면 -1을 반환한다. 다음의 errno는 이 함수를 위해 정의된 에러 상황이다.

EBADF : filedes 인수는 유용한 파일기술자가 아니다.

ENOTTY : filedes는 터미널과 아무런 연관이 없다.

EINVAL : when 인수값이 유용하지 않거나, termios_p인수에 잘못된 데이터가 잘못되었거나.

tcgetattr 과 tcsetattr 이 filedes를 터미널과 연관된 파일기술자로 정했다고 할지라도, 그 속성은 파일기술자의 것이 아니라 터미널 디바이스 그 자체의 속성이다. 이것은 변경한 터미널 디바이스의 속성들이 불변함을 의미한다; 만일 나중에 다른 프로세스가 터미널 파일을 개방하면, 그것은 심지어 개방한 파일 기술자로 아무 일도 하지 못할지라도 변경된 속성들을 보일 것이다. 유사하게, 만일 단일 프로세스가 동일한 터미널 디바이스에 다중 또는 복제된 파일기술자들을 가졌다면, 터미널 속성 변경은 이 파일기술자 모두의 입력과 출력에 영향을 미친다. 이 의미는, 예를 들어, 당신은 한 개의 파일 기술자만을 개방할 수 없거나, 또는 반향 모드(echoed mode)로 보통의 라인 버퍼의 입력을 터미널로부터 읽기 위한 스트림을 개방할 수 없다; 그리고 동시에 당신은 비반향 모드로 동일한 터미널로부터 단일 문자를 읽기 위해 사용되는 다른 파일기술자를 가진다. 대신에 당신은, 두 개의 모드 사이에 어떤 것이 전면이고, 어떤 것이 후면인지 정확히 해야한다.

12. 4. 3 적당하게 터미널 모드 설정하기

당신이 터미널 모드를 설정할 때, 당신은 첫째로 특별한 터미널 디바이스의 현재의 모드를 얻기 위해 tcgetattr을 호출하고, 그 다음 실제로 당신이 관심을 가진 모드들을 갱신하고, tcseattr에 그 결과를 저장한다.

속성들의 설정을 선택하기 위해 구조체 struct termios를 간단히 초기화하고 직접적으로 tcsetatt에 인수로 주는 것은 나쁜 방법이다. 당신의 프로그램은 이 매뉴얼에 문서화되지 않은 멤버들을 지원하는 시스템에서 지금으로부터 수년 후에 실행되어질지도 모른다. 이해할 수 없는 값들로 이들 멤버들은 설정하는 것을 피하기 위한 방법은 그들을 변경하는 것을 피하는 것이다. 즉, 다른 터미널 디바이스는 적당하게 설정된 다른 모드를 필요로 한다. 그래서 당신은 하나의 터미널 디바이스로부터 다른 터미널 디바이스로 그냥 속성들을 카피하는 것은 피해야한다.

한 멤버가 c_iflag, c_oflag 그리고 c_cflag 등의 독립적 플래그들의 집합일 때, 그들 전체 멤버들을 설정하려하는 것은 잘못된 생각이다, 왜냐하면, 특별한 운영체제는 그들 자신만의 플래그들을 갖고 있기 때문이다. 대신에, 당신은 멤버의 현재의 값들을 얻어서, 당신의 프로그램에서 다룰 수 있는 플래그만 변경하고, 다른 플래그들을 변경하지 않도록 해라. 이곳에 구조체 struct termios의 다른 데이터를 그대로 유지하고, 오직 한 플래그(ISTRIP)만을 어떻게 변경하는지에 대한 예가 있다.

```
int
set_istrip (int desc, int value)
{

        struct termios settings;
        if (tcgetattr (desc, &settings) < 0) {

                perror ("error in tcgetattr");
                return 0;

        }
        settings. c_iflag &= ~ISTRIP;
        if (value)

                settings. c_iflag |= ISTRIP;

        if (tcgetattr (desc, &settings) < 0) {

                perror ("error in tcgetattr");
                return;

        }
        return 1;

}
```

12. 4. 4 입력 모드들

이 절은 입력 프로세싱의 저수준 관점을 완전히 제어하는 터미널 속성 플래그들은 설명하고 있다: 에러 핸들링, 멈춤 신호들, 제어 흐름, 그리고 RET 와 LFD 문자들.

이들 플래그 모드는 구조체 struct termios 의 c_iflag안의 비트들이다. 그 멤버는 정수이고, 당신은 오퍼레이터 &, | 그리고 ^ 를 사용해서 플래그들은 변경할 수 있다. c_iflag_instead를 위한 전체 값을 정하여 시도하지 말고, 나머지는 손대지 말고 남겨둬라. (12. 4. 3절 [Setting Modes] 참조.)

INPCK

만일 이 비트가 설정되면, 입력 패리티 체크가 가능하다. 만일 설정되지 않으면, 입력에서 패리티 에러가 났는지 체

크하지 않는다; 즉 그 문자들은 그 어플리케이션에 간단히 주어진다. 입력 프로세싱에서 패리티 체크는 패리티 검출의 여부에 독립하여 있고, 터미널 하드웨어에서의 패리티 발생은 가능하다; 12. 4. 6절 [Control Modes] 참조. 예를 들어, 당신은 INPCK 입력 모드 플래그를 소거하고 입력에서 패리티 에러들은 버리기 위해 PARENB 제어 모드 플래그를 설정할 수 있지만, 여전히 출력에서 패리티 에러는 발생한다.
만일 이 비트가 설정되면, 패리티 에러가 발생했을 때 IGNPAR 이나 PARMRK 비트가 설정되었는지에 의존하여 무슨 일이 발생한다. 이 비트를 설정하지 않는다면, 패리티 에러를 위한 한 바이트를 '\0'으로 응용프로그램에 주어야 한다.

IGNPAR

만일 이 비트가 설정되면, 구성(framing) 이나 패리티 에러를 위한 바이트가 무시되어진다. 이것은 INPCK 가 설정되어야지 만 유용하다.

PARMRK

만일 이 비트가 설정되면, 패리티나 구성 에러가 있는 입력 바이트들이 프로그램에 표시된다. 이 비트는 INPCK 가 설정되고 IGNPAR이 설정되지 않았을 때 유효하다. 잘못된 바이트를 표시하는 방법은 두 개의 선행 바이트, 377과 0을 사용하는 것이다. 그래서, 프로그램은 터미널로부터 받은 잘못된 한 개의 바이트로부터 세 개의 바이트들을 읽게 되는 것이다. 만일 유용한 바이트가 0377의 값을 가지고, ISTRIP(밑을 보라.) 이 설정되지 않는다면, 그 프로그램은 그것을 패리티 에러 표시와 혼동할 것이다. 그래서 유용한 바이트 0377은 두 개의 바이트로 프로그램에 주어진다, 즉 이 경우에는 0377 0377로. . .

ISTRIP

만일 이 비트가 설정되면, 유용한 입력 바이트들이 일곱 비트로 구성되어 있다; 그렇지 않다면, 여덟 비트 모두가 읽혀서 프로그램에서 사용된다.

IGNBRK

만일 이 비트가 설정되면, 멈춤(break)의 상황이 무시된다.
멈춤(break) 상황은 한 바이트보다 긴 0-값을 가진 비트들의 열들을 비동기적 직렬 데이터 전송의 방법으로 정의된다.

BRKINT

만일 이 비트가 설정되고, IGNBRK 가 설정되지 않는다면, 멈춤(break)의 상황은 터미널 입력과 출력의 큐를 소거하고, 터미널과 연관있는 전면 프로세스 그룹을 위해서 SIGINT 신호를 발생한다. 만일 BRKINT 나 IGNBRK 가 설정되지 않았다면, 멈춤(break) 상황은 만일 PARMRK가 설정되지 않으면, 단일 문자 '\0'로 응용프로그램에 주어진다, 그렇지 않으면, 세 개의 문자열 '\377', '\0', '\0'로 주어진다.

IGNCR

만일 이 비트가 설정되면, 캐리지반환 문자('\r')는 입력에서 버려진다. 버려진 캐리지반환은 당신이 RET 키를 칠 때 캐리지반환과 라인피드(linefeed) 이 둘을 보낸 터미널에서 유용하게 될 것이다.

ICRNL

만일 이 비트가 설정되고 IGNCR 이 설정되지 않으면, 입력으로 받은 캐리지반환문자를('\r') 새줄문자('\n')로해서 응용프로그램에 주어진다.

INLCR

만일 이 비트가 설정되면, 입력으로 받은 새줄문자('\n')는 캐리지반환문자('\r')로 응용프로그램에 주어진다.

IXOFF

만일 이 비트가 설정되면, 입력의 정지/시작의 제어가 가능하다. 즉, 컴퓨터가, 프로그램이 처리하는 속도보다 더 빠르게 데이터가 도착하는 것을 방지하기 위해 STOP과 START 문자들을 보내는 것이다. 입력 데이터를 생성하는 실제 터미널 하드웨어에서 STOP 문자에 응답하여 전송을 중단하고, START 문자에 응답하여 다시 전송을 재개한다. 12. 4. 9. 4절 [Strat/Stop Characters] 참조.

IXON

만일 이 비트가 설정되면, 출력의 시작/정지의 제어가 가능하다. 즉, 만일 컴퓨터가 STOP 문자를 받으면, START 문자를 받을 때까지 출력을 중단한다. 이 경우, STOP 과 START 문자들은 결코 응용프로그램에 주어지는 것이 아니다. 만일 이 비트가 설정되지 않았다면 START와 STOP는 원래의 문자들로 읽혀진다. 12. 4. 9. 4절 [Start/Stop Characters] 참조.

IXANY

만일 이 비트가 설정되면, 출력이 STOP를 통해 정지되어져 있을 때, 어느 문자를 가지고도 출력을 재개할 수 있다. 즉, 오직 START 문자만 출력을 재개하는 것이 아니라는 것이다.

IMAXBEL

만일 이 비트가 설정되면, 벨이 울리도록 터미널에 BEL문자(code 007)를 보내어 터미널의 입력 버퍼를 가득 채운다.

12. 4. 5 출력 모드들

이 절은 출력 문자들을 어떻게 해석하고 화면을 채울 것인지를 제어하는 터미널 플래그와 필드들에 대해 설명한다. 이들 모두는 구조체 struct termios의 c_oflag 멤버에 들어있다.

c_oflag 멤버 자신은 정수형이고, 당신은 그 플래그들과 필드들을 오퍼레이터 &, |, 그리고 ^를 사용해서 갱신할 수 있다. c_oflag의 전체의 값을 변경하려 시도하지 말고_대신에 오직 정해진 하나의 플래그만 변경하고 나머지는 손대지 말고 남겨둬라 (12. 4. 3절 [Setting Modes] 참조.

매크로 : int OPOST

이 비트가 설정되면, 출력 데이터는 터미널 디바이스에 적당하게 표시되는 그런 특별히 정해지지 않은 방법으로 처리된다. 이것은 새줄문자 ('\n')를 캐리지반환과 라인피드의 쌍으로 대치시켜 포함한다. 만일 이 비트가 설정되지 않는다면, 그 문자들은 그대로 전송된다.

다음 세 개의 비트들은 BSD를 위한 것으로, BSD 가 아닌 시스템에는 아무런 영향을 주지 않는다. 모든 시스템에서, 그들은 OPOST가 설정 되어있어야 효과를 발휘한다.

매크로 : int ONLCR

만일 이 비트가 설정되면, 출력에 나타난 새줄문자를 문자의 쌍(pair)인 캐리지반환과 라인피드로 변환한다.

매크로 : int OXTABS

만일 이 비트가 설정되면, 출력에 나타난 탭 문자들을 8칼럼의 탭을 구현하는 적당한 공백으로 변환한다.

매크로 : int ONOEOT

만일 이 비트가 설정되면, 출력에 나타나는 C-d 문자(code 004)들을 버린다. 이들 문자들은 dial-up 터미널의 연결을 단절시키기 때문이다.

12. 4. 6 제어 모드들

이 절은 비동기 직렬 데이터 전송에 관계된 제어 파라미터인 터미널 플래그와 필드들을 설명한다. 이들 플래그들은 터미널 포트(네트웍에 연결된 가상-터미널처럼)의 다른 종류에는 통하지 않을지도 모른다. 이들 모두는 구조체 struct termios의 c_cflag 멤버에 존재한다.

c_cflag 멤버 자체는 정수형으로, 당신은 그 플래그와 필드들을 오퍼레이터 &, |, 그리고 ^ 을 사용해서 갱신할 수 있다. c_cflag의 전체 값들을 정하려 시도하지 말고, 대신에 오직 정해진 플래그들만 변경하고, 나머지는 손대지 말고 남겨둬라 (12. 4. 3절 [Setting Modes] 참조.)

CLOCAL

만일 이 비트가 설정되면, 터미널이 "국부적으로" 연결되어 있고, 모뎀 상태 라인들은(캐리어 검출과 같은)무시됨을 의미한다. 만일 이 비트가 설정되지 않고 당신이 O_NONOBLOCK 플래그를 설정하지 않고 open을 호출하면, open은 모뎀이 연결될 때까지 블록 된다.
만일 이 비트가 설정되지 않고, 모뎀 연결이 끊어지면, SIGHUP 신호를 터미널(만일 그것이 하나를 가진다면)을 위한 제어 프로세스 그룹에 보낸다. 보통, 이것은 탈출(exit) 하려는 프로세스에 때문이다; 21장 [Signal Handling] 참조. 연결이 끊어진 후에 터미널로부터 읽기는 파일끝 상황을 발생하고, 쓰기는 반환될 EIO 에러를 발생한다. 터미널 디바이스는 반드시 닫혀져야 하고, 그 상황을 소거하기 위해 재개방되어져야 한다.

HUPCL

만일 이 비트가 설정되면, 모든 프로세스가 폐쇄된 파일인 터미널 디바이스를 갖거나, 빠져나갈 때 모뎀 단절(disconnect)이 발생한다.

CREAD

만일 이 비트가 설정되면, 입력이 터미널로부터 읽혀질 수 있다. 그렇지 않다면, 입력은 그것이 도착했을 때 버려진다.

CSTOPB

만일 이 비트가 설정되면, 두 개의 stop 비트가 사용된다. 그렇지 않다면 오직 한 개의 stop 비트가 사용된다.

PARENB

만일 이 비트가 설정되면, 패리티 비트의 생성과 검출이 가능하게 된다. 12. 4. 4절 [Input Modes] 를 참조로 입력 패리티 에러가 어떻게 다루어지는지를 보아라.
만일 이 비트가 설정되지 않으면, 출력 문자들에 패리티 비트가 더해지지 않고, 입력 문자들에서는 패리티에러를 체크하지 않는다.

PARODD

이 비트는 PARENB가 설정됐을 때만 유용하다. 만일 PARODD가 설정되면, 홀수 패리티가 사용되고, 그렇지 않으면 짝수 패리티가 사용된다.
제어 모드 플래그들은 문자당 비트들의 개수를 나타내는 필드를 갖고 있다. 당신은 그 값을 추출하기 위해서 CSIZE 매크로를 사용할 수 있다. 이처럼 : settings. c_cflag & CSIZE

CSIZE : 이것은 문자당 비트들의 개수를 위한 마스크이다.

CS5 : 바이트당 다섯 비트를 정한다.

CS6 : 바이트당 여섯 비트를 정한다.

CS7 : 바이트당 일곱 비트들을 정한다.

CS8 : 바이트당 여덟 비트들을 정한다.

CCTS_OFLOW

이 비트가 설정되면, CTS와이어(RS232 프로토콜)에 기반한 출력의 흐름제어가 가능하다.

CRTS_IFLOW

이 비트가 설정되면, RTS 와이어(RS232 프로토콜)에 기반한 입력의 흐름제어가 가능하다.

MDMBUF

이 비트가 설정되면, 출력의 캐리어-기반 흐름제어가 가능하다.

12. 4. 7 국소 모드들

이 절은 구조체 struct termios 의 c_lflag 멤버의 플래그들을 설명한다. 이들 플래그들은 일반적으로 12. 4. 4절 [Input Modes] 에서 설명한, 모드 플래그들보다는 반향, 신호들, 그리고 정규와 비정규입력의 선택 등과 같은 입력 프로세싱의 고-수준 관점을 제어한다.

c_flag 멤버 그 자체는 정수형이고, 당신은 오퍼레이터 &, |, 그리고 ^를 사용해서 그 플래그들과 필드를 변경할 수 있다. c_lflag의 전체 값들을 정하려 시도하지 말고 대신에, 정해진 플래그들만 변경하고, 다른 나머지 것들에는 손대지 말라.

ICANON

이 비트가 설정되면, 정규입력 프로세싱 모드로 된다. 그렇지 않다면, 입력은 비정규 모드로 처리된다. 12. 3절 [Canonical or Not] 참조.

ECHO : 이 비트가 설정되면, 입력문자가 반향된다.

ECHOE

이 비트가 설정되면, 스크린의 현재의 라인에 있는 마지막 문자를 지우는 역할을 하는 ERASE 문자를 사용 할 때 그

마지막 문자를 화면상에서 실제로 없앰으로서 사용자에게 문자가 실제로 지워졌음을 확인시킨다. 그렇지 않다면 지워진 문자는 다시 반향된다(프린팅 터미널에 적당한). 이 비트는 오직 앞에 표시된 것만을 제어한다. ECHOE를 사용하지 않고도, ICANON 비트는 ERASE문자를 실제로 인식하고 입력을 지우는 것을 그 자체에서 제어하고 있다.

ECHOK

이 비트는 KILL 문자에 대한 표시를 가능하게 한다. 이것을 할 수 있는 두 가지 방법이 있다. KILL 문자가 눌려진 곳의 전체 라인을 화면에서 지우는 것이 좀더 나은 방법이다. 좀더 안 좋은 방법은 KILL 문자를 반향한 후에 새줄로 옮기는 것이다. 어떤 시스템은 한가지를 허용하고, 어떤 시스템은 다른 하나를 허용하고, 어떤 것은 당신에게 두가지 중 하나를 선택하도록 허용한다. 이 비트가 설정되지 않으면, KILL 문자는 KILL 문자가 존재하지 않는 것처럼 단지 그것을 반향한다. 그러면 전의 입력을 지웠던 KILL 문자를 기억하는 것은 사용자의 몫이다; 화면에서 아무런 표시가 없기 때문이다. 이 비트는 오직 전의 표시된 것을 제어한다. ICANON 비트는 그 자체로 KILL 문자를 실제로 인식하고, 입력을 지운다.

ECHONL

이 비트가 설정되고 ICANON 비트가 또한 설정되면 새줄 ('\n') 문자는 심지어 ECHO 비트가 설정되지 않았을 때도 반향된다.

ISIG

INTR, QUIT, 그리고 SUSP 문자들을 인식하는지의 여부에 대한 제어 비트이다. 이 문자들과 연관된 함수들은 이 비트가 설정됐을 때만 수행된다. 규정이나 비규정 입력에 대한 것은 이들 문자들을 해석하는데 아무런 영향이 없다. 당신은 이들 문자들의 인식을 불가능할 때 주의해서 사용해야한다. 사용자에 의해 인터럽트될 수 없는 프로그램은 사용자에게 매우 불친절한 것이다. 만일 당신이 이 비트를 소거하면, 당신의 프로그램은 사용자에게 이들 문자들과 연관있는 신호를 보내도록 허용하거나, 그 프로그램으로부터 탈출하는 인터페이스를 제공해야한다. 12. 4. 9절 [Signal Characters] 참조.

IEXTEN

이 비트는 ISIG와 유사하지만, 특별한 문자들에 정의된 제어실행에 대한 것이다. 만일 이 비트가 설정되면, ICANON과 ISIG 국소 모드 플래그와 IXON과 IXOFF 입력모드 플래그를 위한 디폴트 동작을 무효로 할 것이다.

NOFLSH

보통, INTR, QUIT, 그리고 SUSP 문자들은 터미널의 입력과 출력큐를 소거하게 한다. 만일 이 비트가 설정되면 그 큐들은 소거되지 않는다.

TOSTOP

이 비트가 설정되고 시스템이 작업 제어를 지원하면, SIGTTOU신호가 터미널에 쓰기를 시도하는 배경 프로세스에 의해 발생되어진다. 24. 4절 [Access to the Terminal] 참조.

다음 비트들은 BSD 확장이다. GNU 라이브러리는 당신이 그들을 요청하는 어느 시스템 상에서도 사용하도록 이들 심볼들을 정의했지만, BSD 시스템과 GNU시스템을 제외한 다른 시스템에는 아무런 영향을 미치지 않는 비트 설정이다.

ECHOKE

BSD 시스템에서, 이 비트는 ECHOK를 설정할 때, KILL 문자를 표시하는 두 가지 방법중 하나를 선택하게 한다. 만일 ECHOKE 가 설정되면, KILL문자는 화면에서 라인 전체를 지우고 그렇지 않으면 KILL문자는 화면의 다음 라인으로 옮긴다. ECHOKE의 설정은 ECHOK가 설정됐을 때만 유효하다.

ECHOPRT

이 비트는 하드카피 터미널을 조정하는 방법으로 ERASE 문자의 표시를 가능하게 한다.

ECHOCTL

만일 이 비트가 설정되면, 해당하는 텍스트 문자가 따르는 control문자를 '^'표시로 반향한다. 그래서 control-A 는 '^A'처럼 나타난다.

ALTWERASE

이 비트는 WERASE 문자가 지울 것인지 결정한다. 단어의 시작점에서 뒤쪽으로 문자를 지운다. 어디에서 단어를 시작할 것인지 의문이 생긴다. 만일 이 비트가 설정되면, 단어의 시작점은 공백문자 다음의 비공백 문자가 된다. 만일 이 비트가 설정되지 않으면, 단어의 시작점은 영숫자 문자이거나, 또는 그들이 없는 문자 다음의 underscore이다.

FLUSHO

이 비트는 사용자가 DISCARD 문자를 입력할 때 토글 된다. 이 비트가 설정되어 있는 동안에, 모든 출력은 버려진다. 12. 4. 9. 5절 [Othr Special] 참조.

NOKERNINFO

이 비트의 설정은 STATUS 문자의 처리가 불가능하도록 한다. 12. 4. 9. 5절 [Other Special] 참조.

PENDIN

이 비트가 설정되면, 다시 프린트할 필요가 있는 입력 라인이 있음을 알린다. REPRINT문자를 치면 이 비트가 설정된다. 그 비트는 재프린트가 종료될 때까지 설정상태가 지속된다. 12. 4. 9. 2절 [BSD Editing] 참조.

12. 4. 8 라인 속도

터미널 라인 속도는 터미널 상에서 얼마나 빨리 데이터를 읽고 쓸 수 있는지를 컴퓨터에게 알린다. 만일 터미널이 직렬 라인으로 연결되면, 터미널 속도를 그 라인의 실제 속도에 맞도록 지정해야지, 터미널 자신의 임의대로 그 속도를 지정할 수 없고, 만약 그렇게 했다면 통신은 단절된다. 실제 직렬 포트는 오직 어떤 표준 속도만을 받아들인다. 어떤 특별한 하드웨어는 심지어 모든 표준 속도를 지원하지 않을 것이다. 제로(zero)로 속도를 정하는 것은 연결된 상태를 끊고 모뎀 제어 신호를 끄는 것이다.

만일 터미널이 직렬 라인이 아니라면(예를 들어, 그것이 네트웍 연결이라면), 라인의 속도는 실제 데이터 전송속도에 영향을 받지 않지만, 어떤 프로그램에서는 필요한 채워넣기(padding)의 양을 결정하기 위해 그것을 사용할 것이다. 그것은 실제 터미널의 속도에 대응되는 라인 속도 값을 정하기에 가장 좋지만, 당신은 채워넣기(padding)의 다양한 양을 다양한 값으로 안전하게 시험해야 한다.

각 터미널을 위한 두 개의 라인 속도가 있는데, 입력을 위한 것과 출력을 위한 것이다. 당신은 그들을 독립적으로 정할 수 있지만, 대부분의 터미널에서는 동일한 속도를 사용한다.

속도 값은 구조체 struct termios 에 저장되어 있지만, 직접적으로 구조체 struct termios안에 있는 그들을 직접적으로 억세스 하려 시도하지 말라. 대신에, 당신은 그들을 읽고 저장하기 위해서 다음 함수를 사용하라.

함수 : speed_t cfgetospeed(const termios *termios_p)

이 함수는 구조체 *termios`p안에 저장되어 있는 출력 라인 속도를 반환한다.

함수 : speed_t cfgetispeed (const struct termios *termios_p)

이 함수는 구조체 *termios`p안에 저장되어 있는 입력 라인 속도를 반환한다.

함수 : int cfsetospeed (struct termios *termios_p, speed_t speed)

이 함수는 출력 속도로 *termios`p 에 speed를 반환한다. 보통의 반환 값은 0이고, 에러가 발생하면 -1을 반환한다. 만일 speed가 속도 값이 아니면, cfsetospeed는 -1을 반환한다.

함수 : int cfsetispeed (struct termios *termios_p, speed_t speed)

이 함수는 입력 속도로 *termios`p에 speed를 저장한다. 보통의 반환 값은 0이고 에러가 발생하면 -1을 반환한다. 만일 speed가 속도 값이 아니라면, cfsetopeeed는 -1을 반환한다.

함수 : int cfsetspeed (struct termios *termios_p, speed_t speed)

이 함수는 입력과 출력의 속도 둘을 위해 *termios`p로 speed를 저장한다. 보통 반환 값은 0이고, 에러가 발생하면 -1을 반환한다. 만일 speed가 속도 값이 아니라면, cfsetspeed는 -1을 반환한다. 이 함수는 4. 4 BSD 확장이다.

데이터타입: speed_t

speed_type은 unsigned integer형으로 라인의 속도를 나타내기 위해 사용된다.

cfsetospeed 와 cfsetispeed 함수는 그 시스템이 간단히 취급할 수 없는 속도 값일 경우에만 에러를 표시한다. 만일 당신이 기본적으로 받아들일 수 있는 속도 값을 지정하면, 이 함수는 성공할 것이다. 그러나 그들은 어떤 특별한 하드웨어가 정해진 속도를 실제로 지원할 수 있는지를 체크할 수 없고, 실제로 그들은 당신이 속도를 설정하려 계획하는 디바이스를 알지 못한다. 만일 당신이 처리될 수 없는 값으로 특별한 디바이스의 속도를 설정하려고 tcsetattr을 사용하면, tcsetattr은 -1을 반환한다.

이식성 노트: GNU 라이브러리에서, 함수들은 입력과 출력으로 초(second)당 비트로 계산된 속도를 받아들인다. 다른 라이브러리들은 속도를 특별한 코드로 지정해준다. POSIX. 1과 이식성을 위해서는, 당신은 속도를 나타내기 위한 다음의 심볼들 중의 하나를 사용해야만 한다; 그들의 정확한 숫자적 값들은 시스템_의존적이지만, 각 이름은 정해진 의미를 갖고 있다. B110은 110 bps를 위한 것이고, B300은 300 bps를 위한 것이고. . 등등. . 이것은 속도를 나타내기 위한 다른 방법들과 전혀 이식성이 없지만, 그들은 특별한 직렬 라인을 지원할 수 있는 속도이다.

B0 B50 B75 B110 B134 B150 B200

B300 B600 B1200 B1800 B2400 B4800

B9600 B19200 B38400

BSD는 유사이름으로 두 개의 부가적 속도 심볼들은 정의한다. EXTA 는 B19200과 같고, EXTB는 B38400과 같다. 이들 유사어는 오래된 것이다.

함수 : int cfmakeraw (struct termios *termios_p)

이 함수는 BSD에서 전통적으로 원래의 모드"로 불려지는 모드로 *termios`p를 설정하는 쉬운 방법을 제공한다. 그것은 정확히 이런 일을 한다.

```
termios_p->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP
|INLCR|IGNCR|ICRNL|IXON);
termios_p->c_oflag &= ~OPOST;
termios_p->c_lflag &= ~(ECHO|ECHONL|ICANON|ISIG|IEXTEN);
termios_p->c_cflag &= ~(CSIZE|PARENB);
termios_p->c_cflag |= CS8;
```

12. 4. 9 특별 문자들

정규입력에서, 터미널 구동기는 다양한 제어 함수들을 수행하는 특별문자들의 수를 인식한다. 이들은 편집입력을 위한 ERASE 문자(보통은 DEL)와 그리고 다른 편집문자들이 포함된다. SIGINT 신호를 보내기 위한 INTR 문자(보통 C -c)와 다른 신호-발생 문자들은, 다른 정규입력이나 비정규 입력 모드에서 유용할 것이다. 모든 이들 문자들은 이 절에 설명되어 있다.

특별한 문자들은 구조체 struct termios의 c_cc 멤버에 정해져있다. 이 멤버는 배열이다; 각 요소는 특별한 규칙을 가진 문자를 정한다. 각 요소는 그 요소의 인덱스를 위한 심볼 상수를 갖는다_예를 들어 INTR은 INTR 문자를 정하는 요소의 인덱스이다, 그래서 INTR 문자로 '='을 정하여 termios. c_cc[INTR]에 '='을 저장한다.

다른 시스템에서, 당신은 _POSIX_VDISABLE를 정하여 특별 문자 함수들을 불가능하게 할 수 있다. 이 값은 어느 가능한 문자 코드와 같지 않다. 27. 2절 [Options for Files] 를 참고로, 운영체제가 _POSIX_VDISABLE를 당신에게 지원하는지의 여부를 어떻게 알 수 있는지를 보아라.

12. 4. 9. 1 입력 편집을 위한 문자들

이들 특별 문자들은 오직 정규(canonical) 입력 모드에서만 작동된다. 12. 3절 [Canonical or Not] 참조.

매크로 : int VEOF

이것은 특별 제어 문자 배열에 있는 EOF 문자를 위한 첨자(subscript)이다. termios. c_cc[VEOF] 는 그 문자 자체를 저장하고 있다. EOF 문자는 오직 정규입력 모드에서만 인식된다. 그 문자는 새줄문자와 같은 방법으로 라인의 종료자(terminator)로서 동작하지만, 만일 EOF 문자가 라인의 처음에 존재된다면, 0바이트를 반환하여, 파일의 끝임을 지적한다. EOF 문자 그 자체는 버려진다. 보통, EOF 문자는 C-d 이다.

매크로 : int VEOL

이것은 특별 제어 문자배열에 있는 EOL 문자를 위한 첨자이다. termios. c_cc[VEOL] 은 문자 그 자체를 저장하고 있다. EOL 문자는 오직 정규입력 모드에서만 인식된다. 그것은 새줄문자처럼 라인 종료자(terminator) 로서 동작한다. 입력 라인에서 마지막 문자로 읽혀진 EOL문자는 버려진다. 한 라인의 끝인 RET을 만들기 위해서 EOL 문자를 사용할 필요가 없다. 단지 ICRNL 플래그를 설정하라. 실제로, 이것이 디폴트 상황이다.

매크로 : int VERASE

이것은 특별 제어문자 배열에 있는 ERASE문자를 위한 첨자이다. termios. c_cc[VERASE] 는 그 문자 자체를 저장한다. ERASE문자는 오직 정규입력 모드에서만 인식된다. 사용자가 erase 문자를 입력할 때, 전에 입력된 문자가 버려

진다. (만일 그 터미널이 다중 바이트 (multibyte)문자열을 발생시킨다면, 이것은 입력에서 버려진 것이 한 바이트 보다 더 많을 수도 있다.) 이것은 텍스트의 현재 라인보다 앞의 것을 지울 수 없다. ERASE 문자 그 자체는 버려진다. 보통 ERASE문자는 DEL 이다.

매크로 : int VKILL

이것은 특별 제어문자 배열에 있는 KILL 문자를 위한 첨자이다. termios. c_cc[VKILL] 은 문자 그 자체를 저장하고 있다. KILL 문자는 오직 정규입력 모드에서만 인식된다. 사용자가 kill 문자를 입력할 때, 입력의 현재라인의 전체의 내용이 버려진다. kill 문자도 버려진다. KILL 문자는 보통 C-u 이다.

12. 4. 9. 2 편집 문자의 BSD 확장

이들 특별 문자들은 오직 정규입력 모드에서만 동작한다. 12. 3절 [Canonical of Not] 참조. 그들은 BSD 확장이다; GNU 라이브러리는 그들을 요청하면, 어느 시스템 상에서든지 그 심볼들을 사용하게 하지만, 그 문자들은 BSD 시스템을 제외하고는 아무 데서도 동작하지 않을 것이다.

매크로 : int VEOL2

이것은 특별 제어문자 배열에 있는 EOL2 문자를 위한 첨자이다. termios. c_cc[VEOL2] 는 문자 그 자체를 저장한다. EOL2문자는 EOL문자처럼 동작하지만(위를 보라) 다른 문자가 될 수 있다. 그래서, 당신이 입력 라인을 끝내기 위해서 두 개의 문자들을 정할 수 있지만, 그들 중 하나를 위해서 EOL 문자를 설정하고 다른 것을 위해서 EOL2 문자를 설정하라.

매크로 : int VWERASE

이것은 특별 제어문자 배열에 있는 WERASE 문자를 위한 첨자이다. termios. c_cc[VWERSE] 는 문자 그 자체를 저장한다. WERASE 문자는 오직 정규입력 모드에서만 동작한다. 그것은 이전 입력의 전체 단어를 지운다.

매크로 : int VREPRINT

이것은 특별 제어문자 배열에 있는 REPRINT문자를 위한 첨자이다. termios. c_cc[BREPRINT]는 문자 그 자체를 저장한다. REPRINT 문자는 오직 정규입력 모드에서만 인식된다. 그것은 현재의 입력라인을 다시 프린트한다.

매크로 : int VLNEXT

이것은 특별 제어문자 배열에 있는 LNEXT 문자를 위한 첨자이다. termios. c_cc[VLNEXT]는 문자 그 자체를 저장한다. LNEXT 문자는 오직 IEXTEN이 설정되었을 때만 인식된다. 사용자가 입력한 다음문자의 편집을 불가능하게 한다. 이것은 Emacs에서 C-q 와 유사하다. "LNEXT"는 "literal next"를 나타낸다. LNEXT문자는 보통 C-v 이다.

12. 4. 9. 3 신호를 발생시키는 문자들

이들 특별문자들은 정규모드나 비정규모드를 상관하지 않고 동작하지만, ISIG 플래그가 설정되었을 때만 동작한다. (12. 4. 7절 [Local Modes] 참조.)

매크로 : int VINTR

이것은 특별 제어문자 배열에 있는 INTR 문자를 위한 첨자이다. termios. c_cc[VINTR] 은 문자 그 자체를 저장한다. INTR(interrupt)문자는 터미널과 연관된 전면 작업에 있는 모든 프로세스를 위해 SIGINT 신호를 발생시킨다. INTR 문자 그 자체는 버려진다. 21장 [Signal Handling] 를 참조로, 신호에 대한 정보를 보아라. 특별히, INTR 문자는 C-c 이다.

매크로 : int VQUIT

이것은 특별 제어문자 배열에 있는 QUIT문자를 위한 첨자이다. termios. c_cc[VQUIT] 는 문자 그 자체를 저장한다. QUIT 문자는 터미널과 연관된 전면작업에 있는 모든 프로세스를 위한 SIGQUIT 신호를 발생시킨다. QUIT 문자 그 자체는 버려진다. 21장[Signal Handling] 에서 신호에 대한 더 많은 정보를 참조하라. 특별히, QUIT 문자는 C-\이다.

매크로 : int VSUSP

이것은 특별 제어문자 배열에 있는 SUSP 문자를 위한 첨자이다. termios. c_cc[VSUSP]는 문자 그 자체를 저장한다. SUSP(suspend) 문자는 작업제어를 지원하는 동작에서만 인식된다(24장 [Job Control] 참조). 그것은 터미널과 연관된 전면 작업에 있는 모든 프로세스들에게 보내기 위한 SIGTSTP 신호를 발생시킨다. SUSP 문자 그 자체는 버려

진다. 신호에 대한 자세한 정보는 21장 [Signal Handling] 를 참조하라. 몇몇 응용프로그램에서는 SUSP 문자에 대한 해석을 불가능하게 한다. 만일 당신의 프로그램이 그렇게 한다면, 사용자가 그 작업을 멈출 수 있게 하기 위한 다른 메커니즘을 제공해야 할 것이다. 사용자가 이 메커니즘을 불렀을 때, 프로그램은 단지 그 프로세스 자신이 아닌, 프로세스의 프로세스그룹에게 SIGTSTP 신호를 보낸다. 21. 6. 2절 [Signaling Another Process] 참조.

매크로 : int VDSUSP

이것은 특별 제어문자 배열에 있는 DSUSP 문자를 위한 첨자이다. termios. c_cc[VDSUSP] 는 문자 그 자체를 저장한다. DSUSP(suspend) 문자는 작업 제어를 지원한 는 동작에서만 인식된다(24장 [Job Control] 참조). 그것은 SUSP 문자처럼 SIGTSTP 신호를 보내지만, 프로그램이 입력으로 그것을 읽으려 시도하는, 잘못된 행동을 취할 때 발생한다. 작업제어를 지원하는 모든 시스템에서 DSUSP가 지원되는 것이 아니라 오직 BSD 시스템에서만 지원된다. 신호에 대한 자세한 정보는 21장 [Signal Handling] 를 참조하라. 특별히, DSUSP 문자는 C-y 이다.

12. 4. 9. 4 흐름 제어를 위한 특별 문자들

이들 특별 문자들은 정규입력 모드나 비정규입력모드에 상관없이 동작되지만, 그들의 사용은 IXON과 IXOFF 플래그에 의해 제어된다. (12. 4. 4절 [Input Modes] 참조.)

매크로 : int VSTART

이것은 특별 제어문자 배열에서 START 문자를 위한 첨자이다. termios. c_cc[VSTART] 는 문자 그 자체를 저장한다. START 문자는 IXON 과 IXOFF 입력 모드를 지원하기 위해서 사용된다. 만일 IXON이 설정되고, START 문자를 받으면 보류된 출력을 다시 시작한다. 이때 START 문자는 버려진다. 만일 IXOFF 가 설정되면, 시스템은 터미널에 START 문자를 전송할 것이다. START 문자의 보통의 값은 C-q 이다. 당신이 무엇으로 정하든 지에 상관없이 하드웨어에 C-q로 정해져 있다면 당신은 이 값을 변경할 수 없을 것이다.

매크로 : int VSTOP

이것은 특별 제어문자 배열에 있는 STOP 문자를 위한 첨자이다. termios. c_cc[VSTOP] 는 문자 그 자체를 저장한다. STOP 문자는 IXON 과 IXOFF 입력 모드를 지워하기 위해 사용된다. 만일 IXON이 설정되고, STOP 문자를 받으면 출력을 보류시킨다; 이때 STOP 문자 자체는 버려진다. 만일 IXOFF 가 설정되면, 시스템은 입력큐에서 오버플로우가 발생하는 것을 방지하기 위해 터미널에 STOP 문자를 전송할 것이다. STOP를 위해서 보통 사용되는 값은 C-s 이다. 당신은 당신이 무엇으로 이 값을 변경하든 지에 상관없이 하드웨어에서 C-s로 고정되어 있다면, 이 값을 변경할 수 없다.

12. 4. 9. 5 다른 특별 문자들

이곳에서는 BSD 시스템에서 의미가 있는 두 개의 부가적 특별 문자들을 설명한다.

매크로 : int VDISCARD

이것은 특별 제어문자 배열에 있는 DISCARD 문자를 위한 첨자이다. termios. c_cc[VDISCARD] 는 문자 그 자체를 저장한다. DISCART문자는 IEXTEN이 설정되었을 때만 인식된다. 그 영향은 discard-output 플래그를 토글하게된다. 이 플래그가 설정되면, 모든 프로그램 출력은 버려진다. 플래그 설정은 또한 출력 버퍼안에 현재 존재하는 모든 출력을 버린다.

매크로 : int VSTATUS

이것은 특별 제어문자 배열에 있는 STATUS 문자를 위한 첨자이다. termios. c_cc[VSTATUS] 는 문자 그 자체를 저장한다. STATUS 문자의 영향은 현재 프로세스가 어떻게 동작하고 있는지에 대한 상황 메시지를 출력하기 위한 것이다. STATUS 문자는 오직 정규 입력 모드에서만 인식된다.

12. 4. 10 비정규입력

비정규입력 모드에서, ERASE 와 KILL 과 같은 특별 편집 문자들을 무시된다. 입력 편집을 위해 사용자에게 부여된 시스템 도구들이 비정규입력 모드에서는 불가능하게 된다. 그래서 모든 입력 문자들(만약 그들이 신호나 흐름제어 목적을 가진 것이 아니라면)은 응용 프로그램에 정확히 친 대로(typed) 인식된다. 응용 프로그램에서 사용자에게 입력을 편집하는 방법을 제공하는 것이 좋다.

비정규입력 모드는 유용한 입력이 있을 때까지 기다릴 것인지와 얼마나 기다릴 것인지를 제어하기 위한 MIN 과 TIME 이라 불리는 파라미터가 있다. 당신은 유용한 입력이 있을 때, 또는 없을 때, 즉시 반환되도록 해서 무작정 기다리는 것을 피하는 데도 그들을 사용할 수 있다.

MIN과 TIME은 구조체 struct termios의 멤버인 c_cc 배열의 요소로서 저장되어진다. 이 배열의 각 요소는 특별한 규칙을 가지고 있고, 각 요소는 그 요소의 인덱스로 대표되는 심볼 상수를 가지고 있다. VMIN 과 VMAX는 MIN과 TIME 슬롯의 배열의 인덱스들을 위한 이름들이다.

매크로 : int VMIN

이것은 c_cc 배열안의 MIN 슬롯을 위한 첨자이다. 그래서 termios. c_cc[VMIN]은 그 값 자체이다. MIN 슬롯은 비정규입력 모드에서만 유용하다; read가 반환되기 전에 입력큐에서 받아들여야만 하는 유용한 바이트의 최소 개수를 정하는데 사용된다.

매크로 : int VTIME

이것은 c_cc 배열에 있는 TIME 슬롯을 위한 첨자이다. 그래서, termios. c_cc[VTIME] 는 그 값 자체이다. TIME 슬롯은 비정규입력 모드에서만 유용하다; 그것은 0. 1초의 단위로 반환하기 전에 입력을 얼마나 기다릴 것인지를 정한다. MIN 과 TIME값은 read가 반환할 때를 위한 표준을 정하는데 영향을 미친다; 그들의 정확한 의미는 그들이 가진 값에 따른다(0이냐 0이 아니냐. .) 4개의 가능한 경우가 있다.

MIN 과 TIME 둘다 영일 때. . .

이 경우, read는 큐에 요청된 개수를 넘어서는, 유용한 입력이 있을 때 즉시 반환한다. 만일 아무런 입력이 없어도 즉시 반환하는데, 이때 read는 0의 값을 반환한다.

MIN 은 0이지만 TIME은 0이 아닐 때.

이 경우, read는 유용한 입력이 될 때까지 TIME 시간동안 기다린다; 단 한 개의 바이트도 요청한 read를 만족시키기에 충분하고, read는 반환한다. 그것이 반환할 때 요청된 개수의 유용한 문자를 반환한다. 만일 시간이 다할 때까지 유용한 입력이 없으면 read는 0을 반환한다.

TIME 이 0이지만 MIN은 0이 아닐 때.

이 경우, read는 적어도 큐에 유용한 입력이 MIN 바이트가 될 때까지 기다린다. 그 시간동안, read는 요청된 개수의 유용한 문자들을 반환한다. read는 만일 큐에 MIN보다 더 많은 문자가 발생했다면 MIN 문자보다 더 많은 문자를 반환할 수 있다.

TIME 과 MIN 둘다 영이 아닐 때.

이 경우, TIME은 만일 입력이 도착하면, 그 첫 번째 도착한 입력부터 얼마동안 기다릴 것인가를 정한다. read는 MIN 바이트의 입력이 도착하거나, 또는 TIME이 더 이상 아무런 입력 없이 경과되었을 때까지 기다림을 유지한다. read는 TIME이 첫 번째 입력이 도착하기 전에 경과되면 입력 없이 반환할 수 있다. read는 MIN 보다 더 많은 입력이 큐 안에 발생하면 그것을 반환할 수 있다 만일 MIN이 50이고 당신이 단지 10 바이트만 읽기를 요청하면 무슨 일이 발생할 것인가?
보통, read는 버퍼에 50바이트가 찰 때까지 기다린다(또는 더 일반적으로는, 위에 설명된 기다림의 상황이 만족된다.), 그리고 나서, 그들 중 10개를 읽고, 나머지 버퍼에 있는 40개는 read의 연속적 호출에서 사용하기 위해서 운영체제 안에 남겨둔다.

이식성 노트: 어떤 시스템에서, MIN 과 TIME 슬롯은 실제로 EOF 와 EOL 슬롯과 같다. MIN 과 TIME은 비정규 입력에서만 사용되고, EOF와 EOL은 정규입력에서만 사용되기 때문에 심각한 문제는 없지만, 완전한 것은 아니다. GNU 라이브러리는 이렇게 사용하기 위해서 슬롯들을 분리해서 할당한다.

---

12. 5 라인 제어 함수들

이들 함수들은 터미널 디바이스 상에서 갖가지 제어 동작을 수행한다. 터미널 억세스에 관하여, 그들은 출력을 하는 것처럼 취급된다: 만일 그들 함수중 어떤 것이 터미널을 제어중인 배경 프로세스에서 사용된다면, 보통, 프로세스 그룹의 모든 프로세스들은 SIGTTOU 신호를 받는다. 예외적으로 호출한 프로세스 자신이 무시되거나, SIGTTOU 신호에 의해 블록되어 있다면, 그 경우 명령은 수행되고 아무런 신호를 받지 않는다. 24장 [JobControl] 참조.

함수 : int tcsendbreak(int filedes, int duration)

이 함수는 파일 기술자 filedes와 연관된 터미널에 0 비트의 스트림을 전송함으로써 멈춤(break) 상황을 발생시킨다.

멈춤의 존속시간은 duration인수에 의해 제어된다. 만일 duration 이 0이면, 존속시간은 0. 25 와 0. 5초 사이이다. 0이 아닌 값의 의미는 운영체제에 의존된다. 이 함수는 만일 그 터미널이 비동기적 직렬 데이터 포트가 아니면 아무 일도 하지 않는다.
반환 값은 보통 0이고, 에러가 발생하면 -1을 반환한다. 다음의 errno는 이 함수를 위해 정의된 에러 상황이다.

EBADF : filedes 가 유용한 파일 기술자가 아니다.

ENOTTY : filedes 가 터미널 디바이스와 연관이 없다.

함수 : int tcdrain (int filedes)

tcdrain 함수는 큐에 저장된 모든 출력이 터미널 filedes에 모두 전송되어 질 때까지 기다린다.
반환 값은 보통 0이고, 에러가 발생하면 -1을 반환한다. 다음의 errno는 이 함수를 위해 정의된 에러상황이다.

EBADF : filedes 가 유용한 파일 기술자가 아니다.

ENOTTY : filedes는 터미널 디바이스와 연관이 없다.

EINTR : 그 명령은 신호에 의해 인터럽트 되어졌다.

21. 5절 [Interrupter Primitives] 참조.

함수 : int tcflush (int filedes, int queue)

tcflush 함수는 터미널 파일 filedes와 연관된 입력 그리고/또는 출력큐를 소거하기 위해 사용된다. queue 인수는 소거할 큐를 정하고, 다음 값들중 하나를 사용할 수 있다.

TCIFLUSH : 받았지만, 아직 읽지않은 입력 데이터를 소거하라

TCOFLUSH : 쓰여졌지만, 아직 전송되지 않은 출력데이터를 소거하라.

TCIOFLUSH

> 큐에 저장된 입력과 출력을 모두 소거하라. 반환 값은 보통 0이고 에러가 발생하면 -1을 반환한다. 다음의 errno는 이 함수를 위해 정의된 에러상황이다.

EBADF : filedes가 유용한 파일 기술자가 아니다.

ENOTTY : filedes는 터미널 디바이스와 연관이 없다.

EINVAL

> 적당하지 못한 값이 queue인수로써 공급되었다. 이 함수의 이름이 tcflush라고 지어진 것은 유감스러운데, 왜냐하면 한정 "flush"는 보통 모든 출력이 전송되고, 다른 명령을 사용하기 전에 혼동될 입력이나 출력을 버리는데 사용된다. 유감스럽게도, tcflush는 POSIX로부터 유래됐고, 우리는 그것을 변경할 수 없다.

함수 : int tcflow (int filedes, int action)

tcflow 함수는 filedes로 정해진 터미널 파일에서 XON/XOFF 흐름제어에 해당하는 명령을 수행하기 위해 사용된다. action인수는 무슨 명령을 수행할 것인지를 정하고, 다음 값들중 하나를 가질 수 있다.

TCOOFF 출력의 전송을 중단하라.

TCOON 출력의 전송을 다시 시작하라.

TCIOFF STOP 문자를 전송하라.

TCION START 문자를 전송하라.

STOP 와 START에 대한 자세한 정보를 12. 4. 9절 [Special Characters] 를 참조하라.

반환 값은 보통 0이고 에러가 발생하면 -1이 반환된다. 다음의 errno는 이 함수를 위해 정의된 에러상황이다.

EBADF filedes 가 유용한 파일 기술자가 아니다.

ENOTTY filedes 가 터미널 디바이스와 연관이 없다.

EINVAL 적당하지 못한 값이 action인수로 주어졌다.

12. 6 비정규 모드의 예

이곳의 예는 비정규입력모드에서 반향 없이 단일 문자들을 읽기 위해서 터미널 디바이스를 어떻게 맞출 것인지를 보여주고 있다.

```
#include <unistd. h>
#include <stdio. h>
#include <stdlib. h>
#include <termios. h>
/* 원래의 터미널 속성들을 기억하기 위해서 이 변수를 사용하라 */
struct termios saved_attributes;
void
reset_input_mode (void)
{

        tcsetattr (STDIN_FILENO, TCSANOW, &saved_attributes);

}
void
set_input_mode (void)
{

        struct termios tattr;
        char *name;
        /* stdin이 터미널인지 확인하라 */
        if (!isatty (STDIN_FILENO)) {

                fprintf (stderr, "Not a terminal. \n");
                exit (EXIT_FAILURE);

        }
        /* 그들을 나중에 다시 저장 할 수 있도록 터미널 속성들을 저장하라. */
        tcgetattr (STDIN_FILENO, &saved_attributes);
        atexit (reset_input_mode);
        /* 재미있는(?) 터미널 모드를 설정하라. */
        tcgetattr (STDIN_FILENO, &tattr);
        tattr. c_lflag &= ~(ICANON|ECHO); /* Clear ICANON and ECHO. */
        tattr. c_cc[VMIN] = 1;
        tattr. c_cc[VTIME] = 0;
        tcsetattr (STDIN_FILENO, TCSAFLUSH, &tattr);

}
int
main (void)
{

        char c;
        set_input_mode ();
        while (1) {

                read (STDIN_FILENO, &c, 1);
                if (c == '\004') /* C-d */

                        break;

                else

                        putchar (c);

        }
        return EXIT_SUCCESS;

}
```

이 프로그램은 신호와 함께 빠져나가거나 종료되기 전에 원래의 터미널 모드를 재 저장하도록 주의를 기울여야 한다. 이것을 확실하게 하기 위해서는 atexit 함수를(22. 3. 3절 [Cleanups on Exit] 참조.) 사용하라. 쉘은 한 프로세스가 멈추거나 지속될 때 터미널 모드의 재설정에 조심한다. 24장 [Job Control] 참조. 그러나 어떤 쉘들은 실제로 이것을 하지 않는다, 그래서 당신은 터미널 모드를 재설정하는 작업 제어 신호를 위한 핸들러를 만들어야 할 것이다. 위의 예는 그렇게 한다.