대한민국 전산망표준

KIS-2-0010('93)

개방형 운영체제 인터페이스 (POSIX.1)표준

1993. 12

체 신 부

전 문

응용 시스템의 이식성을 제고하기 위한 세계적인 움직임인 포직스(POSIX)는 그 영향력과 시의성에서 가장 주목을 받고 있는 활동이라고 볼 수 있다. 고성능 중형컴퓨터와 워크스테이션의 발전에 따라 가속화되고 있는 UNIX 운영체제의 보급 또한 개방화의 중요한 구동체가 되고 있다. 이러한 상황에서 국가기간전산망사업을 이끌어 나가는데 있어 위에서 기술한 두개의 기술발전을 근간으로 하는것은 매우중요한 일이다.

이러한 운영체제 인터페이스의 표준을 제정하지 않은 상태에서 국가기간전산망을 수행한다면 전산망의 연결은 물론이고, 응용소프트웨어의 호환성에 이르기까지 매우 많은 난관에 부딪히게 될 것이다. 또한 사용자 재교육에 드는 비용과 시간은 전산망사업의 지연을 초래하는 한 원인으로 되는것은 당연하다.

따라서 본 표준을 제정하여 국가기간전산망사업의 효율성을 증가시키고, 또한 다가오는 멀티미디어 시대에 대비하는 것은 국가차원에서 반드시 실행하여야 하는 중대한 과제이다.

국가기간전산망사업의 효율성 제고를 위하여 본 표준은 IEEE(The Institute of Electrical and Electronics Engineers)에서 개발하고 ANSI(American National Standards Institute)에서 1988년에 승인한 POSIX.1(ANSI/IEEE Std 1003.1, IEEE Standard Portable Operating System Interface for Computer Environments)표준을 근간으로 하고 있다.

목 차

제 1 장 총 칙

1.1 목적

전세계적으로 UNIX시장은 년 20%이상의 신장률을 보이고 있다. UNIX의 이러한 경이적인 신장은 UNIX의 우수한 이식성, 개방성, 그리고 윈도우, RISC, 네트워크등과 같은 신기술에 대한 우수한 적응력등에 기인한다. 이와같은 장점으로 인하여 외국은 물론이고 국내에서도 행정전산망용 운영체제로 채택이 되는 등 국내 시장기반도 꾸준히 넓혀가고 있다. 이러한 급속한 신장은 UNIX의 다양한 변종을 만들어 내었으며, 차츰 UNIX에서의 응용프로그램의 혼환성이 줄어들었다. 이러한 상황에 대처하고 그리고 국가기간 전산망 사업을 효과적으로 수행할 수 있도록 하기 위하여 UNIX에 기초한 개방형 운영체제 인터페이스(POSIX.1)표준을 만드는 것을 목적으로 한다.

.

1.2 필요성

기계가 바뀌게 되면 그에 따라 운영체제가 바뀌게 되며, 이에따라 사용자교육 비용과 소프트웨어 비용이 크게 늘어나게 된다. 따라서 이식성이 높은 표준 운영체제와 그와 관련된 표준 인터페이스의 제정이 필수적으로 필요하게 된다. 이러한 조건이 마련될 때 사용자교육비용과 소프트웨어의 변환(conversion)비용이 줄어들며 소프트웨어의 시장이 넓어지게 된다.

1.3 적용대상 및 범위

본 표준은 국가기간전산망에서 사용되는 주전산기급의 운영체제와 그 위에서 실행되는 응용프로그램을 적용대상으로 한다.

본 표준안은 원시 코드 수준에서 응용의 이식성을 지원하기 위한 표준 운영체제 인터페이스와 환경을 정의한다. 이는 응용 개발자와 시스템 구현자 모두를 위한 것이다.

초기 표준안으로서 본 표준안의 초점은 C-언어 인터페이스를 통한 표준화된 서비스들의 제공이다. 앞으로의 개선, 보완들에는 C-언어뿐만이 아닌 다른 언어의 바인딩도 포함될 것으로 기대된다. 이것은 표준안을 프로그래밍 언어와 독립적인 핵심 요구사항을 정의하는 부분과 프로그래밍 언어 바인딩 부분의 두 부분으로 나눔으로서 이루어질 수 있다.

핵심 요구 사항 부분은 본 표준안에 언어 바인딩될 것으로 기대되는 모든 프로그래밍 언어에 공통으로 요구되는 서비스들의 집합을 정의할 것이다. 이들 서비스들은 기능적인 요구사항으로 표현되며 프로그래밍 언어에 의존적인 인터페이스를 정의하지는 않을 것이다.

언어 바인딩은 두가지 중요 부분으로 구성된다. 하나는 이 표준의 프로그래밍 언어와 독립적인 핵심 요구 사항 부분에서 정의된 핵심서비스들을 접근할 수 있는 프로그래밍 언어의 표준화된 인터페이스를 포함할 것이다. IEEE Std 1003.1-1988에 맞는 어느 언어의 바인딩을 사용한 구현도 언어 바인딩의 양 부분에 따라야 한다.

본 표준안은 네 가지의 주요 구성 요소로 이루어진다.

(1) 용어, 개념과 구조, 헤더, 환경 변수 및 관련 요구 사항들에 대한 정의와 사양
(2) 시스템 서비스 인터페이스와 서브루틴들에 대한 정의
(3) C-언어를 위한 언어 관련 시스템 서비스
(4) 이식성, 에러 처리 및 에러 복구를 포함하는 인터페이스 사항들

아래의 영역들은 보 표준안의 범주를 벗어난다.

(1) 사용자 인터페이스(shell)와 관련 명령들
(2) 네트워크 프로토콜과 그에 관련된 시스템 호출 인터페이스
(3) 그래픽 인터페이스
(4) 데이타 베이스 관리 시스템 인터페이스
(5) 레코드 I/O 고려 사항
(6) 목적 또는 이진(binary)코드의 이식성
(7) 시스템 구성(configuration)과 자원 가용성(availability)
(8) 하나의 프로세스내에서 동시성(concurrency)을 지원하는 시스템들의 시스템 서비스 행동 특성

본 표준안은 이러한 성능을 구현하기 위한 내부구축 기술보다는 응용 개발자에게 중요한 외형적인 특성과 기능들을 기술한다. 특히 광범위하고 다양한 상업적 응용들에 요구되는 함수들과 기능들이 강조되었다.

본 표준안은 원시 코드 수준에 대해서만 정의되어 있다. 본 표준안의 목적은 'POSIX를 엄격히 준수'하는 응용 소스 프로그램이 규정대로 구현된 곳에서 실행되도록 할 수 있게 하는 데 있다.

국내의 모든 운영체제와 응용프로그램 개발자 및 판매자들은 이 표준을 만족함으로써 안정적인 국내시장 확보 및 세계시장으로의 활로를 개척할 수 있다.

제 2 장 근거 및 관련 표준

○ ANSI/IEEE Std 1003.1-1988, IEEE Standard Portable Operating System Interface For Computer Environments ANSI/IEEE Std 1003.1

○ Draft Guide to the POSIX Open Systems Environment, P1003.0

○ UNIX Seventh Edition, UNIX System III, UNIX System V, 4.2 BSD와 4.3 BSD

제 3 장 용어해설 및 일반 요구사항

3.1 용어

구현시 정의 - 어떤 수치나 작동방식이 구현시 정의된다는 것은 구현물에 의하여 정확한 프로그램의 구조나 옳바른 자료 등에 대한 요구사항이 정의되거나 문서화됨을 의미한다.

선택사항 - 선택사항이란 본 표준에서는 요구되지는 않으나 **선택**적으로 제공될 수 있음을 의미한다. *POSIX*를 엄격히 준수하는 응용항목에서는 **선택**사항이 허용되지 않는다.

필수사항 - 필수사항이란 구현시의 요구사항 또는 *POSIX*를 엄격히 준수하는 응용항목에 명시된 요구사항을 **필수적**으로 준수해야 함을 의미한다.

권장사항 - 권장사항이란 요구조건은 아니나 구현시 **권장**될 수 있는 사항이다. 즉, 일반응용 프로그램 작성시의 '프로그램 구조'와 같은 **권장**사항과 POSIX를 엄격히 준수하는 응용항목의 요구사항을 의미한다.

지원 - 본 표준에서 일부 기능들은 **선택**적으로 구현될 수 있으나 이들의 구현을 위한 인터페이스는 **필수**적으로 마련되어야 한다. 즉, 어떤 기능이 지원된다는 것은 본 표준에서 규정한 사항에 따라 인터페이스가 동작함을 의미한다. 단, 지원사항이 아닌 기능을 위하여 마련된 인터페이스는 기능이 지원되지 못함에 의해 유발되는 오류 내용을 알려줄 필요가 없으며, 단지 해당기능이 지원되지 않음을 알려주기만 하면 된다.

미정의 - 만일 어떤 응용에 대하여 오류가 있는 프로그램 구조, 잘못된 자료 또는 부정확한 값에 의하여 결정되는 수치나 작동방식에 대한 요구사항이 명시되지 않은 상태를 해당 값이나 작동방식이 미정의되었다고 한다. 구현물(또는 이외의 표준)에서는 그러한 수치의 사용과 작동방식에 따른 결과를 명시해도 좋다. 이러한 사항들은 **확장난을 이용한 POSIX 준수응용** [3.2.2.3]에서 정의한 확장방법에 따라 기술한다.

미규정 - 만일 어떤 응용에 대하여 정확한 프로그램 구조나 수치를 사용함에 따른 아무런 요구사항이 명시되지 않은 것을 해당 수치나 작동방식이 '미규정' 되었다고 한다. 구현물 (또는 이외의 표준) 에서는 그러한 수치를 사용함에 따른 결과나 작동방식을 유발하는 결과를 명시해도 좋다. 특별한 작동방식이 필요한 경우에는 이러한 기능을 사용함에 따른 현상을 **확장을 이용한 POSIX 준수응용** [3.2.2.3]에서 정의되어야 한다.

3.2 적합성

3.2.1 구현의 적합성

3.2.1.1 요구사항

적합한 구현은 아래의 모든 항목을 만족해야 한다.

(1) 시스템은 본 표준에서 정의된 모든 인터페이스를 **필수적**으로 지원해야 한다. 이들 인터페이스는 본 표준에서 언급된 기능에 대한 작동방식의 지원이 **필수**적이다.

(2) 시스템은 본 표준에서 요구되지 않은 부수적인 함수나 기능등을 **선택**적 으로 지원한다. 표준이 아닌 확장사항은 시스템 문서에 기재된 것과 같이 기록되어야 한다. 이러한 비표준 확장사항이 사용될 때에는 본 표준에 명시된 함수들이나 기능의 작동방식이 바뀌기도 한다. 그러한 경우에는 본 표준에 명시된 대로 실행되도록 하는 환경에 대한 정의가 **필수**적으로 시스템 문서상에 수록되어야 한다. 그러나 어떠한 경우에도 이러한 조건으로 인하여 *POSIX*를 엄격히 준수하는 응용의 변경을 초래해서는 않된다.

3.2.1.2 문서화

아래의 내용을 수록한 문서는 구현물이 IEEE Std 1003.1-1988에의 적합성을 확인하는데 **필수**적인 문서이다. 이 문서는 본 표준과 동일한 구조를 가져야 하며, 해당절에서 필요로하는 정보를 제공하여야 한다. 또한 문서는 본 표준의 범위를 벗어나는 확장된 제반사항에 대한 정보는 포함하지 않아야 한다.

이 문서에는 적용되는 표준서의 제목 (정식명칭을 기입), 표준의 번호, 제정일자등의 자료를 명시하여야 한다. 문서의 내용중 적합성에 관한 항목에는 *POSIX*를 준수하는 응용이 사용가능한 ISO나 ISO회원기구에 의해 승인된 소프트웨어 표준의 목록을 (**선택**적으로) 수록할 수도 있다. 또한 이들 표준이나 나라별 정부기구표준 중에서 필요로하는 사항들도 본문서에 **선택**적으로 수록할 수 있다.

문서에는 **<limits.h>** 와 **<unistd.h>** 의 헤더(header), 지시치(stating value), 이러한 수치들이 바뀔 수 있는 조건 및 필요하다면 수치의 변화한계 등을 필수적으로 기록하고 있어야 한다.

문서에는 본 표준에서 지시된 구현시 정의된 모든 기능의 구현에 따르는 작동방식을 **필수**적으로 기술하여야 한다. 그러나 본 문서에는 미정의 또는 미규정된 기능들에 관해서는 기술할 필요가 없다. 또한 본 문서에는 구현물이 다를 수 있음을 언급하는 항목에 구현물의 작동방식을 기록할 것을 **권장**한다.

3.2.1.3 구현선택사항의 적합성 보장

아래의 심볼 상수는 *POSIX*를 준수하는 응용 또는 POSIX 시스템에 적합한 요구사항 또는 두가지 요구사항을 모두 보장하는 본 표준의 구현선택사항을 반영한다.

{NGROUPS_MAX}
다중 그룹 옵션 (**실행시 증가하는 수치** [3.9.3] 참조)

{_POSIX_JOB_CONTROL}

작업 제어 옵션 (**컴파일시 심볼상수**[3.10.3] 참조)
{_POSIX_CHOWN_RESTRICTED}
관리/보안 옵션 (**실행시 심볼상수**[3.10.4] 참조)

컴파일시 심볼상수 [3.10.3] 및 **실행시 심볼상수** [3.10.4] 항목에 수록된 나머지 심볼상수들은 실험이나 여러가지 편리한 작동방식들을 이용하는 지침으로 사용될 수 있다. 그러나 이들 심볼상수는 *POSIX*를 준수하는 응용의 요구사항을 보장하기 위하여 또는 적합한 구현들을 구별하는 데 충분한 기능적 상이성을 제시하지는 않는다.

어떤 **선택**사항을 제거함에 따라 본 표준에 수록된 함수들이 정의되지 못하는 경우에는 (비록 구현상 함수가 아무런 작업을 수행하지 않으며 해당 함수의 리턴값이 에러뿐이라고 하더라도) 구현물은 본 표준에 정의된 철자에 따라 해당 함수를 불러낼 수 있도록 **필수**적으로 구현되어야 한다.

3.2.2 응용의 적합성

본 표준에의 적합성을 요구하는 모든 응용은 **C-프로그래밍 언어를 위한 언어관련 서비스** [3.2.3]의 만을 이용하여야 하며, 아래의 항목중 하나를 만족하여야 한다.

3.2.2.1 POSIX를 엄격히 준수하는 응용

*POSIX*를 엄격히 준수하는 응용이란 본 표준에 기술된 기능과 적용가능한 표준언어만을 사용하는 응용이어야 한다. 이러한 응용은 본 표준에 명시된 모든 작동방식들을 구현시 정의되는 사항으로 간주하고, 모든 심볼상수는 본 표준이 정하는 범위의 수치를 허용한다. 이러한 응용은 그 가용성이 **<limits.h>** [3.9]와 **<unistd.h>** [3.10]에 지시된 기능의 가용성을 원용하도록 허용된다.

3.2.2.2 POSIX를 준수하는 응용

3.2.2.2.1 ISO형 POSIX 준수응용

*ISO*형 *POSIX* 응용은 본 표준에 기술된 기능과 ISO 표준으로 승인된 적합성이 입증된 언어(Conforming Language) 바인딩(binding)만을 사용하는 응용이다. 그러한 응용은 **선택**사항과 제한된 종속성 및 이외의 사용된 ISO표준에 대한 모든 사항을 문서화한 문장을 포함하여야 한다.

3.2.2.2.2 <단체>형 POSIX 준수응용

<단체>형 *POSIX* 준수응용이란 응용이 <**단체**>로 구분되는 IOS회원기구의 표준을 **선택**적으로 적용할 수 있다는 점에서 *ISO*형 *POSIX* 준수응용과 다르다. 그러한 응용은 모든 **선택**사항과 제한된 종속성 및 사용되는 이외의 모든 <단체> 표준을 문서화한 문장을 포함하여야 한다.

3.2.2.3 확장을 이용한 POSIX 준수응용

확장을 이용한 *POSIX* 준수응용이란 본 표준과 일관성을 유지하는 특별한 비표준 기능을 사용하는 것 외에는 *POSIX*를 준수하는 응용과 동일하다. 그러한 응용에 대해서는 *POSIX*를 준수하는 응용 항목에 추가하여 확장부분에 대한 요구사항도 충분히

문서화하여야 한다. 확장을 사용한 *POSIX* 준수응용은 **확장을 이용한 *ISO*형 POSIX 준수응용**[3.2.2.2.1]이나 **확장을 이용한 <단체>형 POSIX준수응용**[3.2.2.2.2]중의 하나이어야 한다.

3.2.3 C-프로그래밍 언어를 위한 언어관련 서비스

새로운 버전의 C-표준(*ANSI/X*3.159- 198x 표준 C-언어)가 공인되면 IEEE Std 1003-1988에 수록된 관련 항목에서의 요구사항도 결정된다. 본 표준에서 필요한 표준 C-언어에 대한 사항은 본 표준의 제 9 장에 수록하였다. 이외에도 제 9 장에서는 C-언어와 관련된 추가사항을 포함하고 있다. 따라서 IEEE Std 1003.1-1988와 C-언어 바인딩을 준수하는 모든 구현물은 제 9 장에서 추가하도록 요구된 사항들을 포함하여 제 9 장에 명시된 모든 기능들을 **필수**적으로 제공하여야 한다.

비록IEEE Std 1003.1-1988이 필요에 의하여 C-표준의 일부분을 참조하지만, IEEE Std 1003.1-1988에의 적합성을 준수하기 위해서 C-표준에의 적합성을 만족시킬 필요는 없다. 또한 제 9 장에 기술한 여러가지 기능을 제공하는 C-언어의 구현물은 본 표준에의 적합성을 준수하고 있으나, 구현된 C-언어가 C-표준에 따르고 있음을 의미하지는 않는다.

3.2.3.1 적합성의 종류

IEEE Std 1003.1-1988과 C-언어 바인딩을 준수하는 모든 구현물은 IEEE Std 1003.1-1988, C-언어 바인딩(C-표준언어 관련 시스템 지원) 또는 IEEE Std 1003.1-1988 (일반적인 사용법과 관련한 C-언어 관련 시스템 지원)중의 하나를 준수해야 한다.

3.2.3.2 C-표준언어 관련 시스템 지원

구현자는 C-표준을 참조하는 경우 제 9 장에 수록된 요구사항을 **필수**적으로 만족시켜야 한다. 또한 제 9 장의 요구사항을 모두 충족시키는 (사용된) C-표준의 버젼을 문서에 기재하여야 한다.

구현시 C-표준이 공인되기까지는 1988. 5.13 (X3J11/88-002) 일자 문서초안을 참고한다. 만일 C-표준초안을 이용하여 적합성을 입증하려면 IEEE Std 1003.1-1988의 C-언어 바이딩(C-표준 언어 관련 시스템 지원)을 준수하는 가를 밝혀야 한다. 또한 구현시에 C-표준초안을 참고하여 제 9 장을 구현하는 경우 적용된 초안이 1988.5.13일자 초안인가를 분명히 문서화하여야 한다(**필수사항**). 공인에 앞선 C-표준초안을 이용한 구현물은 공인된 C-표준에서의 변경사항을 반영하여 수정하도록 **권장**된다.

3.2.3.3 일반적인 사용법과 관련한 C-언어 관련 시스템 지원

구현자는 단순히 C-표준을 참고하도록 하는 대신 일반적인 사용법으로 사용하기 위하여 제 9 장에서 요구하는 루틴과 지원사항을 **필수**적으로 제공하여야 한다. 즉, 구현자는 어느 부분이 C-표준을 참고하고 있는가를 제외하고는 제 9 장의 모든 요구사항을 **필수**적으로 만족하여야 한다. C-표준이 참고되는 부분마다 구현자는 C-언어의 일반적인 사용법과 일관성을 유지하면서 동등한 지원을 **필수**적으로 제공하여야 한다. 구현자는 제공된 인터페이스간의 상이점 및 제공되는 인터페이스가 일반적인 사용법

대신에 참고되는 C-표준을 가지고 있음에 대한 사항을 문서화하여야 한다. 구현자는 인터페이스 상의 차이점을 기술할 때 참고된 C-표준의 버젼을 분명히 명시해야 하며, C-표준의 모든 새로운 상위 버젼에 대해서도 이들간의 차이점에 대한 사항을 **필수**적으로 기술하여야 한다.

C-표준이 공인되기까지는 IEEE Std 1003.1-1988의 C-언어상에서의 바이딩(일반적인 상용법으로서의 C-언어 관련 시스템 지원)에 적합함을 주장하기 위해서는 (제 9 장의 요구조건에 따라 만들어진 구현물과 C-표준을 근거로 하여 제 9 장의 요구조건에 따라 만들어진 구현물에 있어서 인터페이스상의 차이를 문서화할 때에는) 1988.5.13일자 C-표준초안을 **필수**적으로 참고하여야 한다.

C-표준이 함수들을 정의하고, 따라서 함수를 가르키는 일반적인 사용법이 제공되지 않는 경우, 만일 이러한 함수의 구현은 C-표준에 기술된 바에 따라 구현되어야만 한다(**필수**사항). 만일 이 함수가 구현되지 않는다면, 언급한 바와 같이 C-표준과의 차이점에 대한 사항이 문서화되어야 한다(**필수사항**).

3.2.4 기타 C-언어 관련 사항

아래의 규칙들은 C-언어용 라이브러리 함수의 사용법과 관련된 사항이다. 아래의 항목은 달리 특별한 규정이 없는 한 제 4 장과 10 장 사이에 설명되는 함수에 적용된다.

(1) 만일 함수들의 인수(argument)가 부당한 값일 때(예를들어 인수가 함수가 다루는 수의 범위를 벗어나거나 포인터가 프로그램이 주소번지를 벗어날 때, 또는 함수가 널 포인터(Null pointer)를 인수로 가질 수 있음이 명확히 정의되지 않았을 때의 널 포인터 등)에 해당 함수의 작동방식이 정의되지 않는다.

(2) 어떤 함수도 헤더에서 매크로(macro)로 구현될 수 있다. 응용시에 매크로를 사용하지 않으려면 #undef을 사용하여 매크로를 제거하고 실제함수가 참조되도록 해야 한다. 또한 본 표준에서 응용은 어떤 함수를 선언하기에 앞서서 #undef를 사용할 것을 **권장**한다.

(3) 매크로로 구현된 라이브러리 함수를 불러내는 경우에는 함수의 인수를 단 일회 평가하는 코드를 실행한다. 필요하면 인수를 괄호로 묶을 수 있으며, 따라서 일반적으로 임의의 수식을 인수로 사용할 수 있다.

(4) 만일 어떤 라이브러리 함수가 헤더에 정의된 특정한 형식을 참조하지 않고도 선언될 수 있다면 함수는 명시적이거나 묵시적으로 선언될 수 있어야 하며, 또한 함수의 헤더를 생략한채로 사용할 수 있어야 한다.

(5) 만일 인수의 갯수가 변화하는 어떤 함수가 (명시적으로 또는 부속된 헤더에 포함되는 방식으로) 선언되지 않으면 이 함수의 작동방식은 정의되지 않는다.

3.3 일반 용어

아래의 용어는 본 표준에서 특별히 사용되는 용어들이다

경로명(pathname) - 파일을 식별하는 문자열로, 널 문자를 포함하여 {PATH_MAX}바이트 길이의 문자열로 구성된다. 경로명은 사선으로 시작할 수 있으며, 여러개의 파

일명이 사선으로 구분되어 첨가될 수 있다. 경로명이 사선으로 끝나는 경우는 디렉토리를 나타낸다. 연속된 몇 개의 사선은 하나의 사선으로 간주되나 두개의 연속된 사선으로 시작되는 경로명은 '구현시 정의'된 바에 따른다. 그러나 세개 이상의 사선으로 시작되는 경우에는 하나의 사선으로 처리된다. 경로명의 처리는 **경로명 해석** [3.4]에서 자세히 다룬다.

경로명 구성요수(pathname component) - 파일명 참조.

경로명 접두어(path prefix) - 사선(/)으로 끝나는, 디렉토리를 의미하는 경로명.

고아프로세스 그룹(orphaned process group) - 어떤 프로세스 그룹에 속한 프로세스의 부모 프로세스(parent process)가 그룹에 속한 멤버자신이던가 또는 그룹의 세션에 속한 프로세스가 아닌 프로세스 그룹.

공 디렉토리(empty directory) - 도트파일과 도트-도트 파일 이외의 다른 파일을 위한 디렉토리 엔트리가 없는 디렉토리.

공백 문자열(Null string) - 첫문자가 널 문자(또는 공백 문자)인 문자열

그룹 ID(group ID)-시스템 사용자는 최소한 하나의 그룹에 소속된다. 그룹은 양의 정수로 나타내지는 그룹 *ID*로 식별되며, *gid_t* 형식으로 표현된다. 어떤 그룹의 식별자가 프로세스에 부속된 부속된 경우에 그룹 *ID*의 값은 실제그룹 *ID*(real group *ID*), 유효 그룹 *ID*(effective group *ID*), 추가 그룹 *ID*(supplementary guoup *ID*)의 하나이거나 세이브드 셋-그룹 *ID*(saved set-group-*ID*)중의 한가지를 나타낸다.

기타 파일 클라스(file other class) - 프로세스가 파일 소유자 클라스(file owner class)나 파일 그룹 클라스(file group class)에 속하지 않으면 이 프로세스는 기타 파일 클라스(file other class)에 속한다고 정의한다.

널 스트링(Null string) - 공백 문자열 참조.

도트 파일(dot) - 파일명의 하나의 도트(.)인 파일. (**경로명 해석** [3.4] 참조)

도트-도트 파일(dot-dot) - 파일명이 두개의 도트 (..)인 파일 (**경로명 해석** [3.4] 참조)

디바이스(device) - 컴퓨터 주변장치 또는 컴퓨터를 사용할 때 주변장치의 형대로 사용되는 장치류

디렉토리(directory) - 디렉토리 엔트리(directory entry)를 가지고 있는 파일.
하나의 디렉토리는 동일한 이름으로 된 두개의 디렉토리 엔트리를 가지지 못한다.

디렉토리 엔트리(directory entry) - 파일명을 이용하여 해당 파일을 지정하는 관계. 여러개의 디렉토리 엔트리가 하나의 파일을 지정할 수 있다.

루트 디렉토리(root directory) - 어떤 프로세스의 경로명에 사선으로 시작되는 경로명에 대해 그 경로명을 해석하는데 사용되는 디렉토리. [3.4]

링크(link) - 디렉토리 엔트리 참조.

링크 수(link count) - 특정한 파일을 참조하는 디렉토리 엔트리의 갯수.

모드(mode) - 파일의 형식과 파일접근허용을 지정하는 속성의 집합. **파일접근허용** [3.4] 참조.

문자(character) - 단일 그래픽 기호를 표현하는 하나 이상 바이트들의 집합.

문자형 특수파일(character special file) - 디바이스를 참조하는 파일로 그중 대표적인 문자형 특수파일이 터미날 디바이스 파일(**터미날 인터페이스 일반사항** [8.1] 참조)이다. 이외의 문자형 특수 파일은 (본 표준에서 그 구조가 정의되지 않고) 구현시에 정의된다.

백그라운드 프로세스 그룹(background process group) - 포어그라운드 프로세스 그룹(foreground process group)이 아닌 제어 터미날(controlling terminal)과의 연결을 설정한 세션(session)에 속하는 프로세스 그룹.

부모 디렉토리(parent directory) - 작업중인 디렉토리를 지정하는 디렉토리 엔트리를 가지고 있는 디렉토리. 디렉토리가 아닌 파일에 대해서는 작업중인 파일의 디렉토리 엔트리를 가지고 있는 디렉토리. 이 개념은 도트 파일이나 도트-도트 파일에는 해당되지 않는다.

부모 프로세스(parent process) - 프로세스 참조

부모 프로세스 ID(parent process ID) - 현재 작동중인 프로세스에 의하여 새로운 프로세스가 생성된다고 할 때, 새로운 프로세스를 생성한 프로세스의 *ID*를 부모 프로세스 *ID*라고 하며, 이것은 부모 프로세스의 수명동안 유지된다. 부모 프로세스의 수명이 끝나면 부모 프로세스 *ID*는 구현시 정의된 시스템 프로세스의 프로세스 *ID*와 같아진다.

블럭형 특수파일(block special file) - 디바이스를 참조하는 파일로 디바이스의 하드웨어 특성을 노출시키지 않으면서 디바이스로의 접근을 제공하는 점이 문자형 특수파일(character special file)과 다른 점이다.

사선(slash) - 문자 '/'. 본 문자는 ISO 8859/1의 solidus로 알려진 문자이다

사용자 ID(user ID) - 각각의 시스템 사용자는 음수가 아닌 정수값으로 표현되는 사용자 *ID*를 가진다. 이 값은 uid_t형식으로 지정된다. 프로세스와 관련하여 사용자를 나타낼 때 사용자 *ID*는 실제 사용자 *ID*, 유효사용자 *ID* 또는 (**선택**적으로)세이브드 셋-사용자-*ID*를 나타낸다.

상대 경로명(relative parhname) - **경로명 해석** [3.4] 참조.

세션(session) - 각 프로세스 그룹이 세션에 포함된다. 어떤 프로세스가 세션에 포함된다는 것은 해당 프로세스 그룹이 세션에 소속됨을 의미한다. 새로 생성된 프로세스는 이 프로세스를 생성한 프로세스가 속한 세션에 소속된다. 또한 프로세스는 세션에의 소속관계를 변경할 수 있다(setsid() [5.3.2]참조). setpgid() [5.3.3]을 지원하는 구현은 동일한 세션에 여러개의 프로세스 그룹을 포함시킬 수 있어야 한다.

세션 리더(session leader) - 세션을 생성하는 프로세스. setsid() [5.3.2]참조.

세션 수명(session lifetime) - 세션이 생성되어 이 세션에 포함된 모든 프로세스의 수명이 종료될 때까지의 시간.

세이브드 셋-그룹-ID(saved set-group-ID) - *setgid*() [5.2.2] 및 *exec*[4.1.2]에 기술된 바와 같이, 세이브드 셋-그룹-*ID* **선택**사항이 구현될 때 유효 그룹 *ID*속성*(attribute)*을 융통성있게 지정(assignment)할 수 있는 프로세스의 한 속성(attribute).

세이브드 셋-사용자-ID(saved set-user-ID) - *setuid*() [5.2.2] 및 *exec*[4.1.2]에 기술된 바와 같이, 세이브드 셋-사용자-*ID* **선택**사항이 구현될 때 유효 사용자 *ID*속성*(attribute)*을 융통성있게 지정(assignment)하는 프로세스의 한 속성(attribute).

시스템(system) - 본 표준에 의하여 구현된 운영체제

시스템 프로세스(system process) - 응용 프로그램을 실행하는 프로세스가 아니며, 프로세스 *ID*를 가지는 시스템이 정의한 프로세스.

신호(signal) - 시스템에서 발생한 어떤 사건을 어떤 프로세스에게 알려주거나 영향을 미칠 수 있는 메키니즘. 이러한 사건의 예로는 하드웨어상의 예외 발생이나 프로세스에 의한 특별한 조치 등이다. 신호라는 단어는 이러한 사건 자체를 의미하기도 한다.

실제그룹 ID(real group ID) - 사용자 그룹을 표시하는 프로세스의 속성으로 프로세스가 생성되는 시점에서 프로세스를 생성한 사용자 그룹을 의미하는 프로세스의 속성(attribute)이다. 그룹*ID*참조. 이 값은 *setgid*() [5.2.2]에 기술된대로 프로세스 수명중에 변경될 수 있다.

에폭시각 - 1970년 1월 1일 0시 0분 0초로 설정된 국제협약시각. 에폭이후 경과시간(seconds since the epoch) 참조.

에폭이후 경과시간(second since the Epoch) - 에폭시각으로부터 특정한 시각까지 경과한 시간을 초로 환산한 값. 이 값은 초(tm_sec), 분(tm_min), 시간(tm_hour), 1월 1일 이후의 일자(tm_year)로 표시되는 국제협약시각(Coordinated Universal Time)으로부터 아래와 같은 방식으로 계산된다. 즉, 년도가 1970이전이면 그 값은 음수이며 그 의미는 일정하지 않다. 만일 년도가 1970년 이후이면 그 값은 양의 정수이다. 시간은 다음식으로 결정된다

tm_sec + tm_min*60 + tm_hour*3600 + tm_year*86400
+ (tm_year-70)*3153600 + ((tm_year-69)/4)*86400

열린 파일(open file) - 현재 작업중인 파일 서술자(file descriptor)가 가르키는 파일.

열린 파일서술자(open file descriptor) - 프로세스 또는 프로세스 그룹들이 파일을 접근하는 방법을 목록화한 것. 각각의 파일 서술자(file descriptor)는 항상 하나의 열린 파일서술(open file descriptor)을 참조하나 하나의 열린파일서술은 여러개의 파일 서술자에 의하여 참조될 수 있다. 열린파일서술의 속성으로는 파일 오프셋, 파일상황(file status) (**파일제어절차를 위한 자료정의** [7.5.1]의 표 7-5 참조) 및 파일접근모드(표7-5)이 포함된다.

유효 그룹 ID(effective user ID) - **파일접근허용** [3.4]에 나타난 여러가지 접근허용사항을 결정하기 위하여 사용되는 프로세스의 속성의 한가지. 그룹 *ID* 참조. 이 값은 *setgid*() [5.2.2]와 *exec* [4.1.2]에 기술된 대로 프로세스 수명중에 변경될 수 있다.

유효 사용자 ID(effective user ID) - **파일접근허용** [3.4]에 나타난 여러가지 접근 허용사항을 결정하기 위하여 사용되는 프로세스의 속성의 한가지. 사용자 *ID*참조. 이 값은 *setuid*() [5.2.2]와 *exec* [4.1.2]에 기술된 대로 프로세스 수명중에 변경될 수 있다.

읽기전용 파일 시스템(read only file system) - 구현시 파일내용의 변경을 제한하는 특성을 가지도록 정의된 파일 시스템

자식 프로세스(child process) - 프로세스 참조

작업 디렉토리(working directory) - 프로세스와 관련된 디렉토리로 경로명(pathname) 해석에 사용되는 사선으로 시작되지 않는 경로명. **경로면 해석** [3.4] 참

조.

작업 제어(job control) - 사용자로 하여금 **선택**적으로 프로세스들의 실행을 중지시키고 후에 계속하도록 허용하는 기능. 사용자는 터미날 I/O 드라이버와 명령어 인터프리터와 함께 대화식 인터페이스를 통하여 이러한 기능을 실행한다. 적합한 구현은 **선택**적으로 작업제어기능(job control facility)을 지원한다. 즉, 이러한 **선택**사항이 존재하는지 여부는 컴파일시 또는 실행시에 {_POSIX_JOB_CONTROL}심볼의 정의로 알 수 있다. **심볼상수** [3.10] 참조

적절한 권한(appropriate privileges) - 어떤 프로세스와 관련하여 본 표준에서 정의된 함수의 호출(function call)이나 함수의 호출의 **선택**사항 등에 특별한 권리를 부여하였을 때 프로세스별로 접근허용되는 권리를 구현시 정의의 관점에서 본 의미.

절대 경로명(absolute pathname) - **경로명의 해석** [3.4] 참조.

접근 모드(access mode) - 어떤 파일을 다루는데 허용되는 접근허용의 형태.

정규 파일(regular file) - 시스템에 의하여 자료구조의 형태가 지정되지 않고, 임의의 순서로 내용을 접근할 수 있는 파일.

제어 프로세스(controlling process) - 제어터미날(controlling terminal)과 연결을 설정한 세션리더(session leader). 만일 어떤 터미날이 해당 세션용 제어 터미날로서의 역할을 완료하면 세션리더는 더이상 제어 프로세스가 되지 못한다.

제어 터미날(controlling terminal) - 세션에 연결된 터미날로 각 세션에는 하나의 제어 터미날만이 허용되며, 하나의 제어 터미날은 항상 하나의 세션에만 관계된다. 제어 터미날로 부터의 입력 절차(**터미날 인터페이스 일반사항** [8.1] 참조)는 제어터미날에 부속된 모든 프로세스로 보내는 특별한 '신호'를 발생하기도 한다.

주소공간(address space) - 프로세스에 의하여 참조되는 메모리상의 위치.

추가 그룹 ID(supplementary group ID) - 프로세스는 유효그룹 *ID*이외에도 파일접근허용을 결정하는 데 사용되는 {NGROUPS_MAX}개의 추가그룹 *ID*를 가진다. 어떤 프로세스가 새로이 생성되면 이 프로세스의 추가그룹 *ID*는 이 프로세스를 생성한 부모프로세스의 추가그룹 *ID*와 같다. 여기서 프로세스의 유효그룹 *ID*가 추가그룹 *ID*의 목록상에 포함되는지 아니면 제거되는지는 정의되지 않는다.

클럭 수(clock tick) - {CLK_TCK}에 정의된 단위초당 파형 변화의 갯수로 *clock_t*형

식의 값으로 표현된다.

터미날(terminal) - **터미날 인터페이스 일반사항** [8.1]의 규정에 따르는 문자형 특수 파일.

터미날 디바이스(terminal device) - 터미날 참조.

특권(privilege) - 적절한 권한(appropriate privilege) 참조.

특성시험 매크로(feature test macro) - 특정한 특성들이 헤더(header)에 포함되는지를 결정하기 위하여 사용되는 '#defined'심볼. **C-표준에서의 심볼** [3.8.1] 참조.

파이프(pipe) - *pipe*() 함수에 의해 발생된 두개의 파일 서술자중의 하나에 의하여 접근되는 대상으로 파이프가 생성되면 파일 서술자는 생성된 파이프를 다루기 위하여 사용될 수 있으며, 이때 파이프는 FIFO형 특수 파일과 동일한 작동방식으로 운영된다. 파이프는 파일 계층내에서 이름(name)을 갖지 않는다.

파일(file) - 읽거나 쓰이는, 또는 두가지가 가능한 대상. 파일은 파일 접근허용, 형식과 같은 특별한 속성이 주어진다. 파일의 형식으로는 정규파일(regular file), 문자형 특수파일, 블럭형 특수파일, FIFO형 특수파일 및 디렉토리등이 있다. 이외의 파일 형식에 대해서는 구현시 정의된다.

파일 그룹 클라스(file group class) - 만일 어떤 프로세스가 파일 소유자 클라스(file owner class)에 포함되지 않고 그 프로세스의 유효그룹 *ID*(effective group ID)나 추가그룹 ID(supplementary group ID)가 그 파일과 관련된 그룹 ID와 일치한다면 이 프로세스는 해당파일의 파일 그룹 클라스에 속한다고 정의한다. 이외의 클라스 멤버에 대해서는 구현시 정의된다.

파일 서술(file description) - 열린파일서술(open file description) 참조.

파일 서술자(file descriptor) - 각 프로세스 별로 파일에의 접근을 위하여 열린 파일을 지정하는 음이 아닌 정수.

파일명(filename) - 파일의 이름으로{NAME_MAX}길이의 바이트로 구성된다. 이름은 구성하는 문자는 사선(/)과 널문자(Null character)를 제외한 모든 문자가 사용된다. 도트파일과 도트-도트 파일은 특별한 의미를 가진다. **경로명 해석** [3.4] 참조. 파일명은 때때로 경로명 구성요소로 사용된다.

파일 모드(file mode) - 파일접근허용비트(file permission bit)와 파일의 성격을 규정하는 사항으로 **<sys/stat.h>** [6.6.1]에 기술된다.

파일 소유자 클라스(file owner class) - 만일 어떤 프로세스의 유효 사용자 *ID*(effective userr *ID*)가 사용자 *ID*(user *ID*)와 일치하면 이 프로세스를 해당파일의 파일 소유자 클라스에 속한다고 정의한다.

파일 오프셋(file offset) - 파일에서 다음번 I/O연산이 시작되는 바이트 위치를 의미하며, 정규파일과 관련한 각각의 열린 파일 서술(open file description), 블럭형 특수파일(block special file), 또는 디렉토리(directory)는 파일 오프셋을 가진다. 터미날 디바이스를 지정하지 않는 문자형 특수파일(character special file)은 파일 오프셋을 가질 수도 있으며, 파이프(pipe)나 *FIFO*형 특수파일(FIFO special file)은 파일 오프셋이 정의되지 않는다.

파일접근허용비트(file permission class) - 사용되는 파일에 관한 여러가지 정보들 특히 어떤 파일에 대하여 프로세스가 읽기(read), 쓰기(write) 또는 실행(*exec*ution)/탐색(search)을 허용하는 가를 알려주는 정보. 파일접근허용비트는 소유자(owner), 그룹(group), 및 기타(other)의 세부분으로 구분된다. 이들 비트는 **<sys/stat.h>** [6.6.1]에 기술된 대로 파일 모드(file mode)에 포함된다. 파일접근허용비트에 대한 세부적인 내용은 **파일접근허용** [3.4] 참조.

파일 일련번호(file serial number) - 파일시스템에서 파일당 주어진 하나의 번호. 파일 일련번호는 파일시스템 내에서는 일정하다.

파일시스템(file system) - 파일들과 이들에 대한 속성들의 집합. 파일시스템은 특정한 파일을 지정하는 파일 일련번호가 기록될 이름공간(name space)을 가진다.

포어그라운드 프로세스 그룹(foreground process group) - 각 제어터미날(controlling terminal)에 연결된 각 세션은 해당 세션의 프로세스 그룹중에서 단 하나의 프로세스 그룹만이 해당 제어 터미날의 포어그라운드 프로세스 그룹이 된다. 포어그라운드 프로세스 그룹은 백그라운드 프로세스 그룹(background process group)에 의하여 거절된 제어 터미날을 접근할 때 특정한 권리를 가진다. **터미널 접근 제어** [8.1.1.4] 참조.

포어그라운드 프로세스 그룹 ID(foreground process group ID) - 포어그라운드 프로세스 그룹(foreground process group)의 프로세스 그룹 *ID*.

프로세스(process) - 주소 공간(address space)과 (주소 공간내에서 실행되는) 단일 제어 스레드(single control thread) 및 이들을 위해 필요한 시스템 자원. 어떤 프로세스는 fork() 함수를 호출하여 또다른 프로세스를 생성할 수 있다. 여기서 fork()를 실행한 프로세스가 부모 프로세스(parent process)가 되며, fork()에 의하여 새로이 생성된 프로세스는 자식 프로세스(child process)가 된다.

프로세스 그룹(process group) - 시스템에서 각각의 프로세스는 (프로세스 그룹 *ID*로 구분되는) 프로세스 그룹에 소속된다. 이러한 그룹화는 관련된 프로세스들을 불러낼 수 있도록 돕는다. 새로이 생성된 프로세스는 자신을 생성한 프로세스의 프로세스 그룹에 소속된다.

프로세스 그룹 리더(process group leader) - 프로세스 *ID*가 프로세스 그룹 *ID*와 같은 프로세스.

프로세스 그룹 수명(process group lifetime) - 프로세스 그룹이 생성된 시점으로부터 프로세스의 종료나 *setsid*() 또는 *serpgid*()함수를 실행함에 따라 프로세스 그룹을 구성하는 마지막 프로세스가 그룹을 떠나는 시점까지의 기간.

프로세스 그룹 ID(process group *ID*) - 시스템에서 각 프로세스 그룹이 존재하는 동안 해당 프로세스 그룹은 프로세스 그룹 *ID*로 불리는 양의 정수값 pid_t로 나타내어진다. 시스템에서 하나의 프로세스 그룹 *ID*는 해당 프로세스 그룹 수명이 끝날때까지 재사용될 수 없다.

프로세스 수명(process lifetime) - *fork*()함수에 의하여 어떤 프로세스가 생성되면 이 프로세스는 실행가능한 것으로 간주된다. 실행가능한 프로세스의 제어 스레드(controll thread)와 해당 주소공간은 이 프로세스가 종료될 때까지 존재한다. 여기서 만일 프로세스가 (프로세스가 보유하고 있던 일부 자원들이 시스템으로 되돌려지는) 비실행상태로 돌입하더라도 프로세스 *ID*등의 일부 자원들은 그대로 사용될 수 있다. 또다른 프로세스가 비실행중인 프로세스에 대하여 *wait*() 또는 *waitpid*() 함수를 실행시키면 비실행중인 프로세스가 유지하던 나머지 자원들도 시스템으로 되돌려진다. 시스템으로 되돌려지는 마지막 자원은 프로세스 *ID*이며, 비로소 프로세스 수명이 종료된다.

프로세스 ID(process ID) - 시스템의 각 프로세스는 프로세스 수명기간 동안 특별한 양의 정수로 된 번호로 표시된다. 이때 프로세스 *ID*는 id_t에 그 값이 저장된다. 시스템에서 프로세스 *ID*는 프로세스 수명이 종료될 때까지 중복되어 사용되지 못한다. 또한 프로세스 그룹 *ID*가 프로세스 *ID*와 같은 경우에는 프로세스 그룹 수명이 종료될 때까지 동일한 프로세스 *ID*를 중복하여 사용할 수 없다. 시스템 프로세스가 아닌 프로세스는 프로세스 *ID*가 '1'이 될 수 없다.

C-표준(C-Standard) - *ANSI/X*3.159-198x 프로그래밍 언어 C-표준의 줄임말.

FIFO형 특수파일(FIFO special file) - 파일의 한 종류로 먼저 쓰여진 것이 먼저 읽히는 방식(선입선출)으로 사용되는 파일. 이외에도 FIFO의 특성에 대해서는 *open*() [6.3.1], *read*() [7.4.1], *write*() [7.4.2] 및 *lseek*() [7.5.3] 참조.

POSIX 파일명용 문자집합(portable filename character set) - IEEE Std 1003.1-1988에 적합한 파일명에 사용될 수 있는 문자의 종류는 다음과 같다.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

여기서 마지막 세문자는 각각 마침표, 밑줄표 및 하이픈을 의미한다. 하이픈은 파일명의 첫자로는 사용되지 않는다. 대문자와 소문자는 적합성의 구현방법에 따라 상이한 문자로 간주된다. 경로명의 경우에는 사선도 사용할 수 있다.

3.4 일반 개념

확장 보안 제어 - 구현시 정의된 확장된 보안 제어를 위하여 접근제어(파일접근 허용 참조)와 특권(privilege)메카니즘(적절한 권한 [3.3] 참조)이 정의된다. 이들은 본 표준에 기술된 것과 상이한 보안대책을 가능케하는 보안 메카니즘을 제공하도록 한다. 이때 이러한 메카니즘은 본 표준에 수록된 기 정의된 함수들의 의미(semantics)를 변경하거나 무시하지 않아야 한다.

파일접근허용 - 표준 파일 접근 제어 메타니즘은 아래에 설명한 바와 같은 파일허용 비트를 사용한다. 파일 허용비트는 파일이 *open*() [6.3.1], *creat*()

[6.3.2], *mkdir*() [6.4.1] 및 *mkfifo*() [6.4.2]등의 함수에 의하여 생성될 때 설정(set)되고, *chmode*() [6.6.4]함수에 의해 변경될 수 있다. 또한 이들 파일허용비트는 *stat*() 또는 *fstat*() [6.6.2]에 의해 읽힌다.

구현물은 표준파일접근 제어방법 이외에 추가적인 파일접근제어나 대치되는 파일접근 제어방법 중의 한가지나 두가지 모두를 **선택**적으로 제공할 수 있다. 이 경우 추가적인 접근제어 메카니즘은 파일허용 비트로 표시한 접근허용을 더욱 제한하는 사항이어야 한다(필수사항). 대치되는 접근제어 메카니즘은 다음 사항을 필수적으로 준수하여야 한다.

(1) 대치되는 접근제어 메카니즘은 파일의 접근허용에 대해 (*stat*()나 *fatat*() 함수의 복귀값으로 되돌려지는) 해당파일의 파일 소유자 클라스, 파일 그룹 클라스 및 기타 파일 클라스별로 접근허용비트를 분명히 규정하여야 한다.
(2) 대치되는 접근제어 메카니즘은 각 파일별로 파일 소유자나 적절한 특권을 가진 사용자의 명백한 작업에 의해서만 처리될 수 있어야 한다.
(3) *chmod*() 로 어떤 파일의 파일허용비트들을 변경하면 이 파일에 대해서는 대치되는 접근제어 메카니즘이 사용될 수 없다. 대치되는 접근제어 메카니즘의 사용을 제한하기 위하여 추가적인 접근허용 메카니즘의 사용을 제한할 필요는 없다.

어떤 프로세스가 읽기, 쓰기 또는 실행/탐색을 위한 파일접근 허용을 요구할때마다 만일 추가적인 메카니즘이 접근을 거절하지 않는다면 접근모드는 다음과 같이 결정된다.

(1)프로세스가 적절한 권한을 가진다면
  (a) 만일 읽기, 쓰기 또는 디렉토리 탐색이 요구되면 접근을 승인한다.
  (b) 프로세스에 대한 실행 허용이 요구될 때, 만일 파일허용비트나 대치되는 접근제어방법에 의해 하나 이상의 사용자에게 실행허용이 주어진다면 실행 허용 접근이 승인된다. 이외의 경우에는 실행허용 접근이 부정된다.
(2)이외의 경우
  (a) 파일은 파일 소유자 클라스, 파일 그룹 클라스 및 기타 파일 클라스 별로 읽기, 쓰기, 실행/탐색 허용 등의 파일허용비트를 가진다
  (b) 대치되는 접근제어 메카니즘이 사용되지 못하고 요구된 접근허용 비트가 프로세스에 속한 클라스에 대해 설정되었거나, 또는 대치되는 접근제어 메카니즘이 적용가능하고 접근제어가 요구된 접근을 허용하는 두가지 경우에는 접근이 허용된다. 이외의 경우에는 접근이 거절된다.

파일계층 - 시스템상의 모든 파일은 계층구조로 구성된다. 즉, 파일 시스템에서 터미날 노드들은(디렉토리가 아닌) 여러가지 형식의 파일을 의미하며, 터미널 노드가 아닌 파일은 디렉토리이다. 여러개의 디렉토리 엔트리들이 하나의 파일을 지정할 수 있으므로 파일계층은 방향성 그래프(directed graph)로 설명될 수 있다.

파일명의 이식성 - 지정된 문자 이외의 다른 문자들을 파일명에 사용하면 의미가 혼동되거나 문맥이 모호하기 때문에 파일명은 POSIX용 파일명 문자집합으로 구성되어야 한다.

파일시각의 갱신 - 각각의 파일은 다음과 같은 경우에 갱신되는 세가지 시각을 가지고 있다. 즉, 파일이 악세스될 때, 파일이 변경될 때 그리고 파일 상태가 바뀔 때 등이다. 이러한 수치들은 **<sys/stat.h>** [6.6.1]에 기술된 바와 같은 파일특성구조(file characteristics structure)로 출력된다.

파일에는 파일자료의 읽기 또는 쓰기, 파일 상태의 변경 등과 같이 본 표준에 수록된 함수에 의하여 파일시각이 갱신되도록 표시(marked for update)되어야 한다. 파일시각의 갱신은 갱신시마다 즉시 갱신되거나 일정한 주기로 갱신되도록 구현될 수 있다. 시각의 갱신은 현재시각을 기록하고 갱신되어야 함을 알려주는 표식이 지워진다. 갱신되어야 함을 알려주는 표식들은 파일이 어떤 프로세스에 의해서 더 이상 열리지 않거나 해당 파일에 대하여 *stat*() [6.6.2]또는 *fstat*()가 실행될 때에 변경된다. 갱신이 완료된 이후의 시각은 규정되지 않는다. 갱신은 읽기전용파일 시스템상의 파일에서는 실행되지 않는다.

경로명 해석 - 경로명 해석은 경로명으로 부터 파일계층에 있는 특정한 파일을 구별해내는 절차이다. 따라서 동일한 파일을 의미하는 여러개의 경로명이 존재할 수 있다.

경로명상의 각 파일명은 파일명의 상위파일명이 가르키는 디렉토리 내에 위치한다. 예를 들면 경로명의 일부분인 'a/b'에서 파일 'b'는 디렉토리 'a'내에 위치한다 이러한 절차가 성공적으로 완료되지 못하면 경로명 해석이 완성되지 않는다. 사선(slash)으로 시작되는 경로명에서는 첫번째 파일명의 상위파일명은 프로세스의 루트 디렉토리로 간주된다(그러한 경로명을 절대경로명이라고 함). 사선으로 시작되지 않는 경로명에서는 첫번째 파일명의 상위파일명이 프로세스의 작업 디렉토리로 간주된다(그러한 경로명을 상대경로명이라고 한다)

경로명 구성요소의 해석은 해당 구성요소의 경로명접두어에 관련된 값인 {NAME_MAX}와 {_POSIX_NO_TRUNC}에 의해 결정된다. 임의의 경로명 구성요소의 길이가 {NAME_MAX}이 가르키는 값 보다 크고, 따라서 주어진 구성요소의 경로명접두어가{_POSIX_NO_MAX}에 영향을 미친다면 (*pathconf*() [6.7.1] 참조), 구현물에서는 이러한 상태를 에러조건(error condition)으로 간주한. 이외의 경우에서는 경로명 구성요소의 첫{NAME_MAX}바이트가 사용된다.

도트파일(.)은 상위파일명이 지시하는 디렉토리을 의미한다. 도트-도트파일(..)은 상위파일명이 지시하는 디렉토리의 부모 디렉토리를 의미한다. 루트 디렉토리에서의 도트-도트파일은 루트 디렉토리 자신을 의미한다.

하나의 사선만을 가지는 경로명은 해당 프로세스의 루트 디렉토리로 간주된다. 널 경로명(null pathname)은 사용될 수 없다.

3.5에러번호

대부분의 함수는 다음과 같이 정의된 외부 변수 *errno*에 의하여 에러번호를 제공한다.

extern int errno;

변수 *errno*의 값은 어떤 함수를 호출하였을 때, 이 함수에 대해 명확히 지정된 값으로 주어지며, 다른 함수를 호출함에 의하여 변경될 때까지 일정한 값으로 유지된다. 변수 *errno*는 함수에 의하여 되돌려지는 때에만 에러번호가 정확한 것으로 검증된다. 본 표준에서 정의된 모든 함수는 에러번호로 영('0')을 가지지 않는다.

함수의 호출을 실행하는 중에 하나 이상의 에러가 발생하는 경우에 본 표준에서는 에러들이 어떤 순서로 발견되었는가를 정의하지 않는다. 따라서 여러가지 에러번호중의 하나가 임의로 되돌려진다.

구현시에는 본 항목의 목록에 수록되지 않은 에러들이 추가될 수 있으며, 본 항목에 설명된 환경이외의 경우에 본 항목에 명시된 에러들이 출력될 수 있으며, 또는 어떤 경우에 에러가 발생하는 것을 방지하기 위한 확장이나 제한이 취해질 수도 있다. 에러의 발생조건과 어떤 에러가 구현시 정의된(implementation-defined) 것인지에 대해서는 각 함수를 설명하는 절의 '**에러**' 항목에 기술된다. 구현물은 본 표준에 명시된 에러조건으로 인하여 본 항목에서 기술한 에러번호와 다른 에러번호를 출력하지 않아야 한다(필수사항).

아래의 심볼명은 본 표준에서 특별히 정의된 함수의 이름 별로 발생가능한 에러번호를 나타낸다. 이들은 함수별로 '**에러**' 항목에서 보다 자세히 기술된다. 심볼명이 의미하는 번호들은 구현시에 정의되므로 이들 심볼명들은 프로그램 내에서만 사용하도록 권장된다. 본 항목에 수록된 모든 수치는 서로 상이한 값이어야 하며, 구현시 정의된 이들 수치는 각 심볼명별로 헤더 **<errno.h>**에 수록된다.

[E2BIG] Arg의 길이가 너무 길다
새로운 프로세스 이미지의 인수 항목과 환경 항목(environment list)으로 사용된 바이트의 합이 시스템이 부여한 {ARG_MAX}바이트 길이보다 더 길다.

[EACCES] 허용금지
파일접근 허용이 허용되지 않는 방법으로 파일에 접근하는 시도가 행해진다

[EAGAIN] 자원이 잠시 활용될 수 없다
일시적인 상태이므로, 이후에 동일한 루틴을 불러서 정상적인 방법으로 종료될 수 있다.

[EBADF] 잘못된 파일 서술자
파일 서술자의 인수가 범위를 벗어나, 열리지 않는 파일을 참조하는 경우, 또는 읽기(또는 쓰기)만을 위해 열린 파일에 쓰기(또는 읽기)를 시도중이다.

[EBUSY] 자원이 사용중(Resource Busy)

어떤 프로세스가 어떤 자원의 사용을 요구함에 의하여 충돌을 일으킬 수 있는 자원을 하나의 프로세스가 사용하고 있으므로 인하여 시스템 자원의 사용이 불가능한 자원을 사용하려고 한다.

[ECHILD] 자식 프로세스가 없음
어떤 프로세스에서 자식 프로세스가 존재하지 않거나 자식 프로세스의 종료를 기다리지 않는 경우에, 이 프로세스가 *wait*()나 *waitpid*()함수를 실행하였음.

[EDEADLK] 자원의 교착상태(deadlock)가 회피됨
교착상태를 야기할지도 모르는 시스템 자원을 잠그려 한다.

[EDOM] 영역 에러
C-표준에 정의된, 수학용 함수의 입력인수가 함수의 정의구역을 벗어남을 의미한다.

[EEXIST] 파일존재
존재하는 파일을 잘못된 문맥으로 규정하는 경우. 예를 들어 *link*()함수에서의 새로운 링크명을 호출하는 경우 발생한다.

[EFAULT] 주소오류
시스템이 어떤 함수의 호출시 사용하려는 인수의 주소가 유효하지 못함을 검출한 경우. 이 에러를 정확히 발견하는 방법에 관한 것은 구현시 정의된다. 그러나, 이 조건을 검출하기 위해서는 이 수치를 사용하여야 한다

[EFBIG] 파일이 너무 큼
파일의 크기가 구현시 정의된 파일의 최대 크기를 넘는다.

[EINTR] 함수의 호출중 인터럽트기 수행됨
실행중 인터럽트가 걸릴 수 있는 함수의 실행중 프로세스에 의하여 비동기 신호(SIGINT, SIGQUIT 등 - 헤더 **<signal.h>** [4.3.1] 참조)가 들어온다. 이때, 신호처리기(signal handler)가 정상적인 방법에 의하여 복귀하면 호출중 인러럽트가 걸린 함수는 이 에러 조건을 복귀한다.

[EINVAL] 유효하지 않은 인수
유효하지 못한 인수가 제공되었다. 예를 들어 *signal*()나 *kill*()함수에 정의되지 않은 신호를 명시하는 경우에 발생한다.

[EIO] 입출력 오류
어떤 물리적인 입출력 오류가 발생하였다. 이 오류가 발생하면 동일한 파일 서술자의 계속적인 참고시에도 보고된다. 동일한 파일 서술자에 대해 상이한 오류가 발생되면 새로운 오류로 인해 [EIO] 에러를 잃어버린다.

[EISDIR] 디렉토리임
디렉토리를 쓰기 모드로 열려는 시도가 행해진다

[EMFILE] 열려는 파일이 너무 많다.
하나의 프로세스가 열수 있는 파일의 수가{OPEN_MAX}이 나타내는 최대 파일 서술자의 갯수를 초과하였다.

[EMLINK] 링크의 수가 너무 많음

하나의 파일을 지시할 수 있는 링크의 수가 {LINK_MAX}이 나타내는 최대링크의 수를 초과하였다.

[ENAMETOOLONG] 파일명이 너무 김
경로명의 문자열 길이가 {PATH_MAX}값을 초과하거나 경로명 구성요소가{NAME_MAX}보다 길고 이 파일에 대하여 {POSIX_NO_TRUNC}값이 유효하다.

[ENFILE] 시스템에 너무 많은 파일이 열려 있다.
시스템상에서 현재 너무 많은 파일이 열려 있다. 시스템에서 동시에 열수 있는 파일의 크기가 미리 정의된 수치에 도달하여 새로운 파일을 임시로 열려는 요청을 실행할 수 없다

[ENODEV] 해당 디바이스가 없음
어떤 디바이스에 대하여 부적당한 함수를 실행하려는 경우, 예를 들면 프린터와 같은 쓰기전용 장치를 읽으려는 시도를 수행한다.

[ENOENT] 해당화일이나 디렉토리가 없음
명시된 경로명의 구성요소가 존재하지 않거나 경로명이 공백문자열이다.

[ENOEXEC] Exec 포맷오류
적절한 허가를 갖고있는 파일이지만, 파일의 내용은 실행가능한 형태로 존재하지 않아 실행할 수 없는 파일을 요구하였다

[ENOLOK] 잠금장치가 더이상 없음
동시에 가능한 파일과 레코드의 잠금 수가 시스템이 정한 한계에 도달해서 더 이상 활용할 수 없다

[ENOMEM] 충분한 공간(space)이 없음
새로운 프로세스이미지가 요구하는 메모리의 크기가 하드웨어나 시스템이 정한 메모리 관리상의 한계를 초과하여 요구된다

[ENOSPC] 디바이스를 위한 공간이 없음
정규 파일에 대해 *write*() 함수를 실행하거나 디렉토리를 확장할 때 해당 디바이스에 남아있는 여분의 공간이 없음을 의미한다

[ENOSYS] 함수가 구현되지 않았음
구현되지 않아 사용할 수 없는 함수를 호출하였음을 의미한다

[ENOTDIR] 디렉토리가 아님
명시된 파일명의 구성요소가 있으나 디렉토리로 예상된 구성요소가 디렉토리가 아님을 의미한다

[ENOTEMPTY] 디렉토리가 비어있지 않음
비어있는 디렉토리라고 예상된 디렉토리에 도트파일과 도트-도트파일이 외의 파일이 존재한다

[ENOTTY] 부적당한 I/O제어조작
일반적인 파일이나 특수파일에서 허용되지 않는 작업을 실행하려는 시도가 행해진다

[ENXIO] 디바이스나 주소가 없음
존재하지 않는 디바이스를 의미하는 특수파일에 대한 입출력이나,디바

이스의 한계를 벗어나는 요구가 행해진다. 이 오류는 예를 들어 테이프장치가 작동상태(on-line)가 아니거나 장치에 디스크 팩이 올려져 있지 않을 때 발생된다

[EPERM] 조작이 허용되지 않음
적절한 특권을 가진 프로세스 또는 어떤 파일이나 기타 자원의 소유자로 제한된 조작을 수행하도록 하기 위한 것이다

[EPIPE] 부서진 파이프
자료를 읽는 프로세스가 없어 파이프나 FIFO로 쓰기를 시도한다

[ERANGE] 결과가 너무김
C-표준에 정의된바와 같이 함수의 결과가 너무 길어서 가용공간내에 수록될 수 없음을 의미한다

[EROFS] 읽기전용 파일시스템
파일시스템 상에서 읽기전용 파일이나 디렉토리를 수정하려고 시도한다.

[ESPIPE] 부적절한 seek함수
파이프나 FIFO상에서 *lseek*()함수를 시도한다

[ESRCH] 해당 프로세스가 없음
주어진 프로세스 ID에 의해서 명시된 프로세스를 발견할 수 없다.

[EXDEV] 잘못된 링크
다른 파일시스템 상의 파일을 링크하려고 시도한다

3.6. 기본적 시스템 데이타 형식

여러가지 시스템 함수에 의해 사용되는 몇몇 자료형식(data types)은 본 표준의 일부로 정의되지 않고 구현시에 정의된다. 즉, 이들 형식은 헤더 **<sys/stat.h>**에서 정의되며, 그 내용은 표 3-1과 같다

<표 3-1> **기본적인 시스템 자료 형식**

정의된 형식	설 명
dev_t	디바이스 (device)번호에 사용됨
gid_t	그룹 ID에 사용됨
ino_t	파일 일련번호에 사용됨
mode_t	파일 속성(예를 들어 파일 형식, 파일 접근 허용등)에 사용됨
link_t	링크 수(link count)에 사용됨
off_t	파일 그기에 사용됨.
pid_t	프로세스 ID와 프로세스 그룹 ID에 사용됨
uid_t	사용자 ID에 사용됨

표 3-1에 나열된 모든 형식은 산술연산이 가능한 형식이다. 예를 들어 pid_t는 부호를 포함하는 산술연산이 가능한 형식이다.

추가적으로 구현시 정의된 형식에 대한 정의(definition)는 이 헤더에 수록된다. 이러한 선언은 _t로 끝나는 명칭을 가진다. 그러한 심볼들은 **<sys/typet.h>**이 포함되더라도 특성시험 매크로(feature test macro)의 가시성(visiblity)이 요구되어야 할 필요성이 없다.

3.7 환경 서술 (environment description)

환경(environment)으로 불리는 문자자열 배열은 프로세스가 시작될 때 가용하게 된다. 이 배열은 외부변수 *environ*에 의해 지정되며, 다음과 같이 정의된다.

```
extern chatr **environ;
```

이들 문자열은 "name = value"의 형태를 가지며 여기서 name은 문자 '='를 포함하지 않는다. 또한 환경에서의 문자열의 순서에는 아무런 의미를 부여하지 않는다. 만약 프로세스 환경내에 여러 문자열이 동일한 name으로 정의되어도 그 순서는 미정의된다. 아래의 명칭들은 정의될 수 있으며, 정의되었을 경우에 지정된 의미를 갖는다.

HOME	사용자 데이타베이스(user database)로 부터 알아낸 사용자의 최초 작업 디렉토리명(**<pwd.h>**) 헤더의 내용 [10.2.2]참조).
LANG	지역성(locale)을 지정한 명칭.
LC_COLLATE	대조정보를 위한 지역변수명
LC_CTYPE	문자부류를 위한 지역변수명
LC_MONETARY	금액에 관계된 수치 편집정보를 포함하는 지역변수명
LC_NUMERIC	수치 편집(기수 문자 등)정보를 포함하는 지역변수명
LC_TIME	날짜/시간을 구성하는 정보를 위한 지역변수명
LOGNAME	사용자 데이타베이스(헤더 **<pwd.h>** 참조)내에 수록된 등록명(login name)과 일치하는 사용자의 등록계정(login account)의 명칭. 이 명칭은 **POSIX 파일명용 문자집합** [3.3]에 포함되는 문자들로 구성된다
PATH	파일명으로 구분되는 실행파일을 찾는 과정중에 검색되어지는 경로명 접두어들의 나열. 경로명 접두어들은 콜론(:)으로 구분된다. 접두어의 길이가 영보다 큰 경우에는 해당접두어와 파일명사 이에 사선(/)이 삽입된다. 접두어의 길이가 영인 접두어는 현재 작업 디렉토리를 나타낸다. 접두어의 길이가 영인 접두어는 두개의 인접한 콜론(::)이나, 하나의 콜론과 이어지는 접두어의 나열, 또는 접두어의 마지막에 이어지

는 콜론으로 나타내어진다. 경로명 접두어들은 지정된 명칭으로 실행가능한 프로그램이 발견될 때까지 접두어들을 처음부터 끝까지 탐색한다. 경로명을 알아내려는 파일의 경로명에 사선이 포함되어 있다면 경로명 접두어를 탐색하는 절차는 수행되지 않는다.

TERM 정해진 형태로 자료를 출력하기 위한 터미날의 형식. 이 정보는 각 단말기별로 지니고 있는 특별한 기능을 지원하도록 명령어나 응용 프로그램에 의해 사용된다.

TZ 시간대역정보(Time Zone Information). 이 문자열의 형식은 **시각에 관한 함수의 확장** [9.1.1]에 정의되어 있다.

응용에 의하여 사용되거나 생성되는 환경변수 *name*은 호환성을 위하여 POSIX파일명용 문자집합의 문자들만 포함하여야 한다. 이외의 문자들은 구현물에 의해 사용이 허용될 수도 있으며, 응용들은 그런 이름의 사용을 허용해야 한다. 대문자나 소문자는 각 각 별개로 처리되며 같은 문자가 아니다. 시스템상에서 정의된 환경 변수는 대문자나 밑줄로 시작하며 대문자, 밑줄, 번호로만 구성된다.

환경 변수가 가질 수 있는 *value*는 변수값의 마지막을 공백문자로 구분되는 것과 환경과 프로세스 인수의 길이가 {ARG_MAX} 바이트로 제한되는 것 외에는 제한없이 사용된다.

이외의 *name* = *value* 쌍은 *environ* 변수를 조작하거나 프로세스를 생성할 때 (*exec* [4,1,2] 참조) *envp* 인수를 사용하여 환경내에 수록할 수 있다.

3.8 C-언어 정의

3.8.1 C-표준에 의한 심볼

본 표준에서 사용되는 용어와 심볼은 C-프로그래밍 언어에 의해 정의된 것이다. 다음의 용어들은 C-표준에서 정의된다: *CLK_TCK, NULL, byte, character array*, *clock_t*, *header*, *null character, string, time_t*.

본 표준에서 **널 포인터**(NULL pointer)라는 용어는 C-표준에서 사용되는 *null*와 동일하다. **널(NULL)** 또는 공백은 (C-표준에서 자체적으로 요구되는 여러곳 이외에) C-표준과 동일한 값을 가지도록 **<unistd.h>**에 선언되어야 한다.

추가적으로 밑줄로 시작하는 심볼들의 조건은 다음과 같이 정의한다.

(1) 밑줄로 시작하는 모든 외부 식별자(identifier)는 사용법이 지정된다.

(2) 이외의, 밑줄로 시작하고 대문자나 또다른 밑줄로 시작하는 모든 식별자는 사용법이 지정된다.

(3) 프로그램에서 어떤 외부 식별자를 지정된 외부 식별자와 같은 이름으로 정의하면 의미적으로는 동일하여도 그 기능은 미정의된다.

특별한 이외의 이름 공간(namespace)도 C-표준에서 지정된다. 이들 지정내용은 본 표준에서도 적용된다. 또한 C-표준에서는 헤더를 하나이상 포함하는 것이 가능해야하며, 심볼은 여러개의 헤더내에서 정의될 수 있다. 이러한 요구사항은 본 표준의 헤더

에도 해당된다.

3.8.2 POSIX 심볼

본 표준내의 특정한 심볼들은 헤더에서 정의된다. 몇몇 헤더들은 본 표준에서 정의된 심볼이외의 심볼들도 정의할 수 있다. 그러나 이 경우 응용 프로그램에서는 사용되는 심볼간의 충돌 가능성이 존재한다. 또한 다른 표준에서는 어떤 심볼들의 가시성(visibility)을 제어할 수 없으면 이들 심볼들은 헤더내에 포함시킬 수 없는 경우에도 본 표준에서는 이들 심볼들에 대해서 정의한다.

특성시험 매크로(feature test macro)라고 불리는 심볼은 헤더내에 포함될 수 있는 심볼의 가시성을 제어하는데 사용된다. 현 표준에 의한 구현물, 발전된 버젼에 의한 구현물 및 이외의 표준에서는 특성시험 매크로를 추가적으로 정의할 수 있다. 특성시험 매크로는 어떤 심볼이 응용의 전체가 아닌 일부분에서만 보이는 어떤 헤더의 #include 에 앞서서 응용의 컴파일시에 정의되어야 한다. 만약 매크로의 정의가 해당 #include에 선행되지 않는다면 그 결과는 미정의된다. 특성시험 매크로는 밑줄 문자(_)로 시작된다.

구현물에서는 심볼들을 추가하거나 특성시험 매크로를 이용하여 심볼들의 가시성의 제어를 생략하고도 이들 심볼들을 포함시킬 수 있다.

아래와 같은 특성시험 매크로가 정의된다.

명 칭	설 명
_POSIX_SOURCE	프로그램에서는 본 표준에 의해 정의된 심볼이 환경에 의해 제공되는 것으로 간주한다. 헤더내에서 학장이 허용되나 본 표준에서 명칭이 형태상의 명확한 규제사항이 없으면, 확장은 본 특성시험 매크로에 의해 가시성을 가지도록 할 수 없다.

특성시험 매크로의 정확한 의미는 선택되어 지원되는 C-언어 형식에 달려있다.

3.8.2.1 C-표준 언어 관련 지원사항

프로그램내에 특성시험 매크로가 없다면 C-표준에 의해 정의된 심볼들의 집합만이 존재한다. 각각의 특성시험 매크로에 대하여, 해당 특성시험 매크로에 의하여 규정된 심볼들과 C-표준의 심볼들은 헤더가 포함될 때에 정의되어야 한다.

3.8.2.2 공통사용 C-언어 관련 지원사항

특성시험 매크로 _POSIX_SOURCE 가 프로그램내에서 정의되지 않았다면, 본 표준의 요구사항에 벗어나는, 각 헤더내에 정의된 심볼들은 구현시 정의된다.

어떤 헤더가 포함되기 전에 _POSIX_SOURCE가 정의되면 C-표준에 의한 심볼들과 (_POSIX_SOURCE를 포함하여) 프로그램을 위하여 정의된 특성시험 매크로에 의해 가시성이 주어진 심볼들 외에는 어떤 심볼들도 가시성이 없다.

어떤 헤더가 포함되기 전에 _POSIX_SOURCE가 정의되지 않으면 그 작동방식은 미정의 된다.

3.8.3 헤더와 함수원형(prototype)

C-표준 언어 관련 지원사항을 따르는 구현물은 모든 함수에 대한 함수원형을 필수적으로 선언하여야 한다.

공통사용 C-언어 관련 지원사항을 따르는 구현물에서는 함수값이 '일반적인' 정수형(int)이 아닌 모든 함수의 함수값은 필수적으로 선언하여야 한다.

이들 함수원형은 (필요시) 아래에 나열된 헤더내에 수록된다. 아래에 나열되지 않은 함수의 경우에는 함수원형을 <unistd.h>에 포함시킨다. 즉, 선언된 함수가 사용될 때마다 해당 함수의 **용례**에 언급되었는지에 관계없이 해당 함수가 <unistd.h>에 #include문으로 포함될 것이다. C-표준에서도 기술하고 있는 함수(**C-언어를 위한 언어 관련 서비스** [9] 참조)는 C-표준에서 이들 함수를 정의하는 헤더에 함수원형을 수록한다. **POSIX 심볼** [3.8.2]내의 심볼들의 가시성에 관한 요구사항은 지켜져야 한다.

<dirent.h> *opendir*()[6.1.2], *readdir*()[6.1.2]
rewinddir()[6.1.2], *closedir*()[6.1.2]

<fcntl> *open*()[6.3.1], *creat*()[6.3.2], *fcntl*()[7.5.2]

<grp.h> *getgrgid*()[10.2.1], *getgrnam*()[10.2.1]

<pwd.h> *getpwuid*()[10.2.2], *getpwnam*()[10.2.2]

<setjmp.h> *sigsetjmp*()[9.3.1], *siglongjmp*()[9.3.1]

<signal.h> *kill*()[4.3.2], *sigemptyset*()[4.3.3],
sigfillset()[4.3.3], *sigaddset*()[4.3.3],
sigdelset()[4.3.3], *sigismember*()[4.3.3],
sigaction()[4.3.4], *sigprocmask*()[4.3.5],
sigpending()[4.3.6], *sigsuspend*()[4.3.7]

<stdio.h> *fileno*()[9.2.1], *fdopen*()[9.2.2]

<sys/stat.h> *umask*()[6.3.3], *mkdir*()[6.4.1], *mkfifo*()[6.4.2],
stat()[6.6.2], *fstat*()[6.6.2], *chmod*()[6.6.4]

<sys/timers.h> *times*()[5.5.2]

<sys/utsname.h> *uname*()[5.4.1]

<sys/wait.h> *wait*()[4.2.1], *waitpid*()[4.2.1]

<termios.h> *cfgetospeed*()[8.1.2.7], *cfsetospeed*()[8.1.2.7],
cfgetispeed()[8.1.2.7], *cfsetispeed*()[8.1.2.7],
tcgetattr()[8.2.1], *tcsetattr*()[8.2.1],
tcsendbreak()[8.2.2], *tcdrain*()[8.2.2],
tcflush()[8.2.2], *tcflow*()[8.2.2]

<time.h> *time*()[5.5.1], *tzset*()[9.3.2]

<utime.h> *utime*()[6.6.6]

3.9 수의 제한

다음의 소절은 특정한 구현에 의하여 제한되는 수치들에 대해 기술한다. 중괄호로 구분되는 값(예를 들어 {LIMIT}이 그것이며, 이때 중괄호는 명칭을 구성하는 문자로 사용되지 않는다.

3.9.1 C-언어에서의 수치제한

본 표준에서 사용되는 수치의 제한값은 반드시 C-프로그래밍 언어에서 정의된 것으로 간주한다. C-표준에서 정의된 수치의 제한값은 다음과 같다. {CHAR_BIT}, {CHAR_MAX}, {CHAR_MIN}, {INT_MAX}, {IMT_MIN}, {LONG_MAX}, {LONG_MIN}, {MB_LEN_MAX}, {SCHAR_MAX}, {SCHAR_MIN}, {SHRT_MAX}, {SHRT_MIN}, {UCHAR_MAX}, {UCHAR_MIN}, {ULONG_MAX}, {ULONG_MIN},

3.9.2 최소값

표 3-2의 심볼은 주어진 값으로 **<limits.h>**에 정의된다. 이들은 본 표준을 준수하는 시스템에서 일정한 작동방식에 대한 가장 제한적인 값을 가지는 심볼의 명칭이다. 관련된 심볼들은 (실직적인 구현을 반영하고, 그리고 비제한적일 수도 있는)본 표준내에서 정의된다. 본 표준에 적합한 구현은 최소한 주어진 크기의 값을 제공하여야 한다. 이식가능한 응용의 정확한 동작을 보장하려면 주어진 값보다 큰 값으로 규정하지 않아야 한다.

<표 3-2> **최 소 값**

명 칭	설 명	값
{_POSIX_ARG_MAX}	환경 자료와 *exec* 함수들 중의 하나에서의 인수들의 길이(바이트 단위).	4096
{_POSIX_CHILD_MAX}	실제 사용자 ID당 동시에 실행될 수 있는 프로세스의 수.	6
{_POSIX_LINK_MAX}	파일의 링크 수	8
{_POSIX_MAX_CANON}	터미널의 표준 입력 큐의 바이트 수.	255
{_POSIX_MAN_INPUT}	터미널의 표준 입력 큐에 사용 가능 가용 공간의 바이트 수.	255
{_POSIX_NAME_MAX}	파일명의 바이트 수.	14
{_POSIX_NGROUPS_MAX}	프로세스당 동시에 존재하는 추가 그룹 ID의 수.	0
{_POSIX_OPEN_MAX}	하나의 파일이 동시에 열 수 있는 파일 수.	16
{_POSIX_PATH_MAX}	경로명의 바이트 수.	255
{_POSIX_PIPE_BUF}	파이프(pipe)에 기록할 때 동시에	512

	기록되는 '나뉠 수 없는(atomic)'정보의 바이트 수.

3.9.3 실행시 증가되는 수치

표 3-3 에 보인 값의 크기는 특정한 구현에 의해 결정된다. **POSIX**를 엄격히 준수하는 **응용**은 특정한 구현에서 **<limits.h>**으로 제공되는 수치는, 해당 구현하에서 POSIX를 엄격히 준수하는 **응용**이 실행될 때마다 유지해야 하는 값의 최소치로 가정한다.

<표 3-3> **실행시 증가되는 값**

명 칭	설 명	최 소 값
{NGROUPS_MAX}	프로세스당 동시에 존재하는 보완 그룹 ID의 최대값.	{_POSIX_NGROUPS_MAX}

특정한 구현의 특정한 상태에서는 수치가 해당구현을 위한 **<limits.h>**에 의해 제공되는 값으로 부터 증가되기도 한다. 특정한 상태하에서 지원되는 실제수치는 *sysconf*() [5.8.1]함수에 의해 제공된다.

3.9.5 실행시 변하지 않는 값

표 3-4에서 하나의 수치에 대한 정의는, 대응되는 값이 최소값 이상이나 그 값이 결정되지 않는 특정한 구현물 상의 **<limits.h>**로 부터 제거하여야 한다.

이것은 특정한 구현물에서 특정한 상태하에서 가용 기억공간의 크기에 달려있다.

특정한 상태에 의해 지원되는 실제수치는 *sysconf*() [5.8.1]함수로 제공된다.

<표 3-4> **실행시 변하지 않는 값**

명 칭	설 명	최 소 값
{ARG_MAX}	환경 자료를 포함하여, *exec*함수 함수들 중의 하나에서 인수들의 바이트 길이	{_POSIX_ARG_MAX}
{CHILD_MAX}	실제 사용자 ID당 동시에 존재하는 프로세스의 최대값.	{_POSIX_CHILD_MAX}
{OPEN_MAX}	한 프로세스가 동시에 열 수 있는 파일의 최대 갯수.	{_POSIX_OPEN_MAX}

3.9.5 경로명 변수값

표 3-5의 수치는 구현상에서 상수이거나, 경로명 별로 상이할 수 있다. 예를 들어, 파일 시스템이나 디렉토리는 서로 다른 특징을 가질 수 있다.

표 3-5에서 하나의 수치에 대한 정의는, 대응되는 값이 최소값 이상이나 적용되는 파일에 따라 그 값이 변화하는 특정한 구현물 상의 **<limits.h>**로 부터 제거하여야 한다. 특정한 경로명을 지원하는 실제수치는 *pathconf*() [6.7.1]함수로 부터 제공된다.

<표 3-5> **경로명 변수값**

명 칭	설 명	최 소 값
{LINK_MAX}	파일의 링크 수의 최대값	{POSIX_LINK_MAX}
{MAX_CANON}	터미널 기본 입력줄의 최대 바이트 수. (**표준 모드 입력 프로세스** [8.1.1.6] 참조)	{_POSIX_MAX_CANON}
{MAX_INPUT}	터미널 입력 큐에서 사용가능한 공간의 최소 바이트 수. 즉 이식성 있는 응용이 입력을 읽어들이기 전에 이들을 입력으로 쳐넣을 수 있는 최대 바이트의 길이.	{_POSIX_MAX_INPUT}
{NAME_MAX}	파일명의 최대 바이트 길이. (문자열의 길이가 아니며, 공백문자는 제외)	{_POSIX_NAME_MAX}
{PATH_MAX}	경로명의 최대 바이트 길이. (문자열의 길이가 아니며, 공백문자는 제외)	{_POSIX_PATH_MAX}
{PIPE_BUF}	파이프(pipe)에 기록할 때 동시에 기록되는 나뉠 수 없는 (atomic) 정보의 최대 바이트 길이.	{_POSIX_PIPE_MAX}

3.10 심볼상수

적합한 구현은 헤더 **<unistd.h>**를 필수적으로 포함하여야 한다. 이 헤더는 심볼상수들과 본 표준에서 다루는 심볼들의 구조에 대하여 정의한다. 이 헤더에 의해 정의된 상수는 아래와 같다. 이들 상수의 실제값은 구현시 정의된다.

3.10.1 *access*()함수용 심볼상수

access()함수에 의해 사용되는 상수는 표 3-6과 같다.

<표 3-6> ***access*()함수용 심볼상수**

상 수	설 명
R_OK	읽기 허용을 위한 시험.
W_OK	쓰기 허용을 위한 시험.
X_OK	실행 또는 탐색을 위한 시험.
F_OK	파일의 존재확인을 위한 시험.

상수 F_OK, R_OK, W_OK 및 X_OK, 그리고 식

R_OK|W_OK

R_OK|X_OK

및

R_OK|W_OK |X_OK

는 각각 서로 다른 값을 가져야 한다. 식에서 '|'는 비트단위의 OR 연산자이다.

3.10.2 *lseek*()함수용 심볼상수

lseek()함수에 의해 사용되는 상수들은 표 3-7과 같다.

<표 3-7> ***lseek*()함수용 심볼상수**

상 수	설 명
SEEK_SET	파일 오프셋을 *offset*값으로 기록한다.
SEEK_CUR	파일 오프셋을 현재의 오프셋 값에 *offset* 값을 더한값으로 기록한다.
SEEK_END	파일 오프셋을 EOF에 *offest* 값을 더한 값으로 기록한다.

3.10.3 POSIX용 컴파일시 심볼상수

표 3-8의 상수는 컴파일시에 어떤 부가적인 기능들이 존재하는가와 구현시 어떤 동작을 수행하여야 하는지를 결정하도록 응용에서 사용될 수 있다.

비록 POSIX를 엄격히 준수하는 응용에서 (어떤 구현물에서도 모든 값들의 이식성을 보장하기 위하여) 헤더 <unistd.h>로 부터 컴파일된 수치들은 아무런 의심없이 사용될 수 있지만, 실행시 현 시스템 구성상의 이점을 이용하기 위하여 어떤 값을 조사하도록 POSIX를 엄격히 준수하는 응용을 선택할 수 있다. *sysconf*()[5,8,1] 참조.

<표 3-8> **컴파일 시에 필요한 심볼상수**

이 름	설 명
{_POSIX_JOB_CONTROL}	이 심볼이 정의되면 심볼은 구현물이 작업제어(job control)를 지원함을 의미함.
{_POSIX_SAVED_IDS}	이 심볼이 정의되면 각 프로세스는 세이브드 셋-사용자-ID(saved set-user-ID)와 세이브드 셋-그룹-ID(saved set-group-ID)를 가짐.
{_POSIX_VERSION}	정수값 198808L임. 이 수치는 IEEE 표준국이 언제 표준을 승인하였는지를 나타내는 (4개 숫자로 표시되는) 년도와 (2개 숫자의)달로, 발간되는 표준별로 또는 본 표준의 개역시 변경된다.

3.10.4 POSIX용 실행시 심볼상수

표 3-9의 상수는 실행시 어떠한 부가적인 기능들이 제공되는지와 구현시 정의에 따라 본 표준에서 정의된 특정한 상태하에서 어떤 동작을 해야하는지를 결정하기 위하여 응용에서 사용될 수 있다.

표 3-9 의 어떠한 상수도 헤더 **<unistd.h>**에서 정의되지 않으면 이들 수치는 적용되는 파일에 따라 변화한다. *pathconf*() [6.7.1]을 참조하라.

표 3-9의 어떠한 상수도 헤더 **<unistd.h>**에서 -1을 가지도록 정의되면, 구현물은 어떠한 파일에 대해서도 선택사항을 제공하지 않는다. 즉, 헤더 **<unistd.h>**에서 -1이 아닌 다른 수치를 가지도록 정의되었다면, 구현물은 모든 적용가능한 파일에 대하여 선택사항을 제공해야 한다.

pathconf() 또는 *fpathconf*() 함수는 특정한 파일에 관한 표 3-9에 수록된 모든 상수의 값을 (이들이 **<unistd.h>**에 정의되었는지에 관계없이) 조사할 수 있다.

<표 3-9> **실행시 심볼상수**

이 름	설 명
{_POSIX_CHOWN_RESTRICTED}	*chown*()[6.6.5]함수의 사용은 특별한 권한을 가진 프로세스 및 파일의 그룹 ID를 프로세스의 유효(effective) 그룹 ID이나 보완 그룹 ID중의 하나로 변경하는 것으로 제한된다.
{_POSIX_NO_TRUNC}	경로명 구성요소가 {NAME_MAX}보다 길면 에러를 생성한다.
{_POSIX_VDISABLE}	이 값을 사용하면 **<termios.h>** [8.1.2]에서 정의된 터미날형 특수문자를 사용하지 못하도록 할 수 있다. *tcgetattr*() 과 *tcsetattr*()[8.2.1] 참조

제 4장 프로세스관련 기본 함수

본 장에서는 프로세스, 프로세스간의 신호 및 시간과 관련된 함수 등과 같은 기본적인 운영체제용 함수등에 대하여 기술한다. 본 표준에 설명된 프로세스의 모든 속성은, 해당 속성이 변경되었음을 명백히 기술하고 있지 않으면, 프로세스와 관련된 함수에 의하여 변경되지 않고 그대로 유지된다.

4.1 프로세스의 생성과 실행

4.1.1. 프로세스의 생성

함수 : *fork*()

4.1.1.1.용례

#include <sys/types.h>
pid_t fork()

4.1.1.2 설명

fork()함수는 새로운 프로세스를 생성한다. 새로 생성된 (자식)프로세스는 아래의 사항을 제외하고는 호출한 (부모) 프로세스와 동일한 복사본이어야 한다:

(1) 자식 프로세스는 별개의 프로세스 ID를 가진다. 또한 자식 프로세스 ID는 활동중인 모든 프로세스 그룹 ID와도 일치하지 않는다.
(2) 자식 프로세스는 부모 프로세스 ID와 다른 프로세스 ID를 가진다
(3) 자식 프로세스는 부모 프로세스의 파일 서술자를 복사한 자신의 복사본을 가진다. 각각의 자식 프로세스의 파일 서술자는 부모 프로세스의 파일 서술자가 가르키는 동일한 열린파일서술자를 참조한다.
(4) 자식 프로세스는 부모 프로세스의 열린 디렉토리 스트림(open directory stream)을 복사한 자신의 복사본을 가진다 (**디렉토리관련 함수**[6.1.2] 참조). 자식 프로세스에 속한 열린 디렉토리 스트림은 부모 프로세스의 디렉토리 스트림이 가르키는 디렉토리 스트림을 공유할 수 있다.
(5) 자식 프로세스에서 *tms_utime*, *tms_stime*, *tms_cutime* 및 *tms_cstime*등의 수치는 0이 된다(*times*() [5.5.2 참조].
(6) 부모 프로세스에 의해 기 설정된 파일 잠금(file lock)은 자식 프로세스로 상속되지 않는다(*fcntl*() [7.5.2] 참조)
(7) 실행되지 못한 경보(alarm)는 자식 프로세스에 의해 삭제된다(*alarm*() [4.4.1] 참조)
(8) 자식 프로세스를 기다리는 대기중인 신호(pending signal)는 없는 것으로 초기화된다(**<signal.h>** [4.3.1] 참조).

본 표준에 의해 정의된 프로세스 특성들 이외의 모든 사항은 부모 프로세스와 자식 프로세스에서 모두 동일하다. 본 표준에서 정의되지 않은 프로세스 특성의 계승에 대해서는 구현시 정의된바에 따른다

4.1.1.3 복귀값

fork()함수는 (실행이 성공적으로 완료되면) 자식 프로세스에게 0값을 복귀시키고, 부모 프로세스에게는 자식 프로세스의 프로세스 ID를 복귀시키며, 한 쌍의 (부모와 자식)프로세스는 *fork*()함수 이후 계속 실행된다. *fork*()함수의 실행이 실패하면, 부모 프로세스로 -1값을 되돌리고, 자식 프로세스는 생성되지 않는다. 이때, *errno*는 지정된 에러값으로 설정된다.

4.1.1.4 에러

아래에 주어진 하나의 조건이 발생하면 *fork*()함수의 복귀값은 -1이 되고, *errno*는 상응하는 수치로 설정된다.

[EAGAIN] 시스템에서 다른 프로세스를 생성하기 위한 자원이 부족하거나 하나의 사용자가 실행시킬 수 있는 프로세스의 총수가 시스템이 정한 갯수를 초과한다.

[ENOMEM] 프로세스가 요구하는 메모리 공간이 시스템에서 제공할 수 있는 공간을 초과한다

4.1.1.5 참고

alarm() [4.4.1], *exec*() [4.1.2], *fcntl*() [7.5.2], *kill*() [4.3.2], *time*() [5.5.2], *wait*() [4.2.1]

4.1.2 파일의 실행

함수: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

4.1.2.1 용례

int *execl*(*path*, *arg*,*arg*,...,*argn*,**(char *) 0)**
char **path*,**arg*0,*arg*1,...,**argn*;
int *execv*(*path*,*argv*)
char **path*,**atfv*[];
int *execle*(*path*,*arg*0,*arg*1,...,*argn*,**(char *)0,envp)**
char **path*,**arg*0,*arg*1,...,*argn*,*envp*[];
int *execve*(*path*,*argv*,,*envp*);
char **path*,**argv*[],**envp*[];
int *execlp*(*file*,*arg*0,*arg*1,...,*argn*,**(char *) 0)**
char **file*,*arg*0,*arg*1,...,*argn*
int *execvp*(*file*,*argv*)
char **file*,**argv*[];

exetern char ***environ** ;

4.1.2.2 설명

*exec*관련 함수는 현재의 프로세스 이미지를 새로운 프로세스 이미지로 대치한다. 새로운 이미지는 새로운 프로세스 이미지 파일(new process image file)이라 불리는 정규의 실행파일로부터 구성된다. *exec*함수의 성공적인 실행시에, 호출 프로세스 이미지는 새로운 프로세스 이미지에 의해 중복(overlay)되므로 복귀값은 없다.

이 함수의 호출 결과로 C 프로그램이 실행되면 다음과 같은 C-언어 함수의 호출과 같이 수행개시된다.

int main(*argc,argv*)
int *argc*;
char ***argv*;

여기서 *argc*는 인수의 갯수, *argv*는 인수를 수록하는 문자열의 배열이다. 추가로 다음 변수

extern char ***environ*;

는 환경 문자열(*environment string*)를 가르키는 문자 포인터의 배열을 가르키는 포인터로 초기화된다. *argv*와 *environ*배열은 각각 **널** 포인터로 끝난다. *argv*배열의 끝을 나타내는 널 포인터는 *argc*가 가지는 인수의 갯수에 포함되지 않는다.

프로그램이 *exec*함수에 명시한 인수는 *main*()의 인수와 동일한 방식으로 새로운 프로세스 이미지에 전달된다.

인수 *path*는 새로운 프로세스 이미지 파일을 지정하는 경로명을 가리킨다

인수 *file*은 새로운 프로세스 이미지 파일을 지정하는 경로명을 구성하는데 사용된다. 만일 *file*인수가 사선(/)을 포함하지 않으면 이 파일의 경로명 접두어는 환경 변수 **PATH** (**환경 서술** [3.7] 참조)로 주어진 디렉토리를 탐색하여 얻어진다. 만일 이러한 환경 변수가 존재하지 않으면 탐색의 결과는 구현시 정의된 바에 따른다.

인수 *arg*0, *arg1, ..., argn*은 **널(NULL)**로 끝나는 문자열을 가리키는 포인터이다. 이들 문자열은 새로운 프로세스 이미지에서 사용가능한 인수의 목록을 구성한다. 인수의 목록의 마지막은 널 포인터로 끝난다. 인수 *argv*0는 *exec*함수류에 의해 실행이 시작되는 프로세스의 파일명을 가르킨다.

인수 *argv*는 **널(NULL)**로 끝나는 문자열까지를 가르키는 문자 포인터의 배열이다. 이 배열이 마지막은 널 포인터로 끝난다. 이들 문자열은 새로운 프로세스 이미지에서 사용가능한 인수의 항목을 구성한다. *argv*[0]의 값은 *exec* 함수류에 의해 실행이 시작되는 프로세스와 관련된 파일명을 기르킨다.

인수 *envp*는 **널(NULL)**로 끝나는 문자열까지를 가르키는 문자 포인터의 배열이다. 이들 문자열은 새로운 프로세스 이미지를 위한 환경을 구성한다. *envp*배열의 마지막은 널 포인터이다

envp 포인터를 가지지 않는 형태의 함수류(*execl*(), *execv*(), *execlp*() 및 *execvp*()) 상에서의 새로운 프로세스 이미지 환경은 호출 프로세스에서의 외부변수 *environ*에 의해 얻어진다

새로운 프로세스의 인수와 환경목록을 표시하는 바이트의 최대길이는 {ARG_MAX}이다. 구현시에는 공백문자, 포인터, 또는 정렬 바이트(alignment byte)가 바이트 길이의 합계에 포함되는가를 시스템 문서(**문서화** [3.2.1.2] 참조)상에 기술하여야 한다.

호출 프로세스 이미지내에서 열린 파일 서술자는 close_on_*exec* 플래그인 FD_CLOEXEC가 세트된 경우(*fcntl*() [7.5.2], **<fcntl.h>** [7.5.2] 참조)를 제외하고는 새로운 프로세스 이미지에서도 열린채로 남아있는다. 파일 서술자가 열린채로 존재하기 위해서는 파일 잠금(*fcntl*() [7.5.2] 참조)을 포함하여, 열린 파일 서술의 모든 속성이 해당 함수의 호출에 의해 변경되지 않아야한다.

호출 프로세스 이미지상에서 '미리 정해진' (default) 동작을 수행하도록 지정된 신호(SIG_DFL)들은 새로운 프로세스 이미지상에서도 미리 정해진 동작을 수행하도록 지정되어야 한다. 호출 프로세스에 의해 무시되도록 지정된 신호들(SIG_IGN)은 새로운 프로세스 이미지상에서도 무시되도록 지정되어야 한다. 호출 프로세스 이미지에 의하여 받아들이기로 정해진 신호는 새로운 프로세스 이미지상에서도 '미리 정해진'동작을 수행하도록 지정된다(**<signal.h>** [4.3.1] 참조)

만일 새로운 프로세스 이미지 파일의 셋-사용자-ID (set-user-ID) 모드의 비트가 1로 세트되면(*chmod*() [6.6.4] 참조), 새로운 프로세스 이미지의 유효 사용자 ID는 새로운 프로세스 이미지 파일의 소유자 ID로 지정된다. 마찬가지로, 새로운 프로세스 이미지 파일의 셋-그룹-ID (set-group-ID) 모드의 비트가 1로 세트되면 새로운 프로세스이미지의 유효 그룹 ID는 새로운 프로세스 이미지 파일의 그룹 ID로 지정된다. 새로운 프로세스 이미지의 실제사용자 ID, 실제 그룹 ID 및 추가 그룹ID는 각각 호출 프로세스 이미지상에서의 대응되는 ID로 지정된다. 만일 {_POSIX_SAVED_IDS)가 정의되면, 새로운 프로세스 이미지의 유효 사용자 ID와 유효 그룹 ID는 (세이브드 셋-사용자-ID와 세이브드 셋-그룹-ID에서와 같이) *setudk*()함수에 의해 저장된다.

또한 새로운 프로세스 이미지는 다음과 같은 속성을 호출 프로세스 이미지로부터 계승한다:

(1) 프로세스 ID
(2) 부모 프로세스 ID
(3) 프로세스 그룹 ID
(4) 세션 멤버쉽(session membership)
(5) 실제 사용자 ID
(6) 실제 그룹 ID
(7) 추가 그룹 ID
(8) 경보시각신호까지 남은 시간(*alarm*() [4.4.1] 참조)
(9) 현재 작업 디렉토리
(10) 루트 디렉토리
(11) 파일 모드 생성 마스크(*umask*() [6.3.3] 참조)
(12) 프로세스 신호 마스크(*sigprocmask*() [4.3.5] 참조)
(13) 대기중인 신호류(*sigpending*() [4.3.6] 참조)
(14) *tms_utime*, *tms_stime*, *tms_cutime* 및 *tms_cstime* (*times*() [5.5.2] 참조)

본 표준에 의해 정의되었고 본절에 기술되지 않은 프로세스의 모든 속성들은 새로

운 프로세스 이미지와 원래의 프로세스 이미지상에서 같은 값을 지닌다. 본 표준에 의해 정의되지 않은 프로세스의 속성에 대한 계승은 구현시 정의된 바에 따른다.

exec 함수가 성공적으로 수행이 완료되면 *exec* 함수는 파일의 갱신을 위하여 *st_atime* 부분에 표식을 한다. 만일 *exec* 함수가 성공적으로 수행되지 못하나 프로세스 이미지 파일을 위치시키는 것이 가능하다면, 파일의 갱신을 위해 *st_atime* 부분이 표시되는지 여부는 규정되지 않는다. 만일 *exec* 함수가 성공적이면 해당 프로세스 이미지 파일은 *open*()된 것으로 간주한다. 어떤 파일이 열리면, 이후에 이에 상응하는 *close*() 함수가 실행되는 것으로 간주한다.

4.1.2.3 복귀값

exec 함수들 중의 하나가 호출 프로세스로 어떤 값을 복귀하면, 에러가 발생했음을 의미한다. 즉, 복귀값은 -1이고 *errno*는 에러의 종류를 가리킨다.

4.1.2.4 에러

아래 조건중의 한가지가 발생하면, *exec* 함수의 복귀값은 -1이 되고 *errno*는 상응하는 값으로 설정된다.

[E2BIG] 새로운 프로세스 이미지의 인수 목록과 환경목록에 의해 사용되는 바이트의 길이가 시스템이 제한하는 길이인 {ARG_MAX}보다 크다.

[EACCES] 새로운 프로세스 이미지 파일의 경로명 접두어에 수록된 디렉토리에 대한 탐색 허용(search permission)을 부정, 새로운 프로세스 이미지 파일이 실행 허용(execution permission)을 부정, 또는 새로운 프로세스 이미지 파일이 정규 파일이 아니고 구현물이 해당 형식을 가진 파일의 실행을 지원하지 못한다.

[ENAMETOOLONG] 경로나 파일 인수의 길이, 또는 어떤 파일 앞에 첨가한 환경변수 **PATH**의 요소가 {PATH_MAX} 길이를 초과하거나, 경로명 구성요소가 {NAME_MAX} 길이보다 크고 해당 파일에 대하여 {_POSIX_NO_TRUNC}수치가 영향을 미친다.

[ENOENT] 새로운 프로세스 이미지 파일의 경로명에서 하나 이상의 구성요소가 존재하지 않거나, 경로 또는 파일 인수가 공백문자열을 가리킨다.

[ENOTDIR] 새로운 프로세스 이미지 파일의 경로명 접두사의 구성요소가 디렉토리가 아니다.

아래의 조건이 발생하면, *execl*(), *execv*(), *execle*() 및 *execve*() 함수는 -1을 되돌리고, *errno*는 상응하는 값으로 설정된다

[ENOEXEC] 새로운 프로세스 이미지 파일이 적절한 접근허용을 가지고 있으나,적당한 형식이 아니다

아래의 조건이 발생하면, *exec* 함수는 -1을 되돌리고, *errno*는 상응하는 값으로 설정된다.

[ENOMEM] 새로운 프로세스 이미지가 하드웨어나 시스템에 의해 제한되는 메모리에 비하여 더 많은 메모리를 요구한다.

4.1.2.5 참고

alarm() [4.4.1], *chmod*() [6.6.4], *_exit*() [4.4.2], *fcntl*() [7.5.2], *fork*() [4.1.1], *setuid*() [5.2.2], **<signal.h>** [4.3.1], *sigprocmask*() [4.3.5], *sigpending*() [4.3.6], *stat*() [6.6.2], **<sys/stat.h>** [6.6.1], *times*() [5.5.2], *umask*() [6.3.3], **환경 서술** [3.7].

4.2 프로세스의 종료

프로세스의 종료에는 두가지 방법이 있다

(1) 정상적 종료는 *main*()으로부터의 되돌림에 의하여, 또는 *exit*()나 *_exit*()함수에 의하여 요구될 때 발생함.

(2) 비정상적 종료는 *abort*()함수에 의하여 요구되거나 어떤 신호를 받았을 때 (**<signal.h>** [4.3.1] 참조)발생함

exit()와 *abort*() 함수는 C-표준에서 설명된 것과 같다. *exit*()와 *abort*()는 모두 *_exit*() [4.2.2]에 규정된 결과에 따라 프로세스를 종료한다. 그러나, *abort*()에 의하여 *wait*()나 *waitpid*()가 가용하게 되는 상태가 SIGQBRT신호에 의해 종료되는 프로세스의 상태일 때는 제외한다.

부모 프로세스는 *wait*()나 *waitpid*()함수를 이용하여 자식 프로세스가 종료되기를 기다리도록 할 수 있다.

4.2.1 프로세스의 종료 대기

함수: *wait*(), *waitpid*()

4.2.1.1 용례

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait (stack_loc)
int *stat_loc;

pid_t watipid (pid,stat_loc,options)
pid_t pid;

int *stat_loc;
int options;
```

4.2.1.2 설명

호출 함수에서 *wait*()와 *waitpic*() 함수는 호출함수의 자식 프로세스에대한 상태정보를 제공한다. 여러가지 부가사항들을 이용하여 종료 또는 일시 중지된 자식 프로세스의 상태정보를 알아낸다. 만일 두개 이상의 자식 프로세스에 관한 가용한 상태정보가 존재하면 이들의 상태를 알려주는 순서는 규정되지 않는다

wait() 함수는 다음과 같은 상태에서 호출 프로세스의 실행을 지연 시킨다. 즉, 호출 프로세스의 자식 프로세스들 중에서 종료된 하나의 프로세스에 대한 상태정보가 가용하기 전까지, 또는 신호포착 함수(signal-catching function)를 실행시키거나 해당 프로세스를 종료시키는 동작을 수행하는 신호가 전달될 때까지 이들 호출 프로세스의 실행이 지연된다. 만일 상태정보가 *wait*()의 호출에 앞서서 가용하다면, 프로세스로부터의 복귀는 즉시 수행된다.

만일 *pid* 인수가 -1값을 가지며 *options* 인수가 0값을 가지면, *waitpid*() 함수는 *wait*()와 동일하게 동작한다. 이외의 경우에서는 *pid*의 값과 *options* 인수에 의하여 작동방식이 변경된다.

pid 인수는 (상태가 요구되는) 자식 프로세스의 집합을 규정한다. *waitpid*() 함수는 다만 이 집합에 속한 자식 프로세스의 상태를 복귀한다.

(1) *pid*가 -1 이면 어떤 자식 프로세스를 위한 상태를 알고자함. 이때 *waitpid*() 는 *wait*() 동일함.
(2) 만약 *pid*가 0 보다 크면 이것은 상태정보를 알고자하는 단일 자식 프로세스의 프로세스 ID를 의미함
(3) 만약 *pid*가 0 이면 프로세스 그룹 ID가 호출 프로세스의 그것과 같은 자식 프로세스의 상태를 알고자 함
(4) 만약 *pid*가 -1 보다 작으면 프로세스 그룹 ID와 *pid*값의 절대치가 같은 자식 프로세스의 상태를 알고자함.

options 인수는 헤더 **<sys/wait.h>**에 정의된 바와 같이 아래의 플래그와의 비트단위 OR-연산으로 구성된다

WNOHANG *pid*로 규정된 자식 프로세스 중의 하나에 대한 상태정보가 즉시 가용하지 않으면, *waitpid*()함수는 호출 프로세스의 실행을 지연시키지 못한다.

WUNTRACED 구현물이 작업제어를 지원한다면, *pid*로 규정된(종료된) 자식 프로세스와 종료된 이후로 상태가 보고된 적이 없는 프로세스의 상태정보는 정보를 요구하는 프로세스로 함께 제공되어야 한다.

만일 자식 프로세스의 상태정보가 가용하다면 함수 *wait*()나 *waitpid*() 함수는 자식 프로세스의 프로세스 ID 값을 복귀한다. 이 경우에 인수 *stat_loc* 값이 **널(NULL)**이 아니면 정보는 *stat_loc*이 지시하는 장소에 저장된다. 어떤 복귀값이 *main*()으로부터 0 값을 돌아온 자식 프로세스나, *exit*()나 *_exit*()의 *status* 인수로써 0값을 전달한 종료된 자

식 프로세스로 부터의 되돌린 값이 될 필요충분조건은 *stat_loc*이 지시하는 위치의 값이 0 이 되어야 하는 것이다. 이 정보는 주어진 수치의 값에 관계없이 아래의 매크로에 의해 해석되어진다. 여기서 매크로는 **<sys/wait.h>**에 정의된다. 또한 매크로의 *stat_val* 인수는 *stat_loc*에 의해 지시되는 정수값이다.

WIFEWITED(*stat_val*)
정상적으로 종료된 자식 프로세스에 상태가 돌아오면 이 매크로는 0이 아닌 수치로 평가된다.

WEXITSTATUS(*stat_val*)
만약 이 매크로의 값이 0이 아니면 이 매크로는 자식 프로세스가 *_exit*()나 *exit*()로 전달하는 *status* 인수의 하위 8비트를 취하거나, 자식 프로세스가 *main*()로 복귀값으로 평가된다

WIFSIGNALED(*stat_val*)
이 매크로는 (만일 포착되자 않은 신호의 수신(**<signal.h>** [4.3.1] 참조)으로 인하여 종료된 자식 프로세스에 대한 상태를 복귀하면) 0이 아닌 값을 가진다.

WTERMSIG(*stat_val*)
WIFSIGNALED(*stat_val*) 값이 0이 아니면 이 매크로는 자식 프로세스를 종료시킨 신호의 번호로 평가된다.

WIFSTOPPED(*stat_val*)
이 매크로는 (현재 중지된 자식 프로세스에 대한 상태를 복귀하면) 0이 아닌 값을 가진다.

WSTOPSIG(*stat_val*)
만일 WIFSTOPPED(*stat_val*)의 값이 0이 아니면 이 매크로는 자식 프로세스를 종료시킨 신호의 번호로 평가된다.

*stat_loc*으로 지시되는 위치에 수록된 정보가 WUNTRACED 플래그를 규정하는 waitpid()함수를 호출함에 의하여 이곳에 수록되었다면, WIFEXITED(**stat_val*), 또는 WIFSIGNALED(**stat_val*), 또는 WIFSTOPPED(**stat_val*) 중에서 단지 하나의 매크로만이 0이 아닌 값으로 평가된다. *stat_loc*이 지시하는 위치에 수록된 정보가 WUNTRACED 플래그를 규정하지 않는 *waitpid*()함수나 *wait*()함수의 호출에 의하여 이곳에 수록되었다면, WIFEXITED(**stat_val*)과 WIFSIGNALED(**stat_val*) 중에서 단지 하나의 매크로만이 0이 아닌 값으로 평가된다.

어떤 구현에서는 *wait*()나 *waitpid*()가 알려주는 특정한 상태들에 대해서도 추가적으로 정의할 수 있다. 이것은 호출 프로세스 또는 호출 프로세스의 자식 프로세스중의 어떤 것이 표준이 아닌 확장을 명시적으로 사용하지 않는한 발생하지 않는다. 이 경우, 알려진 상태에 대한 해석방법은 구현시 정의된 바에 따른다.

만일 모든 자식 프로세스의 종료를 기다리지 않은채 부모 프로세스가 종료되면, 남겨진 자식 프로세스는 구현시 정의된 시스템 프로세스에 대응하는 새로운 부모 프로세스 ID가 주어진다.

4.2.1.3 복귀값

wait()나 *waitpid*()함수가 가용한 자식 프로세스의 상태를 복귀하면, 이 함수는 상태를 알려주는 자식 프로세스의 프로세스 ID 값을 복귀한다. 만일 *wait*() 또는 *waitpid*() 함수가 호출 프로세스의 신호전달에 의하여 어떤 복귀값을 가진다면, -1이 되고, *errno*는 [EINTR] 값이 된다. 만일 *waitpid*() 함수가 *options*을 WNOHANG로 설정하고 호출

되면, 이 함수는 상태가 가용하지 않은 프로세스의 *pid*로 규정된 최소한 하나의 자식 프로세스를 가지며. *pid*로 규정된 어떤 프로세스에 대한 상태도 가용하지 않으며, 복귀값은 0이 된다. 이외의 경우에 복귀값은 -1이 되고 *errno*는 지정된 에러값으로 설정된다.

4.2.1.4 에러

다음 조건하에서 *wait*()함수의 복귀값은 -1이 되고, *errno*는 상응하는 값으로 설정된다

[ECHILD] 호출 프로세스가 대기중인 자식 프로세스를 가지고 있지 않다
[EINTR] 함수가 어떤 신호에 의해 인터럽트를 받는다. *stat_loc*에 의해 지시하는 위치에 수록된 값이 미정의된다.

다음 조건하에서 *waitpid*()함수의 복귀값은 -1 이며, *errno*는 상응하는 값으로 설정된다

[ECHILD] 프로세스 또는 *pid*로 규정된 프로세스 그룹이 존재하지 않거나, 이들이 호출 프로세스의 자식 프로세스가 아니다
[EINTR] 함수가 어떤신호에 의해 간섭됨. *stat_loc*에 의해 지시되는 위치에 수록된 값이 미정의된다
[EINVAL] options 인수의 값이 유효하지 않다

4.2.1.5

_exit() [4.2.2], *fork*() [4.1.1], *pause*() [4.4.2], *times*() [5.5.2],
<signal.h> [4.3.1].

4.2.2 프로세스의 종료

함수: *_exit*()

4.2.2.1 용례

void _exit (*status***)**
int *status*;

4.2.2.2 설명

_exit()함 수는 아래의 절차에 따라 호출 프로그램을 종료시킨다
(1) 호출 프로세스내의 모든 열린 파일 서술자와 디렉토리 스트림(stream)이 닫혀짐.
(2) 호출 프로세스의 부모 프로세스가 *wait*()나 *waitpid*()를 실행중이면, 호출 프로세스의 종료를 알려주고, *status*의 하위 8비트가 가용하도록 한다. *wait*() [4.2.1]

참조.

(3) 호출 프로세스의 부모 프로세스가 *wait*()나 *waitpid*()를 실행중이 아니면, 부모 프로세스가 적절한 순서로 *wait*()나 *waitpid*()를 실행할 때마다 부모 프로세스로 복귀시키기 위한 *status*의 값이 저장된다.

(4) 어떤 프로세스의 종료는 자식 프로세스를 직접적으로 종료시키지 않는다. 아래에 기술한 바와 같이 SIGHUP신호를 송신함으로써 어떤 상태하에서 자식 프로세스를 간접적으로 종료시킨다. 종료된 프로세스의 자식 프로세스는 구현시 정의된 시스템 프로세스에 상응하는 새로운 부모프로세스 ID가 주어진다.

(5) 구현물이 SIGCHLD신호를 지원하면, SIGCHLD신호는 부모 프로세스로 전송된다

(6) 제어 프로세스의 경우에 SIGHUP신호는 호출 프로세스에 속하는 제어 터미널의 포어그라운드 프로세스 그룹내의 각 프로세스로 전송된다

(7) 제어 프로세스의 경우에, 세션과 관련된 제어 터미널은 세션으로부터 분리되며, 이때 새로운 제어 프로세스에 의해 제어 터미날이 얻어지도록 한다

(8) 구현물이 작업제어를 지원하고, 프로세스를 빠져나옴(exit of process)으로 인해 프로세스 그룹을 고아 프로세스가 되도록하며, 새로운 고아 프로세스 그룹에 속한 어떤 프로세스가 중지되면, SIGCONT신호에 이은 SIGHUP신호는 새로운 고아 프로세스 그룹에 속한 각 프로세스로 전송된다.

이 들 과정은 어떠한 이유에 의해서든지 프로세스가 종료될 때 수행된다.

4.2.2.3 복귀값

_exit()함수는 이 함수의 호출 프로세스로 돌아가지 않는다.

4.2.2.4 참고

close() [7.3.1], *sigaction*() [4.3.4], *wait*() [4.2.1].

4.3 신호류

4.3.1 신호의 개념

4.3.1.1 신호명

헤더 **<signal.h>**는 *sigset_t*형식과 *sigaction*의 구조를 선언한다. 또한 헤더에서는 *void*(*)()형식의 독특한 상수표현방식으로 확장하는 아래와 같은 심볼 상수를 정의한다. 여기서 *void*(*)()형식의 값은 선언 가능한 함수와 일치되지 않는다.

심볼상수	설 명
SIG_DFL	자동적으로 결정되는(default) 신호 처리에 대한 요구
SIG_IGN	신호가 무시되어야 함을 요구

*sigset_t*형식은 신호의 집합을 나타내는데 사용된다. 이 형식은 항상 복합형(integral)이거나 구조형식이다. *sigset_t*형식의 대상을 다루는데 사용되는 여러가지 함수는 sigsetops [4.3.3]에서 정의된다.

헤더 **<signal.h>**에서는 시스템내에서 발생하는 신호를 참조하는데 사용되는 상수들도 선언한다. 본 표준에 의해 정의되고 구현물에 의해 지원되는 각각의 신호는 독특한 양의 정수값을 가진다. 0 값은 널 신호로 예약되어 있다. (*kill*() [4.3.2] 참조) 구현물에서는 시스템에서 발생가능한 추가적인 신호들을 정의할 수 있다.

표 4.1에 수록한 상수는 모든 구현물에서 지원되어야 한다

표 4.2에 수록한 상수는 모든 구현물에서 정의되어야 한다. 그러나 작업제어를 지원하지 않는 구현물에서는 이들 신호를 지원할 필요가 없다. 구현물에서 이들 신호가 지원되지 않는다면, 이들은 본 표준에 따라 동작하여야 한다. 이외의 경우에서, 구현물은 이들 신호를 생성하지 않아야 하며, 이들 신호를 전송하려고 하거나 이들의 동작에 대한 시험이나 규정하려는 시도는 에러조건을 복귀시켜야 한다.

kill() [4.3.2] 및 *sigaction*() [4.3.4] 참조.

<표 4-1> **요구되는 신호류**

심볼상수	자동적으로 수행되는 동작	내 용
SIGABRT	1	비정상적인 종료신호, 예를 들어 C-표준에 정의된 *abort*() 신호에 의해 발생됨.
SIGALRM	1	시간이 종료되었음. 예를 들어 *alarm*() 함수에 의해 발생됨 (*alarm*() [4.4.1] 참조)
SIGFPE	1	잘못된 계산, 예를 들어 0으로 나누는 나눗셈 또는 오버플로우를 야기하는 연산등.

SIGHUP	1	제어 터미날과 연결된 회선이 끊어짐 발생 (**모뎀의 단절** [8.1.1.10] 참조)
SIGILL	1	부적당한 하드웨어 명령어의 검출
SIGINT	1	인터액티브 종료 신호 (**특수 문자** [8.1.1.9] 참조)
SIGKILL	1	(포착될 수 없거나 무시될 수 없는) 종료신호
SIGPIPE	1	읽기가 허용되지 않는 파이프에의 쓰기 (write [7.4.2] 참조).
SIGQUIT	1	인터액티브 종료 신호 (특수 문자 [8.1.1.9] 참조)
SIGSEGV	1	부적당한 메모리 참조의 검출
SIGTERM	1	종료신호
SIGUSR1	1	응용에 따른 신호용(1)으로 지정됨
SIGUSR2	1	응용에 따른 신호용(2)으로 지정됨

표 4.1과 4.2에서 자동적으로 수행되어야 하는 동작은 다음과 같다.

1. 프로세스를 비정상적으로 종료함.
2. 신호를 무시함.
3. 프로세스를 중지함.
4. 현재 중지된 프로세스는 재개하고, 이외의 경우에서는 신호를 무시함.

<표 4-2> 작업제어신호류

심볼상수	자동적으로 수행되는 동작	내 용
SIGCHLD	2	자식 프로세스가 종료되거나 중지됨
SIGCONT	4	중지시 재개됨
SIGSTOP	3	(포착될 수 없거나 무시될 수 없는) 중지신호
SIGTSTR	3	인터액티브 중지 신호 (특수 문자 [8.1.1.9] 참조)
SIGTTIN	3	백그라운드 프로세스 그룹의 프로세스에 의한 제어 터미날의 읽기 (**터미날 접근제어** [8.1.1.4] 참조)
SIGTTOU	3	백그라운드 프로세스 그룹의 프로세스에 의한 제어 터미날의 쓰기 (**터미날 접근제어** [8.1.1.4] 참조)

4.3.1.2 신호의 생성과 전달

어떤 신호를 처음 생성하는 사건(event)이 발생될 때를 해당 프로세스에 필요한 신호가 생성되었다고 한다. 이러한 사건의 예로, *kill*()함수의 호출을 포함하여, 하드웨어의 고장발견, 타이머의 종료(time expiration) 및 터미널 관련활동 등이 있다. 어떤 환경에서는 하나의 사건이 여러개의 프로세스에 필요한 신호들을 생성하기도 한다.

신호별로 이에 응답하는 각각의 프로세스마다의 동작은 시스템에 의해 정의된다(**신호에 의한 작동** [4.3.1.3] 참조). 이때, 해당되는 프로세스가 정상적으로 동작하고 신호가 처리되면 신호가 프로세스로 전달되었다고 한다.

신호의 생성으로부터 전달될 때까지를 대기중(pending)이라고 한다. 보통 이 시간간격은 응용에 의해 측정될 수 없다. 그러나 신호가 어떤 프로세스로 전달되는 과정에서 차단(blocked)될 수 있다. 만일 어떤 차단된(blocked) 신호에 의해 취해지는 동작이 이 신호를 무시하는 동작이 아니며 만일 이 신호가 해당 프로세스를 위하여 생성되었다면, 이 신호는 차단상태가 해제되던가 이 신호에 관계된 동작이 이 신호를 무시하도록 지정되지 않는 한 이 신호는 계속 대기중(panding)이어야 한다. 만일 차단된 신호와 관계된 동작이 신호를 무시하고 만일 이 신호가 해당 프로세스를 위하여 생성되었다면, 신호가 생성되자마자 곧 폐기되던지 또한 계속 대기중으로 남아 있게 되는 지는 규정되지 않는다.

각 프로세스는 프로세스로 전달되는 신호가 차단되어 있는지를 정의하는 신호 마스크(signal mask)를 가지고 있다. 어떤 프로세스의 신호 마스크는 이 프로세스의 부모 프로세스에 의하여 초기화된다. *sigaction*(), *sigprocmask*() 및 *sigsuspend*() 함수들은 신호 마스크의 조작을 제어한다.

어떤 신호에 응답하여 취해지는 동작을 결정하는 것은 신호가 전달되는 시점에서 이루어진다. 이 결정은 최초에 신호가 생성되는 수단과는 별개이다. 대기중인 신호가 계속해서 발생하면 이들 신호가 한차례 이상 전달되는 것에 관해서는 구현시 정의된 바에 따른다. 동시에 여러개의 대기중인 신호가 하나의 프로세스로 전달되는 순서는 규정되지 않는다.

어떤 프로세스를 위하여 어떤 중지신호(SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU)가 생성될 때, 이 프로세스를 위한 모든 대기중인 SIGCONT 신호들은 폐기된다. 반대로, 어떤 프로세스를 위한 SIGCONT 신호가 생성될 때에는, 모든 대기중인 중지 신호들은 폐기되어야 한다. 중지된 프로세스를 위하여 SIGCONT신호가 생성될 때에는, 비록 SIGCONT신호가 차단되거나 무시되더라도 이 프로세스는 재개되어야 한다. 만일 SIGCONT가 차단되나 무시되지 않으면, 이신호는 차단이 해제되거나 이 프로세스를 위하여 중지신호가 생성될 때까지 이 신호는 계속 대기중으로 남는다.

구현물은 어떤상태에서 신호들을 생성하는 지에 대하여 본 표준에 규정되지 않은 모든 조건에 대하여 문서화하여야 한다(**문서화** [3.2.1.2] 참조).

4.3.1.3 신호에 의한 작동

각각의 신호와 관계된 동작은 SIG_DFL, SIG_IGN 및 함수에의 포인터(pointer to a function)의 세가지로 분류된다. 모든 신호는 최초 *main*()루틴에 진입하기 전에 SIG_DFL 또는 SIG_IGN 으로 설정된다(*exec* [4.1.2] 참조). 이 값에 의해 수행되는 동작은 다음과 같다

(1) SIG_DFL — 신호에 직접 관련된, 자동적으로 지정되는 동작
   (a) 본 표준에 정의된 신호에 대해 자동적으로 지정되는 동작은 전술한 목록에 규정되어 있다
   (b) 만일 자동적으로 지정되는 동작이 해당 프로세스를 중지시킨다면, 그 프로

세스의 실행은 잠시 동안 멈춘다. 어떤 프로세스가 중지될 때, 이 프로세스의 부모 프로세스가 SA_NOCLDSTOP 플래그를 세트하지 않으면, 부모 프로세스를 위한 SIGCHILD 신호가 생성된다(*sigaction*() [4.3.4] 참조). 어떤 프로세스가 중지된 동안 이 프로세스로 보내지는 (SIGKILL을 뺀) 추가적인 신호들은 이 프로세스가 재개될 때까지 전달되지 않아야 한다. 고아 프로세스 그룹에 속한 프로세스는 SIGTSTP, SIGTTIN 또는 SIGTTOU신호에 응답하여 중지되는 것은 허용되지 않는다. 이들 신호중의 하나가 전달됨에 의하여 그러한 프로세스를 중지시킬 경우 이 신호는 폐기되어야 한다.

(c) 대기중인, 자동적으로 지정되는 동작이 어떤 신호(예를 들어 SIGCHILD)를 무시하는 신호에 의한 동작을 SIG_DFL로 설정하는 것은 이 신호 차단에 관계없이 대기중인 신호의 폐기를 야기한다.

(2) SIG_IGN – 신호의 무시

(a) 이 신호의 전달은 프로세스에 아무런 영향을 끼치지 않는다. 이 신호가 (C-표준으로 정의된 *kill*()함수나 *raise*()함수에 의해 생성되지 않은) SIGFPE, SIGILL 또는 SIGSEGV 신호를 무시한 후에는 해당 프로세스의 작동방식이 미정의 된다.

(b) 시스템은 SIGKILL 또는 SIGSTOP 신호가 SIG_IGN로 설정하는 동작을 허용하지 않는다.

(c) 대기중인 어떤 신호의 동작을 SIG_IGN으로 설정하는 것은 그것이 차단되느냐에 관계없이 대기중인 신호의 폐기를 야기한다.

(d) 어떤 프로세스가 SIGCHLD 신호에 의한 동작을 SIG_IGN로 설정하면, 작동방식은 규정되지 않는다.

(3) 함수로의 포인터(pointer to a function) – 신호의 포착

(a) 신호의 전달시, 신호를 수신하는 프로세스는 특정한 번지의 신호포착 함수를 실행한다. 신호포착 함수로부터 복귀한 후에 수신 프로세스는 이 프로세스가 인터럽트(interrupt)된 지점에서 실행이 재개된다.

(b) 신호포착 함수는 다음과 같이 C-언어 함수로 호출한다.

void **func**(*signo*)
int *signo*;
여기서 *func*는 규정된 신호포착 함수이며, *signo*는 전달되어 온 신호의 신호번호이다

(c) 어떤 프로세스 (C-표준에 의해 정의된 *kill*()함수나 *raise*()함수에 의해 생성되지 않은 SIGFPE, SIGILL 또는 SIGEGV신호에 대한) 신호포착함수로부터 정상적으로 돌아 온 후에는 이 프로세스의 작동방식은 정의되지 않는다.

(d) 시스템은 어떤 프로세스가 SIGKILL 또는 SIGSTOP 신호를 포착하는 것을 허용하지 않는다

(e) 어떤 프로세스가 종료된 자식 프로세스에 대한 대기(wait)를 하지 않고 SIGCHLD 신호에 대해서 신호포착 함수를 수행한다면. SIGCHLD신호가 해당된 자식 프로세스를 지시하기 위하여 생성되는지 여부를 규정되지 않는

다

(f) 신호포착 함수가 프로세스 실행과 동기하여 호출되지 않을 때 만일 신호포착 함수가 본 표준에서 정의된 몇몇 함수들을 호출하면 이들 함수들의 작동방식은 규정되지 않는다. 다음 표는 신호와 관련된 재진입가능(reentrant) 함수들을 정의한다. 이것은 응용이 이들을 제한없이 신호포착 함수로 부터 호출할 수 있음을 의미한다.

_exit()	*getegid*()	*rename*()	*tcdrain*()
access()	*geteuid*()	*rmdir*()	*tcflow*()
alarm()	*getgid*()	*setgid*()	*tcflush*()
cfgetispeed()	*getgoups*()	*setpgid*()	*tcgetattr*()
cfsetispeed()	*getpgrp*()	*setsid*()	*tcgetpgrp*()
cfgetispeed()	get*pid*	*setuid*()	*tcsendbreak*()
cfgetispeed()	*getppid*()	*sigaction*()	*tcsetattr*()
chdir()	*getuid*()	*sigaddset*()	*tcsetpgrp*()
chmod()	*kill*()	*sigdelset*()	*time*()
chown()	*link*()	*sigemptyset*()	*times*()
close()	*lseek*()	*sigfillset*()	*umask*()
creat()	*mkdir*()	*sigismember*()	*uname*()
dup2()	*mkfifo*()	*sigpending*()	*unlink*()
dup()	*open*()	*sigprocmask*()	*ustat*()
*exec*le()	*pathconf*()	*sigsuspend*()	*utime*()
execve()	*pause*()	*sleep*()	*wait*()
fcntl()	*pipe*()	*stat*()	*waitpid*()
fork()	*read*()	*sysconf*()	*write*()
fstat()			

위의 항목에서 누락된 모든 IEEE std 1003.1-1988함수와 C-표준에 의해 정의된 함수중에서 신호포착 함수로부터 호출가능한지가 언급되지 않은 모든 함수는 신호에 관한한 불확실한 (unsafe)것으로 간주된다. 만일 불확실한 어느 함수를 호출한 신호포착 함수가 어떤 불확실한 함수를 인터럽트(interrupt)하면, 작동방식은 정의되지 않는다.

4.3.1.4 기타 함수에서의 신호효과

본 표준에서 정의한 특정한 함수를 실행하는 어떤 프로세스로 신호가 전달되면 이 신호는 이들 함수의 작동방식에 영향을 미친다. 만일 신호의 동작이 프로세스를 종료시키려고 하면, 프로세스는 종료되고 함수는 복귀하지 않는다. 신호의 동작이 프로세스를 중지시키려고 하면, 이 프로세스가 다시 수행되거나 종료될 때까지 이 프로세스는 중지된다. 어떤 프로세스에 대한 SIGCONT신호의 생성은 이 프로세스가 계속 수행되도록 하며, 원래의 함수는 이 프로세스가 중지된 지점에서 재개된다. 신호의 동작이 신호포착 함수를 호출하는 것이라면, 신호포착 함수가 호출된다. 이 경우 원래의 함수

가 신호에 의해 인터럽트되었다고 한다. 만일 신호포착 함수가 복귀되면 인터럽트된 함수의 작동방식은 해당 함수별로 서술된 바에 따른다. 무시되는 신호는 모든 함수의 작동방식에 아무런 영향을 미치지 못한다. 또한 차단된 신호도 이들 신호가 전달될 때까지 모든 함수의 작동방식에 영향을 미치지 못한다.

4.3.2 프로세스로의 신호전송

함수: *kill*()

4.3.2.1 용례

#include <sys/types.h>
#include <signal.h>
int kill (*pid*,sig)

pid_t *pid*;
int sig;

4.3.2.2 설명

kill()함수는 *pid*로 규정된 프로세스나 프로세스 그룹으로 신호를 전송한다. 전송되는 신호는 *sig*로 규정되고, 이것은 **<signal.h>** [4.3.1]에 수록된 항목중의 하나이거나 0이어야 한다. *sig*가 0(널신호 - null signal)이면, 에러검사가 수행되나 신호는 실제로 전송되지 않는다. 널 신호는 *pid*의 유효성을 검사하는데 이용된다.

*pid*로 지정된 프로세스로 신호를 전송하도록 허용된 어떤 프로세스에 대하여 적절한 권한이 없는 한, 신호를 전송하는 프로세스의 실제 사용자 ID 또는 유효 사용자 ID는 특별한 권한을 가지지 않으면 신호를 수신하는 프로세스의 유효 사용자 ID 또는 유효 사용자 ID와 일치되어야 한다. 만일 {_POSIX_SAVED_IDS}가 정의되면, 수신 프로세스의 세이브드 셋-사용자-ID는 유효 사용자 ID를 대신해서 검사된다. 만일 어떤 수신 프로세스의 유효 사용자 ID가 S_ISUDI 모드 비트 (**<sys/stat.h>** [6.6.1] 참조)의 이용을 통하여 변경되었으면, 구현물에서는 응용이 부모 프로세스 또는 동일한 실제 사용자 ID를 가진 프로세스에 의해 전송된 신호를 수신하는 것을 허용할 수 있다

만약 *pid*가 0보다 크면, *sig*는 프로세스 ID가 *pid*와 같은 프로세스로 보내진다

만약 *pid*가 0이면 *sig*는 구현시 정의되는 시스템 프로세스를 제외한 (프로세스 그룹 ID가 전송프로세스의 프로세스 그룹 ID와 같고 해당 프로세스로 신호를 전송하도록 허용된) 모든 프로세스로 전송된다.

만약 *pid*가 -1이면 *kill*()함수의 작동방식은 규정되지 않는다.

만약 *pid*가 음수이나 -1이 아니면, *sig*는 (프로세스 그룹 ID가 *pid*의 절대값과 같으며, 신호를 전송하도록 허용된) 모든 프로세스로 전송된다.

*pid*값에 의해 전송 프로세스로 보내질 *sig*가 생성되었으며, 또한 *sig*가 차단되지 않았다면, *sig* 또는 적어도 대기중인 차단되지 않은 신호중의 하나는 *kill*()함수가 복귀되기 전에 전송프로세스로 전달된다.

SIGCONT신호를 지원하는 구현의 경우에, 위에서 설명한 사용자 ID의 검사는 SIGCONT를 전송한 프로세스와 동일한 세션의 프로세스로 보낼때에는 적용되지 않는다.

확장된 보안제어를 제공하는 구현의 경우에는 널신호를 포함하여 신호의 전송에 대하여 구현시 정의된 더많은 제한이 부과된다. 특정한 경우에 시스템은 *pid*로 규정된 일부 또는 전체 프로세스의 존재를 부정할 수 있다.

만일 어떤 프로세스가 *pid*로 규정된 프로세스로 *sig*를 전송할 수 있는 권한이 있으

면 *kill*()함수는 성공적으로 수행된다. 만일 *kill*()함수가 수행되지 못한다면 아무런 신호도 전송되자 않는다.

4.3.2.3. 복귀값

함수가 성공적으로 완료되면 복귀값은 0이 된다. 그 외의 경우에는 -1이 복귀되며, *errno*는 상응하는 에러값으로 설정된다.

4.3.2.4 에러

다음 조건이 발생하면 *kill*()함수의 복귀값은 -1이고, *errno*는 상응하는 값으로 설정된다.

[EINVAL] *sig*인수의 값이 부적당하거나 지원되지 않는 번호의 신호이다
[EPERM] 프로세스는 수신 프로세스로 신호를 전송하는 권한을 가지고 있지 않다.
[ESRCH] *pid*로 규정된 상응하는 프로세스나 프로세스 그룹을 찾을 수 없다.

4.3.2.4 참고

getpid() [5.1.1], *setpid*() [5.3.2], sigadtion [4.3.4], **<signal.h>** [4.3.1].

4.3.3 신호집합에 관한 연산 (Manipulate Signal Sets)

함수: *sigemptyset*(), *sigfillset*(), *sigaddset*(), *sigdelset*(), *sigismember*()

4.3.3.1 용례

```
#include <signal.h>

int sigemptyset(set)
sigset_t *set;

int sigfillset(set)
sigset_t *set;

int sigaddset(set,signo)
sigset_t *set;
int *signo;
int sigdelset(set,signo)
sigset_t *set;
int *signo;
int sigismember(set,signo)
```

sigset_t **set*;
int **signo*;

4.3.3.2 설명

sigsetops 함수는 신호집합에 관한 연산을 취급한다. 이들의 연산의 대상이 되는 것은 응용에 의해 주소를 부여할 수 있는 (addressable) 데이타 개체뿐이며, 프로세스의 전달이 차단된 신호집합은 어떤 프로세스를 대기중인 집합(**<signal.h>** [4.3.1] 참조)과 같은 시스템에 알려진 신호집합은 포함되지 않는다.

sigemptyset() 함수는 본 표준에서 정의된 모든 신호를 제외한 인수 *set*이 지시하는 신호집합을 초기화한다.

sigfillset() 함수는 본 표준에서 정의된 모든 신호를 포함하여 인수 *set*이 지시하는 신호집합을 초기화한다.

응용은 모든 *sigset_t*형식의 개체(이 개체를 달리 사용하기 전에)에 대해 *sigemptyset*()이나 *sigfillset*()를 적어도 한번 호출한다. 만일 그러한 개체가 이러한 방법으로 초기화되지 않으며, 또한 어떤 것도 *sigaddset*(), *sigdelset*(), *sigismember*(), *sigaction*(), *sigprocmask*(), *sigpending*(), 또는 *sigsuspend*() 함수의 인수로도 제공되지 않으면, 그 결과는 정의되지 않는다.

sigaddset()과 *sigdelset*()함수는 각각 (인수 *set*이 지시하는 신호집합의 인수인 *signo* 의값으로 규정된 개별적인) 신호를 더하거나 뺀다.

sigismember() 함수는 인수 *signo*의 값으로 규정된 신호가 인수 *set*이 지시하는 집합에 포함되는가를 시험한다.

4.3.3.3 복귀값

함수가 정상적으로 완료되면 *sigismember*() 함수의 복귀값은 규정된 신호가 규정된 집합에 포함되면 1 을, 포함되지 않으면 0 이 된다. 정상적인 종료시 다른 함수의 복귀값은 0 이 된다. 상기의 모든 함수에서 에러가 발견되면 복귀값으로 -1을 반환하며 *, *errno*는 지정된 에러번호로 설정된다.

4.3.3.4 에러

다른 조건이 발견되면 *sigaddset*(), *sigdelset*(), *sigismember*()함수는 복귀값으로 -1을 반환하고 *errno*는 상응하는 값으로 설정된다

[EINVAL] 인수 *signo*의 값이 부적당하거나 지원되지 않는 신호번호이다.

4.3.3.5 참고

sigaction() [4.3.4], **<signal.h>** [4.3.1], *sigpending*() [4.3.6],
sigprocmask() [4.3.5], *sigsuspend*() [4.3.7].

4.3.4 신호 동작의 점검과 변경

함수: *sigaction*()

4.3.4.1 용례

#include <signal.h>

int *sigaction*(*sig*,*act*,*oact*)
int *sig*;
struct *sigaction* **act*, **oact*;

4.3.4.2 설명

sigaction() 함수는 호출 프로세스가 특정한 신호와 관련된 동작을 시험하거나 규정하는 것 (또는 두가지 모두를) 허용한다. 인수 *sig*는 신호를 규정한다. 즉, 수용 가능한 값은 **<signal.h>** [4.3.1]에서 정의된다.

허용된 동작을 기술하는데 사용되는 구조 *sigaction*()은 헤더 **<signal.h>**에서 정의되며, 최소한 다음과 같은 항목을 포함한다.

항목형식	항목명칭	내 용
void (*)()	*sa_handler*	SIG_DFL, SIG_IGN, 또는 함수로서의 포인터.
sigset_t	*sa_amsk*	신호포착 함수의 실행시 차단되어야 하는 신호의 추가적인 집합.
int	*sa_flags*	신호의 작동방식에 영향을 미치는 특정한 플래그.

만일 인수 *act*가 널(NULL)이 아니면 *act*는 규정된 신호와 관련된 동작을 규정하는 구조를 지시한다. 만일 인수 *oact*가 **널**이 아니면 신호와 관련된 이전의 동작은 인수 *oact*이 지시하는 위치에 저장된다. 만일 인수 *act*가 **널**이면 이 함수의 호출에 의하여 신호의 처리과정은 변경되지 않는다. 즉, 이 함수의 호출은 현재 주어진 신호의 처리과정을 조회하는데 사용될 수 있다. *sigaction*() 구조의 sa_handler 부분은 규정된 신호와 관련된 동작을 지정한다. 만일 sa_handler부분이 신호포착 함수를 규정한다면, sa_mask 부분은 (신호포착 함수가 호출되기 전에 프로세스의 신호 마스크에 추가되는) 신호의 집합을 지정한다. SIGKILL과 SIGSTOP신호는 이러한 메카니즘을 사용하는 신호마스크에 추가되지 않는다. 시스템에서는 이것을 지시하는 에러를 야기하지 않도록 하여야 한다. sa_flags 부분은 규정된 신호의 작동방식을 변경하는데 사용될 수 있다.

헤더 **<signal.h>**에 정의된 아래의 프래그 비트는 sa_flags내에 설정될 수 있다.

심 볼 상 수	내 용
SA_NOCLDSTOP	자식 프로세스가 중지될 때 SIGGHLD를 생성하지 않음.

만일 *sig*가 SIGGHLD이고 *sa_flags*내에서 SA_NOCLDSTOP이 설정되지 않는 경우, 그리고 구현시에 SIGGHLD신호가 지원되면, (호출 프로세스의 자식 프로세스들이 중지될 때마다) 호출 프로세스에 대한 SIGGHLD 신호가 생성된다. 만일 *sig*가 SIGGHLD이고 *sa_flags*내의 SA_NOCLDSTOP 플래그가 설정되는 경우, 구현물에서는 이러한 방법으로 SIGGHLD신호를 생성하지 않는다.

어떤 신호가 *sigaction*() 함수에 의한 신호포착 함수에 의해 잡힌 경우, 신호포착 함수가 실행되는 기간동안(또는 *sigprocmask*() 또는 *sigsuspend*() 함수의 호출이 되기까지) 새로운 신호 마스크가 계산되어 만들어진다.

이 마스크는 현재의 신호 마스크와 전달중인 신호의 *sa_mask*의 값의 합집합을 취한 것과 전달중인 신호를 포함하여 구성된다. 사용자의 신호 처리기가 정상적으로 복귀하면 신호 마스크는 원래의 값으로 재구성된다.

특정한 신호에 대하여 하나의 동작을 수행하도록 하면 이 동작은 (새로운 *sigaction*() 함수를 호출함에 의하여) 새로운 동작을 수행하도록 분명히 요구되던가 또는 *exec*함수중의 하나가 호출되기 전까지는 원래의 동작을 수행하도록 한다.

만일 *sig*를 위한 전처리 동작이 C-표준에서 정의된 *signal*()함수에 의해 만들어지면, *oact*가 지시하는 구조로 복귀되는 부분의 값은 규정되지 않으며, 특히 *oact->sa_handler*는 *signal*() 함수로 전달되는 값과 항상 같을 필요는 없다. 그러나, 만일 동일한 구조에의 포인터 또는 이들과 동일한 내용물이 *act*인수를 통하여 계속적으로 호출되는 *sigaction*() 함수로 전달된다면, 신호의 처리는 원래의 *signal*()함수의 반복적인 호출과 동일이다.

만일 *sigaction*()함수가 실패하면, 아무런 새로운 신호의 처리도 수행되지 않는다.

4.3.4.3 복귀값

성공적인 완료시에 복귀값은 0 이 된다. 이외의 경우에는 -1을 복귀하고 *errno*는 지정된 에러값으로 설정된다.

4.3.4.4 에러

다음 조건이 발생하면, *sigaction*() 함수의 복귀값은 -1이 되고 *errno*를 상응하는 값으로 설정한다.

[EINVAL] *sig*인수의 값이 부적당하거나 지원되지 않는 신호번호임. 또는 포착될 수 없는 신호를 포착하려 하거나 무시될 수 없는 신호를 무시하려고 한다. **<signal.h>** [4.3.1] 참조.

4.3.4.5 참고

kill() [4.3.2], **<signal.h>** [4.3.1], *sigprocmask*() [4.3.5], *sigsetops*() [4.3.3], *sigsuspend*() [4.3.7].

4.3.5 차단된 신호에 대한 점검과 변경(Examine and Change Blocked Signals)

함수: *sigprocmask*()

4.3.5.1 용례

```
#include <signal.h>

int sigprocmask (how,set,oset)
int how;
sigset_t *set, *oset;
```

4.3.5.2 설명

sigprocmask() 함수는 호출 프로세스의 신호마스크를 조사하거나 변경 (또는 두가지) 경우에 사용된다. 만일 인수 *set*의 값이 **널**이 아니면, 이것은 현재 차단된 집합을 변경하는데 사용되는 신호의 집합을 지시한다.

인수 *how*의 값은 이 집합의 변경되는 과정을 지시하며, **<signal.h>** [4.3.1]에 지정된 바와 같이 아래의 값을 갖는다.

명 칭	내 용
SIG_BLOCK	결과로 얻어진 집합은 현재 집합과 인수 *set*이 지정하는 신호 집합간의 합집합임.
SIG_UNBLOCK	결과로 얻어진 집합은 현재 집합과 인수 *set*이 지정하는 신호 집합의 여집합간의 교집합임.
SIG_SETMASK	결과로 얻어진 집합은 인수 *set*이 지정하는 신호 집합임.

만일 인수 o*set*이 **널**이 아니면, 이전의 마스크는 o*set*이 지시하는 공간에 저장된다. 만일 인수 *set*의 값이 **널**이면, 인수 *how*의 값은 중요하지 않고 프로세스의 신호 마스크는 이 함수의 호출에 의해 변경되지 않는다. 따라서 이 호출은 현재 차단된 신호를 조사하는데 사용될 수 있다.

만일 *sigprocmask*() 함수를 호출한 후에 대기중인 차단된 신호가 있다면 이들 신호 중 적어도 하나는 *sigprocmask*() 함수가 복귀되기 전에 전달되어야 한다.

SIGKILL과 SIGSTOP신호는 차단되지 못한다. 시스템은 이러한 경우가 발생하지 않도록 하여야 한다.

만일 SIGFPE, SIGILL 또는 SIGSEGV 신호가 차단된 동안 다시금 이들이 생성되는 경우는 C-표준에서 정의된 *kill*()함수나 *raise*() 함수를 호출함에 의하여 신호가 생성되

지 않는 한, 그 결과는 정의되지 않는다.

만일 *sigprocmask*() 함수가 실패하면, 프로세스의 신호 마스크는 이 함수 호출로 변경되지 않는다.

4.3.5.3 복귀값

함수의 성공적인 완료시 복귀값은 0이며, 이외의 경우에는 -1 이며, *errno*는 지정된 에러값으로 설정된다.

4.3.5.4 에러

다음 조건이 발생한다면, *sigprocmask*()함수의 복귀값은 -1이고 *errno*는 상응하는 값으로 설정된다.

[EINVAL] 인수 *how*의 값이 정의된 값과 다르다.

4.3.5.5 참고

sigaction() [4.3.4], **<signal.h>** [4.3.1], *sigpending*() [4.3.6], *sigsetops*() [4.3.3], *sigsuspend*() [4.3.7].

4.3.6 대기중인 신호의 점검 (Examine Pending Signals)

함수: *sigpending*()

4.3.6.1 용례

#include <signal.h>

int sigpending(*set*)
sigset_t **set*;

4.3.5.2 설명

sigpending() 함수는 전달이 차단되고 호출 프로세스를 대기중인 신호의 집합을 인수 *set*이 지시하는 공간에 저장한다.

4.3.6.3 복귀값

이 함수의 성공적인 완료시 복귀값은 0 이고. 이외의 경우에는 -1이며, *errno*는 지정된 에러값으로 설정된다.

4.3.6.4 에러

본 표준에서는 *sigpending*() 함수에서 검출되는 어떠한 에러 조건도 규정하지 않는다. 몇몇 에러는 구현시 정의된 바에 따라 검출된다.

4.3.6.5 참고

<signal.h> [4.3.1], *sigprocmask*() [4.3.5], *sigsetops*() [4.3.3].

4.3.7 신호의 대기(Wait for a Signal)

함수: *sigsuspend*()

4.3.7.1 용례

#include <signal.h>
int sigsuspend(*sigmask*)
sigset_t **sigmask*;

4.3.7.2 설명

sigsuspend() 함수는 프로세스의 신호 마스크를 인수 *sigmask*가 지시하는 신호들의 집합으로 교체한 후, 신호포착 함수를 실행시키거나 프로세스를 종료시키는 동작을 하는 신호가 전달될 때까지 프로세스를 일시 정지 시킨다.

만일 동작이 프로세스를 종료시키려고 하면, *sigsuspend*() 함수는 복귀하지 않는다. 만일 동작이 신호포착 함수를 실행시키려고 하면, 신호포착 함수가 복귀된 이후에 *sigsuspend*() 함수는 *sigsuspend*()를 호출하기 전의 신호 마스크를 원상으로 회복시킨 후에 복귀한다

<signal.h> [4.3.1]에서 문서화된 것처럼 무시될 수 없는 신호들을 차단하는 것은 불가능하다. 즉, 시스템은 이러한 에러를 발생시키지 않아야 한다.

4.3.7.3 복귀값

sigsuspend()함수가 프로세스의 실행을 무제한 중지시키므로 이 함수의 실행이 완료되어도 어떤 값을 복귀하지 못한다. 이 함수의 실행이 실패하면 복귀값은 -1이 되고 *errno*는 상응하는 에러값으로 설정된다.

4.3.7.4 에러

아래의 조건이 발생하면, *sigsuspend*() 함수의 복귀값은-1 이고, *errno*는 상응하는 값으로 설정된다

[EINTR] 어떤 신호가 호출 프로세스에 의하여 포착되고, 신호포착 함수로부터 제어가 반환된다.

4.3.7.5 참고

pause() [4.4.2], *sigaction*() [4.3.4], **<signal.h>** [4.3.1], *sigpending*() [4.3.6], *sig*p*rocmask*() [4.3.5], *sigsetops*() [4.3.3].

4.4 시각 운용

프로세스는 *sleep*() 함수를 이용하여 일정한 시간만큼 일시적으로 중지되거나 *pause*()함수에 의하여 어떤 신호가 도달할 때까지 무한히 중지될 수 있다. 여기서

alarm()함수는 특정한 시간에 어떤 신호가 도착하도록 계획할 수 있으므로 *pause*()함수에 의한 일시 중지가 무한히 계속되지 않도록 할 수 있다.

4.4.1 경보의 계획

함수: *alarm*()

4.4.1.1 용례

unsigned int alarm(*seconds*)
unsigned int *seconds*;

4.4.1.2 설명

alarm()함수는 시스템으로 하여금(seconds로 규정된 시간(초)이 경과한 후에) SIGALRM신호를 호출 프로세스에 보내도록 한다.

프로세스의 계획에 의한 지연은 프로세스로 하여금 신호의 실제적인 처리를 원하는 시간까지 개시되지 못하도록 할 수 있다.

경보의 요구는 누적되지 않는다. 즉, 하나의 SIGALRM 생성만이 이 방식으로 계획된다. 만일 SIGALRM이 아직 생성되지 않았다면 경보의 호출은 SIGALRM이 생성되는 시간을 재계획하는 결과를 초래한다.

만일 *seconds*가 0이면, 이전에 발생된 *alarm*()요구는 모두 취소된다.

4.4.1.3 복귀값

alarm()함수는 (시스템이 SIGALRM신호를 생성하도록 계획되기까지) 남은 시간(초) 값을 복귀하거나, *alarm*()요구가 없다면 0값을 복귀한다.

4.4.1.4에러

alarm()함수는 항상 성공적으로 실행되며 따라서 에러에 대한 복귀값이 지정되지 않는다.

4.4.1.5 참고

exec() [4.2.1], *fork*() [4.1.1], *pause*() [4.4.2], *sigaction*() [4.3.4],
<signal.h> [4.3.1].

4.4.2 프로세스 실행의 중지 (Suspend Process Execution)

함수: *pause*()

4.4.2.1 용례

int pause()

4.4.2.2.설명

pause()함수는 신호포착 함수를 실행시키거나 프로세스를 종료시키는 동작을 수행

하는 신호가 전달될 때까지 호출 프로세스를 일시 정지 시킨다.

만일 신호의 동작이 프로세스를 종료시키려고 하면, *pause*() 함수는 복귀하지 않는다.

만일 신호의 동작이 신호포착 함수를 실행시키려고 하면 *pause*() 함수는 신호포착 함수가 복귀한 다음 복귀한다.

4.4.2.3 복귀값

pause()함수가 프로세스의 실행을 무한히 중지시킨 이후에는 이 함수가 성공적으로 완료되는 경우에는 아무런 값도 반환하지 않는다. 이외의 경우에서 복귀값은 -1이며. *errno*는 지정된 에러값으로 설정된다.

4.4.2.4 에러

다음 조건이 발생하면, *pause*()함수의 복귀값은 -1 이고 *errno*는 상응하는 값으로 설정된다.

[EINTR] 신호가 호출 프로세스에 의해 포착되고 신호포착 함수로부터 제어가 복귀된다.

4.4.2.5 참고

alarm() [4.4.1], *kill*() [4.3.2], *wait*() [4.2.1], **기타 함수에서의 신호효과** [4.3.1.4].

4.4.3 프로세스 실행의 지연

함수: *sleep*()

4.4.3.1 용례

unsigned int sleep(*seconds*)
unsigned int *seconds*;

4.4.3.2 설명

sleep()함수는 인수 *seconds*로 지정된 시간(초)이 경과되거나 어떤 신호가 호출 프로세스에 전달되고, 이 신호에 의한 동작이 신호포착 함수를 호출하거나 프로세스를 종결시킬 때까지 현재 수행중인 프로세스의 실행을 일시적으로 중지시킨다. 일시중지 시간은 시스템상의 여타 동작들로 인하여 계획된 것보다 길다.

만일 *sleep*()함수의 실행중 SIGALRM신호가 호출 프로세스를 위해 생성되며, 또한 SIGALRM신호가 전달로 부터 무시되거나 차단되면, SIGALRM신호가 계획될 때 *sleep*()함수가 복귀할 것인가는 규정되지 않는다. 만일 신호가 차단되면 *sleep*()함수가 복귀한 후에 신호가 신호가 계속 대기중인지 또는 폐기되는 지는 규정되지 않는다.

만일 *sleep*()함수의 실행중(이전의 *alarm*()함수를 호출함에 의하여 발생된 경우를 제외하고) SIGALRM신호가 호출 프로세스를 위해 생성되고, 또한 SIGALRM신호가 전달중에 무시되지 않고 차단되지 않으면, 그 신호가 *sleep*()함수가 복귀하는 것 외에 어떤 영향을 미치는지에 대해서는 규정되지 않는다.

만일 신호포착 함수가 *sleep*() 함수에 인터럽트를 걸고, SIGALRM이 만들어 지도록 계획된 시간과 SIGALRM에 관련된 행동에 관하여, 또는 SIGALRM신호가 전달로부터 차단되었는가에 관하여 조사(examine) 또는 변경하려고 하면, 그 결과는 규정되지 않는다.

만일 신호포착 함수가 *sleep*() 함수에 인터럽트를 걸고, 신호포착 함수가 sleep() 함수의 호출에 앞서 저장하였던 환경으로 재구성하기 위하여 *sig*longjmp() 또는 longjmp() 함수를 호출한다면, SIGALRM신호와 관련된 동작과 SIGALRM 신호가 생성되기로 계획된 시간은 규정되지 않는다. 또한 프로세스의 신호마스크가 환경의 일부로써 재구성되지 않는 한 SIGALRM신호가 차단되는 지도 규정되지 않는다(*sigset*jmp() [9.3.1] 참조).

4.4.3.3 복귀값

만일 요구된 시간이 경과되어 *sleep*()함수가 복귀하면 복귀값은 0 이어야 한다. 만일 *sleep*()함수가 어떤 신호의 전달에 의해 복귀한다면, 복귀값은 일시 중지하려고 계획된 시간에서 실제로 경과한 시간을 제외한 나머지 시간(초)이 된다.

4.4.3.4 에러

sleep() 함수는 항상 성공적이며 따라서 에러에 따른 복귀값은 지정되지 않는다.

4.4.3.5 참고

alarm() [4.4.1], *pause*() [4.4.2] *sigaction*() [4.3.4].

제 5장 프로세스 환경

5.1 프로세스 식별(Process Identification)

5.1.1 프로세스 ID 및 부모 프로세스 ID의 획득

함수: *getpid*(), *getppid*()

5.1.1.1 용례

#include<sys/types.h>
pid_t getpid()
pid_t getppid()

5.1.1.2 설명

getpid()함수의 복귀값은 호출 프로세스의 프로세스 ID이며, *getppid*()함수의 복귀값은 호출 프로세스의 부모 프로세스 ID이다.

5.1.1.3 복귀값 - 설명참조

5.1.1.4 에러

getpid()와 *getppid*()함수는 항상 성공적으로 수행되므로 에러를 지시하는 복귀값은 지정되지 않는다.

5.1.1.5 참고

exec() [4.1.2], *fork*() [4.1.1], *kill*() [4.3.2].

5.2. 사용자 식별 (User Iedntification)

5.2.1 실제 사용자 ID, 유효 사용자 ID, 실제 그룹 ID 및 유효 그룹 ID의 획득

함수: *getuid*(), *geteuid*(), *getgid*(), *getegid*()

5.2.1.1 용례

#include<sys/types.h>
uid_t getuid()
uid_t geteuid()
gid_t getgid()
gid_t getegid()

5.2.1.2 설명

getuid() 함수의 복귀값은 호출 프로세스의 실제 사용자 ID이다.
geteuid() 함수의 복귀값은 호출 프로세스의 유효 사용자 ID이다.
getgid() 함수의 복귀값은 호출 프로세스의 실제 그룹 ID이다.
getegid() 함수의 복귀값은 호출 프로세스의 유효 그룹 ID이다.

5.2.1.3 복귀값 - 설명참조

5.2.1.4 에러

getuid(), *geteuid*(), *getgid*() 및 *getegid*() 함수는 항상 성공적으로 수행되므로 에러를 지시하는 값은 지정되지 않는다.

5.2.1.5 참고

setuid() [5.2.2]

5.2.2 사용자 ID 및 그룹 ID의 설정

함수: *setuid*(), *setgid*()

5.2.2.1 용례

#include<sys/types.h>
int setuid(*uid*)
uid_t *uid*;
int setgid(gid)
gid_t gid;

5.2.2.2 설명

{_POSIX_SAVED_IDS}가 정의되는 경우:

(1) 만일 프로세스가 적절한 권한을 가진다면, *setuid*() 함수는 실제 사용자 ID, 유효 사용자 ID 및 세이브드 셋-사용자-ID(saved *set*-user-ID)를 *uid*로 설정한다.
(2) 만일 프로세스가 적절한 권한을 가지지 않으나 *uid*가 실제 사용자 ID 또는 세이브드 셋-사용자-ID와 동일하다면, *setuid*()함수는 유효 사용자 ID를 uid로 설정함. 여기서 실제 사용자 ID와 세이브드 셋-사용자-ID는 이 함수의 호출에 의해 변경되지 않는다.
(3) 만일 프로세스가 적절한 권한을 가진다면, *setgid*()함수는 실제 그룹 ID, 유효 그룹 ID와 세이브드 셋-사용자-ID를 gid로 설정한다.
(4) 만일 프로세스가 적절한 권한을 가지지 않으나 gid가 실제 그룹 ID 또는 세이브드 셋-사용자-ID와 동일하다면, *setgid*()함수가 유효그룹 ID를 gid로 설정한다. 여기서 실제 그룹 ID와 세이브드 셋-사용자-ID는 이 함수 호출에 의해 변경되지 않는다.

{_POSIX_SAVED_IDS}이 정의되지 않는 경우:

(1) 만일 프로세스가 적절한 권한을 가진다면, *setuid*() 함수는 실제 사용자 ID와 유효 사용자 ID를 uid로 설정한다.
(2) 만일 프로세스가 적절한 권한을 가지지 않으나 uid가 실제 사용자 ID와 동일하다면, *setuid*()함수는 유효 사용자 ID를 uid로 설정한다. 여기서 실제 사용자 ID는 이 함수의 호출에 의해 변경되지 않는다.
(3) 만일 프로세스가 적절한 권한을 가진다면, *setgid*() 함수는 실제그룹 ID와 유효 그룹 ID를 gid로 설정한다.
(4) 만일 프로세스가 적절한 권한을 가지지 않으나 gid가 실제그룹 ID와 동일하다면, *setgid*()함수가 유효그룹 ID를 gid로 설정한다. 여기서 실제 그룹 ID는 이 함수의 호출에 의해 변경되지 않는다.

호출 프로세스의 추가 그룹 ID도 이 함수의 호출에 의해 변경되지 않는다.

5.2.2.3 복귀값

성공적인 완료시 0값이 복귀된다. 이외의 경우에는 -1 값이 복귀되고 *errno*는 지정된 여러값으로 설정한다.

5.2.2.4 에러

다음 조건이 발생하면 *setuid*() 함수의 복귀값은 -1이고 *errno*는 상응하는 값으로 설정한다.

[EINVAL] uid 인수의 값이 유효하지 않고 구현물에서 지원되지 않는다.
[EPERM] 프로세스가 적절한 권한을 가지지 않았고, uid가 실제 사용자 ID(또는 {_POSIX_SAVED_IDS}가 정의된 경우에는 세이브드 셋-사용자-ID)와 부합되지 않는다.

다음 조건이 발생하면 *setgid*()함수의 복귀값은 -1이고 *errno*는 상응하는 값으로 설정한다.

[EINVAL] gid인수의 값이 유효하지 않고 구현물에서 지원되지 않는다.
[EPERM] 프로세스가 적절한 권한을 가지지 않았고, gid가 실제 그룹 ID(또는 {_POSIX_SAVED_IDS}가 정의된 경우에는 세이브드 셋-사용자-ID)와 부합되지 않는다.

5.2.2.5 참고

exec() [4.1.2], *setuid*() [5.2.1].

5.2.3 추가 그룹 ID의 획득

함수: *getgroups()*

5.2.3.1 용례

#include<sys/types.h>

int getgroups(*gidsetsize*, *grouplist*)
int *gidsetsize*;
gid_t *grouplist*[];

5.2.3.2 설명

getgroups()함수는 배열 *grouplist*의 내용을 호출 프로세스의 추가 그룹 ID로 채운다. *gidsetsize*인수는 제공된 배열 *grouplist*에 포함된 원소의 수를 규정한다. 즉, 추가 그룹 ID들의 실제 갯수가 복귀된다. 여기서 복귀된 값이상인 인덱스로 지정되는 배열의 내용물은 정의되지 않는다.

호출 프로세스의 유효그룹 ID가 추가 그룹 ID의 복귀된 목록에 포함되어 있는지 아니면 생략되었는 지는 규정되지 않는다.

특별한 경우로 만일 *gidsetsize*인수가 0이면, *getgroups*()는 *grouplist*인수가 지시하는 배열을 수정하지 않고 호출 프로세스와 관련된 보완 그룹 ID의 수를 복귀한다

5.2.3.3 복귀값

성공적인 완료시 추가 그룹 ID의 갯수가 복귀된다. {NGROUPS_MAX}가 0이면 이 값은 0이 된다. 복귀값이 -1이면 이 함수의 실행이 실패했음을 의미하며 *errno*는 상응하는 에러값으로 설정한다.

5.2.3.4 에러

다음 조건이 발생하면, *getgroups*() 함수의 복귀값은 -1이 되고 *errno*는 상응하는 값으로 설정한다.

[EINVAL] gidsetsize()인수가 0 이 아니며 추가 그룹 ID의 갯수보다 적다.

5.2.3.5 참고

setgid() [5.2.2]

5.2.4 사용자 명의 획득

함수: *getlogin*(), *cuserid*()

5.2.4.1 용례

char *getlogin()

#include<stdio.h>

char cuserid(*s*)
char **s*;

5.2.4.2 설명

이 함수는 현재 수행중인 프로세스와 관련된 사용자 명을 알려주는 문자열을 복귀한다. *cuserid*() 함수는 프로세스의 유효 사용자 ID와 관계된 이름을 복귀하고, *getlogin*()함수는 제어 터미널을 소유한 로그인(login) 작업요소(*act*ivity)와 관계된 이름을 복귀하다.

사용자 명을 획득하기 위하여 추천하는 절차는 다음과 같다. 즉, *cuserid*()나 *getlogin*() 함수를 호출하거나, 만일 이것이 실패하면 *getuid*()함수의 복귀값으로 *getwuid*() 함수를 호출하는 것이다.

getlogin() 함수는 사용자의 로그인명으로의 포인터를 복귀한다. 동일한 사용자 ID는 여러 로그인명으로 공유될 수 있다. 따라서, 정확한 사용자 데이타 베이스를 찾아내기 위해서는 *getpwnam*() 함수와 함께 *getlogin*() 함수를 사용해야 한다.

만일 *getlogin*()이 널이 아닌 포인터를 복귀하면, 이 포인터는 (비록 동일한 사용자 ID로 여러 로그인 명이 존재하더라) 로그인하는 데 이용된 사용자 명을 지시해야 한다.

cuserid() 함수는 현재 프로세스의 소유자 로그인명을 문자로 표시한다. 만일 *s*가 널포인터가 아니면 *s*는 적어도 L_cuserid 바이트 크기의 배열을 지시한다. 여기서 로그인명을 나타내는 문자열은 이 배열에 저장되어 복귀된다. 심볼상수 L_cuserid는 **<stdio.h>**에서 정의되며 0보다 큰 값을 가진다.

5.2.4.3 복귀값

getlogin() 함수의 복귀값은 사용자의 로그인명을 포함하는 문자열을 지시하는 포인터 또는 사용자의 로그인명을 찾을 수 없을 때에는 널포인터가 된다. 또한 *getlogin*() 함수의 복귀값이 정적 데이터(static data)위치를 지시하므로 각각의 호출시마다 이곳을 반복해서 사용할 수 있다.

만일 *s*가 널포인터이면, *cuserid*()의 결과는 그 주소가 정적주소공간에 생성된다 만일 로그인명을 찾을 수 없으면 *cuserid*()함수의 복귀값은 널포인터이다. 만약 *s*가 널포인터가 아니면, *s*가 복귀된다. 만일 로그인명을 찾을 수 없으면 **s*에 널문자를 저장한다. *cuserid*()의 복귀값은 정적 데이터를 지시할 수 있고, 따라서 각 함수의 호출에 의해 반복해서 쓰여질 수 있다.

cuserid() 함수의 구현시의 *getwnam*() 함수를 사용할 수 있으며, 따라서 이들 루틴에 대한 사용자 호출의 결과는 후속의 다른 루틴의 호출에 의하여 반복적으로 겹쳐 쓰여질 수 있다.

5.2.4.4 에러

본 표준은 *cuserid*()나 *getlogin*() 함수에 의하여 발견되어야 하는 에러조건을 규정하지 않는다. 몇 몇 에러들은 구현시 정의된 바에 따라 검출된다.

5.2.4.5 참고

getpwnam() [10.2.2], *getpwuid*() [10.2.2]

5.3 프로세스 그룹

5.3.1 프로세스 그룹 ID의 획득

함수: *getpgrp*()

5.3.1.1 용례

#include <sys/types.h>
pid_t getpgrp()

5.3.1.2 설명

getpgrp() 함수는 호출 프로세스의 프로세스 그룹 ID를 복귀한다.

5.3.1.3 복귀값 - 설명참조

5.3.1.4 에러

getpgrp() 함수는 항상 성공적으로 수행되므로 에러값은 지정되지 않음.

5.3.1.5 참고

getpgrp() [5.3.3], *setsid*() [5.3.2], *sigaction*() [4.3.4].

5.3.2 세션의 생성과 프로세스 그룹 ID의 생성(creat)

함수: *setsid*()

5.3.2.1 용례

#include <sys/types.h>
pid_t setsid()

5.3.2.2 설명

호출 프로세스가 프로세스 그룹 리더가 아니면 *setsid*() 함수는 새로운 세션을 생성한다. 여기서 호출 프로세스는 새로운 세션의 세션 리더 및 새로운 프로세스 그룹의 프로세스 그룹 리더가 되며, 제어 터미널을 소유하지 않는다. . 호출 프로세스의 프로세스 그룹 ID는 호출 프로세스의 프로세스 ID와 동일하게 설정되어야 한다. 호출 프로세스는 새로운 프로세스 그룹에 포함된 유일한 프로세스이며 새로운 세션내의 유일한 프로세스이다.

5.3.2.3 복귀값

성공적인 완료시 *setsid*() 함수의 복귀값은 호출 프로세스의 프로세스 그룹 ID 값이 된다.

5.3.2.4 에러

다음조건이 발생하면 *setsid*() 함수의 복귀값은 -1이 되고 *errno*는 상응하는 값으로 설정된다.

[EPERM] 호출 프로세스가 이미 프로세스 그룹 리더이거나 호출 프로세스가 아닌 어떤 프로세스의 프로세스 그룹 ID가 호출 프로세스의 프로세스 ID와 일치한다.

5.3.2.5 참고

exec [4.1.2], _exit() [4.2.2], *fork*() [4.1.1], *getpid*() [5.1.1], *kill*() [4.3.2], *setpgid*() [5.3.3], *sigaction* [4.3.4].

5.3.3 작업 제어를 위한 프로세스 그룹 ID의 설정

함수: *setgid*()

5.3.3.1 용례

#include <sys/types.h>

int setpgid(*pid*, *pgid*)
pid_t *pid*, *pgid*;

5.3.3.2 설명

{_POSIX_JOB_CONTROL}이 정의된 경우:
setpgid() 함수는 존재하는 프로세스 그룹을 묶거나 호출 프로세스의 세션내에 새로운 프로세스 그룹을 생성한다. 세션 리더의 프로세스 그룹 ID는 변경되지 않는다. 함수의 성공적인 완료시 *pid*와 일치하는 프로세스 ID를 가진 프로세스의 프로세스 그룹 ID는 *pgid*로 설정된다. 특별히 *pid*가 0인 경우에는 호출 프로세스의 프로세스 ID가 사용된다. 또한 *pgid*가 0이면 지정된 프로세스의 프로세스 ID가 사용된다.
이외의 경우:
위에서 설명한 바에 따라 구현물이 *setpgid*() 함수를 지원하던가 또는 *setpgid*()함수의 실행이 실패한다.

5.3.3.3 복귀값

함수의 성공적인 완료시 *setpgid*() 함수의 복귀값은 0이 된다. 이외의 경우에는 -1값을 qhrrnl하고 *errno*는 지정된 에러값으로 설정된다.

5.3.3.4 에러

아래의 조건이 발생하면 *setpgid*() 함수의 복귀값은 -1이 되고 *errno*는 상응하는 값으로 설정된다.

[EACCES] *pid* 인수의 값이 호출 프로세스의 자식 프로세스의 프로세스 ID와 일치하고 자식 프로세스는 EXEC함수중의 하나를 성공적으로 실행한다.

[EINVAL] *pgid*인수의 값이 0보다 작거나 구현물에 의해 지원되지 않는 값이다.

[ENOSYS] *setpgid*() 함수가 이 구현물에 의해 지원되지 않는다.

[EPERM] *pgid* 인수에 의해 지정된 프로세스가 세션 리더이다. *pid* 인수의 값은 적합하나 호출 프로세스의 자식 프로세스의 프로세스 ID와 일치하고, 자식 프로세스가 호출 프로세스와 같은 세션내에 있지 않다. *pgid* 인수의 값이 *pid*인수가 지시하는 프로세스의 프로세스 ID와 일치하지 않고 *pgid*인수의 값과 일치하는 프로세스 그룹 ID를 가지는 프로세스와 호출 프로세스와 동일한 세션에 존재하지 않는다.

[ESRCH] *pid*인수의 값이 호출 프로세스의 프로세스 ID 또는 호출 프로세스의 자식 프로세스의 프로세스 ID와 일치하지 않는다.

5.3.3.5참고

getpgrp() [5.3.1], *setsid*() [5.3.2], *tcsetpgrp*() [8.2.4], *exec* [4.2.1]

5.4 시스템의 식별

5.4.1 시스템 명

함수: *uname*()

5.4.1.1 용례

#include<sys/utsname.h>

int uname(*name***)**
struct *utname* **name*;

5.4.1.2 설명

uname() 함수는 현재의 운영체제를 식별하는 정보를 인수 *name*이 지시하는 구조내에 저장한다.

uts*name* 구조는 헤더 **<sys/utsname.h>**에 정의되며, 표 5-1에 수록된 항목을 포함하여야 한다.

<표 5-1> **uname() 구조의 항목**

항 목	내 용
sysname	운영체제의 구현물의 명칭
nodename	구현물별에 따른 통신 네트웍내의 본 노드의 명칭
release	본 구현물의 현 발매 수준(release level)
version	본 발매의 현 버전 수준
machine	시스템이 실행되는 하드웨어 형식의 명칭

이들 자료 항목은 널문자로 끝나는 문자배열이다
각 항목의 포맷은 구현시 정의된 바에 따른다. 시스템 문서(**문서화** [3.2.1.2] 참조)

는 각 항목의 소스(source)와 포맷을 규정하며 각 항목의 나타내는 값의 범위를 규정할 수 있다.

본 구조내에 *nodename* 항목이 포함되는 것은 통신 네트웍에의 접속을 위한 충분한 정보가 주어짐을 의미하지는 않는다.

5.4.1.3복귀값

함수의 성공적인 완료시 복귀값으로 음수가 아닌 값을 복귀한다 . 이외의 경우에는 -1값을 복귀하고 *errno*는 지정된 에러값으로 설정된다.

5.4.1.4에러

본 표준은 *uname*() 함수에서 검출되어야 하는 어떠한 에러조건도 규정하지 않는다. 몇몇 에러는 구현시 정의된 바에 따라 검출된다.

5.5 시각

5.5.1 시스템 시각의 획득

함수: *time*()

5.5.1.1 용례

#include <time.h>
time_t time (*tloc*)
time_t **tloc*

5.5.1.2 설명

time()함수의 복귀값은 **에폭이후 경과시간** [3.3]값이다.

인수 *tloc*은 복귀된 값이 저장된 위치를 지시한다. 만일 *tloc*이 **널**포인터이면 아무런 값도 저장되지 않는다.

5.5.1.3 복귀값

함수의 성공적인 완료시 *time*()함수의 복귀값은 시각에 관한 값을 가진다. 이외의 경우에는 ((*time_t*)-1)값은 복귀하고 *errno*는 지정된 에러값으로 설정된다.

5.5.1.4 에러

본 표준에서는 *time*() 함수에서 검출되는 어떠한 에러조건도 규정하지 않는다. 몇몇 에러는 구현시 정의된 바에 따라 검출된다.

5.5.2 프로세스 시각

함수: *times*()

5.5.2.1 용례

#include <times.h>

clock_t times (*buffer*)
struct *tms* **buffer*;

5.5.2.2. 설명

times() 함수는 사용시간 계수 정보를 수록하는 버퍼에 의해 지시되는 구조에 수록된다. clcok_t 형과 *tms* 구조는 <sys/times.h>에서 정의된다. *tms*구조에는 아래와 같은 항목이 포함된다.

모든 시간은 초당 {CLK_TCK}개의 간격으로 표현된다.

종료된 자식 프로세스의 시각은, *wait*()나 *waitpid*()함수가 종료된 자식 프로세스의 프로세스 ID를 반환할 때 부모 프로세스의 *tms_cutime*과 현재의 *tms_cstime*항목에 포함된다. wait [4.2.1] 참조. 만일 자식 프로세스가 그 자
신의 자식 프로세스(자식의 자식)들이 종료됨을 기다리지 않았다면 자식의 자

항 목	형 식	내 용
clock_t	*tms_utime*	사용자 CPU 시간
clock_t	*tms_utime*	시스템 CPU 시간
clock_t	*tms_utime*	종료된 자식 프로세스의 사용자 CPU 시간
clock_t	*tms_utime*	종료된 자식 프로세스의 시스템 CPU 시간

식에서의 시간은 자식 프로세스의 시간계산에 포함되지 않는다.

tms_utime 값은 사용자 인스트럭션의 실행을 위하여 사용된 CPU 시간이다.

tms_stime 값은 프로세스를 처리하기 위하여 시스템이 수행한 CPU 시간이다.

*tms_cutime*은 자식 프로세스의 *tms_utime*과 *tms_cutime*을 더한 값이다.

*tms_cstime*은 자식 프로세스의 *tms_stime*과 *tms_cstime*을 더한 값이다.

5.5.2.3 복귀값

함수의 성공적인 완료시 *times*()는 어떤 임의의 시각 (예를 들어 시스템의 동작개시 시각 등) 이후로 경과한 (초당 {CLK_TCK} 간격으로 표현되는) 실제 시간을 복귀한다. 지정된 시각은 프로세스내에서 *times*()를 호출함에 의하여 변경되지 않는다. 복귀값은 *clock_t*형의 표현가능한 범위를 초과 할 수 있다. 만일 *times*()함수의 실행이 실패하면 복귀값으로 ((*clock_t*)-1)값이 복귀되고 *errno*는 지정된 에러값으로 설정된다.

5.5.2.4 에러

본 표준에서는 *times*() 함수에서 검출되는 어떠한 에러조건도 규정하지 않는다. 몇몇 에러는 구현시 정의된 바에 따라 검출된다.

5.5.2.5 참고

exec [4.1.2], *fork*() [4.1.1], *times*() [5.5.1], *wait*() [4.2.1]

5.6 환경 변수

5.6.1 환경관련 함수

함수: *getenv*()

5.6.1.1 용례

#include <stdlib.h>

char *getenv(*name*)
char **name*;

5.6.1.2 설명

getenv()함수는 *name* = *value* 형의 문자열을 환경 목록으로부터 찾아내어 *val*ue를 지시하는 포인터를 복귀한다. 만일 기술된 *name*을 찾을 수 없다면 **널**포인터를 복귀한다.

5.6.1.3 복귀값

함수의 성공적인 완료시 *getenv*()함수는 규정된 *name*를 나타내는 *val*ue를 포함하는 문자열의 포인터를 복귀하거나 규정된 *name*을 찾을 수 없으면 **널**포인터를 복귀한다. *getenv*()로부터의 복귀값은 정적 데이터를 지시할 수 있으며, 따라서 이는 여러번의 호출에 의해 겹쳐 쓰여질 수 있다. 성공하지 못한 종료는 결과적으로 **널**포인터를 복귀한다.

5.6.1.4 에러

본 표준은 *getenv*() 함수에서 검출되는 어떠한 에러조건도 규정하지 않는다. 몇몇 에러는 구현시 정의된 바에 따라 검출된다.

5.6.1.5 참고

environ [4.1.2], **환경 서술 [3.7]**

5.7 터미날의 식별

5.7.1 터미날 경로명의 생성

함수: *ctermid*()

5.7.1.1 용례

#include <stdio.h>

char *ctermid(*s*)
char **s*;

5.7.1.2 설명

ctermid() 함수는 현재 작업중인 프로세스의 제어 터미날을 가르키는 (경로명으로 사용되는) 문자열을 생성한다.

만일 *ctermid*() 함수가 경로명을 복귀하면 파일에의 접근은 보장되지 않는다.

5.7.1.3 복귀값

*s*가 **널**포인터이면 문자열은 정적 지역에 생성되고 이 지역의 주소가 복귀된다. 이외의 경우에 *s*는 최소한 L_ctermid 바이트 길이의 문자배열을 지시하는 것으로 간주한다. 즉, 문자열은 이 배열내에 위치하고 *s*의 값이 복귀된다. 심볼상수 L_ctermid는 **<stdio.h>**에서 정의된 0보다 큰 값이다.

ctermid() 함수는 제어 터미널을 가르키는 경로명이 결정될 수 없거나 성공적으로 수행되지 않으면 공백 문자열을 복귀한다

5.7.1.4 에러

본 표준은 *ctermid*() 함수에서 검출되는 어떠한 에러조건도 규정하지 않는다. 몇몇 에러는 구현시 정의된 바에 따라 검출된다.

5.7.1.5 참고

ttyname() [5.7.2]

5.7.2 터미날 디바이스명의 결정

함수: *ttyname*(), *isatty*()

5.7.2.1 용례

char *ttyname(*fildes*)
int *fildes*;

int isatty(*fildes*)
int *fildes*;

5.7.2.2 설명

ttyname() 함수는 파일 서술자 *fildes*에 따른 터미날의 (널로 종료되는) 경로명을 포함하는 문자열로의 포인터를 복귀한다.

ttyname()의 복귀값은 반복해서 겹쳐서 쓰일 수 있는 정적 데이터를 지시할 수 있다.

isatty() 함수의 복귀값은 *fildes*가 어떤 터미날의 유효한 파일 서술자일 경우에는 1이 되고, 이외의 경우에는 0이다.

5.7.2.3 복귀값

ttyname() 함수의 복귀값은 *fildes*이 어떤 터미날의 유효한 파일 서술자이거나 경로명을 결정할 수 없을 경우에는 **널**포인터가 된다

5.7.2.4 에러

본 표준은 *ttyname*() 또는 *isatty*() 함수에서 검출되는 어떠한 에러조건도 규정하지 않는다. 몇몇 에러는 구현시 정의된 바에 따라 검출된다.

5.8 설정 가능한 시스템 변수

5.8.1 설정 가능 시스템 변수의 획득

함수: *sysconf*()

5.8.1.1 용례

#include <unistd.h>

long sysconf(*name*)
int *name*;

5.8.1.2 설명

sysconf() 함수는 응용으로 하여금 설정가능한 시스템 한계나 부수사항(예를 들어 변수들)의 현재 값을 결정하는 방법을 제공한다.

name 인수는 조사되어야 하는 시스템 변수를 나타낸다. 구현물은 표5-2에 나열된 모든 변수를 지원하여야 하며, 이외의 사항도 부수적으로 지원할 수 있다. 표 5-2의 변수는 **<limits.h>** [3.9] 또는 **<unistd.h>** [3.10] (또는 C-표준의 {CLK_TCK}와 같이 **<time.h>**)으로부터 발췌한 것이며, *name*과 함께 사용되는 **<unistd.h>**에 정의된 심볼상수이다.

{CLK_TCK}값은 C-표준에 따라 (본 표준에서도) 실행시에 값이 평가되도록 허용된다.

sysconf()의 복귀값인 {SC_CLK_TCK} 값은 {CLK_TCK}으로 복귀된 값과 같다

<표5-2> 설정가능한 시스템 변수

변수	*name* 값
{ARG_MAX}	{_SC_ARG_MAX}
{CHILD_MAX}	{_SC_CHILD_MAX}
{CLK_TCK}	{_SC_CLK_TCK}
{NGROUPS_MAX}	{_SC_NGROUPS_MAX}
{OPEN_MAX}	{_SC_OPEN_MAX}
{_POSIX_JOB_CONTROL}	{_SC_POSIX_JOB_CONTROL}
{_POSIX_SAVED_IDS}	{_SC_POSIX_SAVED_IDS}
{_POSIX_VERSION}	{_SC_POSIX_VERSION}

5.8.1.3 복귀값

만일 *name*이 부적당한 값이면 *sysconf*()의 복귀값은 -1이다. 만일 *name*에 상응하

는 변수를 시스템에서 정의하지 않았다면 *sysconf*()는 *errno*값을 그대로 두고 -1값을 복귀한다.

이외의 경우에, *sysconf*() 함수의 복귀값은 시스템에서의 현재 변수의 값이 된다. 복귀되는 값은 변수들이 구현물의 **<limit.h>** [3.9] 또는 **<unistd.h>** [3.10]에 의하여 해석되어질 때, 응용에서 묘사된 상응하는 값보다 더 제한적이지 않아야 한다. 이 값은 호출 프로세스의 수명주기동안 변하지 않는다.

5.8.1.4 에러

아래의 조건에서 *sysconf*()함수의 복귀값은 -1이고 *errno*는 상응하는 값으로 설정된다.

[EINVAL] *name*인수의 값이 부적당하다.

제 6장 파일과 디렉토리

본 장의 함수는 파일과 디렉토리의 생성 및 삭제, 파일 특성의 검사 및 변경을 처리하는 운영체제 서비스를 수행하는 것이다. 이 함수들은 프로세스가 출력을 수행하기 전에 파일과 디렉토리에 접근하기 위한 주된 방법을 제공한다. (7장 **입출력 관련기본 함수** 참조)

6.1 디렉토리

6.1.1 디렉토리 엔트리의 포맷

헤더 **<dirent.h>**는 directory 루틴이 사용하는 구조와 자료형을 정의한다. **디렉토리**의 내부 포맷은 구현시 정의된다.

readdir() 함수는 다음 원소들을 포함하는 struct dirent형의 객체에 대한 포인터를 복귀값으로 한다.

원소형	원소명	설 명
char[]	*d_name*	파일 명 (널로 끝남)

문자배열 d_*name*의 크기가 정해져 있는 것은 아니지만, 마지막의 널문자를 제외한 바이트 수가 반드시 {NAME_MAX}를 넘지 않아야 한다.

6.1.2. 디렉토리 관련 함수

함수: *opendir*(), *readdir*(), *rewinddir*(), *closedir*()

6.1.2.1 용례

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(dirname)
char *dirname;

struct dirent *readdir(dirp)
DIR *dirp;

void rewinddir(dirp)
DIR *dirp;

int closedir(dirp)
DIR *dirp;
```

6.1.2.2 설명

헤더 **<dirent.h>** [6.1.1] 에서 정의된 자료형 DIR은 **디렉토리 스트림**을 나타내는데, 디렉토리 스트림이란 특정 디렉토리 내의 모든 디렉토리 엔트리들이 순서대로 나열된 것을 말한다. 디렉토리 엔트리는 파일을 표현한다. 파일은 본 절에서 설명하는 함수와 비동기적으로 디렉토리에서 제거되거나 추가될 수 있다. DIR형은 화일 서술자를 사용하여 구현할 수 있다.

이 경우 응용이 열 수 있는 파일과 디렉토리는 모두 합쳐서 최대 *{OPEN_MAX}*개까지이다. *(open () [6.3.1] 참조)*. 파일 서술자를 사용할때는 반드시 그 파일 서술자의 FD_CLOEXEC 플래그를 1로 해야 한다.(**<fcntl.h>** [7.5.1] 참조)

opendir()함수는 인수 *dirname*가 지정하는 디렉토리에 해당하는 디렉토리 스트림을 열고 첫번째 엔트리를 디렉토리 스트림의 현재 위치로 한다.

readdir()함수는 *dirp*가 가리키는 디렉토리 스트림에서 현재 위치의 디렉토리 엔트리가 표시하는 구조에 대한 포인터를 복귀값으로 하고, 현재 위치를 다음 엔트리로 바꾼다. 디렉토리 스트림의 끝에 도달하면 복귀값은 NULL포인터가 된다.

readdir()함수가 빈(empty) 이름을 포함하는 디렉토리 엔트리를 복귀값으로 해서는 안된다. 도트 파일이나 도트 도트 파일을 위한 엔트리가 존재하면, 도트 파일을 위한 엔트리 하나와 도트 도트 파일을 위한 엔트리 하나가 복귀값으로 되어야 한다. 그렇지 않은 경우에는 이들을 복귀값으로 해서는 안된다.

readdir()의 복귀값이 되는 포인터는 같은 디렉토리 스트림에 대한 또 다른 *readdir*()호출에 의해서 덧씌워질 수 있는 데이터를 가리킬 수도 있다. 그러나 이 데이터가 다른 디렉토리 스트림에 대한 *readdir*()호출에 의해 덧씌워지는 일은 없어야 한다.

readdir()함수는 실제 읽기 함수 하나에 대해 여러 디렉토리를 버퍼하도록 할 수도 있다. 디렉토리를 실제 읽을 때마다 *readdir*()함수는 디렉토리의 *st_atime*필드가 갱신되도록 표시를 해놓아야 한다.

rewinddir()함수는 *dirp*가 가리키는 디렉토리 스트림의 현재 위치를 디렉토리의 맨 앞으로 바꾼다. *opendir*()과 마찬가지로 *rewinddir*()함수도 디렉토리 스트림이 해당 디렉토리의 현재 상태를 가리키게 하지만 복귀값은 없다.

가장 최근의 *opendir*()이나 *rewinddir*()함수 호출 이후에 파일이 디렉토리에서 제거되거나 추가되었을 때, 다음 *readdir()* 이 그 파일에 해당하는 엔트리를 복귀값으로 갖게 할 것인지는 미규정이다.

closedir()함수는 *dirp* 가 가리키는 디렉토리 스트림을 닫는다. 성공적으로 끝났으면 복귀값이 0이되고, 아니면 -1로 되어 에러가 있음을 표시한다. 복귀할 때 *dirp*는 DIR형의 객체를 가리키지 않을 수도 있다. DIR형을 구현하는데 파일 서술자를 사용했으면 그 파일 서술자를 닫아야 한다.

위의 함수들에 전달되는 *dirp* 인수가 현재 열린 디렉토리 스트림을 가리키지 않을 때의 효과는 미정의이다.

*exec*계열의 함수 중 어느 하나를 수행한 후에 디렉토리 스트림을 사용한 결과는 미정의이다. *fork*() 함수는 호출한 후에 부모 프로세스나 자식 프로세스가 *readdir*()이나 *rewinddir*(), 또는 두 함수 모두를 사용해서 디렉토리 스트림에 대한 처리를 계속하게

할 수 있다. 그러나 부모와 자식 프로세스가 모두 처리를 계속할 수는 없다. 만일 부모와 자식 프로세스갸 모두 이 함수들을 사용하면 그 결과는 미정의이다. 단 *closedir*()은 둘 중에 하나 또는 둘 다 사용할 수도 있다.

6.1.2.3 복귀값

성공적으로 끝나면 *opendir*()은 DIR형의 객체에 대한 포인터를 복귀시킨다. 아니면 **NULL**이 복귀값이 되고 에러 종류를 표시하도록 *errno*가 정해진다.

readdir()함수가 성공적으로 끝나면 struct dirent형의 객체에 대한 포인터가 복귀값이 된다. 에러가 발생하면 복귀값은 **NULL**이 되고 에러에 상응하는 값을 *errno*에 설정한다. 디렉토리의 끝을 만나면 복귀값이 **NULL**이 되지만 이 함수 호출에 의해 *errno*의 값이 변하지는 않는다.

closedir()함수가 성공적으로 끝나면 복귀값이 0가 된다. 아니면 -1이 복귀값이 되고 에러에 상응하는 값을 *errno*에 설정한다.

6.1.2.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면. *opendir*()의 복귀값이 NULL이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] *dirname*의 구성 요소 중 탐색 접근이 허용되지 않는 것이 있거나 *dirname*에 대한 읽기가 허용되지 않는다.

[ENAMETOOLONG] *dirname*인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명의 구성 요소가 {NAME_MAX}보다 길다.

[ENOENT] 지정된 디렉토리가 존재하지 않는다.

[ENOTDIR] *dirname*의 구성 요소 중 디렉토리가 아닌 것이 있다.

[EMFILE] 이 프로세스가 너무 많은 파일 서술자를 열고 있다.

[ENFILE] 이 시스템 내에 너무 많은 파일 서술자가 열려있다

다음 중 어느 조건에라도 해당하는 일이 발생하면, *readdir*()의 복귀값이 NULL이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EBADF] *dirp*인수가 열린 디렉토리 스트림을 가리키고 있지 않다.

다음 중 어느 조건에라도 해당하는 일이 발생하면, *closedir*()의 복귀값이 -1이 되면서 에러 조건에 따라 *errno*값이 정해져야 한다.

[EBADF] *dirp*인수가 열린 디렉토리 스트림을 가리키고 있지 않다.

6.1.2.5 참고

<dirent.h> [6.1.1]

6.2 작업 디렉토리

6.2.1 현재 작업 디렉토리의 변경

함수: *chdir*()

6.2.1.1 용례

int chdir(*path*)
char **path*;

6.2.1.2 설명

*path*인수는 디렉토리 경로명을 가리킨다. *chdir*()함수는 지정된 디렉토리가 현재 작업 디렉토리가 되게 한다. 즉 사선으로 시작하지 않은 경로명의 경로 탐색을 여기서부터 시작하게 한다.

chdir()함수가 실패하면 현재 작업 디렉토리는 변경되지 않는다.

6.2.1.3 복귀값

성공적으로 끝나면 복귀값이 0이 된다. 아니면 -1이 복귀값이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

6.2.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *chdir*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로명의 구성 요소 중 탐색 접근이 허용되지 않는 것이 있다.
[ENAMETOOLONG] *path* 인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명 구성 요소중 {NAME_MAX}보다 긴 것이 있다.
[ENOTDIR] 경로명의 구성 요소 중 디렉토리가 아닌 것이 있다.
[ENOENT] 지정된 디렉토리가 존재하지 않거나 *path*가 공백 문자열이다.

6.2.1.5 참고

getcwd() [6.2.2]

6.2.2 작업 디렉토리 경로명

함수: *getcwd*()

6.2.2.1 용례

char *getcwd(*buf*,*size*)
char **buf*;
int *size*;

6.2.2.2 설명

getcwd()함수는 현재 작업 디렉토리의 절대 경로명을 인수 *buf*가 가리키는 문자배열에 복사한 후, 그 포인터를 복귀시킨다. *size*인수는 *buf*인수가 가리키는 문자배열의 크기를 바이트로 나타낸 것이다. *buf*가 NULL포인터일 때 *getcwd*()의 작동 방식은 미정의이다.

6.2.2.3 복귀값

성공적으로 끝나면 *buf*인수가 복귀값이 된다. 에러가 발생했으면 널포인터를 복귀값으로 하면서 에러에 상응하는 값을 *errno*에 설정한다. 에러가 발생한 후의 *buf*내용은 미정의이다

6.2.2.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, 반드시 *getcwd*()의 복귀값을 NULL로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EINVAL] *size*인수의 값이 0보다 작거나 같다.

[ERANGE] *size*인수의 값이 0보다 크지만 경로명에 1 더한 값보다 작다.

[EACCES] 경로명의 구성 요소 중 읽기 또는 탐색 접근이 허용되지 않는 것이 있다.

6.2.2.5 참고

chdir() [6.2.1]

6.3 일반 파일의 생성

6.3.1 파일의 열기

함수: *open*()

6.3.1.1 용례

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(path,oflag,...)
char *path;
int oflag;
```

6.3.1.2 설명

open()함수는 파일과 파일 서술자 사이의 연결을 이루어준다. 이 함수는 파일을 가리키는 열린 파일 서술과 열린 파일 서술을 가리키는 파일 서술자를 만든다. 이 파일 서술자는 다른 입출력 함수가 이 파일을 참조하는데 사용된다. *path* 인수는 파일 을

지정하는 경로명을 가리킨다

open()함수는 반드시 이 프로세스가 아직 열지 않은 가장 작은 값의 파일 서술자를 지정된 파일의 파일 서술자로 정해서 복귀값으로 해야 한다. 열린 파일서술은 새로운 것이므로 시스템내의 다른 프로세스와 파일 서술자를 공유하지 않는다. 파일 오프셋은 반드시 파일의 맨 앞으로 정해져야 한다

열린 파일 서술의 파일 상태 플래그와 파일 접근 모드는 oflag값에 따라 정해져야 한다. oflag의 값은 다음 리스트에 나열된 값들의 조합으로 이루어진다. 심볼상수의 정의에 관해서는 **<fcntl.h>** [7.5.1]을 참고하라. 응용은 oflag값에 다음의 처음 세값(파일 접근 모드) 중에서 하나를 반드시 지정해야 한다. 둘 이상을 지정해서는 안된다.

O_RDONLY	읽기 전용으로 열기
O_WRONLY	쓰기 전용으로 열기
O_RDWR	읽기와 쓰기겸용으로 열기. FIFO파일에 이 플래그를 적용하면 그 때의 결과는 미정의

나머지 플래그들에 대해서는 어떤 조합도 oflag내에 지정할 수 있다

O_APPEND	이 플래그가 1이면 파일에 쓰기 전에 파일 오프셋을 파일 맨 뒤로 바꾸어야 한다.
O_CREAT	이 옵션은 세번째 인수 *mode*를 필요로 한다. *mode*는 *mode_t*형이다. 있는 파일이면 다음 설명할 O_EXCL과 연관된 경우를 제외하고는 아무 효과도 없다. 없는 파일이면 새로 파일을 만들고 파일의 사용자 ID를 프로세스의 유효 사용자 ID와 같게 해야 한다. 파일 그룹 ID는 파일이 만들어진 디렉토리의 그룹 ID나 프로세스의 유효그룹 ID과 같게 해야 한다. 프로세스의 파일 모드 생성 마스크(umask [6.3.3] 참조)에서 정해지는 것을 제외하고 파일 접근 허용비트들은 (**<sys/stat.h>** [6.6.1] 참조) *mode*값에 따라 정해져야 한다. *mode*에서 파일 접근 허용 비트가 아니 비트가 1로 되어 있을 때의 효과는 구현시 정의된다. 파일이 읽기나 쓰기, 겸용 중 어느 방식으로 열리는 지에는 *mode*가 영향을 미치지 않는다
O_EXCL	O_EXCL과 O_CREAT가 모두 1이면 있는 파일에 대해 *open*()을 수행할 때 실패하도록 해야한다. 파일이 있는가 검사하고 새 파일을 만드는 작업은 O_EXCL과 O_CREAT가 1 이며 같은 디렉토리 내의 같은 파일을 지정하는 *open*()을 수행하는 다른 프로세스들에 의해 나뉘어지지 않아야(atomic)한다. O_EXCL이 1이고 O_CREAT가 0일때의 결과는 구현시 정의된다.
N_NOCTTY	이 플래그가 1이고 *path*가 터미날 장치를 지정하면 *open*() 함수는 그 터미날 장치가 이 프로세스의 제어 터미날이 되지 않도록 해야 한다. (**제어 터미날** [8.1.1.3] 참조)
O_NONBLOCK	(1) O_RDONLY또는 O_WRONLY가 1인 상태에서 FIFO파일을 열때

(a) O_NONBLOCK이 1이면
읽기전용의 *open*()은 즉시 복귀해야 하고, 쓰기 전용의 *open*()은 읽기 위해 이 파일을 연 프로세스가 없을 때 에러상태로 복귀해야 한다

(b) O_NONBLOCK이 0 이면
읽기 전용의 *open*()은 어떤 프로세스가 쓰기 위해 파일을 열할 때까지 차단 되어야 한다. 쓰기 전용의 *open*()은 어떤 프로세스가 읽기 위해 열 때까지 차단되어야 한다.

(2) 차단되지 않는 열기를 지원하는 블럭형 특수 또는 문자형 특수 파일을 열때

(a) O_NONBLOCK이 1 이면
open()은 장치가 준비 상태 또는 가용 상태가 될 때까지 기다리지 않고 즉시 복귀해야 한다. 장치의 이후 작동 방식은 장치에 따라 다르다

(b) O_NONBLOCK이 0 이면
open()은 장치가 준비 상태 또는 가용 상태가 될 때까지 기다렸다가 복귀해야 한다

(3) (1),(2)의 경우가 아닐 때 O_NONBLOCK의 작동 방식은 미규정이다. O_TRUNC파일이 존재하고, 정규 파일이며, O_RDWR이나 O_WRONLY로 그 파일이 성공적으로 열리었으면, 파일의 길이는 0으로 줄어들어야 한다. 이 함수 호출에 의해서 모드나 소유자가 바뀌어서는 안 된다. FIFO형 특수파일이나 디렉토리에 대해 O_TRUNC는 아무 효과도 없어야 한다. 다른 종류의 파일에 대한 효과는 구현시 정의된다. O_RDONLY와 O_TRUNC를 함께 사용할 때의 결과는 미정의이다

O_CREAT가 1이고 없는 파일일 때 *open*() 함수가 성공적으로 수행되면, 파일의 *st_atime*, *st_ctime*, *st_mtime* 필드와 부모 디렉토리 *st_ctime*, *st_mtime* 필드가 갱신되도록 표시를 해두어야 한다.

O_TRUNC가 1이고 없는 파일일 때 *open*() 함수가 성공적으로 수행되면, 파일의 *st_ctime*과 *st_mtime*필드가 갱신되도록 표시를 해두어야 한다.

6.3.1.3 복귀값

성공적으로 끝나면 파일을 열고 사용되지 않은 최소수의 파일 서술자를 표시하는 음이 아닌 정수를 복귀값으로 해야한다. 아니면 -1을 복귀값으로 하고 에러에 상응하는 값을 *errno*에 설정한다. 복귀값이 -1일 때는 파일이 새로 만들어지거나 변경되지 않아야 한다.

6.3.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면 반드시 *open*()의 복귀값을 -1로

하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성 요소에 대한 탐색 접근이 허용되지 않거나, 파일은 있으나 oflag에 표시된 접근이 허용되지 않거나 파일이 없는데 파일을 만들어야 할 부모 디렉토리에 대한 쓰기 접근이 허용되지 않거나 O_TRUNC가 1인데 쓰기 접근이 허용되지 않는다.

[EEXIST] O_CREAT와 O_EXCL이 1 이고 지정된 파일이 존재한다.

[EINTR] *open*()이 신호에 의해 중단되었다.

[EISDIR] 지정된 파일이 디렉토리인데 oflag 인수가 쓰기가 읽기/쓰기 겸용의 접근을 지정하고 있다.

[EMFILE] 이 프로세스가 너무 많은 파일 서술자를 사용하고 있다.

[ENAMETOOLONG] *path* 문자열의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENFILE] 이 시스템내에 너무 많은 파일이 열려 있다.

[ENOENT] O_CREAT가 0 이고 지정된 파일이 존재하지 않거나, O_CREAT가 1이고 경로 접두어가 존재하지 않거나 *path* 인수가 공백 문자열을 가리킨다.

[ENOSPC] 새 파일을 포함시켜야 할 디렉토리나 파일 시스템을 확장할 수 없다.

[ENOTDIR] 경로명의 구성 요소 중 디렉토리가 아닌 것이 있다.

[ENXIO] O_NONBLOCK이 1 이고, 지정된 파일이 FIFO이며, O_WRONLY가 1일 때 그 파일을 읽기 위해 연 프로세스가 없다.

[EROFS] 지정된 파일을 읽기 전용 파일 시스템내에 있으며, oflag 인수의 O_WRONLY, O_RDWR, O_CREAT(없는 파일인 경우), O_TRUNC 중 적어도 하나가 1이다.

6.3.1.5 참고

close() [7.3.1], *creat*() [6.3.2], *dup*() [7.2.1], *exec* [4.1.2], *fcntl*() [7.5.2], **<fcntl.h>** [7.5.1], *lseek*() [7.5.3], *read*()[7.4.1], **<sys/stat.h>** [6.6.1], *write*() [7.4.2], *umask*() [6.3.3], **신호가 기타 함수에 미치는 영향** [4.3.1.4]

6.3.2 새로운 파일의 생성 및 기존 파일의 새로 쓰기

함수: *creat*()

6.3.2.1 용례

```
#include <sys/types.h>
#include <sys/stat.h>

#include <fcntl.h>
```

int creat(*path*,*mode*)
char **path*;
mode_t *mode*;

6.3.2.2 설명

creat (*path*, *mode*);
는
open (*path*, **O_WRONLY | O_CREAT | O_TRUNC**, *mode*);
와 같다.

6.3.2.3 참고

open() [6.3.1], **<sys/stat.h>** [6.6.1]

6.3.3 파일 생성 마스크의 설정

함수: *umask*()

6.3.3.1 용례

#include <sys/types.h>
#include <sys/stat.h>

mode_t umask(*cmask*)
mode_t *cmask*;

6.3.3.2 설명

umask()는 프로세스 파일 모드 생성 마스크를 *cmask*로 바꾸고 바꾸기 전의 마스크 값을 복귀값으로 한다. *cmask*의 파일 접근 허용 비트(**<sys/stat.h>** [6.6.1] 참조) 만 사용한다. 다른 비트의 의미는 구현시 정의된다.

프로세스의 파일 모드 생성 마스크는 *open*(),*creat*(),*mkdir*(),*mkfifo*() 함수의 호출 중에 공급된 *mode* 인수의 허가 비트를 0으로 하기 위해 사용한다. *cmask*에 1로 되어 있는 비트의 위치에 해당하는 새 파일의 모드 비트는 0이 된다.

6.3.3.3 복귀값

파일 모드 생성 마스크의 변경 전 값이 복귀값이다.

6.3.3.4 에러

umask() 함수는 항상 성공하므로 에러를 표시하기 위한 값을 정해 놓지 않는다.

6.3.3.5 참고

chmod() [6.6.4], *creat*()[6.3.2], *mkdir*() [6.4.1], *mkfifo*() [6.4.2], *open*() [6.3.1], **<sys/stat.h>** [6.6.1]

6.3.4 파일의 링크

함수 : *link*()

6.3.4.1 용례

int link (*path*1, *path*2)
char **path*1, **path*2;

6.3.4.2 설명

인수 *path*1은 존재하는 파일을 지정하는 경로명을 가리킨다. 인수 *path*2는 새로 만들어질 디렉토리 엔트리를 지정하는 경로명을 가리킨다. 다른 파일 시스템에 속한 파일들간의 연결을 지원하도록 구현하는 것은 선택 사항이다. *link*() 함수는 기존 파일에 새로운 링크를 만들고, 이 파일의 링크 수를 하나 증가시켜야 하는데, 이 작업은 반드시 나뉘어 지지 않게 이루어져야 한다.

link()함수가 실패하면, 아무 링크도 만들어지지 않고 그 파일의 링크 수는 변화가 없어야 한다.

사용자가 적절한 권한을 갖고 있으며 그 디렉토리에서 *link*()를 이용하는 것을 구현이 지원하는 디렉토리만을 *path*1 인수로 사용해야 한다. *path*1에 해당하는 파일에 접근이 허용되는 프로세스만이 호출할 수 있도록 구현하는 것은 선택사항이다.

성공적으로 끝나면 *link*()함수는 파일의 *st_ctime* 필드가 갱신되도록 표시해 두어야 한다. 또 새 엔트리를 포함하는 디렉토리의 *st_ctime*과 *st_mtime* 필드도 갱신되도록 표시해야 한다.

6.3.4.3 복귀값

성공적으로 끝나면, *link*()의 복귀값은 0이 되어야 한다. 아니면, -1이 복귀값이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

6.3.4.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *link*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EAACCES] 양 경로 접두어의 구성요소 중 탐색이 허용되지 않는 것이 있거나, 쓰기 접근을 허용하지 않는 디렉토리에 쓰는 작업을 해야 하거나, 호출한 프로세스가 *path*1인수의 파일에 대한 접근 권한이 없는데 구현이 이를 요구하는 경우이다.

[EEXIST] *path*2가 지정하는 링크가 이미 존재한다.

[EMLINK] *path*1이 지정하는 파일에 대한 링크수가 {LINK_MAX}를 넘게된다.

[ENAMETOOLONG] *path*1이나 *path*2문자열의 길이가 {NAME_MAX}보다 길거나, {_POSIX_TRUNC}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[EENOENT] 양 경로 접두어 중 존재하지 않는 것이 있거나. *path*1이 저장하는 파일이 존재하지 않거나. *path*1이나 *path*2가 공백 문자열을 가리킨다.

[ENOSPC] 링크를 포함시켜야 할 디렉토리를 확장할 수 없다.

[ENOTDIR] 양 경로 접두어의 구성 요소 중 디렉토리가 아닌 것이 있다.

[EPERM] *path*1이 지정하는 파일이 디렉토리이고, 호출한 프로세스가 적절한 권한을 갖지 못했거나 그 디렉토리에서 *link*()를 사용할 수 없도록 구현되었다.

[EROFS] 읽기 전용 파일 시스템 상의 디렉토리에 쓰기 작업을 해야 한다.

[EXDEV] ath2가 지정하는 링크와 *path*1이 지정하는 파일이 서로 다른 파일 시스템에 있는데, 다른 파일 시스템간의 링크를 지원하지 않도록 구현되었다.

6.3.4.5 참고

rename() [6.5.3], *unlink*() [6.5.1]

6.4 특수 파일의 생성

6.4.1 디렉토리 만들기

함수: *mkdir*()

6.4.1.1 용례

#include <sys/types.h>
#include <sys/stat.h>

int mkdir(*path*,*mode*)
char **path*;
mode_t *mode*;

6.4.1.2 설명

mkdir()루틴은 *path*를 이름으로 하는 새 디렉토리를 만든다. 새 디렉토리의 파일 접근 허용 비트는 *mode*에 따라 초기화한다. *mode*인수의 파일 접근 허용 비트는 프로세스의 파일 생성 마스크(*umask*() [6.3.3] 참조)로 변경한다. 파일 접근 허용 비트가 아니 다른 비트가 1로 되어 있을 때의 의미는 구현시 정의된다.

디렉토리의 소유자 ID는 프로세스의 유효 사용자 ID와 같은 값으로 정해진다. 디렉토리의 그룹 ID는 이 디렉토리가 만들어진 디렉토리의 그룹 ID나 프로세스의 유효 그룹 ID로 된다.

새로 만든 디렉토리는 공 디렉토리이다.

성공적으로 끝나면 *mkdir*()함수는 디렉토리의 *st_atime*, *st_ctime*, *st_mtime*필드가 갱신되도록 표시해 두어야 한다. 또한 새 엔트리가 들어갈 디렉토리의 *st_atime*, *st_ctime*, *st_mtime* 필드도 갱신되도록 표시해 두어야 한다.

6.4.1.3 복귀값

복귀값 0 은 성공을 표시한다. -1은 에러가 있었음을 표시하며 이 때 에러 코드는 *errno*에 기억시킨다. 에러가 있을 때는 아무 디렉토리도 만들어 지지 않아야 한다.

6.4.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *mkdir*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성요소 중 탐색 접근이 허용되지 않는 것이 있거나, 새로 만들 디렉토리의 부모 디렉토리에 대한 쓰기 접근이 허용되지 않는다.

[EEXIST] 지정된 파일이 이미 존재한다.

[EMLINK] 부모 디렉토리의 링크수가{LINK_MAX}를 넘게 된다.

[ENAMETOOLONG] *path* 인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] 경로 접두어의 구성 요소중 존재하지 않는 것이 있거나, *path*인수가 공백 문자열을 가리킨다.

[ENOSPC] 새 디렉토리의 내용을 기억하거나, 새 디렉토리의 부모 디렉토리를 확장할 만한 공간이 파일 시스템 내에 없다.

[ENOTDIR] 경로 접두어의 구성 요소 중 디렉토리가 아닌 것이 있다.

[EROFS] 새 디렉토리의 부모 디렉토리가 읽기 전용 파일 시스템에 있다.

6.4.1.5 참고

chmod() [6.6.4], *stat*() [6.6.2], **<sys/stat.h>** [6.6.1], *umask*() [6.6.3]

6.4.2 FIFO형 특수 파일 만들기

함수: *mkfifo*()

6.4.2.1 용례

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(path,mode)
char *path;
mode_t mode;
```

6.4.2.2 설명

mkfifo()는 *path*가 가리키는 경로명이 지정하는 FIFO형 특수 파일을 새로 만든다. 새 FIFO의 파일 접근 허용 비트는 *mode*에 따라 초기화한다. *mode*인수의 파일 접근 허용 비트는 프로세스의 파일 생성 마스크 (umask() [6.3.3] 참조)로 변경한다. 파일 접근 허용 비트가 아닌 다른 비트가 1로 되어 있을 때의 효과는 구현시 정의된다.

FIFO의 소유자 ID는 프로세스의 유효 사용자 ID와 같은 값으로 정해져야 한다. FIFO의 그룹 ID는 이 FIFO가 만들어진 디렉토리의 그룹 ID나 프로세스의 유효그룹 ID로 된다.

mkfifo()함수가 성공적으로 끝나면 파일의 *st_atime*, *st_ctime*, *st_mtime*필드가 갱신되도록 표시해 두어야 한다. 또한 새 엔트리가 들어갈 디렉토리의 *st_ctime*, *st_mtime*필드도 갱신되도록 표시해 두어야 한다.

6.4.2.3 복귀값

성공적으로 끝나면, 0 이 된다. 아니면, -1 이 되면서, FIFO는 생성되지 않고, 에러에 상응하는 값을 *errno*에 설정한다.

6.4.2.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *mkfifo*()의 복귀값이 -이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성 요소 중 탐색 접근이 허용되지 않는 것이 있다.

[EEXIST] 지정된 파일이 이미 존재한다.

[ENAMETOOLONG] *path* 문자열의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] 경로 접두어의 구성 요소 중 존재하지 않는 것이 있거나 *path* 인수가 공백 문자열을 가리킨다.

[ENOSPC] 새 파일을 포함시켜야 할 디렉토리를 확장할 수 없거나 파일 시스템에 할당할 자원이 더 이상 없다.

[ENOTDIR] 경로 접두어의 구성 요소 중 디렉토리가 아닌 것이 있다.

[EROFS] 지정된 파일이 읽기 전용 파일 시스템에 있다.

6.4.2.5 참고

chmod() [6.6.4], *exec* [4.1.2], *pipe*() [7.1.1], *stat*() [6.6.2], **<sys/stat.h>** [6.6.1], *umask*() [6.6.3]

6.5 파일의 제거

6.5.1 디렉토리 엔트리의 제거

함수: *unlink*()

6.5.1.1 용례

int unlink(*path*)
char **path*;

6.5.1.2 설명

unlink()함수는 *path*가 가리키는 경로명이 지정하는 링크를 제거하고 이 링크가 가리키는 파일의 링크 수를 하나 감소시킨다.

파일의 링크 수가 0이 되고 이 파일을 연 프로세스가 하나도 없으면, 이 파일이 사용하던 공간을 회수하고 이 파일은 더 이상 접근할 수 없게 해야 한다. 마지막 링크를 제거할 때 이 파일을 연 프로세스가 하나라도 있으면, *unlink*()에서 복귀하기 전에 이 링크는 제거해야 하지만, 이 파일을 연 프로세스가 모두 닫을 때까지 파일 내용을 지우는 것은 연기해야 한다.

프로세스가 적절한 권한은 갖고 있으며 디렉토리에 *unlink*()를 사용하는것을 지원하는 구현에서만 *path* 인수가 디렉토리를 지정할 수 있다. 응용이 디렉토리를 제거하기 위해서 *rmdir*()을 사용할 수 있게 하는 것은 권장 사항이다.

unlink()함수가 성공적으로 끝나면, 부모 디렉토리의 *st_ctime*, *st_mtime* 필드가 갱신되도록 표시해 두어야 한다. 또한 파일의 링크 수가 0 이 아니면, 파일의 *st_ctime*필드도 갱신되도록 표시해 두어야 한다.

6.5.1.3 복귀값

성공적으로 끝나면, 복귀값이 0이 되어야 한다. 아니면, -1이 되어야 하고 에러에 상응하는 값을 *errno*에 설정한다. -1이 되면 지정된 파일은 이 함수 호출에 의해 변하지 말아야 한다.

6.5.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *unlink*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성요소 중 탐색 접근이 허용되지 않는 것이 있거나, 제거할 링크를 기억하고 있는 디렉토리에 대한 쓰기 접근이 허용되지 않는다.

[EBUSY] *path*인수가 지정하는 디렉토리를 시스템이나 다른 프로세스가 사용하고 있어서 링크를 제거할 수 없으며, 이를 에러로 취급하도록 구현하였다.

[ENAMETOOLONG] *path* 문자열의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] 지정된 파일이 존재하지 않거나 *path*인수가 공백 문자열을 가리킨다.

[ENOTDIR] 경로 접두어의 구성 요소 중 디렉토리가 아닌 것이 있다.

[EPERM] *path*가 지정하는 파일이 디렉토리이고, 호출한 프로세스가 적절한 권한을 갖지 못했거나 그 디렉토리에서 *unlink*()를 사용할 수 없도록 구현되었다.

[EROFS] 링크시킬 디렉토리의 엔트리가 전용 파일 시스템에 있다.

6.5.1.5 참고

close() [7.3.1], *link*() [6.3.4], *open*() [6.3.1], *rename*() [6.5.3], *rmdir*() [6.5.2]

6.5.2 디렉토리의 제거

함수: *rmdir*()

6.5.2.1 용례

int rmdir(*path*)
char **path*;

6.5.2.2 설명

rmdir()함수는 *path*가 지정하는 디렉토리를 제거한다. 반드시 공 디렉토리만이 제거되도록 해야 한다.

지정된 디렉토리가 루트 디렉토리이거나 어떤 프로세스의 현재 작업 디렉토리일

때의, 이 함수 효과는 구현시 정의된다.

디렉토리의 링크 수가 0이 되고 이 디렉토리를 연 프로세스가 하나도 없으면, 이 디렉토리가 사용하던 공간을 회수되고 이 파일은 더 이상 접근할 수 없게 해야 한다. 마지막 링크를 제거할 때 이 파일을 연 프로세스가 하나라도 있으면, *rmdir*()에서 복귀하기 전에 도트와 도트 도트 엔트리를 제거해야 한다. 이 디렉토리에는 새로운 엔트리를 만들 수 없게 하여야 하며, 이 디렉토리를 연 프로세스가 모두 닫을 때까지 디렉토리 내용은 제거하지 않는다.

rmdir()함수가 성공적으로 끝나면, 부모 디렉토리의 *st_ctime*, *st_mtime*필드가 갱신되도록 표시해 두어야 한다.

6.5.2.3 복귀값

성공적으로 끝나면, 복귀값이 0이 되어야 한다. 아니면, -1이 되어야 하고 에러에 상응하는 값을 *errno*에 설정한다.

6.5.2.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *rmdir*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로의 구성 요소 중 탐색 접근이 허용되지 않는 것이 있거나, 제거할 디렉토리의 부모 디렉토리에 대한 쓰기 접근이 허용되지 않는다.

[EBUSY] *path*인수가 지정하는 디렉토리를 다른 프로세스가 사용하고 있으며 이를 에러로 취급하도록 구현하였기 때문에 제거할 수 없다.

[EEXIST] 또는 [ENOEMPTY]
*path*인수가 공 디렉토리가 아닌 디렉토리를 지정하고 있다.

[ENAMETOOLONG] *path*인수의 길이가 {PATH_MAX}보다 길거나, {_POSIX_NO_TRUNC}가 발효중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] 없는 디렉토리가 지정되었거나 *path*인수가 공백 문자열을 가리키고 있다.

[ENOTDIR] 경로의 구성 요소 중 디렉토리가 아닌 것이 있다.

[EROFS] 제거할 디렉토리 엔트리가 읽기 전용 파일 시스템에 있다.

6.5.2.5 참고

mkdir() [6.4.1], *unlink*() [6.5.1]

6.5.3 파일명의 변경

함수: *rename*()

6.5.3.1 용례

int rename(*old*,*new*)
char **old*;
char **new*;

6.5.3.2 설명

rename()함수는 파일의 이름을 변경한다. *old*인수는 변경할 파일의 경로명을 가리키며, *new*인수는 파일의 새 경로명을 가리킨다.

old 인수와 *new*인수가 모두 같은 파일에 대한 링크를 가리키면, *rename*()함수는 일단 성공적으로 끝나고 실제는 아무일도 하지 않아야 한다.

*old*인수가 디렉토리가 아닌 파일의 경로명을 가리킬 때, *new*인수가 디렉토리의 경로명을 가리키면 안된다. *new*인수가 지정하는 링크가 이미 있으면, 그 링크를 제거하고 *old*의 이름을 *new*로 바꾼다. 이 경우 *new*가 지정하는 링크는 이후의 작업을 하는 동안 내내 존재해야 하며 *new*가 가리키는 또는 작업이 시작되기 전에 *old*가 가리키던 파일을 나타내야 한다. 따라서 *new*가 이미 존재하는 디렉토리를 지정하면, 공백디렉토리일 것을 권장한다.

*new*경로명은 *old*를 가리키는 경로 접두어를 포함해서는 안된다. *old*를 포함하는 디렉토리와 *new*를 포함하는 디렉토리에 대한 쓰기 접근 권한이 있어야 한다. *old*인수가 디렉토리의 경로명을 가리키면, *old*가 지정하는 디렉토리에 대해 쓰기 접근 허용이 필요할 수가 있으며, *new*가 지정하는 디렉토리가 이미 존재하는 경우는 이에 대한 쓰기 접근 허용도 필요할 수 있다.

new 인수가 지정하는 링크가 존재하고 이 링크를 제거하면 파일의 링크수가 0이 되며 이 파일을 연 프로세스가 없을 때, 이 파일이 차지하고 있던 공간을 회수하며 이파일은 더이상 접근한 수 없도록 해야 한다. 마지막 링크를 제거할 때 이 파일을 연 프로세스가 하나라도 있으면, *rename*()에서 복귀하기 전에 이 링크를 제거해야 하지만 파일 내용을 지우는 것은 이 파일에 대한 모든 참조가 닫아질 때까지 연기해야 한다.

성공적으로 끝나면 각 파일의 부모 디렉토리의 st_stime과 *st_mtime*필드가 갱신되도록 표시해 두어야 한다.

6.5.3.3 복귀값

복귀값 0은 성공을 표시한다. -1은 에러가 있었음을 표시하고 에러 코드는 *errno*에 기억되어 있다.

6.5.3.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *rename*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 두 경로 접두어의 구성 요소 중 탐색이 허용되지 않는 것이 있거나, *old*나 *new*를 포함하는 디렉토리중 쓰기가 허용되지 않는 것이 있거나, 쓰기 접근이 허용되어야 하는데 *old*나 *new*인수가 가리키는 디렉토리

에 접근할 수 없다.

[EBUSY] *old*나 *new*가 지정하는 디렉토리를 시스템이나 다른 프로세스가 사용하고 있어서 이름을 변경할 수 없으며, 이를 에러로 취급하도록 구현하였다.

[EEXIST] 또는 [ENOEMPTY]
*new*가 가리키는 링크가 도트 도트나 도트 파일이 아니 다른 엔트리를 포함하는 디렉토리이다.

[EINVAL] *new*디렉토리 경로명이 *old*디렉토리를 가리키는 경로 접두어를 포함한다.

[EISDIR] *new*인수가 디렉토리를 가리키고 *old*인수는 디렉토리가 아닌 파일을 가리킨다.

[ENAMETOOLONG] *old*나 *new*인수의 길이가 {PATH_MAX}보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] *old*인수가 지정하는 링크가 존재하지 않거나 *old*나 *new*가 공백 문자열을 가리킨다.

[ENOSPC] *new*를기억할 디렉토리를 확장할 수 없다.

[ENOTDIR] 두 경로명 접두어의 구성 요소 중 디렉토리가 아닌 것이 있다. 또는 *old*인수가 디렉토리를 가리키는데 *new*인수는 디렉토리 아닌 파일을 가리킨다.

[EROFS] 요청한 작업이 읽기 전용 파일 시스템의 디렉토리에 쓰는 일을 필요로 한다.

[EXDEV] *new*와 *old*가 가리키는 링크가 다른 파일 시스템에 있는데, 다른 파일 시스템간의 링크를 지원하지 않도록 구현되었다.

6.5.3.5 참고

link() [6.3.4], *rmdir*() [6.5.2], *unlink*() [6.5.1]

6.6 파일 특성

6.6.1 파일 특성 : 헤더와 자료 구조

헤더 **<sys/stat.h>**는 구조 stat을 정의한다. stat는 표 6-1의 원소를 포함하며 함수 *stat*()과 *fstat*()을 이용하여 그 내용을 조회할 수 있다.

<표 6-1> **stat 구조**

원소형	원소명	설 명
mode_t	*st_mode*	파일 모드(**<sys/stat.h> 파일 모드** [6.6.1.2] 참조)
ino_t	*st_ino*	파일의 일련 번호
dev_t	*st_dev*	이 장치를 포함하는 장치의 ID 파일의 이런 번호와 장치 ID를 함께 사용하면 시스템내의 모든 파일을 구별할 수 있다.
nlink_t	*st_nlink*	링크의 갯수
uid_t	*st_uid*	파일 소유자의 사용자 ID
gid_t	*st_gid*	파일 그룹의 그룹 ID
off_t	*st_size*	정규 파일의 경우, 바이트 단위로 표시한 파일 크기. 다른 파일 타입의 경우, 이 필드의 사용은 미규정이다.
time_t	*st_atime*	최근의 접근 시각
time_t	*st_mtime*	최근의 데이터 수정 시각
time_t	*st_ctime*	최근의 파일 상태 변경 시각

표에 있는 원소는 하나도 빠짐없이 stat 구조에 있어야 한다. st_*mode*, st_ino, st_dev, st_*uid*, st_gid, *st_atime*, *st_ctime*, *st_mtime*은 본 표준에서 정의하는 모든 종류의 파일에 대해 반드시 의미있는 값을 가져야 한다. st_link는 이 파일의 링크수와 같은 값을 갖아야 한다.

6.6.1.1 <sys/stat.h> 파일 유형

다음 매크로들은 어떤 파일이 특정한 타입인가를 검사하는 기능을 갖아야 한다. 매크로에 전달되는 값 m은 stat구조의 st_*mode*값이다. 검사 결과가 사실이면 매크로는 0아닌 값을 내고, 사실이 아니면 0값을 낸다.

S_ISDIR(m)	디렉토리 파일인가 검사하는 매크로
S_ISDIR(m)	특별한 문자 파일인가를 검사하는 매크로
S_ISDIR(m)	특별한 블럭 파일인가를 검사하는 매크로
S_ISDIR(m)	정규 파일인가를 검사하는 매크로
S_ISDIR(m)	특별한 파이프 또는 FIFO파일인가를 검사하는 매크로

6.6.1.2 <sys/stat.h> 파일 모드

*mode_t*형인 파일 모드 부분(예를 들면 st_*mode*)의 값은 다음의 마스크와 비트들로 인코드된다.

S_IRWXU	파일 소유자 클래스에 대한 읽기,쓰기,탐색(디렉토리인 경우),실행(디렉토리가 아닌경우)의 허용 마스크
S_IRUSR	파일 소유자 클래스에 대한 읽기 허용 비트

S_IWUSR 파일 소유자 클래스에 대한 쓰기 허용 비트

S_IXUSR 파일 소유자 클래스에 대한 탐색(디렉토리인 경우), 또는 실행(디렉토리가 아닌 경우) 허용 마스크

S_IRWXG 파일 그룹 클래스에 대한 읽기,쓰기,탐색(디렉토리인 경우),실행(디렉토리가 아닌경우)의 허용 마스크

S_IRGRP 파일 그룹 클래스에 대한 읽기 허용 비트

S_IWGRP 파일 그룹 클래스에 대한 쓰기 허용 비트

S_IXGRP 파일 그룹 클래스에 대한 탐색(디렉토리인 경우), 또는 실행(디렉토리가 아닌 경우) 허용 비트

S_IRWXO 기타 파일 클래스에 대한 읽기,쓰기,탐색(디렉토리인 경우),실행(디렉토리가 아닌경우)의 허용 비트

S_IROTH 기타 파일 클래스에 대한 읽기 허용 비트

S_IWOTH 기타 파일 클래스에 대한 쓰기 허용 비트

S_IXOTH 기타 파일 클래스에 대한 탐색(디렉토리인 경우) 또는 실행(디렉토리가 아닌 경우) 허용 비트

S_ISUID 실행 중 사용자 ID의 설정. 파일이 프로그램으로 실행될 때(*exec* 참조) 프로세스의 유효 사용자 ID는 반드시 파일 소유자의 ID로 설정되어야 한다. 정규 파일의 경우 쓰기가 시행된 후에는 이 비트가 0이 되는 것을 권장한다.

S_ISGID 실행 중 그룹 ID의 설정. 파일이 프로그램으로 실행될 때(*exec* 참조) 프로세스의 그룹 ID는 반드시 파일의 그룹으로 설정되어야 한다. 정규 파일의 경우 쓰기가 시행된후에는 이 비트가 0이 되는 것을 권장한다.

S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH, S_ISUID, S_ISGID가 정의하는 각 비트는 모두 서로 달라야 한다. S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR을 비트 단위 OR 연산한 값이어야 한다. S_IRWXG는 S_IRWXG, S_IRWXO값을 정할 때 다른 구현시 정의 비트들을 포함시켜 비트 단위 OR연산해도 좋다. 그러나 이 구현시 정의 비트들이 본 표준에서 정의하는 다른 비트들과 중첩되어서는 안된다. **파일 접근 허용 비트**는 S_IRWXU, S_IRWXG, S_IRWXO를 비트단위 OR 연산한 것으로 정의한다.

6.6.1.3 <sys/stat.h> 시간 엔트리

struct stat에서 시간에 관련되는 필드는 다음과 같다.

st_atime 파일 데이터에 접근한 시각. 예를 들면 *read*().

st_mtime 파일 데이터를 수정한 시각. 예를 들면 *write*().

st_ctime 파일 상태를 변경한 시각. 예를 들면 *chmod*().

이 값들의 갱신 방법은 **파일 시간 갱신 [3.4]**에 설명되어 있다.

이 필드들의 값을 직접 바꾸는 본 표준 내의 모든 함수는 함수 정의에서 기술한

방법으로 값의 변경 내용을 기록한다. *st_atime*, *st_mtime*, *st_ctime*등을 직접 변경하는 여타 함수는 구현시 정의되어야 한다.

모든 시각은 **에폭 이후 경과 시간 [3.3]**으로 표시한다.

6.6.1.4 참고

chmod() [6.6.4], *chown*() [6.6.5], *creat*() [6.3.2], *exec*() [4.1.2], *link*() [6.3.4], *mkdir*() [6.4.1.], *mkfifo*() [6.4.2], *pipe*() [7.1.1], *read*() [7.4.1], *unlink*() [6.5.1], *utime*(), [6.6.6], *write*() [7.4.2], *remove*()(C-표준)

6.6.2 파일 상태 정보의 획득

함수: *stat*(), *fstat*()

6.6.2.1 용례

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(path,buf)
char *buf;
struct stat *buf;

int fstat(fildes,buf)
int fildes;
struct stat *buf;
```

6.6.2.2 설명

*path*인수는 파일을 지정하는 경로명을 가리킨다. 지정된 파일에 대한 읽기나 쓰기, 수행 허용은 필요하지 않으나, 경로명에서 파일 앞에 나오는 모든 디렉토리에 대한 탐색을 할 수 있어야 한다. *stat*()함수는 지정된 파일에 대한 정보를 읽어내서 *buf*인수가 가리키는 장소에 써 넣는다.

fstat()함수도 마찬가지로 파일에 대한 정보를 얻어내는 것인데 파일 서술자*fildes*를 사용해서 열린 파일을 지정하는 점이 다르다.

이 외에 다른 파일 접근 제어 방식을 추가로 제공하거나 파일 접근 제어 방식을 다르게 구현한 경우에는, 구현시 정의한 조건에 따라 *stat*()이나 *fstat*()함수가 실패하게 할 수도 있다. 특히 *path*가 지정하는 파일이 없을 때 시스템이 이를 실패로 만들 수 있다.

두 함수 모두 시간에 관련된 필드를 갱신해서 stat구조에 써 넣는다. **파일 시간 갱신**[3.4]을 참조하라.

*buf*는 헤더 **<sys/stat.h>** [6.6.1]에서 정의한 대로 stat구조에 대한 포인터로서, 이 파일에 대한 정보를 저장할 장소를 가리킨다.

6.6.2.3 복귀값

성공적으로 끝나면, 0이 되어야 한다. 아니면, -1이 되어야 하고 에러에 상응하는 값을 *errno*에 설정한다.

6.6.2.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *stat*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성 요소중 탐색 접근이 허용되지 않는 것이 있다.

[ENAMETOOLONG] *path*인수의 길이가 {PATH_MAX}보다 길거나, {_POSIX_NO_TRUNC}가 발효중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] 지정된 파일이 존재하지 않거나 *path* 인수가 공백 문자열을 가리킨다.

[ENOTDIR] 경로 접두어의 구성 요소 중 디렉토리가 아닌 것이 있다.

다음 중 어느 조건에라도 해당하는 일이 발생하면, *fstat*()의 복귀값이 -1이 되면서 에러 조건에 따라 errno 값이 정해져야 한다.

[EBADF] *fildes*인수가 적절한 파일 서술자가 아니다.

6.6.2.5 참고

creat() [6.3.2], *dup*() [7.2.1], *fcntl*() [7.5.2], *open*() [6.3.1], *pipe*() [7.1.1], **<sys/stat.h>** [6.6.1]

6.6.3 파일 접근 허용 검사

함수: *access*()

6.6.3.1 용례

#include <unistd.h>

int access(*path*,*amode*)
char **path*;
int *amode*;

6.6.3.2 설명

access()함수는 *path*인수가 가리키는 경로명이 지정하는 파일에 대한 접근 허용을 검사한다. 검사할 접근 허용은 *amode*가 지정한다. 이때 유효 사용자 ID대신 실제 사용자 ID를, 유효그룹 ID대신 실제 그룹 ID를 사용한다.

*amode*의 값은 검사할 접근 허용 (R_OK,W_OK.X_OK)들을 비트 단위 포괄적(inclusive)OR 연산한 결과나 존재 여부에 대한 검사(F_OK)들 중에 하나가 된다. 심볼 상수의 설명은 *access*() **함수용 심볼상수 [3.10.1]**를 참조하라.

접근 허용을 검사할 때는 **파일 접근 허용 [3.4]**에서 설명한 바와 같이 각각을 개별

적으로 검사해야 한다. 프로세스가 적절한 권한만 가지고 있다면, 실행 파일 접근허용 비트 중 1로 되어 있는 것이 하나도 없더라도 X_OK를 성공한 것으로 구현할 수도 있다.

6.6.3.3 복귀값

요청한 접근이 허용되면, 0 값을 복귀값으로 한다. 아니면, -1이 되어야 하고 에러에 상응하는 값을 *errno*에 설정한다.

6.6.3.4 에러

다음 조건 중 어느 조건에라도 해당하는 일이 발생하면, *access*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] *amode*가 지정하는 접근이 허용되지 않거나 경로 접두어의 구성 요소 중 탐색 접근이 허용되지 않는 것이 있다.

[ENAMETOOLONG] *path* 인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] *path* 인수가 공백 문자열을 가리키거나 존재하지 않는 파일 이름을 가리킨다.

[ENOTDIR] 경로 접두어의 구성 요소중 디렉토리가 아닌 것이 있다.

[EROFS] 읽기 전용 파일 시스템에 있는 파일에 대한 쓰기 접근을 요청했다.

[EINVAL] *amode*가 적절하지 못한 값을 지정한다.

6.6.3.5 참고

chmo()[6.6.4], *stat*()[6.6.2], **<unistd.h>**[3.10]

6.6.4 파일 모드의 변경

함수 : *chmod*()

6.6.4.1 용례

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod(path,mode)
char *path;
mode_t mode;
```

6.6.4.2 설명

*path*인수는 파일을 지정하는 경로명을 가리켜야 한다. 이 함수를 호출한 프로세스의 유효 사용자 ID가 파일 소유자와 일치하거나 이 프로세스가 적절한 권한을 가지고 있으면, **<sys/stat.h>**[6.6.1]에서 설명한대로 *chmod*()함수는 *mode* 인수의 내용에 따라 지

정된 파일의 S_ISGID, 파일 접근 허용 비트의 값을 설정해야 한다. *mode*의 비트들은 **파일 접근 허용**[3.4]에서 설명한 바와 같이 이 파일과 연관된 사용자 그룹, 기타의 접근 허용을 정의한다. 구현시 정의하는 추가 조건에 의해 *mode*의 S_ISUID와 S_ISGID 비트가 무시되도록 구현할 수도 있다.

호출한 프로세스가 적절한 권한을 갖고 있지 않으며, 파일의 그룹 ID가 유효 그룹 ID나 추가 그룹 ID중 어느 것과도 일치하지 않고, 이 파일이 정규 파일이면, *chmod*()에서 성공적으로 복귀했을 때 파일 모드의 S_ISGID는 반드시 0이 되어 있어야 한다.

chmod() 함수 실행시 파일 열기를 위한 파일 서술자에 미치는 효과는 구현시 정의된다.

성공적으로 끝났을 때, *chmod*() 함수는 파일의 *st_ctime* 필드가 갱신되도록 표시해 놓아야 한다.

6.6.4.3 복귀값

성공적으로 끝났으면, 0 값을 복귀값으로 해야 한다. 아니면, -1 이 되어야 하고 에러에 상응하는 값을 *errno*에 설정한다. -1이 될 때, 파일 모드에는 변화가 없어야 한다.

6.6.4.4 에러

다음 조건 중 어느 조건에라도 해당하는 일이 발생하면, *chmod*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성 요소중 탐색 접근이 허용되지 않는 것이 있다.

[ENAMETOOLONG] *path* 인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNG}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX} 보다 긴 것이 있다.

[ENOTDIR] 경로 접두어의 구성 요소중 디렉토리가 아닌 것이 있다.

[ENOENT] 지정된 파일이 존재하지 않거나 *path*인수가 공백 문자열을 가리킨다.

[EPERM] 유효 사용자 ID가 파일 소유자와 일치하지 않으면 호출한 프로세스가 적절한 권한을 갖고 있지 못하다.

[EROFS] 지정된 파일이 읽기 전용 파일 시스템에 있다.

6.6.4.5 참고

chown()[6.6.5], *mkdir*()[6.4.1], *mkfifo*()[6.4.2], *stat*()[6.6.2], **<sys/stat.h>**[6.6.1]

6.6.5 파일의 소유자 및 그룹 변경

함수 : *chown*()

6.6.5.1 용례

#include <sys/types.h>

int chown(*path*,*owner*,*group*)
char **path*;
uid_t *owner*;
gid_t *group*;

6.6.5.2 설명

*path*인수는 파일을 지정하는 경로명을 가리킨다. 지정된 파일의 사용자 ID와 그룹 ID는 각각 *owner*와 *group*에 기억된 숫자값으로 설정된다.

파일의 사용자 ID와 같은 유효 사용자 ID를 갖거나 적절한 권한을 갖는 프로세스만이 파일의 소유자를 바꿀 수 있다. *path*에 대해 {_POSIX_CHOWN_RESTRICTED}가 발효중이면,

(1) 적절한 권한을 갖는 프로세스만이 소유자를 변경할 수 있도록 제한한다.
(2) 유효 사용자 ID가 파일의 사용자 ID와 같은 프로세스는 소유자를 변경할수 있다.

단 적절한 권한이 없을 때는 *owner*가 파일의 사용자 ID와 같고 *group*이 호출 프로세스의 유효 그룹 ID나 추가 그룹 ID 중 하나와 같을 때만 허용된다.

path 인수가 정규 파일을 가리킬 때, 파일 모드의 S_ISUID와 S_ISGID비트는 *chown*()에서 성공적으로 복귀했을 때 0으로 되어야 한다. 단 적절한 권한을 갖는 프로세스가 호출했을 때 이 비트들을 변경할 것인지 아닌지는 구현시 정의한다. *chown*() 함수가 정규 파일이 아닌 파일에 대해 호출되었고 그 수행이 성공적으로 끝났을 때는 이 비트들을 0으로 하지 않아도 좋다. 이 비트들은 **<sys/stat.h>**[6.6.1]에서 정의한다.

chown() 함수가 성공적으로 수행을 마치면 이 파일의 *st_ctime* 필드가 갱신되도록 표시해 놓아야 한다.

6.6.5.3 복귀값

성공적으로 끝나면, 0이 되어야 한다. 아니면, -1이 되어야 하고 에러에 상응하는 값을 *errno*에 설정한다. 정해진다. -1이 되면 파일의 소유자와 그룹에 변화가 없어야 한다.

6.6.5.4 에러

다음 조건 중 어느 조건에라도 해당하는 일이 발생하면, *chown*()의 복귀값이 -1이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성 요소중 탐색 접근이 허용되지 않는 것이 있다.
[ENAMETOOLONG]*path* 인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.
[ENOTDIR] 경로 접두어의 구성 요소중 디렉토리가 아닌 것이 있다.
[ENOENT] 지정된 파일의 존재하지 않거나 *path* 인수가 공백 문자열을 가리킨다.
[EPERM] 유효 사용자 ID가 파일 소유자와 일치하지 않으며 호출한 프로세스가 적절한 권한을 갖고있지 못하며 {_POSIX_CHOWN_RESTRICTED}가

이런 권한이 필요함을 표시하고 있다.

[EROFS] 지정된 파일의 읽기 전용 파일 시스템에 있다.

[EINVAL] 소유자나 그룹 ID가 적절한 값이 아니며 구현에서 지원하지 않는 값이다.

6.6.5.5 참고

chmod()[6.6.4], **<sys/stat.h>**[6.6.1]

6.6.6 파일 접근 및 수정 시각의 설정

함수 : *utime*()

6.6.6.1 용례

```
#include <sys/types.h>
#include <utime.h>

int utime(path,times)
char *path;
struct utimbuf *times;
```

6.6.6.2 설명

인수 *path*는 파일을 지정하는 경로명을 가리킨다. *utime*()함수는 지정된 파일의 접근및 수정 시각을 설정한다.

times 인수가 **NULL**이면 파일의 접근 및 수정 시각은 현재 시각이 된다. 이런 식으로 *utime*()을 사용하려면 프로세스의 유효 사용자 ID가 파일의 소유자와 일치하거나, 프로세스가 그 파일에 대한 쓰기 허용을 가지고 있거나 적절한 권한을 가지고 있어야 만 한다.

times 인수가 **NULL**이 아니면, times를 utim*buf* 구조의 포인터로 생각하고 지정된 구조에 기억된 값으로 접근 및 변경 시각을 설정한다. 파일의 소유자나 적절한 권한을 갖는 프로세스만이 *utime*()함수를 이런 식으로 사용할 수 있도록 해야 한다.

utim*buf* 구조는 헤더 **<utime.h>**에서 정의되며 다음 원소를 포함한다.

utim*buf* 내의 시각은 **에폭 이후 경과 시간** [3.3]으로 표시한다.

원소형	원소명	설 명
time_t	*actime*	접근 시각
time_t	*modtime*	수정시각

성공적으로 끝나면 그 파일의 *st_ctime* 필드가 갱신되도록 표시해 놓아야 한다.

6.6.6.3 복귀값

성공적으로 끝나면, 0이 되어야 한다. 아니면, -1이 되어야 하고 에러에 상응하는 값을 *errno*에 설정한다. 이 때 파일 시각은 영향을 받지 않아야 한다.

6.6.6.4 에러

다음 조건 중 어느 조건에라도 해당하는 일이 발생하면, *utime*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성 요소중 탐색 접근이 허용되지 않는 것이 있거나, times인수가 **NULL**이고 프로세스의 유효 사용자 ID가 파일의 소유자와 일치하지 않으며 쓰기 접근이 허용되지 않는다.

[ENAMETOOLONG] *path* 인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] 지정된 파일이 존재하지 않거나 *path*인수가 공백 문자열을 가리킨다.

[ENOTDIR] 경로 접두어의 구성 요소중 디렉토리가 아닌 것이 있다.

[EPERM] times 인수가 **NULL**이 아니고 호출한 프로세스의 유효 사용자 ID가 파일의 쓰기 접근 권한을 가지고 있으나 파일 소유자와 일치하지 않고 호출한 프로세스가 적절한 권한을 갖고 있지 않다.

[EROFS] 지정된 파일이 읽기 전용 파일 시스템에 있다.

6.6.6.5 참고

<sys/stat.h>[6.6.1]

6.7 구성 가능한 경로명 변수

6.7.1 구성 가능한 경로명 변수의 획득

함수 : *pathconf*(), *fpathconf*()

6.7.1.1 용례

```
#include <unistd.h>

long pathconf (path,name)
char *path;
int name;

long fpathconf (fildes,name)
int fildes, name;
```

6.7.1.2 설명

pathconf()와 *fpathconf*() 함수는 파일이나 디렉토리와 연관된 구성 가능한 한계와 선택 사항(**변수**)의 **값**을 응용이 지정하는 방법을 제공한다.

pathconf()에서 *path* 인수는 파일이나 디렉토리의 경로명을 가리킨다. *fpathconf*()에서 *fildes*인수는 열린 파일 서술자이다.

name 인수는 지정된 파일이나 디렉토리에 대한 조회를 위한 변수이다. 표 6-2에 있는 모든 변수를 반드시 지원하도록 구현하여야 하며, 기타 다른 변수의 지원은 선택 사항이다. 표 6-2의 변수는 **<limits.h>**[3.9]나 **<unistd.h>**[3.10]에 있는 것이며, 심볼 상수는 **<unistd.h>**에서 정의한 것으로 *name*의 값으로 사용한다.

다음 주는 표 6-2에 적용되는 것이다.

1. *path*나 *fildes*가 디렉토리를 가리킬 때, 복귀값은 디렉토리 자신에 대한 값이다.
2. *path*나 *fildes*가 터미날 파일을 가리킬 때의 작동 방식은 미정의이다.
3. *path*나 *fildes*가 디렉토리를 가리킬 때, 복귀값은 디렉토리 내의 파일명에 대한 값이다.
4. *path*나 *fildes*가 디렉토리를 가리키지 않을 때의 작동 방식은 미정의이다.
5. *path*나 *fildes*가 디렉토리를 가리키면, 지정된 디렉토리가 현재 작업 디렉토리일때 상대 경로명의 최대 길이가 복귀값이 된다.
6. *path*가 FIFO형 파일을 가리키거나,*fildes*가 파이프나 FIFO형 파일을 가리킬 때, 복귀값은 지정된 대상 자체에 대한 것이다.*path*나 *fildes*가 디렉토리를 가리키면 그 안에 있는 임의의 FIFO나 그 안에 만들 수 있는 임의의 FIFO에 대한 정보를 복귀값으로 한다. *path*나 *fildes*가 여타 종류의 파일을 가리킬 때의 작동 방식은 미정의이다.
7. *path*나 *fildes*가 디렉토리를 가리킬 때의 복귀값은 이 디렉토리 내에 있거나, 이 디렉토리 내에 만들 수 있는 파일에 대한 값으로, 본 표준에서 정의한 모든 종류의 파일 중 디렉토리가 아닌 모든 파일에 해당한다.

<표 6-2> **구성 가능한 경로명 변수**

변 수	*name* 값	주
{LINK_MAX}	{_PC_LINK_MAX}	1
{MAX_CANON}	{_PC_MAX_CANON}	2
{MAX_INPUT}	{_PC_MAX_INPUT}	2
{NAME_MAX}	{_PC_NAME_MAX}	3,4
{PATH_MAX}	{_PC_PATH_MAX}	4,5
{PIPE_BUF}	{_PC_PIPE_BUF}	6
{_POSIX_CHOWN_RESTRICTED}	{_PC_CHOWN_RESTRICTED}	7
{_POSIX_NO_TRUNC}	{_PC_NO_TRUNC}	3,4
{_POSIX_VDISABLE}	{_PC_VDISABLE}	2

6.7.1.3 복귀값

*name*이 잘못된 값이면 *pathconf*()나 *fpathconf*()의 복귀값은 -1이 되어야 한다.

*name*에 해당하는 변수가 경로나 파일 서술자에 대해 제한이 전혀 없을 때, *path*conf()나 *fpathconf*()는 *errno* 값을 바꾸지 않은 채 복귀값을 -1로 해야 한다.

*name*의 값을 결정하는데 *path*를 사용할 필요가 있도록 구현되었으나 *name*을 *path*가 지정하는 파일과 연관시키는 것을 지원하지 못할 때, *path*가 지정하는 파일을 조회하기 위한 적절한 권한이 없을 때, *path*가 존재하지 않을 때 등의 경우에 *pathconf*()의 복귀값은 -1이 되어야 한다.

*name*의 값을 결정하는데 *fildes*를 사용할 필요가 있도록 구현되었으나 *name*을 *fildes*가 지정하는 파일과 연관시키는 것을 지원하지 못할 때나 *fildes*가 잘못된 파일 서술자일 때는, *fpathconf*() 복귀값은 -1이 되어야 한다.

그렇지 않으면 *pathconf*()나 *fpathconf*()는 *errno* 값을 변화시키지 않으면서 지정된 파일이나 디렉토리의 현재 변수 값을 복귀값으로 한다. 응용을 구현의 **<limits.h>**[3.9]나 **<unistd.h>**[3.10]과 함께 컴파일했을 때는 복귀값이 응용에 설명한 것보다 더 제한적이어야 한다.

6.7.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *pathconf*()와 *fpathconf*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EINVAL] *name*의 값이 적절하지 못하다.

다음 중 어느 조건에라도 해당하는 일이 발생하면, *pathconf*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 경로 접두어의 구성 요소 중 탐색 접근을 허용하지 않는 것이 있다.

[EINVAL] 지정된 파일과 변수명을 연관짓는 것을 허용하지 않도록 구현되었다.

[ENAMETOOLONG] *path* 인수의 길이가 {PATH_MAX} 보다 길거나, {_POSIX_NO_TRUNC}가 발효 중인데 경로명 구성 요소 중 {NAME_MAX}보다 긴 것이 있다.

[ENOENT] 없는 디렉토리가 지정되었거나 *path* 인수가 공백 문자열을 가리키고 있다.

[ENOTDIR] 경로 접두어의 구성 요소중 디렉토리가 아닌 것이 있다.

다음 중 어는 조건에라도 해당하는 일이 발생하면, *fpathconf*()의 복귀값이 -1이 되면서 에러에 상응하는 값을 *errno*에 설정한다.

[EBADF] *fildes* 인수가 적절한 파일 서술자가 아니다.

[EINVAL] 지정된 파일과 변수명을 연관짓는 것을 허용하지 않도록 구현되었다.

제 7 장 입출력 관련 원시 함수

본 장에서 설명하는 함수는 파일 및 파이프의 입출력을 처리하는 것이다. 파일 서술자와 입출력 활동 간의 협조 및 운영에 관한 함수도 포함한다.

7.1 파이프

7.1.1 프로세스 사이의 채널 생성

함수 : *pipe*()

7.1.1.1 용례

int pipe (*fildes*)
int *fildes* [2];

7.1.1.2 설명

pipe() 함수는 파이프를 만들고 파일 서술자 2개를 만들어서 인수 *fildes*[0]와 *fildes*[1]에 넣는다. 이 인수들은 파이프의 출력(읽기) 측과 입력(쓰기) 측에 열린 파일의 서술자이다. 인수의 값은 정수로서 *pipe*()함수가 실행되는 시점에서 사용 가능한 파일 서술자 중 가장 작은 값 2개로 해야 한다. 양 파일 서술자의 O_NONBLOCK 플래그는 0이 되어야 한다. (이것을 1로 하기 위해서는 *fcntl*() 함수를 사용할 수 있다.)

fildes[0]에서 데이터를 읽고 *fildes*[1]에 데이터를 쓸 수있다. *fildes*[0]에서 읽을 때는 *fildes*[1]에 쓴 데이터를 선입선출법(FIFO)으로 접근해야 한다.

출력 측을 가리키는 파일 서술자 *fildes*[0]을 열린 프로세스는 이 파이프를 읽기용으로 열린 것이다. 마찬가지로 *fildes*[1]을 열린 프로세스는 이 파이프를 쓰기용으로 열린 것이다.

pipe() 함수가 성공적으로 끝나면 파이프의 *st_atime*, *st_ctime*, *st_mtime* 필드가 갱신되도록 표시해 놓아야 한다.

7.1.1.3 복귀값

성공적으로 끝나면 복귀값이 0이 되어야 한다. 아니면 복귀값을 -1로 하고 에러에 상응하는 값을 *errno*에 설정한다.

7.1.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *pipe*()의 복귀값을 -1로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EMFILE] 이 프로세스가 이미 {OPEN_MAX}-2 개보다 많은 파일 서술자를 사용하고 있다.

[ENFILE] 동시에 열린 파일의 개수가 시스템에서 정한 한계를 넘었다.

7.1.1.5 참고

fcntl()[7.5.2], *open*()[6.3.1], *read*()[7..4.1], *write*()[7.4.2]

7.2 파일 서술자의 조작

7.2.1 열린 파일 서술자의 복제(duplicate)

함수 : *dup*(), *dup2*()

7.2.1.1 용례

int dup (*fildes*)
int *fildes*[2];
int dup2 (*fildes*, *fildes*2)
int *fildes*, *fildes*2

7.2.1.2 설명

dup()와 *dup2*() 함수는F_DUPFD 명령을 이용하여 *fcntl*() 함수가 제공하는 서비스에 대한 또 다른 인터페이스를 제공한다.

fid = dup (fildes);

는

fid = fcntl (fildes, F_DUPFD, 0);

와 같아야 하며,

fid = dup2 (fildes, fildes2);

는 아래 세가지 사항을 제외하고는

close (fildes2);
fid = fcntl (fildes, F_DUPFD, fildes2);

와 같아야 한다.

(1) *fildes*2가 0보다 작거나 {OPEN_MAX}보다 크면, *dup2*()의 복귀값은 -1이 되면서 *errno*를 [EBADF]로 해야 한다.
(2) *fildes*가 적절한 파일 서술자이면서 *fildes*2와 같으면, *dup2*()함수는 *fildes*2를 닫지 않으면서 *fildes*2를 복귀값으로 해야한다.
(3) *fildes*가 적절한 파일 서술자가 아니면, *dup2*()는 실패하고 *fildes*2를 닫지 말아야 한다.

7.2.1.3 복귀값

성공적으로 끝나면 파일 서술자가 복귀값이 되어야 한다. 아니면 복귀값을 -1로하고 에러에 상응하는 값을 *errno*에 설정한다.

7.2.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *dup*()와 *dup2*()의 복귀값을 -1로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EBADF] 인수 *fildes*가 적절한 열린 파일 서술자가 아니거나 *fildes*2가 범위를 벗어났다.

[EMFILE]파일 서술자가 {OPEN_MAX}보다 많아지거나 사용할 수 있는 파일 서술자 중 *fildes2*보다 큰 파일 서술자가 없다.

7.2.1.5 참고

close() [7.3.1], *creat*()[6.3.2], *exec*[4.1.2], *fcntl*()[7.5.2], *open*()[6.3.1], *pipe*()[7.1.1]

7.3 할당된 파일 서술자의 회수

7.3.1 파일 닫기

함수 : *close*()

7.3.1.1 용례

int close (*fildes*)
int *fildes*;

7.3.1.2 설명

close()함수는 *fildes*가 지정하는 파일 서술자의 할당을 취소하고 이를 회수하여야 한다. 즉 이 프로세스가 나중에 *open*()등의 함수를 수행할 때 이 파일 서술자를 사용할 수 있게 하는 것이다. 이 프로세스가 갖고 있는 레코드 잠금 장치 중에서 *fildes*가 지정하는 파일에 관한 것은 모두 제거해야 한다.

close()함수가 받아들여야 하는 신호에 의해 인터럽트되면, *errno*를 [EINTR]로 설정하고 복귀값은 -1로 한다. 이때 *fildes*의 상태는 미규정이다.

파이프나 FIFO형 특수 파일을 지정하는 파일 서술자를 닫으면, 파이프나 FIFO에 남아있는 데이터는 모두 없애야 한다.

어떤 열린 파일을 지정하는 파일 서술이 모두 닫히면, 이 열린 파일 서술을 없애야 한다.

링크 수가 0인 파일의 파일 서술이 모두 닫히면, 이 파일이 차지하고 있던 공간을 회수하고 더 이상 이 파일에 접근할 수 없게 해야 한다.

7.3.1.3 복귀값

성공적으로 끝나면 복귀값이 0이 되어야 한다. 아니면 복귀값을 -1로 하고 에러에 상응하는 값을 *errno*에 설정한다.

7.3.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *close*()의 복귀값을 -1로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EBADF] 인수 *fildes*가 적절한 파일 서술자가 아니다.
[EINTR] 신호가 close 함수를 인터럽트했다.

7.3.1.5 참고

creat()[6.3.2], *dup*()[7.2.1], *exec*()[4.1.2], *fcntl*()[7.5.2], *fork*()[4.1.1], *open*()[6.3.1], *pipe*()[7.1.1], *unlink*()[6.5.1], **신호가 기타 함수에 미치는 영향**[4.3.1.4]

7.4 입력과 출력

7.4.1 파일에서 읽기

함수 : *read*()

7.4.1.1 용례

int read (*fildes*, *buf*, *nbyte*)
int *fildes*;
char **buf*;
unsigned *nbyte*;

7.4.1.2 설명

read() 함수는 *fildes*가 지정하는 파일에서 *nbyte*를 읽어 *buf*가 가리키는 버퍼에 넣는다.

*nbyte*가 0이면 *read*()의 복귀값은 0이 되고 그 외에는 아무 결과도 내지 말아야 한다.

정규 파일이나 탐색(seek)이 가능한 다른 파일에서의 *read*()는 *fildes*와 관련된 파일 오프셋이 가리키는 위치에서 시작되어야 한다. 성공한 경우 *read*()에서 복구하기 전에 오프셋을 실제로 읽은 데이터 바이트 수 만큼 증가시켜야 한다.

탐색이 불가능한 파일에서의 *read*()는 현재 위치에서 시작되어야 한다. 이런 파일과 연관되는 파일 오프셋의 값은 미정의이다.

read() 함수가 성공적으로 완료되면 실제로 읽어서 버퍼에 넣은 바이트 수가 복귀값이 되어야 한다. 이 숫자는 절대로 *nbyte*보다 클 수 없다. 파일에 남아있는 데이터가 *nbyte*보다 적거나, 신호가 *read*()를 인터럽트했거나, 파이프(또는 FIFO)나 특수 파일인데 즉시 읽을 수 있는 데이터가 *nbyte*보다 적을 때는 복귀값이 *nbyte*보다 적어 질 수 있다. 예를 들면 터미날과 연관된 파일에서 *read*()하면 입력한 데이터 한 줄에 해당하는 값이 된다.

데이터를 하나도 읽지 못하고 신호에게 인터럽트 당하면 복귀값을 -1로 하고 *errno*를 [EINTR]로 해야 한다.

얼마간 데이터를 읽은 후에 신호에게 인터럽트 당하면 복귀값이 -1이 되면서 *errno*를 [EINTR]로 하던지 그 때까지 읽은 바이트 수를 복귀값으로 하여야 한다. 파이프나 FIFO에서 *read*() 할 때는 데이터를 하나라도 읽었으면 *errno*가 [EINTR]가 되지 않아야 한다.

현재의 파일 끝(end-of-file)을 넘어가면 아무 데이터 전송도 일어나지 말아야 한다. 시작 위치가 파일 끝이거나 이를 지나간 자리이면 복귀값을 0으로 해야 한다. 장

치 특수 파일의 경우 다음 번 *read*()의 결과는 구현시 정의한다. *nbyte*의 값이 {INT_MAX}보다 클 때의 결과는 구현시 정의 한다.

비어 있는 파이프 (또는 FIFO)에서 읽으려 할 때는

(1) 쓰기 위해서 파이프를 연 프로세스가 없으면 *read*()의 복귀값을 0으로 하고 파일 끝임을 알려야 한다.

(2) 쓰기 위해 파이프를 연 프로세스가 있고 O_NONBLOCK가 1이면, *read*()의 복귀값을 -1로 하고 *errno*를 [EAGAIN]으로 하여야 한다.

(3) 쓰기 위해 파이프를 연 프로세스가 있고 O_NONBLOCK가 0이면, 누가 파이프에 데이터를 하나라도 써 넣거나 쓰기 위해 연 프로세스가 모두 파이프를 닫을 때까지 이 프로세스를 차단시켜야 한다.

차단되지 않는(nonblocking) 읽기를 지원하는 파일 (파이프나 FIFO가 아닌)에 데이터가 하나도 없는데 읽으려 하면

(1) O_NONBLOCK이 1이면 *read*()의 복귀값을 -1로 하고 *errno*를 [EAGAIN]으로 해야 한다.

(2) O_NONBLOCK이 0이면 읽을 데이터가 생길 때까지 *read*()를 차단시켜야 한다. 한다.

(3) 읽을 데이터가 있으면 O_NONBLOCK 플래그는 아무 영향을 미치지 않는다.

정규 파일에서 파일 끝에 이른 것은 아니나 아직 데이터를 써넣지 않은 부분에서 *read*()를 하면 0 값이 읽혀야 한다.

read()가 성공적으로 끝나면 파일의 *st_atime* 필드가 갱신되도록 표시해 두어야 한다.

7.4.1.3 복귀값

성공적으로 끝나면 *read*()의 복귀값은 실제 읽은 바이트 수에 해당하는 정수가 되어야 한다. 아니면 복귀값을 -1로 하고 에러에 상응하는 값을 *errno*에 설정한다. 이때 *buf*가 가리키는 버퍼의 내용은 미확정(indeterminate)이다.

7.4.1.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *read*()의 복귀값을 -1로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EAGAIN] 파일 서술자의 O_NONBLOCK 플래그가 1이다. 이 프로세스는 *read*()에서 지연될 것이다.

[EBADF] 인수 *fildes*가 읽기 위해 열린 적절한 파일 서술자가 아니다.

[EINTR] 신호가 읽기 작업을 인터럽트했는데, 데이터가 전혀 전송되지 않았거나 부분적인 전송에 대해서는 보고하지 않도록 구현되었다.

[EIO] 작업 제어를 지원하는 구현에서 백그라운드 프로세스 그룹에 속하는 프로세스가 제어 터미날에서 읽으려 하였는데, 이 프로세스가 SIGTTIN 신호를 무시하거나 차단시켰던지 이 프로세스의 프로세스

그룹이 고아가 되었다. 이 에러는 구현시 정의되는 원인에 의해서도 발생할 수 있다.

7.4.1.5 참고

creat()[6.3.2], *dup*()[7.2.1], *fcntl*()[7.5.2], *lseek*()[7.5.3], *open*()[6.3.1], *pipe*()[7.1.1], **신호가 기타 함수에 미치는 영향[4.3.1.4]**, 터미날 **인터페이스** 특성[8.1.1]

7.4.2 파일에 쓰기

함수 : *write*()

7.4.2.1 용례

int write (*fildes*, *buf*, *nbyte*)
int *fildes*;
char **buf*;
unsigned *nbyte*;

7.4.2.2 설명

write()함수는 *buf*가 가리키는 버퍼에서 *nbyte* 바이트를 읽어서 열린 파일 서술자 *fildes*가 지정하는 파일에 써넣어야 한다.

*nbyte*가 0일 때, 정규 파일이면 0을 복귀값으로 하고 그 외에 다른 일은 전혀 하지 않아야 한다. 정규 파일이 아닐때의 결과는 구현시 정의 한다.

정규 파일이거나 탐색이 가능한 다른 종류의 파일의 경우 *fildes*가 지정하는 파일의 오프셋이 표시하는 위치에서부터 데이터를 써넣기 시작해야 한다. 성공한 경우 *write*()에서 복귀하기 전에 파일 오프셋을 실제로 써넣은 데이터 바이트 수 만큼 증가 시켜야 한다. 정규 파일의 경우 이렇게 증가된 파일 오프셋이 파일의 길이보다 커지면 파일의 길이를 이 파일 오프셋의 값으로 바꾸어 주어야 한다.

탐색이 불가능한 파일에서의 *write*() 현재 위치에서 시작되어야 한다. 이런 파일과 연관되는 파일 오프셋의 값은 미정의이다.

파일 상태 플래그의 O_APPEND 플래그가 1이면 매번 쓰기 시작하기 전에 파일 오프셋을 파일 맨 끝으로 설정해야 한다.

write()가 요구하는 바이트 수 만큼 쓸 자리가 없을때 (예를 들면 기억 매체의 맨 끝에 다다른 경우)는 공간이 남아 있는 만큼만 써야 한다. 예를 들어 20 바이트만 파일에 더 기억하면 꽉 찬다고 하자. 이 때 512 바이트를 쓰라고 하면 *write*()의 복귀값은 20이 된다. 이 다음에 0이 아닌 바이트 수를 지정하여 쓰라고 하면 이 함수는 실패할 것이다. (단 다음에 설명하는 경우는 예외로 한다.)

write() 함수가 성공적으로 완료되면 *fildes*가 가리키는 파일에 실제로 써 넣은 바이트 수가 복귀값이 되어야 한다. 이 숫자는 절대로 *nbyte*보다 클 수 없다.

write()가 데이터를 하나도 쓰지 못하고 신호에게 인터럽트 당하면 복귀값을 -1로 하고 *errno*를 [EINTR]로 해야 한다.

얼마간 데이터를 써넣은 후에 신호에 게 인터럽트 당하면 복귀값이 -1이 되면서 *errno*를 [EINTR]로 하던지 그 때까지 써넣은 바이트 수를 복귀값으로 하여야 한다. 파이프나 FIFO에 대한 *write*()는 데이터를 하나라도 써 넣었고 *nbyte*가 {PIPE_BUF}보다 작거나 같으면 절대로 *errno*를 [EINTR]로 해서는 안된다.

*nbyte*의 값이 {INT_MAX} 보다 클 때의 결과는 구현시 정의한다.

파이프 (또는 FIFO)에 대한 쓰기 요청은 다음과 같은 점을 제외하고는 정규 파일과 같은 방법으로 취급해야 한다.

(1) 파이프에는 파일 오프셋이 없으므로 쓰기 요청은 파이프의 맨 뒤에 덧 붙이 는 형식이 되어야 한다.

(2) {PIPE_BUF} 바이트보다 적거나 같은 데이터의 쓰기 요청의 경우 같은 파이프에 쓰기를 하는 다른 프로세스의 데이터와 섞여서는 안된다. {PIPE_BUF} 바이트보다 많은 데이터는 파일 상태 플래그의 O_NONBLOCK 플래그 값에 상관없이 다른 프로세스의 데이터와 교대로 쓰일 수 있다. 이 때 서로 다른 프로세스의 데이터 간의 경계는 임의로 정할 수 있다.

(3) O_NONBLOCK 플래그가 0이면 쓰기 요청을 했을 때 프로세스가 차단될 수 도 있다. 그러나 정상적으로 종료되었을 때 *nbyte*가 복귀값이 되어야 하는 것은 필수적이다.

(4) O_NONBLOCK 플래그가 1이면 아래 설명과 같이 *write*() 요청을 다른 방법으로 취급해야 한다. *write*() 함수 때문에 프로세스가 차단되지 않게 해야 한다. {PIPE_BUF} 바이트 보다 적은 데이터의 쓰기 요청은, 성공적으로 끝나서 *nbyte*를 복귀값으로 하던지 아니면 에러가 나서 -1을 복귀값으로 하고 *errno*를 [EAGAIN]으로 하던지 해야 한다. {PIPE_BUF} 바이트보다 많은 데이터의 쓰기 요청은, 쓸 수 있는만큼 데이터를 쓰고 실제 쓴 데이터 갯수를 복귀값으로 하던지 전혀 데이터를 쓰지 않는 상태에서 복귀값을 -1, *errno*를 [EAGAIN]으로 하던지 해야 한다. 또 {PIPE_BUF}보다 많은 데이터의 *write*()요청이 있을 때 파이프가 비어있으면 적어도 {PIPE_BUF} 바이트만큼은 전송이 이루어져야 한다.

차단되지 않는 쓰기를 지원하는 파일 서술 (파이프나 FIFO가 아닌)에 대한 쓰기를 하려고 하는데 이 파일이 즉각 데이터를 받을 수 없는 상태일 때

(1) O_NONBLOCK 플래그가 0이면, 데이터를 받을 수 있는 상태가 될 때까지 *write*()는 차단되어야 한다.

(2) O_NONBLOCK 플래그가 1이면, *write*()가 프로세스를 차단시켜서는 안된다. 데이터는 일부라도 프로세스를 차단시키지 않고 쓸 수 있으면 쓸수 있는 만큼은 쓰고 실제 써넣은 데이터 갯수를 복귀값으로 한다. 아니면 -1을 복귀값으로 하고 *errno*를 [EAGAIN]으로 한다.

write()가 성공적으로 끝나면 파일의 *st_ctime*과 *st_mtime*필드가 갱신되도록 표시해 두어야 한다.

7.4.2.3 복귀값

성공적으로 끝나면 *write*()의 복귀값은 실제 써넣은 바이트 수에 해당하는 정수가

되어야 한다. 아니면 복귀값을 -1로 하고 에러에 상응하는 값을 *errno*에 설정한다.

7.4.2.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, wrote()의 복귀값을 -1로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EAGAIN] 파일 서술자의 O_NONBLOCK 플래그가 1이다. 이 프로세스는 *write*()에서 지연될 것이다.

[EBADF] 인수 *fildes*가 쓰기 위해 열린 적절한 파일 서술자가 아니다.

[EFBIG] 구현시 정의한 최대 파일 크기를 넘는 파일에 쓰려고 했다.

[EINTR] 신호가 쓰기 작업을 인터럽트 했는데, 데이터가 전혀 전송되지 않았거나 부분적인 전송에 대해서는 보고하지 않도록 구현되었다.

[EIO] 작업 제어를 지원하는 구현에서 백그라운드 프로세스 그룹에 속하는 프로세스가 제어 터미날에 쓰려고 하였으며 TOSTOP이 1인데, 이 프로세스가 SIGTTOU 신호를 무시하지도 차단시키지도 않고 이 프로세스의 프로세스 그룹이 고아가 되었다. 이 에러는 구현시 정의되는 원인에 의해서도 발생할 수 있다.

[ENOSPC] 파일이 저장된 장치에 빈 기억 공간이 남아있지 않다.

[EPIPE] 아무 프로세스도 읽기용으로 열리지 않은 파이프(또는 FIFO)에 쓰려고 했다. 이 프로세스에는 SIGPIPE 신호가 보내져야 한다.

7.4.2.5 참고

creat()[6.3.2], *dup*()[7.2.1], *fcntl*()[7.5.2], *lseek*()[7.5.3], open()[6.3.1], *pipe*()[7.1.1], **신호가 기타 함수에 미치는 영향** [4.3.1.4]

7.5 파일 제어 조작

7.5.1 파일 제어 조작의 데이터 정의

헤더 **<fcntl.h>**는 *fcntl*()[7.5.2]와 *open*()[6.3.1]함수가 사용하는 요청과 인수를 다음과 같이 정의한다. 표 7-1에서 7-7까지에서 사용되는 값은 모두 서로 달라야 한다. 또 oflag값, 파일 상태 플래그, 파일 접근 모드의 각 엔트리 값도 서로 모두 달라야 한다.

<표 7-1> **fcntl()의 cmd값**

상 수	설 명
F_DUPFD	파일 서술자의 복제
F_GETFD	파일 서술자 플래그 읽어 오기
F_GETLK	레코드 잠금 정보 읽어 오기
F_SETFD	파일 서술자 플래그 설정
F_GETFL	파일 상태 플래그 읽어 오기
F_SETFL	파일 상태 플래그 설정
F_SETLK	레코드 잠금 정보 설정
F_SETLKW	레코드 잠금 정보 설정, 단 차단되면 대기

<표 7-2> **fcntl()이 사용하는 파일 서술자 플래그**

상 수	설 명
FD_CLOEXEC	*exec* 계열 함수 수행 후 파일 서술자 닫기

<표 7-3> **fcntl()이 레코드 잠금에 사용하는 l_type의 값**

상 수	설 명
F_RDLCK	공유 또는 읽기 잠금 장치
F_UNLCK	잠금 장치 해제
F_WRLCK	독점 또는 쓰기 잠금 장치

<표 7-4> **open()의 oflag 값**

상 수	설 명
O_CREAT	없는 파일일 때 새 파일 생성
O_EXCL	독점적 사용 플래그
O_NODTTY	제어 터미날을 할당 않기
O_TRUNC	절단(truncate) 플래그

<표 7-5> open()과 fcntl()이 사용하는 파일 상태 플래그

상 수	설 명
O_APPEND	추가(append) 모드 설정
O_NONBLOCK	지연 없음

<표 7-6> open()과 fcntl()이 사용하는 파일 접근 모드

상 수	설 명
O_RDONLY	읽기 전용으로 열린
O_RDWR	읽기와 쓰기 겸용으로 열린
O_WRONLY	쓰기 전용으로 열린

<표 7-7> 파일 접근 모드가 사용하는 마스크

상 수	설 명
O_ACCMODE	파일 접근 모드용 마스크

7.5.2 파일 제어

함수 : *fcntl*()

7.5.2.1 용례

#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int fcntl (*fildes*, *cmd*, . . .)
int *fildes*, *cmd*;

7.5.2.2 설명

fcntl()함수는 열린 파일을 제어하는 방법을 제공한다. 인수 *fildes*는 파일 서술자이다.

*cmd*에서 사용할 수 있는 값은 헤더 **<fcntl.h>**[7.5.1]에서 정의하며 다음 내용을 반드시 포함해야 한다.

F_DUPFD 아직 사용되지 않은 (즉 열려있지 않은) 파일 서술자 중 세번째 인수인 *arg* (*arg*는 정수형) 보다 작지 않은면서 최소값인 새로운 파일 서술자를 복귀값으로 한다. 새로운 파일 서술자는 원래의 파일 서술자

와 같은 열린 파일 서술을 가리키며 모든 잠금 장치를 공유한다. 새 파일 서술자는 *exec* 계열의 함수 호출이 끝나도 계속 열려 있도록 FD_CLOEXEC 플래그를 0으로 한다.

F_GETFD 파일 서술자 *fildes*와 연관된 파일 서술자 플래그 (표 7-2에서 정의)를 읽어 온다. 파일 서술자 플래그는 한 파일 서술자와만 연관되어 있어서 같은 파일에 대한 파일 서술자에는 영향을 미치지 않는다.

F_SETFD *fildes*와 연관된 파일 서술자 플래그 (표 7-2에서 정의)를 세번째 인수 *arg*에 따라 바꾼다. FD_CLOEXEC 플래그가 0이면 *exec* 계열의 함수 호출이 끝나도 계속 열려있어야 한다. 아니면 *exec* 계열 함수의 실행이 성공적으로 끝나면 파일을 닫아야 한다.

F_GETFL filde와 연관된 열린 파일 서술의 파일접근 모드와 파일 상태 플래그 (표 7-5에서 정의)를 읽어 온다. 파일 접근 모드는 표 7-6에서 정의하는데 O_ACCMODE 마스크를 이용해서 복귀값에서 추출해 낼 수 있다. O_ACCMODE 마스크는 **<fcntl.h>**[7.5.1]에서 정의한다. 파일 상태 플래그와 파일접근 모드는 이 열린 파일 서술과만 연관되어 있어서 다른 열린 파일 서술과 동일한 파일을 참조하는 다른 열린 파일 서술자에는 영향을 미치지 않는다.

F_SETFL *fildes*와 연관된 열린 파일 서술의 파일 상태 플래그 (표 7-5에서 정의)를 세번째 인수 *arg*에 따라 바꾼다. 이 함수 호출에 의해 파일 접근 모드가 변해서는 안된다. *arg*의 다른 비트들의 1로 되어 있을 때의 결과는 구현시 정의한다.

다음 명령들은 advisory 레코드 잠금을 위해 쓸 수 있다. 정규 파일에 대해서는 advisory 레코드 잠금의 지원이 필수적이며, 다른 파일에 대한 지원은 선택 사항이다.

F_GETLK 세번째 인수 *arg*가 가리키는 잠금 장치 서술을 차단시킨 최초의 잠금 장치를 읽어온다. *arg*는 struct flock (아래에서 설명) 형에 대한 포인터이다. 읽어온 정보는 flock 구조에서 *fcntl*()에 전달된 정보를 덧쓴다. 이 잠금 장치를 만들지 못하게 하는 잠금 장치를 찾지 못했을 때는 잠금 장치 타임이 F_UNLCK로 정해져야 하는 것 외에는 flock 구조에 변화가 없어야 한다.

F_SETLK 세번째 인수 *arg*가 가리키는 잠금 장치 서술에 따라 파일 세그먼트 잠금 장치를 1로 또는 0으로 한다. *arg*는 struct flolk 형에 대한 포인터이다. F_SETLK는 공유 (또는 읽기) 잠금 장치 (F_RDLCK) 또는 독점(또는 쓰기)잠금장치(F_WRLCK)를 설정하고 해지(F_UNLCK)하는데 사용된다. F_RDLCK, F_WRLCK, F_UNLCK는 **<fcntl.h>** [7.5.1] 헤더에서 정의한다. 공유 잠금 장치나 독점 잠금 장치를 설정할 수 없을 때는 *fcntl*()이 즉시 복귀해야 한다.

F_SETLKW 공유 잠금 장치나 독점 잠금 장치가 다른 잠금 장치에 의해 차단

되었을 때, 요청이 만족될 때까지 프로세스가 기다려야 하는 점을 제외 하고는 F_SETLK와 같다. *fcntl*()이 어떤 영역에 대해 대기 상태에 있을때 기다리던 신호가 접수되면 *fcntl*()은 인터럽트 되어야 한다. 프로세스의 신호 처리 프로그램에서 복귀하면 *fcntl*()은 복귀값을 -1로하고 *errno*를 [EINTR]로 해야 하며, 잠금 함수는 수행되지 않아야 한다.

<fcntl.h> [7.5.1] 헤더에서 정의하는 flock 구조는 advisory 잠금 장치에 대해 기술한다. 여기에는 표 7-8의 원소들이 포함된다.

<표 7-8> **flock 구조**

원소형	원소명	설 명
short	l_*type*	F_RDLCK, F_WRLCK 또는 F_UNLCK
short	l_*whence*	시작 오프셋에 대한 플래그
off_t	l_*start*	바이트 단위로 나타낸 상대 오프셋
off_t	l_*len*	크기 : 0이면 EOF까지
pid_t	l-*pid*	잠금 장치를 중단시킨 프로세스의 프로세스 ID로서, F_GETLK이 읽어 간다.

파일의 한 세그먼트에 공유 잠금 장치가 설정되어 있을 때, 다른 프로세스도 그 세그먼트 또는 세그먼트의 일부분에 대해 공유 잠금 장치를 설정할 수 있어야 한다. 공유 잠금 장치는 이 잠금 장치가 보호하는 영역의 어느 부분에 대해서도 독점 잠금 장치를 설정할 수 없도록 한다. 파일 서술자가 읽기 접근용으로 열려있지 않을 때 공유 잠금 요청은 실패한다.

독점 잠금 장치는 이 잠금 장치가 보호하는 영역의 어느 부분에 대해서도 다른 프로세스가 공유 잠금 장치나 독점 잠금 장치를 설정하지 못하게 한다. 파일 서술자가 쓰기 접근용으로 열려있지 않을 때 독점 잠금 요청은 실패한다.

l_*whence* 값은 SEEK_SET, SEEK_CUR, SEEK_END 중 하나가 될 수 있다. 이 값들은 상대 오프셋 (l_start 바이트) 을 파일의 처음부터 셀 것인가, 현재 위치에서부터 셀 것인가, 파일 끝에서부터 셀 것인가를 각각 표시한다. l_len 값은 잠글 바이트 수이다. l_len이 음수일 때의 결과는 구현시 정의된다. l_*pid* 필드는 F_GETLK만 사용하는 것으로 차단 잠금 장치를 중단시킨 프로세스의 프로세스 ID를 받아오는 역할을 한다. F_GETLK가 성공적으로 처리된 후 l_*whence*의 값은 SEEK_SET으로 설정된다.

현재의 파일 끝을 넘은 부분에 잠금 장치를 설정할 수도 있으나, 파일 처음보다 더 앞쪽으로 설정할 수는 없다. l_len이 0일 때는 이 파일에 대해 가능한 최대 파일 오프셋으로 확장되도록 잠금 장치가 설정되어야 한다. flock struct의 l_*whence*와 l_start가 파일 처음을 가리키고 l_len이 0이면 전체 파일이 잠김 상태가 되어야 한다.

호출 프로세스는 파일의 각 바이트에 대해 한 타입의 잠금 장치만을 갖고 있어야 한다. F_SETLK이나 F_SETLKW에서 성공적으로 복귀하면 지정된 영역 내의 각 바이트가 갖고 있던 잠금 타입은 새로운 잠금 타입으로 대치된다. 어떤 프로세스에 대해 파일과 연관된 모든 잠금 장치는 이 프로세스가 이 파일 서술자를 닫거나 이 파일의

서술자를 갖고 있는 프로세스가 종료될 때 제거되어야 한다. *fork*() 함수를 이용해서 생성한 자식 프로세스에는 잠금 장치가 상속되지 않는다.

잠금 영역을 제어하는 프로세스가 다른 프로세스의 잠금 영역에 대한 잠금을 시도하다 휴면(sleep) 상태에 들어갈 때 교착상태가 발생할 가능성이 있다. 잠금 영역의 잠금이 풀어질 때까지 교착상태에 빠지게 되는 휴면 상태를 시스템이 찾게되면 *fcntl*()함수는 [EDEADLK] 에러를 발생시키면서 실패하게 해야 한다.

7.5.2.3 복귀값

성공적으로 끝났을 때의 복귀값은 *cmd*에 따라 달라진다. 표 7-9에 여러가지 복귀값을 보였다.

<표7-9> **fcntl()의 복귀값**

요 청	복 귀 값
F_DUPFD	새로운 파일 서술자
F_GETFD	표 7-2에서 정의한 플래그 값 단 음수가 되어서는 안된다.
F_SETFD	-1이 아닌 값
F_GETFL	파일 상태 플래그와 접근 모드 단 음수가 되어서는 안된다.
F_SETFL	-1이 아닌 값
F_GETLK	-1이 아닌 값
F_SETLK	-1이 아닌 값
F_SETLKW	-1이 아닌 값

아니면 복귀값을 -1로 하고 에러에 상응하는 값을 *errno*에 설정한다.

7.5.2.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *fcntl*()의 복귀값을 -1로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EACCES] 나 [EAGAIN]
: 인수 *cmd*가 F_SETLK이고, 잠금장치형(l_*type*)이 공유 잠금 장치 (F_RDLCK) 나 독점 잠금 장치 (F_WRCLK) 이며, 잠그려고 하는 파일 세그먼트에 이미 다른 프로세스가 독점 잠금을 걸었다. 또는 독점 잠금을 하려하는데 이미 다른 프로세스가 공유 잠금이나 독점 잠금을 걸었다.

[EBADF]
: 인수 *fildes*가 쓰기 위해 열린 적절한 파일 서술자가 아니다. 인수 *cmd*가 F_SETLK이나 F_SETLKW이고 l_*type*이 공유 잠금 (F_RDLCK)인데, *fildes*가 읽기 위해 열린 적절한 파일 서술자가 아니다. 인수 *cmd*가 F_SETLK이나 F_SETLKW이고 l_*type*이 독점 잠금 (F_WRLCK)인데, *fildes*가 쓰기 위해 열린 적절한 파일 서술자가 아니다.

[EINTR]
: 인수 *cmd*가 F_SETLKW이고 신호가 이 함수를 인터럽트하였다.

[EINVAL] 인수 *cmd*가 F_DUPFD이고, 세번째 인수가 음수이거나 {OPEN_MAX} 보다 크거나 같다.
인수 *cmd*가 F_GETLK, F_SETLK, F_SETLKW 중 하나이고 *arg*가 가리키는 것이 적절하지 못하거나, *fildes*가 잠금을 지원하지 않는 파일을 가리키고 있다.

[EMFILE] 인수 *cmd*가 F_DUPFD이고 {OPEN_MAX} 개의 파일 서술자를 이 프로세스가 사용하고 있거나, *arg*보다 크거나 같은 파일 서술자 중 지금 사용할 수 있는 것이 없다.

[ENOLCK] 인수 *cmd*가 F_SETLK 이나 F_SETLKW이고 이 잠금 장치 설정 또는 해지 작업을 하면 잠금 영역의 개수가 시스템이 정해 놓은 한계를 넘게 된다.

[EDEADLK] 인수 *cmd*가 F_SETLKW이고 교착 상태 발생 조건이 탐지되었다.

7.5.2.5 참고

close()[7.3.1], *exec*[4.1.2], *open*()[6.3.1],**<fcntl.h>**[7.5.1], **신호가 기타 함수에 미치는 영향**[4.3.1.4]

7.5.3 읽기/쓰기를 위한 파일 오프셋의 변경

함수 : *lseek*()

7.5.3.1 용례

```
#include <sys/types.h>
#include <unistd.h>

off_t lssek (fildes, offset, whence)
int fildes, whence;
off_t offset
```

7.5.3.2 설명

*fildes*인수는 열린 파일 서술자이다. *lseek*() 함수는 *fildes*와 연관된 열린 파일서술의 파일 오프셋을 다음과 같이 설정해야 한다.

(1) *whence*가 SEEK_SET이면, 오프셋은 *offset* 바이트로 한다.

(2) *whence*가 SEEK_CUR이면, 오프셋은 현재의 오프셋에다 *offset*을 더한 값으로 한다.

(3) *whence*가 SEEK_END이면, 오프셋은 파일의 크기에다 *offset*을 더한 값으로 한다.

심볼 상수 SEEK_SET, SEEK_CUR, SEEK_END는 헤더 **<unistd.h>**[3.10]에서 정의한다.

장치 중에는 수색을 할 수 없는 것들이 있다. 이런 장치와 연관된 파일 오프셋 값은 미정의이다. 이런 장치에 대한 *lseek*() 함수의 작동 방식은 구현시 정의한다.

lseek() 함수는 파일에 현재 존재하는 데이터의 끝부분을 넘어서 파일 오프셋을 설정하는 것을 허용해야 한다. 이 위치에 데이터가 기록된 후, 중간에 비어 있는 부분에 대한 읽기는 실제 데이터가 기록되기 전까지 0을 읽게해야 한다.

lseek() 함수 혼자서는 파일의 크기를 늘릴 수 없다.

7.5.3.3 복귀값

성공적으로 끝나면 파일 처음부터 바이트 단위로 된 오프셋 위치가 복귀값이 된다. 아니면 (off_t)-1을 복귀값으로 하고 에러에 상응하는 값을 *errno*에 설정한다. 이때 파일 오프셋은 변하지 말아야 한다.

7.5.3.4 에러

다음 중 어느 조건에라도 해당하는 일이 발생하면, *lseek*()의 복귀값을 -1로 하면서 에러에 상응하는 값을 *errno*에 설정한다.

[EBADF] 인수 *fildes*가 적절한 파일 서술자가 아니다.

[EINVAL] *whence* 인수가 적절한 값이 아니거나 수행 결과의 파일 오프셋이 적절하지 않은 값이다.

[ESPIPE] *fildes* 인수가 파이프나 FIFO와 연관되어 있다.

7.5.3.5 참고

creat()[6.3.2], *dup*()[7.2.1], *fcntl*()[7.5.2], *open*()[6.3.1],*read*()[7.4.1], *sigaction*()[4.3.4], *write*()[7.4.2], **<unistd.h>**[3.10]

제 8장 디바이스와 클래스 관련 함수

8.1 터미날 인터페이스 일반 사항

이 절에서는 터미날 인터페이스 일반 사항을 다루는데, 이 일반 사항은 비동기적 통신 포트를 제어하기 위해서 제공되어야 하는 필수사항이다. 이 인터페이스가 망연결 또는 동기적 포트나 두가지 모두를 지원할지는 구현시 정의된다. 이 장에 있는 어떤 함수들은 프로세스의 제어 터미날에만 적용된다. 이 경우에는 제어 터미날에만 적용된다고 언급된다.

8.1.1 인터페이스 특성

8.1.1.1 터미날 디바이스 파일의 열기

터미날 파일이 열릴 때에는 정상적인 상태에서는 프로세스로 하여금 연결(connection)이 만들어질 때까지 기다리도록 한다. 실제로는 응용 프로그램이 이 터미날 파일을 여는 경우는 거의 없으며 이들은 특수 프로그램에 의해 열리며 응용의 표준 입력, 출력 및 에러 파일이 된다.

open()[6.3.1]에서 설명한 바와 같이 터미날 디바이스 파일을 O_NONBLOCK 플래그를 리셋 상태로 여는 것은 터미날 디바이스가 준비 상태이고 가용할 때까지 프로세스를 차단하도록 한다. CLOCAL 플래그 역시 *open*()에 영향을 미칠 수 있다. **제어 모드**[8.1.2.4]을 참고하시오.

8.1.1.2 프로세스 그룹

터미날은 관련있는 포어그라운드 프로세스 그룹을 가질 수 있다. 이 포어그라운드 프로세스 그룹은 신호을 생성하는 입력 문자들을 취급하는데 있어 특별한 역할을 한다. 이러한 것은 **특수 문자**[8.1.1.9]에서 논의된다. 만약 구현이 작업제어를 지원한다면, (만약 {_POSIX_JOB_CONTROL}이 정의되어 있다면 **심볼 상수**[3.10]을 참고) 작업제어를 지원하는 명령어 해석기 프로세스[1]들이 서로 연관있는 프로세스들을 단일 프로세스 그룹에 놓고 이 프로세스 그룹을 그 터미날과 연결맺음으로써 터미날은 다른 작업이나 프로세스 그룹에 할당할 수 있다. 이 절에 표시되는 허가 요구사항들이 지켜진다면 터미날의 포어그라운드 프로세스 그룹은 하나의 프로세스에 의해 설정되거나 조사될 수 있다. *tcgetpgrp*()[8.2.3]과 *tcsetpgrp*()[8.2.4]을 참고하시오. 터미날 인터페이스는 포어그라운드 프로세스 그룹에 속하지 않는 프로세스에 의한 터미날에의 접근을 제한함으로써 이 할당을 도와준다. **터미날 접근 제어**[8.1.1.4]을 참고하시오.

8.1.1.3 제어 터미날

터미날은 제어 터미날로써 하나의 프로세스에 속할 수 있다. 제어 터미날을 가지

---

1) P1003.2 실무 그룹은 명령어 해석기의 정의와 설명에 대해 연구하고 있다.

고 있는 세션의 각 프로세스들은 같은 제어 터미날을 갖는다. 터미날은 기껏해야 한 세션에 대해 제어 터미날이 될 수 있다. 하나의 세션에 대한 제어 터미날은 구현시 정의되는 식으로 세션 리더에 의해 할당된다. 만약 세션리더가 제어 터미날을 가지고 있지 않고 세션하고 관련있지 않은 터미날 디바이스 파일을 O_NOCTTY 선택 사항을 사용치 않고 연다면 (*open*()[6.3.1] 참고) 그 터미날이 그 세션리더의 제어 터미날이 될지에 대해서는 구현시 정의된다. 만약 세션리더가 아닌 프로세스가 터미날 파일을 열거나 O_NOCTTY 선택 사항이 *open*()에 사용된다면 그 터미날은 호출 프로세스에 제어 터미날이 되어서는 안된다. 제어 터미날이 세션과 연결을 맺을 때에는 포어그라운드 프로세스 그룹은 세션 리더의 프로세스 그룹으로 설정되어야 한다.

제어 터미날은 *fork*() 함수 동안 자식 프로세스가 물려 받는다. 프로세스는 *setsid*() 함수를 이용, 새로운 세션을 생성할 때나 제어 터미날과 관련있는 모든 파일 서술자들이 닫혀질 때 그 프로세스는 제어 터미날을 포기하게 된다.

제어 프로세스가 끝날 때에는 제어 터미날은 현재의 세션과의 연결로부터 풀리어져, 새로운 세션리더가 습득할 수 있도록 된다. 먼저번 세션에 있던 다른 프로세스들에 의한 터미날에의 접근은 거절되며 터미날에의 접근시도는 마치 모뎀의 단절이 일어난 것처럼 취급된다.

8.1.1.4 터미날 접근 제어

만약 어떤 프로세스가 그것의 제어 터미날의 포어그라운드 프로세스 그룹에 속한다면 **입력 처리와 데이타 읽기**[8.1.1.6]에서 서술된 것과 같이 읽기 작업은 허용되어야 한다. 작업 제어를 지원하는 구현들에 대해서는 백그라운드 프로세스 그룹에 속한 프로세스가 제어 터미날로부터 읽기를 시도한다면 그것은 다음의 특수한 경우 중 하나가 적용되지 않는 한 그 프로세스 그룹이 SIGTTIN 신호를 받게 만든다. 즉, 만약 읽기 프로세스가 SIGTTIN 신호를 무시하거나 차단한다면, 또는 만약 읽기 프로세스의 프로세스 그룹이 고아이라면 *read*()는 *errno*를 [E10]로 설정한 채 -1을 복귀값으로 하며 아무 신호도 보내지 않는다. SIGTTIN 신호의 미리 정해진(default) 행위는 그 신호가 보내져야 될 프로세스를 중지시키는 것이다. **신호명**[4.3.1.1]을 참고하시오.

만약 프로세스가 제어터미날의 포어그라운드 프로세스 그룹에 속한다면, **데이타 쓰기 및 출력 처리**[8.1.1.8]에서 설명된 바와 같이 쓰기 작업이 허용된다. 다음의 특수한 경우 중 하나가 적용되지 않는 한 백그라운드 프로세스 그룹에 속한 프로세스가 제어 터미날에 쓰기를 시도하면 그 프로세스 그룹은 SIGTTOU 신호를 받게 된다. 즉, 만약 TOSTOP가 설정되어 있지 않거나 또는 만약 TOSTOP가 설정되어 있으면서 프로세스는 SIGTTOU 신호를 무시하거나 차단한다면 그 프로세스는 터미날에 쓰기가 허용되며 SIGTTOU 신호는 보내지지 않든다. 만약 TOSTOP가 설정되어 있으면 쓰기를 하는 프로세스의 프로세스 그룹이 고아이고 쓰기를 하는 프로세스가 SIGTTOU를 무시하지 않거나 차단하지 않으면 *write*()는 *errno*값을 [E10]로 만든 채 -1을 복귀값을 돌려주며 아무 신호도 보내지 않는다.

터미날 매개변수들을 설정하는 어떤 호출들은 쓰기(write)와 같은 방법으로 취급받는데 TOSTOP가 무시된다는 것만 다르다. 따라서 그 효과는 TOSTOP가 설정되었을때의 터미날 쓰기의 효과와 같다. **제어 함수**[8.2]를 참고하시오.

8.1.1.5 입력 처리와 데이타 읽기

터미날 디바이스 파일과 관련있는 터미날 디바이스는 전이중(full-duplex) 모드로 동작하며 따라서 출력이 일어나고 있는 동안에는 데이타는 도착할 수 있다. 각 터미날 디바이스 파일은 그것과 관련있는 입력 큐를 가지고 있는데 입력 데이타는 프로세스에 의해 읽히기 전에 시스템에 의해 입력 큐에 저장된다. 시스템은 입력 큐에 저장될 수 있는 최대 바이트값으로 {MAX_INPUT}을 부여할 수 있다. 그 최대값이 초과되었을때의 시스템의 행위(반응)은 구현시 정의된다. 일반적인 두 가지 종류의 입력 처리가 이용될 수 있으며 터미날 디바이스 파일이 표준 모드로 동작할 것이냐 또는 비표준 모드 일 것이냐에 따라서 입력 처리 종류는 결정된다. 이 모드들은 **표준 모드 입력 처리**[8.1.1.6]와 **비표준 모드 입력 처리**[8.1.1.7]에서 설명된다. 그 밖에도 입력 문자들은 c_iflag(**입력모드**[8.1.2.2]참고)와 c_iflag(**지역모드**[8.1.2.5]참고) 필드에 따라 처리된다. 그러한 처리는 반향(echoing)을 포함할 수 있는데 이 반향은 일반적으로 입력 문자를 터미날로부터 받았을 때 그 즉시 터미날로 다시 전송하는 것을 의미한다. 이것은 전이중(full-duplex) 모드에서 작동할 수 있는 터미날에 유용하다.

데이타를 터미날 디바이스 파일에서 읽어 프로세스에 제공되는 방법은 터미날 디바이스 파일이 표준 모드이냐 비표준 모드냐에 달려있다.

뿐만 아니라 그 방법은 O_NONBLOCK 플래그가 *open*()이나 *fcntl*()에 의해 설정되어 있는지의 여부에도 의존한다. 만약 O_NONBLOCK 플래그가 리셋이면 읽기 요청은 데이타가 가용하던지 신호가 도착할 때까지 차단되어져야 한다. 만약 O_NONBLOCK 플래그가 설정되면 읽기 요청은 차단없이 다음 세가지 중 한 가지로 완료된다:

(1) 만약 전체 요청을 만족할 만큼 데이타가 충분히 있으면 *read*()는 성공적으로 수행되며 실제 읽은 바이트 수를 복귀값으로 한다.

(2) 만약 전체 요청을 만족시킬 만큼 가용한 데이타가 충분하지 않다면 *read*()는 가능한 한 많은 데이타를 읽고 실제 읽은 바이트 수를 복귀값으로 함으로써 성공적으로 끝낸다.

(3) 만약 가용한 데이타가 전혀 없다면 *read*()는 *errno*를 [EAGAIN]으로 세트한채 -1을 복귀값으로 한다.

데이타가 가용한 시간은 입력 처리 모드가 표준인지 비표준인지에 따라서 다르다. 다음 절인 **표준 모드 입력 처리**[8.1.1.6]와 **비표준 모드 입력 처리** [8.1.1.7]절이 이 입력 처리 모드의 각각에 대하여 다룬다.

8.1.1.6 표준 모드 입력 처리

표준 모드 입력 처리에서는 터미날 입력은 줄(line) 단위로 처리된다. 한 줄은 '\n'(새줄) 문자, EOF(파일끝) 문자, EOL(줄끝) 문자에 의해 끝나게 된다. EOF와 EOL에 대한 자세한 정보를 위해서는 **특수문자**[8.1.1.9]를 참고하라. 이것은 한 줄 전체가 타이핑되었거나 신호를 받을 때까지는 읽기 요청이 완전히 끝나지 않는다는 것을 의미한다. 또한 읽기 호출에서 아무리 많은 바이트가 요청된다고 하더라도 기껏해야 한 줄만 읽혀진다. 그러나 전체 줄을 한 번에 읽을 필요는 없으며 어느 수의 바이트(한 바이트를 포함해서)도 정보를 읽지 않고 읽기에서 요청될 수 있다.

만약 {MAX_CANON}이 이 터미날 디바이스를 위해 정의된다면 이는 한 줄에서

읽을 수 있는 바이트 수의 최대값이 된다. 이러한 최대값을 초과했을 때 시스템의 행동은 구현시 정의된다. 만약 {MAX_CANON}이 정의 되어있지 않으면 그러한 한계값은 없다. **경로명 변수값**[3.9.5]을 참고하라.

지우기와 없애기 처리는 ERASE와 KILL 문자등(**특수 문자**[8.1.1.9]참고) 두가지 특수문자중 하나를 받았을 때 일어난다. 이러한 처리는 NL(새줄), EOF, EOL 문자에 의해 끝나지 않은 입력 큐에 있는 데이타에 영향을 미친다. 이와 같이 끝맺음이 안된 데이타는 현재 줄을 이루게 된다. ERASE 문자는 현재 줄에 문자가 있다면 마지막 문자를 삭제한다. KILL 문자는 현재 줄에 문자가 있다면 모든 데이타를 삭제한다. ERASE와 KILL 문자는 현재 줄에 문자가 없다면 효과가 없다. ERASE와 KILL 문자는 그들 스스로가 입력 큐에 위치하지는 않는다.

8.1.1.7 비표준 모드 입력 처리

비표준 모드 입력 처리에서 입력 바이트들은 줄 단위로 묶여지지 않으며 지우기와 없애기 처리가 일어나지 않는다. c_cc 배열의 MIN, TIME 멤버값들은 받아들인 바이트들을 어떻게 처리할 것인지를 결정하는데 쓰인다.

MIN은 함수 *read*()가 성공적으로 끝날 때 받아야만 하는 바이트의 최소치를 나타낸다. TIME은 0.1초 단위로 구별되며 갑작스럽고 짧은 시간동안의 데이타 송신을 끝내는데 쓰인다. 만약 MIN 값이 {MAX_INPUT}보다 크다면 그와 같은 요청에 대한 반응은 구현시 정의된다. MIN과 TIME및 그들 사이의 상호간섭에 대한 4가지의 가능한 값들이 아래에서 설명된다.

8.1.1.7.1 사례 A : MIN > 0, TIME > 0

이 경우에는 TIME은 바이트간 타이머로서 쓰이며 첫번째 바이트를 받아들였을때 활성화된다. 그것은 바이트간 타이머이기 때문에 한 바이트를 받은 다음에는 리셋된다. MIN과 TIME간의 상호 작용은 다음과 같다. 즉, 한 바이트를 받자마자 바이트간 타이머가 시작된다. 바이트간 타이머가 효력 중지하기 전에 (타임는 각 바이트를 받자마자 리셋된다는 것을 기억하라) MIN개의 바이트를 받는다면 읽기는 만족되었다. 만약 MIN개의 바이트를 받기 전에 타이머가 중지된다면 그 때까지 받았던 문자들은 사용자에게 되돌려진다. 만약 TIME이 효력 중지된다면 적어도 한 바이트는 되돌려진다. 왜냐하면 한 바이트를 받지 않고는 타이머는 동작 가능 상태가 될 수 없기 때문이다. 이와 같은 경우(즉 MIN>0, TIME>0)에는 MIN과 TIME 메카니즘이 첫번째 바이트를 받아서 활성화되던가 신호를 받았을 때까지 읽기는 차단된다.

8.1.1.7.2 사례 B : MIN > 0, TIME = 0

이 경우에는 TIME 값이 0이기 때문에 타이머는 아무 역할을 못하며 MIN만이 중요하다. MIN개의 바이트를 받을 때까지 또는 신호를 받을 때까지 대기 중인 읽기는 수행되지 않는다. 레코드 단위의 터미날 I/O를 읽기 위해 이 경우를 이용하는 프로그램은 읽기 작업에서 무한정으로 차단될 수 있다.

8.1.1.7.3 사례 C : MIN = 0, TIME > 0

이 경우에는 MIN 값이 0이기 때문에 TIME이 더 이상 바이트간 타이머를 뜻하지 않는다. 그것은 이제는 읽기 타이머로 쓰이며 이 타이머는 함수 *read*()가 처리되자마자 활성화된다. 한 바이트가 읽혀지자마자 읽기는 만족이 되거나 읽기 타이머가 종료된다. 이 경우에는 만약 타이머가 종료되면, 한 바이트로 되돌려지지 않는다는 것을 기억하라. 만약 타이머가 종료되지 않는다면 읽기가 만족될 수 있는 유일한 방법은 한 바이트를 받아들이는 것이다. 이 경우에는 읽기는 한 바이트를 기다리는 것을 무한정으로 차단해서는 안된다. 즉, 읽기가 시작된 후 TIME * 0.1 초 안에 한 바이트도 받지 못한다면 *read*()는 0값을 되돌려주며 데이타를 하나도 읽기 않게 된다.

8.1.1.7.4 사례 D : MIN = 0, TIME = 0

요구 바이트수나 현재 사용가능한 바이트수 중 최소값이 더 이상의 바이트가 입력

되는 것을 기다리지 않고 복귀값으로 되돌려져야 한다. 어떠한 문자도 사용가능하지 않다면, *read*()는 어떠한 데이타도 읽지 않은 채 0 값을 복귀값으로 돌려준다.

8.1.1.8 데이타 쓰기 및 출력 처리

프로세스가 터미날 디바이스 파일에 한 바이트 이상을 쓸 때, 그것들은 c_oflag 필드(**출력 모드** [8.1.2.3]를 참고하라.)에 따라 처리된다. 구현은 버퍼 방식 (*buffer*ing mehcanism)을 제공하기도 한다. *write*()에 대한 호출이 완료되었을 때, 쓰여진 모든 바이트들은 디바이스에로의 전송이 예정되었지만 전송이 완료되지 않아도 된다. *write*() 시의 O_NONBLOCK의 효과에 대해서는 *write*()[7.4.2]를 참고하라.

8.1.1.9 특수문자

어떤 문자들은 입력 또는 출력 또는 둘 다에 대한 특수 기능들을 갖는다. 이 함수들은 다음과 같이 요약된다.

INTR 입력에 대한 특수 문자이며, ISIG 플래그(**지역모드** [8.1.2.5]를 참고하라)가 동작 가능시 인식된다. SIGINT 신호를 생성하는데 이 신호는 터미날이 제어 터미날인 포어그라운드 프로세스 그룹의 모든 프로세스에 보내진다. ISIG가 설정되어 있으면, INTR은 처리될 때 버려진다.

QUIT 입력에 대한 특수 문자이며 ISIG 플래그가 동작 가능시 인식된다. SIGQUIT 신호를 생성하는데 이 신호는 터미날이 제어 터미날인 포어그라운드 프로세스 그룹의 모든 프로세스에 보내진다. 만약 ISIG 가 설정되면, QUIT 문자는 처리될 때 버려진다.

ERASE 입력에 대한 특수 문자이며 ICANON 플래그가 설정되어 있으며, 현재 줄의 마지막 문자를 지운다. **표준 모드 입력 처리**[8.1.1.6]를 참고하라. NL, EOF 이나 EOL문자에 의해 경계지어지는, 한 줄의 시작 부분을 초과해서 지워서는 안된다. 만약 ICANON이 설정되어 있으면, ERASE 문자는 처리할 때 버려진다.

KILL 입력에 대한 특수 문자이고 ICANON 플래그가 설정되어 있으면 인식된다. NL, EOF 또는 EOL 문자에 의해 경계 지어지는 한 줄 전체를 지운다. 만약 ICANON이 설정되어 있으면, KILL 문자는 처리할 때 버려진다.

EOF 입력에 대한 특수 문자이고 ICANON 플래그가 설정되어 있으면 인식된다. EOF을 수신했을 때에는, 읽혀지기를 기다리는 모든 바이트들이 새로운 줄을 기다리지 않고 즉시 그 프로세스에 보내진다. 그리고 EOF는 버려진다. 그래서 만약 기다리는 바이트가 없다면 (즉, 한 줄의 시작점에서 EOF가 일어난다면) 바이트 카운트 값 0이 *read*()의 복귀값으로 되돌려지는데 이는 파일의 끝을 나타낸다. ICANON이 설정되어 있으면, 처리시 EOF 문자는 버려진다.

NL 입력에 대한 특수 문자이고, ICANON 플래그가 설정되어 있으면 인식된다. 이것은 줄 경계 문자('\n')이다.

EOL 입력에 대한 특수 문자이고 ICANON 플래그가 설정되어 있으면 인식된

다. NL과 같은 줄 추가시의 줄 경계문자이다.

SUSP 만약 작업 제어가 지원된다면 (**특수 제어 문자**[8.1.2.6]를 참고하라), SUSP 특수 문자는 입력시 인식된다. 만약 ISIG 플래그가 동작 가능 상태로 되어 있으면, SUSP 문자를 입력 받으면 SIGTSTP 신호가 해당 터미널이 제어터미널인 포어그라운드 프로세스그룹의 모든 프로세스에 보내지며 SUSP 문자는 처리시 버려진다.

STOP 입력과 출력에 대한 특수 문자이며 IXON(출력 제어)이나 IXOFF(입력제어) 플래그가 설정되어 있으면 인식된다. 출력을 임시로 중지시키기 위하여 사용될 수 있다. CRT 터미날에서 출력이 읽히기 전에 사라지는 것을 방지할 때 유용하다. IXON이 설정되어 있다면, STOP 문자는 처리시 버려진다.

START 입력과 출력에 대한 특수 문자이며 IXON(출력 제어)나 IXOFF(입력 제어) 플래그가 설정되어 있으면 인식된다. STOP 문자에 의해 임시로 중지되었던 출력을 재개시키기 위해 사용될 수 있다. IXON이 설정되면 START 문자는 처리시 버려진다.

CR 입력에 대한 특수 문자이며 ICANON 플래그가 설정되어 있으면 인식된다. CR은 '\r'이다. ICANON과 ICRNL이 설정되어 있고 IGNCR이 설정되어 있지 않으면, 이 문자는 NL로 해석되고 NL 문자와 똑같은 효과를 낸다.

NL과 CR문자는 바뀔 수가 없다. START와 STOP문자가 바뀔 수 있는지 여부는 구현시 정의된다. INTR, QUIT, ERASE, KILL, EOF, EOL과 SUSP(작업제어만)을 위한 값은 개개인의 취향에 맞게 바뀔 수 이다.

만약 {_POSIX_VDISABLE}가 터미날 파일에 유효하다면, 바꿈이 가능한 특수 제어 문자와 관련된 특수 문자 함수들은 개별적으로 동작 불능 상태가 될 수 있다. **특수 제어 문자** [8.1.2.6]를 참고 하라.

만약 두 개 이상의 특수 문자들이 같은 값을 가지면, 그 문자가 수신될 때 수행되는 함수는 정의되지 않는다. 특수 문자는 그것의 값에 의해서 뿐만 아니라, 그것의 문맥에 의해서도 인식된다. 예를 들면, 바이트들이 개별적으로 고려될 때 갖는 바이트들의 의미와 다른 의미를 갖는 일련의 다중 바이트가 구현시 정의될 수 있다. 또한 추가의 단일 바이트 함수들이 구현시 정의될 수 있다.이 구현시 정의 되는 다중 바이트나 단일 바이트 함수들은 IEXTEN 플래그가 설정되어 있을 때에만 인식된다. 그렇지 않으면, 데이타는 보통의 문자로 해석되거나 이 절에서 정의되는 특수 문자로 해석된다.

8.1.1.10 모뎀의 단절

만약 모뎀이 단절되었음이 제어 터미날을 위한 터미날 인터페이스에 의해 검출되거나 또는 CLOCAL이 터미날을 위한 c_cflag 필드안에 설정되어 있지 않으면(**제어모드**[8.1.2.4]를 참고하라) SIGHUP 신호는 그 터미날과 관련된 제어 프로세스에 보내진다. 만약 다른 준비(arrangement)가 이루어지지 않았다면, 이것은 제어 프로세스가 종료되게 만든다. *_exit*()[4.2.2]를 참고하라. 그 뒤에 따르는 터미날 디바이스로부터의 읽기는 그 디바이스가 닫혀질 때까지 파일 끝(end-of-file) 표시를 되돌려준다.

그래서 터미날 파일을 읽고 파일의 끝을 검사하는 프로세스들은 단절 후에 제대로 종료될 수 있다. 터미날 디바이스에 대한 그 후의 *write*()는 그 디바이스가 닫혀질 때 까지 *errno*값을[E10]로 한 채 -1을 복귀값으로 되돌린다.

8.1.1.11 터미날 디바이스 파일의 닫기

터미날 디바이스 파일을 닫기 위한 마지막 프로세스는 어떠한 출력도 그 디바이스에 보내지도록 하고 어떠한 입력도 버리도록 해야 한다. 따라서, 만약 HUPCL이 제어 모드에서 설정되어 있으면, 통신 포트는 단절 함수를 지원하고 터미날 디바이스는 단절을 수행할 것이다.

8.1.2 설정 가능 매개변수

8.1.2.1 termios 구조체

어떤 터미날 I/O 특성들을 제어할 필요가 있는 루틴들은 헤더 **<termios.h>**에 정의되는 *termios* 구조체를 사용하여 그렇게 한다. 이 구조체의 멤버들은 표8-1에 보여지는 멤버들을 포함하지만 이 멤버들로 제한되지는 않는다.

<표 8-1>　　termios 구조체

멤버 형	배열 크기	멤버 명	설 명
tcflag_t		*c_iflag*	입력 모드
tcflag_t		*c_oflag*	출력 모드
tcflag_t		*c_cflag*	제어 모드
tcflag_t		*c_lflag*	지역 모드
cc_t	NCCS	*c_cc*	제어 문자

형들 tc_flag_t와 cc_t는 헤더 **<termios.h>**에 정의되어져야 한다. 그것들은 부호없는 정수형이어야 한다.

termios 구조체의 전체 크기는 구현시 정의된다.

8.1.2.2 입력모드

표 8-2에 보여지는 c_iflag 필드의 값은 기본 터미날 입력 제어를 기술하며 보여진 마스크들은 비트단위 포괄적(inclusive)OR연산으로 구성된다. 여기서 마스크들은 비트별로 달라야 한다. 이 표의 마스크 명 심볼들은 **<termios.h>**에 정의된다.

비동기 직렬 데이타 전송에서는, 전송 중단 조건은 한 바이트를 보내기 위한 시간보다 더 오랫동안 계속되는 일련의 0값의 비트들로 정의된다. 비록 한 바이트 이상의 시간동안 지속될지라도, 연속된 0 값의 비트들은 하나의 전송 중단 조건으로 해석된다. 비동기 직렬 데이타 전송이 아닌 경우에는 전송 중단 조건의 정의는 구현시 정의된다.

<표 8-2> **termios c_iflag 필드**

마스크 명	설 명
BRKINT	전송 중단시에 인터럽트 신호를 보냄
ICRNL	입력시 CR을 NL로 대응시킴
IGNBRK	전송 중단 조건을 무시
IGNCR	CR를 무시
IGNPAR	패리티 에러가 있는 문자들을 무시
INLCR	입력시 NL을 CR로 대응시킴
INPCK	입력 패리티 검사를 동작 가능 상태로 함
ISTRIP	7비트 문자로 변환 시킴
IXOFF	시작/정지 입력 제어를 동작 가능 상태로 함
IXON	시작/정지 출력 제어를 동작 가능 상태로 함
PARMRK	패리티 에러를 표시

만약 IGNBRK가 설정되어 있으며, 입력시 검출되는 전송 중단 조건은 무시된다. 즉, 입력 큐에 넣어지지 않으며 따라서 어떠한 프로세스에 의해서도 읽혀지지 않는다. 만약 IGNBRK가 설정되어 있지 않고 BRKINT가 설정되어 있으면, 전송 중단 조건은 입력과 출력 큐를 쓸어낼 것이며, 만약 터미날이 포어그라운드 프로세스 그룹의 제어 터미날이라면 전송 중단 조건은 그 포어그라운드 프로세스 그룹에 하나의 SIGINT 신호를 생성해주어야 한다. 만약 IGNBRK와 BRKINT 둘 다 설정되어 있지 않으면, 전송 중단 조건은 하나의 '\0'으로 읽혀지고, 또는 PARMRK가 설정되어 있으면 '\377', '\0', '\0' 으로 읽혀진다.

만약 IGNPAR가 설정되어 있으면, (전송 중단이 아닌) 프레임 에러나 패리티 에러가 있는 바이트는 무시된다.

만약 PARMRK가 설정되어 있고 IGNPAR는 설정되어 있지 않으면, (전송 중단이 아닌) 프레임 에러나 패리티 에러가 있는 바이트는 세 문자 수열 '\377', '\0', X로 응용에 주어진다. 여기서 '\377', '\0'은 각 수열 앞에 나오는 두 문자 플래그이고, X는 잘못 수신된 문자 데이타이다. 이 경우에 모호함을 피하기 위해서, ISTRIP가 설정되어 있지 않으면, '\377'의 올바른 문자가 '\377', '\377'로 응용에 주어진다. 만약 PARMRK와 IGNPAR 둘 다 설정되어 있지 않으면, (전송 중단이 아닌) 프레임 에러나 패리티 에러는 응용에게 단일 문자 '\0'로 주어진다.

만약 INPCK가 설정되어 있으면, 입력 패리티 검사는 동작 가능 상태가 된다. 만약 INPCK가 설정되어있지 않으면 입력 패리티 검사는 동작 불능 상태가 되어 입력 패리티 에러들이 없는 출력 패리티 생성을 허용한다. 입력 패리티 검사가 동작 가능 또는 불능 상태가 되는 것은 패리티 검출이 동작 가능 또는 불능 상태가 되는 것과는 상관이 없다. (**제어모드**[8.1.2.4]참고) 만약 패리티 검출이 동작 가능 상태이지만 입력

패리티 검사가 동작 불능 상태가 되면, 터미날이 연결되는 하드웨어는 패리티 비트를 인식해야 하지만, 터미날 특수 파일은 이 비트가 옳게 설정되어 있는지 여부를 검사하지 않는다.

만약 ISTRIP가 설정되어 있으면, 유효한 입력 바이트들은 7 비트로 벗겨지고 (stripped), 그렇지 않으면 모든 8 비트들이 처리된다.

만약 INLCR이 설정되어 있으면, 수신된 NL문자는 CR문자로 번역된다. 만약 IGNCR이 설정되어 있으면 수신된 CR 문자는 읽혀지지 않고 무시된다. 만약 IGNCR이 설정되어 있지 않고 ICRNL는 설정되어 있으면, 수신된 CR 문자는 NL 문자로 번역된다.

만약 IXON이 설정되어 있으면, 시작/정지 출력 제어가 동작 가능 상태가 된다. 수신된 STOP문자는 출력을 중지시키고 수신된 START 문자는 출력을 재발송시킨다. IXON이 설정되어 있을 때 START, STOP문자들은 읽혀지지 않고 단순히 흐름 제어 기능들을 수행한다. IXON이 설정되어 있지 않으면 START, STOP문자들은 읽혀진다.

만약 IXOFF가 설정되어 있으면 시작/중지 입력 제어는 동작 가능 상태가 된다. 시스템은 터미날 디바이스로 하여금 데이타 송신을 중지하도록 할 목적으로 한 개 이상의 STOP 문자들을 전송하는데 이 STOP 문자들은 입력 큐에 있는 바이트수가 {MAX_INPUT}값을 초과하지 않도록 하기 위해 필요하다. 또한 시스템은 터미날 디바이스가 입력 큐를 오버플로우시키는 위험이 없이 데이타 송신을 계속할 수 있자마자 하나 이상의 START 문자들을 송신하는데 이 문자들은 터미날 디바이스가 데이타 송신을 재개하도록 하는 목적이다. STOP, START 문자들이 송신되는 상세한 조건들은 구현시 정의된다.

open() 후의 초기의 입력 제어 값은 구현시 정의된다.

8.1.2.3 출력 모드

c_oflag 필드의 값들은 기본 터미날 출력에 관한 것이다. 이 값은 각각 고유한 마스크들의 비트 단위 포괄적(inclusive)OR연산으로 구성된다.

마스크 명	설명
OPOST	출력 처리 수행

c_oflag필드를 위한 마스크명 심볼들은 **<termios.h>**에서 정의된다.

OPOST가 설정되어 있다면 출력 데이타들은 구현시 정의된 방법에 따라서 텍스트 라인들이 터미날에 적당하게 보이도록 변환된다. OPOST가 설정되어 있지 않다면 문자들을 아무런 변환없이 출력된다.

함수 *open*()이 호출된 후의 초기 출력 제어 값은 구현시 정의되어 진다.

8.1.2.4 제어 모드

표 8-3에 설명되어 있는 c_cflag 필드 값들은 기본 하드웨어 제어에 관한 것이다.이 값들은 각각 고유한 마스크들의 비트단위 포괄적(inclusive)OR연산으로 구성된다. 명시된 모든 값들이 기본 하드웨어에서 지원되도록 요구되지는 않는다. 이 표에 있는 마스크명 심볼들은 **<termios.h>**에서 정의된다.

CSIZE는 전송시와 수신시의 바이트 크기(비트 단위로)를 의미한다. 이 바이트 크기는 패리티 비트가 있다면 패리티 비트를 포함하지 않는다. CSTOPB이 설정되어 있다면 정지 비트(stop bit)로서 두 개의 비트를 사용하고, 설정되어 있지 않다면 한 개의 정지 비트를 사용한다. 예를 들면 일반적으로 110 보오드(baud)에 두개의 정지 비트가 사용된다.

CREAD가 설정되어 있다면 수신이 가능한 상태가 된다. CREAD가 설정되어 있지 않다면 문자가 수신되지 않는다.

PARENB가 설정되어 있다면 패리티 비트를 생성과 검출이 가능하게 되고 한 비트의 패리티 비트가 각 문자에 더해진다. 패리티가 작동 가능한 경우에는 PARODD가 설정되어 있다면 홀수 패리티를 사용하고, PARODD가 설정되어 있지 않다면 짝수 패리티를 사용한다.

<표 8-3>　　termios c_cflag 필드

마스크 명	설　　명
C*locale*	모뎀 상태 라인을 무시함
CREAD	수신을 가능하게함
CSIZE	바이트당 비트수*
CS5	5 비트
CS6	6 비트
CS7	7 비트
CS8	8 비트
CSTOPB	두개의 정지 비트(stop bit)를 보냄.그렇지 않으면 한 비트임
HUPCL	마지막 닫기에 자동적으로 통신을 끝낼수 있음
PARENB	패리티를 고려함
PARODD	홀수패리티, 그렇지 않으면 짝수 패리티

* CSIZE는 역사적으로 문자 크기(*charact*er *size*)라고도 한다.

HUPCL이 설정되어 있다면 포트 열기를 한 마지막 프로세스가 그 포트를 닫거나, 그 프로세스가 종료될 때 그 포트를 위한 모뎀 제어 라인이 리셋된다.

CLOCAL이 설정되어 있다면 연결 여부는 모뎀 상태 회선의 상태와 무관하다.

CLOCAL이 설정되어 있지 않다면 연결 여부를 위해 모뎀의 상태 회선이 검사된다.

일반적으로 함수*open*()을 호출하면 모뎀이 완전하게 연결될 때까지 기다리게 된다. 그러나 O_NONBLOCK(*open*() [6.3.1] 참고)이 설정되어 있거나 CLOCAL이 설정되어 있다면 함수 *open*()은 모뎀이 완전하게 연결될 때까지 기다리지 않고 즉시 끝낸다.

제어 모드가 설정되어 있는 대상이 비동기 직렬 연결로 되어있지 않다면 몇가지 제어 모드는 무시되어진다. 예를 들자면 네트웨크로 연결된 다른 호스트의 터미날에의 전송 속도를 설정하려 한다면, 그 터미날과 터미날이 직접 연결된 기계 사이의 전송 속도는 설정되어질 수도 있고, 설정되지 않을 수도 있다.

함수 *open*()이 호출된 후의 초기 하드웨어 제어 값은 구현시 정의되어 진다.

8.1.2.5 지역 모드 (locale Modes)

표 8-4에 보여지는 c_lflag 필드의 값들은 여러가지 함수의 제어에 관한 것이다. 이 값들은 각각 고유한 마스크(mask)들의 비트 단위 포괄적(inclusive)OR연산으로 구성된다. 마스크명 심볼들은 **<termios.h>**에 정의되어 진다.

<표 8-4>　　termios c_lflag 필드

마스크명	설　명
ECHO	입력 문자 반향(echo)됨
ECHOE	에러 정정 백스페이스로서 ERASE가 반향됨
ECHOK	KILL이 반향됨
ECHONL	'\n'이 반향됨
ICANON	표준 입력(지우기와 죽이기 처리)
IEXTEN	구현시 정의되는 확장 함수 지원
ISIG	신호 작동 가능
NOFLSH	인터럽트,종료(quit),중지(suspend)후에 쓸어버리기 불가능
TOSTOP	백그라운드 출력을 위한 SIGTTOU전송

ECHO가 설정되어 있다면 입력된 문자들이 화면에 보여진다. ECHO가 설정되어 있지 않다면 문자가 입력되더라도 화면에 보이지 않는다.

ECHO와 ICANON이 설정되어 있다면 ERASE 문자는 화면의 현재 줄의 마지막 문자를 지운다. 현재 줄에 지울 문자가 없다면 아무일도 수행하지 않거나 또는 지울 문자가 없다는 것을 알려줄 수 있는데 이는 구현시 정의된다.

ECHOK과 ICANON이 설정되어 있다면 KILL 문자는 화면의 한 줄을 지우거나 또는 KILL 문자 다음에 '\n'을 출력한다.

ECHONL과 ICANON이 설정되어 있다면 ECHO가 설정되어 있지 않더라도 '\n'이 출력된다.

ICANON이 설정되어 있다면 표준 처리(canonical processing)가 가능하다. 즉 'erase'와 'kill' 편집 기능을 가능하게 하고, 입력된 문자들을 조합하여 줄로 만들어 준다. 이 줄들은 NL, EOF, EOL등으로 구분된다(**표준 모드 입력 처리** [8.1.1.6] 참 고).

ICANON이 설정되어 있지 않다면 읽기 요청은 입력 큐로 부터 직접 처리된다. 최소한 MIN 바이트가 입력되거나, 입력시 문자와 문자 사이의 종료 시간인 TIME이 지나야 읽기가 완료된다. 읽기 시간의 단위는 10 분의 1 초이다. 더 자세한 것을 위해서는 **비표준 모드 입력 처리[8.1.1.7]**을 참고하라.

ISIG이 설정되어 있다면 각 입력 문자는 INTR, QUIT, SUSP(작업 제어에만 해당) 등과 같은 특수 제어 문자와 비교, 검사한다. 이들 입력 문자가 특수 제어 문자의 하나와 일치하면 그것에 해당하는 함수가 수행된다. ISIG이 설정되어 있지 않다면 이러한 검사는 수행되지 않는다. 따라서 특수 입력 함수는 ISIG이 설정되어 있을 경우에만 가능하다.

IEXTEN이 설정되어 있다면 구현시 정의된 함수가 입력 데이타로부터 인식되어야 한다. IEXTEN이 설정되어 있는 것이 ICANON, ISIG, IXON, IXOFF 등과 어떻게 작용할 것인가는 구현시 정의되어 진다. IEXTEN이 설정되어 있지 않다면 구현시 정의된 함수가 인식되지 않고, 입력 문자는 ICANON, ISIG, IXON, IXOFF 등에 설명된 것과

같이 처리된다.

NOFLSH가 설정되어 있다면 INTR, QUIT, SUSP(작업 제어에만 해당) 등과 같은 문자와 관련된 입출력 큐에 대한 정상적인 쓸어버리기는 수행되지 않는다.

TOSTOP이 설정되어 있고 작업 제어(job control)를 지원할 수 있게 구현되었다면 터미날의 포어그라운드 프로세스 그룹에 속하지 않을 때 프로세스의 제어 터미날에 쓰기를 시도하는 프로세스의 프로세스 그룹으로 신호 SIGTTOU가 보내진다. 이 신호는 프로세스 그룹의 멤버들을 중지 시킨다. 그렇지 않으면 그 프로세스에 의해 생성된 출력은 현재 출력 스트림에의 출력이다. SIGTTOU 신호를 차단하거나 무시하는 프로세스들은 예외이며 출력을 하도록 허용되며 SIGTTOU신호는 보내지지 않는다.

함수 *open*()이 호출된 후의 초기 지역 제어값은 구현시 정의된다.

8.1.2.6 특수 제어 문자

특수 제어 문자의 값들은 배역 c_cc에 의해서 정의된다. 표준 모드와 비표준 모드에서 이 배열의 각 원소에 대한 첨자명과 설명이 표 8-5에 보여진다. 이 표에서의 첨자명 심볼은 **<termios.h>**에서 정의된다.

첨자들은 각각 고유한 값을 가진다. 단 VMIN와 VEOF, VTIME과 VEOL은 각각 같은 값을 가진다.

작업 제어를 지원하지 않는 구현에서는 첨자 VSUSP에 의해 인덱스된 c_cc 배열에서의 SUSP 문자값을 무시한다.

NCCS는 c_cc 배열의 원소수를 의미하며 구현시 정의된다.

<표 8-5> termios c_cc 특수 제어 문자
첨자 사용

표준 모드	비표준모드	설명
VEOF		EOF 문자
VEOL		EOL 문자
VERASE		ERASE 문자
VINTR	VINTR	INTR 문자
VKILL		KILL 문자
	VMIN	MIN 값
VQUIT	VQUIT	QUIT 문자
VSUSP	VSUSP	SUSP 문자
	VTIME	TIME 값
VSTART	VSTART	START 문자
VSTOP	VSTOP	STOP 문자

START 문자와 STOP 문자의 변환을 지원하지 않는 구현에서는 함수 *tcsetattr*()이 호출되었을 때 첨자 VSTART와 VSTOP에 의해 인덱스된 c_cc 배열에서의 문자 값들

을 무시하지만 *tcgetattr*()이 호출되었을 때에는 사용중인 값을 함수 값으로 되돌려줘야한다.

터미날 디바이스 파일에 대하여 {_POSIX_VDISABLE}이 정의되어 있고, 변환 가능한 특수 제어 문자(특수 문자[8.1.1.9] 참고) 중 하나의 값이{_POSIX_VDISABLE}이라면 해당 함수는 동작 불능 상태가 된다.

즉 어느 입력 데이타로 동작 불능 특수 문자로서 인식되지 않는다. ICANON이 설정되어 있지 않다면 {_POSIX_VDISABLE}의 값은 배열 c_cc의 VMIN, VTIME입력에 대해 아무런 특별한 의미를 갖지 못한다.

모드 제어 문자의 초기값은 구현시 정의되어진다.

8.1.2.7 전송 속도(baud rate) 함수

함수 : *cfgetispeed*(), *cfgetospeed*(), *cfsetispeed*(), *cfsetospeed*()

8.1.2.7.1 용례

```
#include <termios.h>

speed_t cfgetospeed(termios_p)
struct termios * termios_p ;

int cfsetospeed(terio_p,speed)
struct termios * termios_p ;
speed_t speed ;

speed_t cfgetispeed(termios_p)
struct termios * termios_p ;

int cfsetispeed(termios_p,speed)
struct termios * termios_p ;
speed_t speed ;
```

8.1.2.7.2 설명

*termios*구조에 있는 입/출력 전송 속도 값을 알아내고 설정하기 위해 다음과 같은 인터페이스가 제공된다. 아래에서 설명된 터미날 디바이스에 대한 영향은 *tcsetattr*()함수가 성공적으로 호출될 때까지는 효력이 발생하지 않는다.

입/출력 전송 속도들은 *termios* 구조에 저장된다. 표 8-6에서 보여 주고 있는 값들이 지원되며 표 8-6에 있는 심볼들에 대해서는 **<termios.h>**에 정의되어 있다.

<표 8-6> termios 전송 속도 값들

항목	설명	항목	설명
B0	끊김	B600	600 보오드
B50	50 보오드	B1200	1200 보오드
B75	75 보오드	B1800	1800 보오드
B110	110 보오드	B2400	2400 보오드
B134	134.5 보오드	B4800	4800 보오드
B150	150 보오드	B9600	9600 보오드
B200	200 보오드	B19200	19200 보오드
B300	300 보오드	B38400	38400 보오드

speed_t 형은 **<termios.h>**에 정의되고 부호 없는 정수(un*sig*ned integer)형이어야 한다.

ternios_p 인수는 *termios* 구조에 대한 포인터이다.

cfgetospeed()는 *termios_p*가 가르키고 있는 *termios* 구조에 저장된 출력 전송 속도를 복귀값으로 한다.

cfsetospeed()는 *termios_p*가 가르키고 있는 *termios* 구조에 저장된 출력 전송 속도를 *speed*에 설정한다.

전송 속도값이 0인 B0는 연결을 단절하는데 사용한다. 만약 B0가 명시되어 있으면 모뎀 제어 라인들은 더 이상 작동하지 않고 따라서 연결이 단절되게 된다.

cfgetispeed()는 *termios* 구조체에 저장되어 있는 입력 전송 속도를 함수 값으로 한다. *cfsetispeed*()는 *termios* 구조체에 저장되어 있는 입력 전송 속도 값을 *speed*에 설정한다.

만약 입력 전송 속도가 0으로 설정되면 입력 전송 속도는 출력 전송 속도에 의해 명시될 것이다. *cfsetispeed*()와 *cfsetospeed*()는 만약 성공적으로 수행되면 0의 값을, 에러가 발생하면 -1을 복귀값으로 한다.

지원되지 않는 전송 속도로 설정하려는 시도들은 무시되며, 에러가 *cfsetispeed*(), *cfsetospeed*() 또는 *tcsetattr*() 전부에 의해서 되돌려 지는지 아니면 이들 중 어느 것에 의해서만 되돌려 지는지의 여부는 구현시 정의된다.

이것은 하드웨어에 의해 지원되지 않는 전송 속도들의 변환과 만약 하드웨어가 이것을 지원하지 않으면 입/출력 전송 속도를 다른 값으로 설정하는 변환을 지칭한다.

8.1.2.7.3 복귀값

설명 참고

8.1.2.7.4 에러(error)

표준에서는 *cfgetispeed*(), *cfgetospeed*(), *cfsetispeed*(), *cfsetospeed*() 함수들에 대해 에러가 검출되도록 요구되는 어떠한 에러 조건도 명시하지 않는다. 어떤 에러들은 구현시

정의되는 조건하에서 검출될 수 있다.

8.1.2.7.5 참고

tcsetattr() [8.2.1]

8.2 일반 터미날 인터페이스 제어 함수

일반적인 터미날 기능을 제어하는데 사용되는 함수들이 이 절에서 설명된다. 만약 구현이 작업제어를 지원한다면 특정 명령에 대해 부연 설명되지 않는 한 이러한 함수들은 백그라운드 프로세스에 의한 사용은 제한된다. 이러한 일들을 수행하려는 시도는 그 프로세스 그룹이 SIGTTOU신호를 받도록 해야한다. 만약 호출 프로세스가 SIGTTOU 신호를 차단하거나 무시한다면 그 프로세스는 그 작업을 수행하도록 허락되며 SIGTTOU신호는 보내지지 않는다.

모든 함수에서 *fildes*는 열린 파일 서술자이다. 그러나 그 함수들은 현재 터미날 파일에만 영향을 미치고, 그 파일 서술자와 관련 있는 열린 파일 서술자에는 영향을 미치지 않는다.

8.2.1 속성 관련 함수

함수 : *tcgetattr*(), *tcsetattr*()

8.2.1.1 용례

#include <termios.h>

int tcgetattr (*fildes*,*termios_p*)
int *fildes*;
struct *termios* * *termios_p*;

int tcsetattr (*fildes*, *optional_actions*, *termios_p*)
int *fildes*, *optional_actions*;
struct *termios* * *termios_p*;

8.2.1.2 설명

함수 *tcgetattr*()는 *fildes*에 의해 언급된 객체(object)와 관련 있는 매개 변수를 가져야 하고 그 변수들을 *termios_p*에 의해 언급된 *termios* 구조체에 저장해야 한다.

이 함수는 백그라운드 프로세스로부터 허용된다. 그러나 터미날 속성들은 이후에 포어그라운드 프로세스에 의해 바뀔 수 있다.

함수 *tcgetattr*()는 터미날과 관계 있는 매개 변수들을 *termios_p*에 의해 참고되는 *termios* 구조로부터 다음과 같이 설정해야 한다.(현재 가용하지 않은 기본 하드웨어로부터의 지원이 요구되지 않는한)

(1) *optional_actions*이 TCSANOW이면 매개 변수 설정이 즉각적으로 발생한다.

(2) *optional_actions*이 TCSADRAIN이라면 *fildes*에 기록된 모든 출력이 전송된 다음 매개 변수 설정이 일어난다. 출력에 영향을 미치는 매개 변수를 변경할 때 이 함수는 반드시 사용되어야 한다.

(3) *optional_actions*이 TCSAFLUSH이면 *fildes*에 의해 참고되는 객체에 기록된 모든 출력이 전송된 후에 매개 변수 설정이 발생한다. 전송 받았지만 읽히지 않은 모든 입력은 매개 변수 설정이 일어나기 전에 버려진다.

optional_actions 값을 위한 심볼 상수들은 **<termios.h>**에 정의 된다.

8.2.1.3 복귀값

성공적으로 끝났을때에는 0을 복귀값으로 한다. 그렇지 않으면 -1을 복귀값으로 하며 에러를 나타내기 위하여 *errno*는 설정된다.

8.2.1.4 에러

다음 조건들 중 어느 하나라도 일어난다면 함수 *tcgtattr*()는 -1을 복귀값으로 하며 *errno*를 이에 상응하는 값으로 설정한다.

[EBADF] *fildes* 인수가 올바른 파일 서술자가 아니다.
[ENOTTY] *fildes*와 관련 있는 파일이 터미날이 아니다.

만약 다음 조건들 중 어느 하나라도 일어난다면 함수 *tcsetattr*()는 -1을 복귀값으로 하며*errno*를 이에 상응하는 값으로 설정한다.

[EBADF] *fildes* 인수가 올바른 파일 서술자가 아니다.
[EINVAL] *optional_actions* 인수의 값이 적절하지 않거나 *termios* 구조체에 표현된 속성을 지원되지 않는 값으로 변경 시키고자 하는 시도가
[ENOTTY] *fildes*와 관계있는 파일이 터미날이 아니다.

8.2.1.5 참고

<termios.h> [8.1.2]

8.2.2 회선 제어 함수

함수 : *tcsendbreak*(), *tcdrain*(), *tcflush*(), *tcflow*()

8.2.2.1 용례

```
#include <termios.h>

int tcsendbreak(fildes, duration)
int fildes ;
int duration ;

int tcdrain(fildes)
int fildes ;
```

int tcflush (*fildes*, *queue_selector*)
int *fildes;*
int queue;

int tcflow (*fildes*, *act*ion)
int *fildes;*
int *act*ion;

8.2.2.2 설명

터미날이 비동기 직렬 데이타 전송을 시용하고 있다면 함수 *tcsendbreak*()는 연속된 0 값의 비트들을 일정 시간 동안 전송해야 한다. 만약 *duration*이 0이라면 적어도 0.25초 이상, 0.5초 이하 동안 0 비트들을 전송해야 한다. 만약 *duration*이 0이 아니면 구현시 정의되는 시간 동안 0값의 비트를 전송해야 한다.

만약 터미날이 비동기 직렬 데이타 전송을 하지 않는 다면 함수 tcsendbrsak()가 전송 중단 조건(구현에 의해 정의 되어 진다)을 만들기 위해 데이타를 전송해야 하는지 또는 아무런 행동도 취하지 않고 끝날지의 여부는 구현시 정의된다.

함수 *tcdrain*()는 fides에 의해 참고되는 객체에 기록된 모든 출력이 전송될 때까지 기다려야 한다.

tcflush()함수는 fides에 의해 참고된 객체에 기록되었지만 전송되지 않은 데이타를 전송 받았지만 읽혀지지 않은 데이타를 무시한다. 이때 어느 것을 무시할 것인가는 *queue_selector* 의 값에 달려있다.

(1) *queue_selector*가 TCIFLUSH이라면 전송 받았지만 읽혀지지 않은 데이타를 버려야 한다.
(2) *queue_selector*가 TCOFLUSH이라면 기록 되었지만 전송되지 않은 데이타를 버려야 한다.
(3) *queue_selector*가 TCIOFLUSH이라면 전송받았지만 읽혀지지 않은 데이타와 기록되었지만 전송되지 않은 데이타 모두를 버려야한다.

함수 *tcflow*()는 *act*ion의 값에 따라서 fides에 의해 참고된 객체에 관한 데이타를 전송하거나 전송받는 일을 중단해야 한다.

(1) *act*ion 값이 TCOOFF 이면 출력을 일시 중단해야한다.
(2) *act*ion 값이 TCOON 이면 중단되었던 출력을 재개해야한다.
(3) *act*ion 값이 TCIOFF 이면 시스템은 STOP문자를 전송해야하고 이것은 터미날 디바이스가 시스템에 데이타를 전송하는 것을 중단하도록 하는 목적을 가지고 있다. (**입력 모드 [8.1.2.2.]**의 IXOFF에 대한 설명 참고)
(4) *act*ion 값이 TCION 이면 시스템은 START 문자를 전송해야하고 이것은 터미날 디바이스가 시스템에 데이타 전송을 시작하도록 하는 목적을 가지고 있다.

(**입력 모드** [8.1.2.2.]의 IXOFF에 대한 설명 참고)

queue selector와 *act*ion 값을 위한 심볼 상수들은 〈***termios*. h>**에서 정의된다.
터미날 파일을 여는데 있어서의 기본 양식은 입력및 출력중 어느 것도 중단되지 않는다는 것이다.

8.2.2.3 복귀값

성공적으로 끝났을때는 0 값을 복귀값으로 한다. 그렇지 않으면 -1을 복귀값으로 하며 에러를 나타내기 위해 *errno*가 설정된다.

8.2.2.4 에러

다음 조건들 중 어느 하나라도 일어난다면 함수 *tcsendbreak*()는 -1을 복귀값으로 하며 *errno*는 상응하는 값으로 설정된다.

[EBADF] fildes 인수가 올바른 파일 서술자가 아니다.
[ENOTTY] fildes와 관계 있는 파일이 터미날이 아니다.

다음 조건들 중 어느 하나라도 일어난다면 함수 *tcdrain*()는 -1을 복귀값으로 하고 *errno*는 상응하는 값으로 설정된다.

[EBADF] *fildes* 인수가 올바른 파일 서술자가 아니다.
[EINTR] *tcdrain*() 함수를 어떤 신호가 인터럽트(interrupt) 걸었다.
[ENOTTY] *fildes*와 관련 있는 파일이 터미날이 아니다.

다음 조건들 중 어느 하나라도 일어난다면 함수 *tcflush*()는 -1을 복귀값으로 하고 *errno*는 상응하는 값으로 설정된다.

[EBADF] fides 인수가 올바른 파일 서술자가 아니다.
[EINVAL] queue_selector 인수가 적절한 값을 가지고 있지 않다.
[ENOTTY] *fildes*와 관련 있는 파일이 터미날이 아니다.

다음 조건들 중 어느 하나라도 일어난다면, 함수 *tcflow*()는 -1을 복귀값으로 하고 *errno*는 상응하는 값으로 설정된다.

[EBADF] *fildes* 인수가 올바른 파일 서술자가 아니다.
[EINVAL] *act*ion 인수가 적절한 값을 가지고 있지 않다.
[ENOTTY] fildes와 관련 있는 파일이 터미날이 아니다.

8.2.2.5 참고

<termios.h> [8.1.2]

8.2.3 포어그라운드 프로세스 그룹 ID의 획득

함수: *tcgetpgrp*()

8.2.3.1 용례

#include 〈sys/types.h〉

pid_t tcgetpgrp(*fildes*)
int *fildes*:

8.2.3.2 설명

If {_POSIX_JOB_CONTROL}이 정의되어있다면:

(1) 함수 *tcgetpgrp*()는 터미날과 연관되어있는 포어그라운드 프로세스 그룹의 프로세스 그룹 ID 값을 복귀값으로 한다.

(2) 함수 *tcgetpgrp*()는 백그라운드 프로세스 그룹의 멤버인 프로세스에 의해 행해진다. 그러나 정보는 포어그라운드 프로세스 그룹의 멤버인 프로세스에 의해 그 후에 변경되어질 수도 있다.

정의되어있지 않다면:

위에서 기술한 것과 같이 함수 *tcgetpgrp*()의 기능을 지원하도록 구현하던지 아니면 함수 *tcgetpgrp*()에의 호출은 실패한다.

8.2.3.3 복귀값

성공적으로 완료되면, 함수 *tcgetpgrp*()는 터미날과 관련되어 있는 포어그라운드 프로세서 그룹의 프로세서 그룹 ID를 복귀값으로 한다. 그러나 성공적으로 완료되지 않았으면, -1을 복귀값으로 하며 에러상태를 나타내도록 *errno*를 설정한다.

8.2.3.4 에러

다음 조건들 중 어느 하나라도 일어난다면 함수 tcgetpgrp()는 -1을 복귀값으로 하고 *errno*를 다음의 해당 값으로 설정한다.

[EBADF] *fildes* 인수가 올바른 파일 서술자가 아니다.

[ENOSYS] 함수 *tcgetpgrp*()가 구현되지 않는다.

[ENOTTY] 호출 프로세서가 제어 터미날을 가지고 있지않거나 그 파일이 제어 터미날이 아니다.

8.2.3.5 참고

setsid()[5.3.2], *setpgid*()[5.3.3], *tcsetpgrp*()[8.2.4].

8.2.4 포어그라운드 프로세스 그룹 ID의 설정

함수: *tcsetpgrp*()

8.2.4.1 용례

#include 〈sys/types. h〉

int tsetpgrp (fildes, *pgrp_id*)
int *fildes;*
pid_t *pgrp_id;*

8.2.4.2 설명

만약 { POSIX JOB CONTROL}이 정의되어있다면:

프로세스가 제어 터미날을 가지고 있으면, 함수 *tsetpgrp*()는 그 터미날과 관련되어 있는 포어그라운드 프로세스 그룹 ID를 *pgrp_id* 값으로 설정해야한다. *fildes*와 관련있는 파일은 호출 프로세스의 제어 터미날이어야하며. 제어 터미날은 호출 프로세서의 세션(session)에 현재 연관되어있어야 한다. *pgrp_id* 값은 호출 프로세서와 같은 세션(session)에 있는 프로세스의 프로세스 그룹 ID와 일치해야 한다.

정의되어있지 않다면:

위에서 기술한 것과 같이 함수 *tcsetpgrp*()의 기능을 지원하도록 구현하던지 아니면 함수 *tcsetpgrp*()에의 호출은 실패한다.

8.2.4.3 복귀값

성공적으로 완료되면, 함수 *tcsetpgrp*()는 0을 복귀값으로 한다. 그러나 성공적으로 완료되지않았으면 -1값을 복귀값으로 하고 에러 상태를 나타내도록 *errno*를 설정한다.

8.2.4.4 에러

다음 조건들 중 어느 하나라도 일어난다면 함수 *tcsetpgrp*()는 -1을 복귀값으로 하고 *errno*를 다음의 해당 값으로 설정한다.

[EBADF] *fildes*인수가 올바른 파일 서술자가 아니다.

[EINVAL] *pgrp_id* 인수의 값이 구현에서 지원되지 않는 값이다.

[ENOSYS] 함수 *tcsetpgrp*()가 구현되지 않는다.

[ENOTTY] 호출 프로세스가 제어 터미날을 가지고 있지않거나 그 파일이 제어 터미날이 아니다. 또는 제어 터미날이 더이상 호출 프로세스의 세션(session)과 연관되어지지않는다.

[EPERM] *pgrp_id* 값이 구현에 의해 지원되는 값이지만 호출 프로세스와 같은 세션에 있는 프로세스의 프로세스 그룹 ID와 일치하지 않는다.

제 9장 C-언어를 위한 언어- 명사형 서비스

9.1 참고된 C-언어 루틴

아래 열거된 함수들은 지시되어진 C-표준의 지시된 절들에서 설명되어진다. C-언어 바인딩이 있는 IEEE Std 1003.1-1988은 이 함수들, 이 장에서 기술된 함수들에 대한 확장, 그리고 이 표준에서 규정된 나머지 요구 사항들을 포함하고 있다. 덧셈(+)표시가 덧붙여져있는 함수는 C-표준보다 많은 요구사항을 가지고 있다. C-언어 바인딩에 있는 IEEE Std 1003.1-1988을 수용했다고 주장하는 어느 구현도 이 장의 요구사항과 이 표준에서 규정된 요구사항, C-표준의 해당 절에서의 요구 사항에 따라야 한다.

이 장에서 수용과 관련한 요구사항에 대해서는 **C 프로그램 언어를 위한 언어 관련 서비스**[3.2.3] 과 그 부속절들을 참고하라.

4.2 진단 함수

assert.

4.3 문자 처리 함수

isalnum, isapha, idcntrl, isdigit, islower, islowr, isprint,
ispunct, isspace, isupper, isxdigit, tolwer, toupper.

4.4 지역화 함수

setlocale+.

4.5 수학 함수

acos, asin, atan, atan2, cos, sin, tan, cosh, sinh, tanh, exp,
frexp, ldexp, log, log10, modf, pow, sqrt, ceil, fabs, floor, fmod

4.6 전역 점프(Non-Local Jumps) 함수

setjmp, longjmp

4.9 입출력 함수

clearerr, fclose, feof, ferror, fflush, fgetc, fgets, fopen,
fputc, fputs, fread, freopen, fseek, ftell, fwrite, getc,
gets, perror, printf, fprintf, sprintf, putc, putchar, puts,
remove, rename+, rewind, scanf, fscanf, sscanf, setbuf, tmpfile,
tmpnam, ungetc.

4.10 일반 유틸리티 함수

abs, atof, atoi, atol, rand, srand, calloc, free, malloc, realloc,
abort+, exit, getenv+, bsearch, qsort.

4.11 문자열 처리 함수

strcpy, strncpy, strcat, strncat, strcmp, strncmp, strchr, strcspn,
strpbrk, strrchr, strspn, strstr, strtok, strlen.

4.12 날짜와 시각 함수

time, asctime, ctime+, gmtime+, localetime+, mktime+, strftime+

IEEE Std 1003.1-1988를 수용하는 시스템은 C-표준에서 기술된 “텍스트 스트림”과

“이진 스트림” 간에 구별이 없어야한다.

함수 *fseek*()의 경우에는, 지정된 위치가 파일끝(end-of-file) 다음을 가리킬 경우 함수 *lseek*()(*lseek*() [7.5.3] 참고)에서 기술한 것과 같은 결과가 일어날 것이다.

함수 *exit*()에 의해 사용되는 것과 같이 매크로 EXIT_SUCCESS는 0의 값으로 평가한다.

함수 get*ime*()에 인수로 쓰이는 에폭이후 경과시간(초단위)과 **<time.h>**에서 정의된 tm 구조 간의 관계는 **에폭이후 경과시간**[3.3]의 정의에서 주어지는 표현식에서 명시된 것과 같아야한다. 그 정의에서는 구조에서의 이름들과 표현식에서의 이름들이 대응한다. 만약 시간대 UCTO가 효력이 있다면 이와 같은 관계는 *localetime*()과 *mktime*()에도 옳아야 한다.

9.1.1 시간에 관한 함수의 확장

TZ(**환경변수**[3.7] 참고)라 이름지워진 함수의 내용은 미리 정해진 시간대를 무시하기 위해 *ctime*(), *localetime*(), *strftime*(), *mktime*() 등의 함수들에 의해 사용된다. TZ의 값은 다음의 두 형식 중 하나를 가진다.

:characters

또는

std *offset* dst *offset*, rule

만약 TZ가 첫번째 형식(즉, 첫번째 문자가 콜론(colon))을 가진다면 그 콜론 뒤에 따르는 문자들은 구현시 정의된 방법으로 다루어진다.

확장된 형식(첫번째 문자가 콜론이 아닌 모든 TZ에 해당)은 다음과 같다.

std*offset*[dst[*offset*][,start[/time],end[/time]]]

위 형식에서 각 용어의 의미는 다음과 같다.

*std*와 *dst* 표준(std) 시간대 혹은 여름(dst) 시간대의 지정을 위해 3byte 이상 사용한다. 오직 std만이 요구사항이다. 만약 dst가 없으면 여름 시간은 이 지역(*locale*)에 적용되지 않는다. 대문자, 소문자들이 허용된다. 첫 문자로서 콜론(:)은 제외하며 그 외에는 모든 문자, 숫자, 콤마(,), 빼기(-), 더하기(+), ASCII NUL이 허용된다.

offset 국제협약시간(Coordinated Universal Time:CUT)에 맞추기 위해 지역시간에 더해져야할 값을 나타낸다. *offset*은 다음 형식을 가진다.

hh[:mm[:ss]]

분(mm)과 초(ss)는 선택사항이며, 시(hh)는 필수사항이고 한자리의 숫자여야 한다. std 다음의 *offset*이 필수적이다. 만약 dst 다음에

*offset*이 없으면, 여름 시각은 표준 시각보다 한 시간 앞서는 것으로

정한다. 한자리 이상의 숫자가 사용되는데, 이 값은 항상 십진수로 해석되어진다. 시(hh)는 0 에서 24 사이의 값이어야하며, 분(mm)과 초(ss)는 0에서 59 사이의 값이어야한다. 위의 범위를 벗어나는 값은 예기치 못할 결과를 유발시킬 수 있다. 만약 *offset*앞에 "-"가 있으면,시간대는 본초 자오선(the Prime Meridian)의 동쪽에 있어야하며, "-"가 없으면 (선택인 "+"를 쓴 것과 같음) 서쪽에 있어야한다.

rule 일광 절약 시간으로 언제 바뀌고 언제 다시 정상으로 돌아오는지를 명시한다. rule은 다음 형식을 가진다.

date/time, date/time

여기서 처음의 date는 표준에서 서머타임으로 언제 바뀌는가를 기술하고, 두 번째 date는 언제 다시 원상으로 돌아오는가를 기술한다. 각 time 필드는 언제 다른 시간으로 바뀌는가를 기술하는데, 현재의 지역시간으로 기술한다. date의 포맷은 다음중의 하나 이어야 한다.

Jn 줄리안 날짜(Juilan day) n (1 ≤ n ≤ 365). 윤년일(Leap day)은 계산되지 않는다. 즉 윤년을 포함한, 모든 해에 있어서 2 월 28 일은 59 번째 날이고, 3 월 1 일은 60 번째 날이다. 예외의 2 월 29 일을 명시적으로 언급하는 것은 불가능 하다.

n 0 기준의 줄리안 날짜(zero-based Julian day) (0 ≤ n ≤ 365). 윤년일이 계산되고,2 월 29 일을 언급하는 것이 가능하다.

Mm.n.d 그 해의 m 번째 달의 n번째 주의 d 번째 요일 (0 ≤ d ≤ 6, 1 ≤ n ≤ 5, 1 ≤ m ≤ 12 인데, 다섯째 주(week 5) 는 m 월의 마지막 d요일을 의미한다. 이것은 네번째 혹은 다섯번째 주에 발생함). 첫 주(week 1)는 d 번째 요일이 있는 첫번째 주이다. 요일 0은 일요일이다.

time은 -나 + 기호가 허용되지 않는것을 제외 하고는 *offset*의 포맷과 같다. time이 주어지지 않는다면, 미리 정해진 값 02:00:00 으로 된다.

ctime(), *strftime*(), *mktime*(), *localtime*()이 호출될때마다, 외부 변수인tz*name* 이 포함되어 있는 시간대 이름(time zone *name*s)은 마치 함수 *tzset*()[9.3.2]이 호출된 것처럼 설정된다.

응용들은 TZ을 변경할 수 있도록 명시적으로 허용되어 있고, 변경된 TZ은 자기 자신에게 적용된다.

9.1.2 setlocale() 함수로의 확장

함수 : *setlocale*()

9.1.2.1 용례

#include <locale.h>
char *setlocale (*category,locale*)
int *category* ;
char **locale* ;

9.1.2.2 설명

C-표준은 함수 *setlocale*()에 대해 모국환경(구현시 정의됨)의 명시를 허용하는데, 이것은 특정한 범주를 구현시 정의되는 미리 정해진 값으로 설정하도록 한다. IEEE Std 1003.1-1988 시스템에 있어서, 이것은 환경 변수의 값에 따른다.

locale 인수에 널 스트링의 포인터를 전해줌으로써 특정한 범주는 구현시 정의되는 미리 정해진 값으로 설정된다.

가능한 *category*의 값은 다음과 같다.

LC_CTYPE
LC_COLLATE
LC_TIME
LC_NUMERIC
LC_MONETARY
구현시 정의된 기타 범주

모든 경우에 있어서, *setlocale*()은 먼저 대응하는 환경 변수 값을 체크해야 한다.(예를 들어서 LC_CTYPE 범주의 경우에는 LC_CTYPE). 그리고 유효하다면(즉, 유효한 지역의 이름을 가리킨다면) *setlocale*()은 특정한 범주의 국제환경을 그 값에 설정하고, 지역 설정에 따르는 스트링(즉, 환경변수의 값,""이 아님)을 되돌려 준다. 만약 값이 유효하지 않다면, *setlocale*()은 **NULL** 포인터를 되돌려 주고, 국제환경은 이 함수의 호출에 의해서 변하지 않는다.

만약 특정 범주의 환경 변수가 설정되어 있지 않거나 빈 스트링으로 설정되어 있으면, LANG 환경 변수가 설정되어 있고 유효한 경우가 아닌한 *setlocale*()의 행위는 구현시 정의된다. 만약 LANG 환경 변수가 설정되어 있고 유효하다면 *setlocale*()은 그 범주를 대응하는 LANG의 값으로 설정할 것이다. 어떤 구현에서는 이것을 시스템에 대한 미리 정해진 값으로,또 다른 구현에서는 "C" 지역(*locale*)에 대한 미리 정해진 값으로 한다. 모든 범주를 구현시 정의된 미리 정해진 값으로 설정하는 것은 앞에서 사용한 방법과 비슷하지만, 설정할 특정 값을 결정하기 위해서는 모든 환경 변수에게 물어 보아야 한다. 국제환경에 있는 모든 범주를 설정하기 위해서 *setlocale*()은 다음과 같은 방법으로 호출된다.

setlocale(LC_ALL, "");

이 요청을 만족하기 위해서, *setloc*al()은 먼저 모든 환경 변수를 체크한다. 만약 어

떤 환경 변수라도 유효하지 않다면, *setloc*al()는 널 포인터를 되돌려 주고, 국제환경은 이 함수의 호출에 의해서 변경되지 않는다. 만약 관련된 모든 환경 변수가 유효하다면 , *setlocale*()은 국제 환경을 환경 변수의 값을 반영하도록 설정한다. 범주들은 다음 순서로 결정된다.

LC_CTYPE
LC_COLLATE
LC_TIME
LC_NUMERIC
LC_MONETARY
구현시 정의된 기타 범주

이러한 방법을 이용하여, 환경 변수들에 대응하는 범주들은 특정 범주를 위한 **LANG** 환경 변수의 값을 무시하고 그 위에 쓰게 된다.

LANG 환경 변수가 설정되어 있지 않거나, 빈 스트링으로 설정되어 있는 경우,
setlocale(*category*, "")
의 행위는 구현시 정의된다.

9.2 파일형 C-언어 함수

이 절에서는 C-표준에서 기술한 것과 같이 FILE 형을 참고하는 함수들과 이 표준에서 정의된 다른 함수들과의 상호 작용에 대하여 기술한다.

화일 위치 표시자(file position indicator), 스트림이라는 용어는 C-표준에서 정의된 것이다.

스트림은 단일 프로세스에 대해서 지역적으로 간주된다. *fork*() 호출 후, 각 부모와 자식 프로세스는 개방 파일 서술자를 공유하지만 서로 다른 스트림을 가진다.

9.2.1 스트림 포인터를 파일 서술자로의 매핑

함수 : *fileno*()

9.2.1.1 용례

#include <stdio.h>
int fileno(*stream*)
FILE **stream* ;

9.2.1.2 설명

함수 *fileno*()는 *stream* 과 관련된 정수 파일 서술자를 복귀값으로 한다.(*open*()[6.3.1] 참고).

<unistd.h> [3.10] 에 있는 다음의 심볼들은 응용이 시작될때 C-언어의 stdin, stdout, stderr 와 관련이 있는 파일 서술자들을 정의한다.

이 름	설 명	값
STDIN_FILENO	표준 입력값, stdin	0
STDOUT_FILENO	표준 출력값, stout	1
STDERR_FILENO	표준 에러값, stderr	2

9.2.1.3 복귀값

설명을 참고 하시오. 에러 발생시, -1을 복귀값으로 하며, *errno* 는 에러를 나타내도록 설정된다.

9.2.1.4 에러

이 표준에서는 함수 *fileno*()에서 검출되도록 요구되는 어떤 에러 조건도 명시하지 않는다. 몇가지 에러는 구현시 정의되는 조건들 하에서 검출될 수 있다.

9.2.1.5 참고

open()[6.3.1]

9.2.2 파일 서술자에의 스트림 열기(open)

함수 : *fdopen*()

9.2.2.1 용례

#include <stdio.h>

FILE *fdopen(*fildes*, *type*)
int *fildes*;
char **type* ;

9.2.2.2 설명

fdopen() 루틴은 스트림을 파일 서술자와 연관시켜 준다.
type 인수는 다음중 하나의 값을 갖는 문자 스트링 이다.

"r" 읽기를 위한 열기
"w" 쓰기를 위한 열기
"a" 파일 끝에 쓰기를 위한 열기
"r+" 수정(읽기와 쓰기)을 위한 열기
"w+" 수정(읽기와 쓰기)을 위한 열기
"a+" 파일끝에 수정(읽기와 쓰기)을 위한 열기

이 플래그들의 의미는 "w"와 "w+" 가 파일의 절단을 발생시키지 않는것을 제외하고는, C-표준에서 *fopen*()을 명시한 것과 정확히 같다. *type* 인수를 위한 그 밖의 값들

은 구현시에 정의될 수 있다.

스트림의 *type* 은 열린 파일의 모드에 의해 접근이 허용되어야 한다.

새로운 스트림과 연관된 파일 위치 표시자는 파일 서술자와 연관된 파일 오프셋이 나타내는 위치로 설정된다.

fdopen()은 주어진 파일의 *st_atime*가 갱신되도록 표시되도록 할 수 있다.

9.2.2.3 복귀값

만약 성공적이라면, 함수 *fdopen*()는 스트림의 포인터를 복귀값으로 한다. 성공적이지 않으면 NULL 포인터를 복귀값으로 하며 에러를 나타내도록 *errno*가 설정된다.

9.2.2.4 에러

이 표준에서는 함수 *fdopen*()에서 검출되도록 요구되는 어떤 에러 조건도 명시하지 않는다. 몇가지 에러는 구현시 정의되는 조건들 하에서 검출 될 수 있다.

9.2.2.5 참고

open() [6.3.1], *fdopen*() (C-표준)

9.2.3 다른 파일형 C-함수와의 상호작용

하나의 열린 파일 서술은 스트림과 파일 서술자 두개 모두를 통해서 접근 될 수 있다. 각 파일 서술자나 스트림은 참고하는 열린 파일 서술에게는 **핸들(handle)** 이라고 불리운다. 하나의 열린 파일 서술은 여러개의 핸들을 가질 수 있다.

핸들은, 주어진 열린 파일 서술에 영향을 미치지 않고, 사용자가 생성하거나 파괴할 수 있다. *fcntl*(), *dup*(), *fdopen*(), *fileno*(), 및*fork*() (존재하는 프로세스를 새로운 프로세스에 복제)등을 통해서 핸들들을 생성할 수 있다. 그리고 최소한 *fclose*(), *close*(), *exec*() 함수들(이 함수들은 파일 서술자를 닫고 스트림들을 파괴함) 등을 통해서 핸들을 파괴할 수 있다.

read(), *write*(), 또는 *lseek*()와 같이 파일 오프셋에 영향을 주는 연산에 사용되지 않는 파일 서술자는 여기에서는 핸들(handle)로 생각하지 않지만, *fdopen*(), *dup*(), *fork*()와 같은 연간의 결과는 영향을 미칠 수 있다. *fopen*()나 *fdopen*()에 의해서 생성되던지 상관없이 파일 오프셋에 영향을 미치는 응용에 의해 직접적으로 사용되지 않는 한 이러한 예외는 스트림 하의 파일 서술자도 포함된다. (*read*()와 *write*()는 파일 오프셋에 간접적으로 영향을 미치고 *lseek*()는 직접적으로 영향을 미친다.)

하나의 핸들(활동성 핸들)을 갖는 함수 호출결과는 이 표준의 다른곳에서 정의되지만, 둘 이상의 핸들이 사용되고 그들중 하나가 스트림이라면 그들의 동작은 다음에 기술한바와 같이 조정되어야 한다. 만약 이러한 조정이 이루어지지 않으면, 결과는 정의될 수 없다.

스트림인 핸들의 경우에는 *fclose*() 또는 *freopen*()(*freopen*()의 결과는 이 논의에서는 새로운 스트림이며, 이전의 값과 같은 열린 파일 서술에 대한 핸들이 될 수 없다.)이 수행될때, 또는 그 스트림을 소유하는 프로세스가 *exit*() 또는 *abort*()로 끝날 때 핸들은 닫힌것으로 간주된다. 파일 서술자는 *close*(), *_exit*()에 의해 닫히며 FD_CLOEXEC가

그 파일 서술자에 설정되어 있으면 *exec* 함수중 하나에 의해 닫혀진다.

핸들이 활동성 핸들이 되기 위해서는 다음의 동작들이 현재의 활동성 핸들인 첫번째 핸들의 마지막에서 두번째 사용과 미래의 활동성 핸들인 두번째 핸들의 두번째 사용 사이에서 수행되어져야 한다. 이렇게 되면 두번째 핸들이 활동성 핸들이 된다. 첫번째 핸들에 대한 파일 오프셋에 영향을 주는 응용에 의한 모든 행동은 그 파일이 다시 활동성 핸들이 될때까지 일시정지되어야 한다.(만약 스트림 함수가 파일 오프셋에 영향을 주는 기저 함수를 가지고 있으면, 그 스트림 함수는 파일 오프셋에 영향을 미치는 것으로 간주될 것이다. 기저 함수들은 아래에 서술된다.)

이러한 규칙들이 적용되기 위해서 핸들들이 같은 프로세스에 있을 필요는 없다.

(1) 첫번째 핸들의 경우에 아래에 적은 첫번째 적용가능조건이 적용되어야 한다. 아래의 필요한 행동이 취해진후에도 핸들이 열림으로 되어 있다면 그 핸들은 닫힘으로 된다.
   (a) 핸들이 파일 서술자라면 다른 행동이 불필요하다.
   (b) 만약 열림상태의 이 파일 서술에 대한 모든 핸들에 수행되어야 할 행동이 그것을 닫힘으로 하는것 뿐이라면 어떤 행동도 취해질 필요가 없다.
   (c) 만약 핸들이 버퍼되어 있지 않은 스트림이라면, 어떤 행동도 취해질 필요가 없다.
   (d) 만약 핸들이 라인별로 버퍼된 스트림이고 마지막 연산이 기저함수에 대해 *fgets*()와 *fputs*()와 같은 효과를 준다면, 특별한 행동이 취해질 필요가 없다. *fgets*()의 경우에 위의 효과는 마치 구현에서 선택한 미리 읽기(readahead)가 결코 일어나지 않은 것처럼 해석된다.
   (e) 만약 핸들이 읽기가 아닌 쓰기 또는 추가를 위해 열려있는 스트림이라면 *fflush*()를 행하거나 스트림이 닫혀진다.
   (f) 만약 스트림이 읽기를 의해 열리고 그것이 파일의 끝에 위치해 있다면 (즉, *feof*()가 참이면), 행동이 취해질 필요가 없다.
   (g) 만약 스트림이 읽기를 허용하는 상태로 열리고, 해당되는 열린 파일 서술이 탐색 가능한 장치를 참고한다면 *fflush*()가 행해지거나 그 스트림이 폐쇄될 것이다.
   (h) 그 밖의 경우에는 그 결과는 정의되어 있지 않다.

(2) 두번째 핸들의 경우: 위의 첫번째 핸들을 위해 필요한 함수를 부른것을 제외하고는 어느 이전의 활동성 핸들이 명시적으로 파일 오프셋을 변화시키는 함수를 호출하였다면 그 응용은 적절한 위치에다 핸들의 형(*type*)에 따라 *lseek*() 또는 *fseek*()을 행해야한다.

(3) 만약 위의 첫번째 핸들에 대한 요구 사항이 만족되기전 활동성 핸들이 접근 가능상태를 끝낸다면, 열림 파일 서술의 상태는 정의 되어지지 않은 상태가 된다. 예를 들면, 이러한 상황은 *fork*() 또는 *_exit*() 동안 발생할 수 있다.

(4) *exec* 함수는 어떤 스트림이나 파일 서술자가 새로운 프로세스 이미지(image)에 대해 이용가능할 것인가와는 상관없이 함수가 호출되어지는 시점에서 열려 있는 모든 스트림에의 접근을 허용하지 않는다.

(5) 구현은 하나의 응용이 여러 개의 프로세스로 이루어져 있다 하더라도 위의 규

칙을 따랐을때에는 핸들의 사용 순서와 상관없이 올바른 결과를 만들어 내야 하는것을 확신시켜야 한다. (올바른 결과란 자료의 분실이 없어야하며, 쓰기를 했을때 중복이 없고, 탐색에 의해 요청된것을 제외하고는 모든 자료가 순서에 따라 쓰여져야 하며, 순차적으로 읽기를 할때 모든 자료를 읽을 수 있어야 한다는 것을 말한다.) 만약 위의 규칙들을 따르지 않았을 경우에는, 그 결과는 명시될 수 없다.

(6) 스트림에 대해 연산을 하는 함수는 0개 이상의 **기저 함수(underlying function)**를 갖는다고 말한다. 이것은 스트림 함수는 기저 함수와 어떤 특성을 공유하는것을 의미하나 그 함수들의 구현 사이에 어떤 관련이 있는것을 요구하지는 않는다.

(7) 또한 다음 절에서는 표준 입/출력 루틴에 대해 표준 C에서는 요구하지 않는 추가적인 요구들에 대해 서술하고 있다.

9.2.3.1 fopen()

fopen()을 *open*()과 마찬가지로 파일 서술자를 할당한다. 또한 *fopen*()은 기저 파일의 *st_atime* 필드는 갱신되도록 표시되어야 한다.

기저 함수는 *open*()이다.

9.2.3.2 fclose()

fclose()는 FILE 스트림과 관련있는 파일 서술자에 대해 *close*()를 행한다. 이 또한 스트림이 쓰기 가능이고, 버퍼된 자료가 파일에 아직 쓰여지지 않았다면 기저 파일의 *st_ctime* 필드와 *st_mtime* 필드는 갱신되도록 표시되어야 한다.

기저 함수는 *write*()와 *close*()이다.

만약 파일이 그 끝을 만난 상태가 아니고 탐색(seeking)이 가능한 파일이라면, 열린 기저 파일 서술의 파일 오프셋은 열린 파일서술에 대한 다음 연산이 닫혀지는 스트림으로부터 마지막으로 읽거나 또는 스트림에 마지막 쓰기를 한 바로 다음 바이트를 다룰 수 있도록 조정되어져야 한다.

9.2.3.3 freopen()

freopen()은 *fclose*()와 *fopen*() 두개 모두의 특성을 가지고 있다.

9.2.3.4 fflush()

fflush()는 스트림이 쓰기가능하고, 버퍼된 자료가 파일에 아직 쓰여지지 않았다면, 해당 파일의 *st_ctime* 필드와 *st_ctime* 필드는 갱신되도록 표시되어야 한다.

기저 함수는 *read*(), *write*(), *lseek*()이다.

스트림이 읽기를 위해 열려 있다면 스트림에 버퍼된 데이타중 아직 읽히지 않은 데이타는 무효화된다.

읽기위해 열린 스트림에 있어서, 파일이 그 끝에 도달한 상태가 아니고, 탐색 가능하다면, 해당되는 열린 파일 서술의 파일 오프셋은 열린 파일 서술에 대한 다음 연산이 닫혀지는 스트림으로부터 마지막으로 읽거나 또는 스트림에 마지막 쓰기를 한 바

로 다음 바이트를 다룰 수 있도록 조정되어져야 한다.

9.2.3.5 *fgetc*(), *fgets*(), *fread*(), *getc*(), *getchar*(), *gets*(), *scanf*(), *fscanf*()

이 함수들은 *st_atime* 필드를 갱신되도록 표시할 수 있다. 우선적인 *ungetc*()의 호출에 의해 제공되지 않은 데이타를 복귀값으로 하는 이런 함수들중 하나의 첫번째 성공적인 수행에 의해 *st_atime* 필드를 갱신되도록 표시할 수 있다.

기저함수로는 *read*()와 *lseek*()가 있다.

9.2.3.6 *fputc*(), *fputs*(), *fwrite*(), *putc*(), *putchar*(), *puts*(), *printf*(), *vprintf*(), *vfprintf*()

이러한 함수들중 하나의 성공적인 수행과 같은 스트림에 대한 *fflush*()나 *fclose*()의 호출 또는 *exit*()나 *abort*()의 호출의 성공적인 끝남 사이에 파일의 *st_ctime*과 *st_mtime* 필드가 갱신되도록 표시되어야 한다.

기저함수는 *write*()와 *lseek*()이다.

9.2.3.7 *fseek*(), *rewind*()

이들 함수들은 스트림이 쓰기 가능이고, 버퍼된 자료가 파일에 아직 쓰여지지 않았다면, 파일의 *st_ctime* 필드와 *st_mtime* 필드가 갱신되도록 표시되어야 한다.

기저 함수는 *lseek*()와 *write*()이다.

만약 주어진 스트림에 대해 *ftell*()을 제외하고 최근에 수행한 연산이 *fflush*()이면, 해당되는 열린 파일 서술 파일 오프셋은 *fseek*()에 의해 명시된 위치를 고려하도록 조정되어져야 한다.

9.2.3.8 *perror*()

이 함수가 성공적으로 끝났던 시간과 stderr에 대한 *fflush*()나 *fclose*()의 끝냄 시간 또는 *exit*()나 *abort*()의 끝냄시간 사이에 쓰기가 행해졌던 것으로(*st_ctime*과 *st_mtime*는 갱신되도록 표시되어 있음) 여기고 표준 에러 스트림과 관련된 파일을 마크한다.

9.2.3.9 *tmpfile*()

tmpfile()은 *fopen*() 처럼 파일 서술자를 할당한다.

9.2.3.10 *ftell*()

기저 함수는 *fseek*()이며, *fflush*()후의 *ftell*()의 결과는 *fflush*()전의 결과와 같다.

9.2.3.11 에러 보고

만약 위의 어떤 함수도 위에서 기술한 상응하는 기저 함수에 의해 검출되는 조건에 의해 야기된 에러 표시를 되돌려 준다면, *errno*에 되돌려준 값이 기저함수에 의해 제공된 것이다.

9.2.3.12 *exit*(), *abort*()

exit() 함수는 위에 기술한 *fclose*()의 특성을 가지며 모든 열린 스트림에 대해 *fclose*()와 똑같은 영향을 갖는다. *abort*()에 대한 호출이 프로그램의 종료를 야기한다면, *abort*() 함수 또한 같은 효과를 가진다. (프로그램의 끝남이 일어나지 않는 조건에 대해서는 c-표준을 참고하라.)

9.2.4 파일에 대한 연산 - 함수 *remove*()

remove() 함수는 파일 시간에 대해 *unlink*()와 같은 효과를 가진다.

9.3 기타 C-언어 함수

9.3.1 지역을 벗어나는 점프

함수 : *sigsetjmp*(), *siglongjmp*()

9.3.1.1 용례

#include <setjmp. h>

int sigsetjmp(*env*, savemask)
sigjmp_buf env;
int savemask:

void siglongjmp(*env*, *val*)
sigjmp_buf env;
int val;

9.3.1.2 설명

sigsetjmp() 매크로는 c-표준의 *setjmp*() 매크로의 정의를 따른다. savemask 인수의 값이 0이 아니면, 함수 *sigsetjmp*()는 호출 환경의 일부로써 프로세스의 현재 신호 마스크(**<signal.h>**[4.3.1] 참고)를 저장한다.

siglongjmp() 함수는 c-표준의 *longjmp*()의 정의를 따른다. *env* 인수가 0이 아닌 savemask 인수를 갖고 있는 *sigsetjmp*() 함수에의 호출에 의해 초기화될때에만 *siglongjmp*() 함수는 저장한 신호 마스크를 복원한다.

9.3.1.3 참고

sigaction() [4.3.4], **<signal.h>** [4.3.1], *sigprocmask*() [4.3.5], *sigsuspend*() [4.3.7]

9.3.2 시간대 설정

함수 : *tzset*()

9.3.2.1 용례

#include <time.h>
void tzset()

9.3.2.2 설명

tzset() 함수는 *localtime*(), *ctime*(), *strftime*(), *mktime*()에 의해 사용되는 시간 변환 정보를 설정하기 위해서 환경 변수 TZ의 값을 이용한다. 만약 TZ가 환경에 없다면 구현시 정의되는 미리 정해진 시간대 정보가 이용된다.

tzet() 함수는 외부 변수 tz*name*을 지정한다.

extern *char* *tz*name*[2] = {"std", "dst"}:
여기서 std와 dst는 **시간 함수에의 확장** [9.1.1]에 설명되어 있다.

제10장 시스템 데이타베이스

10.1 시스템 데이타베이스

이 절에서 설명된 함수들은 응용으로 하여금 아래에 설명된 2개의 시스템 데이타베이스를 접근할 수 있도록 허용한다.

그룹 데이타베이스는 각 그룹에 대해서 다음의 정보를 갖고 있다.
(1) 그룹명
(2) 숫자로된 그룹 ID
(3) 그룹내에서 접근 허용된 모든 사용자들의 이름 또는 숫자들의 리스트

사용자 데이타베이스는 각 사용자에 대해서 다음의 정보를 갖고 있다.
(1) 등록명(login *name*)
(2) 숫자로된 사용자 ID
(3) 숫자로된 그룹 ID
(4) 초기 작업 디렉토리
(5) 초기 사용자 프로그램

만약 초기 사용자 프로그램 필드가 널(null)이면, 시스템 디폴트(system default)가 사용된다.
만약 초기 작업 디렉토리 필드가 널(null)이면, 그 필드에 대한 해석은 구현시 정의된다.
이 데이타베이스들은 구현시 정의되는 다른 필드들을 포함할 수 있다.

10.2 데이타베이스에의 접근

10.2.1 그룹 데이타베이스에의 접근

함수 : *getgrgid*(), *getgrnam*()

10.2.1.1 용례

#include <grp.h>
struct group *getgrgid(gid)
gid_t gid;
struct group *getgrnam(*name*)
char **name*;

10.2.1.2 설명

getgrgid()와 *getgrnam*()루틴들은 그룹 데이타베이스로부터 gid 또는 *name*에 일치하는 엔트리(entry)를 포함하는 struct *group*형의 객체에 대한 포인터들을 복귀시킨다. **<grp.h>**에서 정의되는 이러한 구조는 표10-1에서 보여주는 구성원들을 포함한다.

<표 10-1> **group 구조**

구성원 형	구성원 이름	설 명
*char **	*gr_name*	그룹명
gid_t	*gr_gid*	숫자로 된 그룹 ID
*char ***	*gr_mem*	각 구성원 이름들에 대한 포인터벡터(null로 끝난다)

10.2.1.3 복귀값

에러나 요청된 엔트리를 발견하지 못했을 때에는 NULL 포인터를 복귀값으로 한다.이 복귀값들은 정적 데이타를 가리키는데 이 정적 데이타는 각 호출마다 겹쳐 쓰여진다.

10.2.1.4 에러

이 표준은 *getgrgid*() 또는 *getgrnam*() 함수들에 대해 검출되도록 요구되는 어떤 에러 조건도 명시하지 않는다. 어떤 에러들은 구현시 정의되는 조건들 하에서 검출될 수 있다.

10.2.1.5 참고

getlogin() [5.2.4]

10.2.2 사용자 데이타베이스에의 접근

함수 : *getpwuid*(), *getpwnam*()

10.2.2.1 용례

#include <pwd.h>
struct passwd *getpwuid(*uid*)
uid_t *uid*;
struct passwd *getpwnam(*name*)
char **name*;

10.2.2.2 설명

함수 *getpwuid*()와 *getpwnam*()들은 사용자 데이타베이스로부터 *uid* 또는 *name*와 일치하는 엔트리를 포함하는 struct passwd형의 객체에 대한 포인터를 복귀값으로 한다.

<pwd.h>에서 정의되는 이러한 구조는 표 10-2에서 보여주는 구성원들을 포함한다.

<표10-2> **passwd 구조**

구성원 형	구성원 이름	설 명
*char **	*pw_name*	사용자의 로그인 이름
uid_t	*pw_uid*	사용자의 ID 번호
gid_t	*pw_gid*	그룹 ID 번호
*char **	*pw_dir*	초기 작업 디렉토리
*char **	*pw_shell*	초기 사용자 프로그램

cuserid() [5.2.4] 함수의 구현은 *getpwnam*()함수를 이용한다. 따라서 어떤 함수에 대한 사용자 호출의 결과는 다른 함수에 대한 뒤이은 호출에 의해서 겹쳐쓰일 수 있다.

10.2.2.3 복귀값

에러나 요구된 엔트리를 찾지 못했을 경우에는 NULL 포인터를 복귀값으로 한다. 이 복귀값들은 정적 데이타를 가리키는데 이 정적 데이타는 각 호출마다 겹쳐 쓰여진다.

10.2.2.4 에러

이 표준은 *getpwgid*() 또는 *getpwnam*() 함수들에 대해 검출되도록 요구되는 어떤 에러 조건도 명시하지 않는다. 어떤 에러들은 구현시 정의되는 조건들하에서 검출될 수 있다.

10.2.2.5 참고

cuserid() [5.2.4], *getlogin*() [5.2.4]

제11장 데이타의 상호 교환 포맷

11.1 기록 보존/상호 교환 파일 포맷

적합한 시스템은 여기서 기술하는 교환 포맷들을 사용하여 한 매체로부터 파일의 계층구조로 파일을 복사하는 방법과 파일의 계층구조로부터 한 매체로 파일을 복사하는 방법을 제공해야 한다. 이 표준은 이러한 방법을 정의하지 않는다.[1)]

적절한 권한이 없는 프로세스에 의해서 매체로부터 파일을 복사하는데 이 방법이 사용된다면, 확장된 tar 포맷의 모드 필드나 확장된 cpio 포맷의 c_*mode* 필드에 의해 제공되는 파일 접근 허용에 일치하는 모드 수가 주어졌을 때, *creat*()[6.3.2]이 하는 것과 같은 방식으로 소유권이나 접근 허용같은 보호 정보가 설정된다.적절한 권한을 갖는 프로세스는 심볼 사용자와 그룹 ID들이 tar 포맷을 위해 사용될 때를 제외하고는 **확장된 tar 포맷**[11.1.1]에서 기술된 것처럼 매체상에 기록된 것과 똑같이 소유권과 접근 허용을 재저장한다.

포맷 생성 유틸리티는 파일 시스템을 이 절에서 정의된 포맷으로 구현시 정의되는 방법에 의해 번역하는데 사용된다. 그리고 포맷 읽기 유틸리티는 이 절에서 정의된 포맷으로부터 파일 시스템으로 번역하는데 사용된다.

이 포맷들의 헤더들은 ASCII 코드에서 표현된 문자들을 사용하도록 정의된다. 하지만 파일 자체의 내용에 대해서는 아무런 제약도 없다. 파일에서의 데이타는 이진수 데이타일 수 있고 ,사용자에게 가능한 어떤한 포맷으로 표현된 텍스트일 수 있다.소스 단계에서 텍스트를 이동시키는데 이 포맷들이 사용될 때에는 모든 문자들은 ASCII 코드로 표현되어야 한다.

11.1.1 확장된 tar 포맷

확장된 tar 저장 테이프나 파일은 일련의 블럭들을 포함하고 있다. 각 블럭은 512 바이트의 블럭 크기로 고정된다. 비록 이 포맷들이 9 트랙의 산업계 표준인 1/2 인치 자기 테이프에 저장되는 것으로 간주된다고 하더라도, 운반 가능한 다른 종류의 매체들도 배제되지 않는다. 저장된 각 파일은 파일을 기술하는 헤더 블록 뒤에 그 파일의 내용을 갖는 0 개 이상의 블럭으로 나타낸다. 저장 파일의 맨 끝에는 기록보존 끝(end-of-archive) 식별자로 해석되는 이진수 형태의 0 값으로 채워진 두 블럭이 있다.

블럭은 실제의 I/O 연산을 위해 그룹화될 수 있다. n 개의 블럭으로 이루어진 각 그룹(n은 저장파일을 생성하는 응용 유틸리티에 의해 설정됨)은 하나의 *write*() 연산으로 쓸 수 있다. 자기 테이프상에서, 이 쓰기의 결과는 한 테이프 레코드이다. 블럭의 마지막 그룹은 항상 정규 크기이며, 0 값만을 갖는 두 개의 블럭뒤의 블럭들은 정의되지 않은 데이타를 갖는다.

헤더 블럭은 표 11-1에서 보는 바와 같은 구조로 되어 있다. 모든 길이와 오프셋은 십진수 값이다.

---

1) 1003.2 실무 그룹이 이 방법에 관하여 작업중이다.

<표 11-1>　　tar 헤더 블럭

필드명	바이트 오프셋	길이(바이트로표현)
name	0	100
mode	100	8
uid	108	8
gid	116	8
size	124	12
mtime	136	12
chksum	148	8
typeflag	156	1
linkname	157	100
magic	257	6
version	263	2
uname	265	32
gname	297	32
devmajor	329	8
devminor	337	8
prefix	345	155

이 헤더 블럭에서 사용되는 심볼 상수들은 다음과 같이 <tar.h> 헤더에서 정의된다.

```
#define TMAGIC          "ustar"     /*ustar와 null*/
#define TMAGLEN         6
#define TVERSION        "00"        /*00와 no null */
#define TVERSLEN        2

/* typeflag 필드에서 사용되는 값 */
#define REGTYPE         '0'         /* 정규 파일     */
#define AREGTYPE        '/0'        /* 정규 파일     */
#define LNKTYPE         '1'         /* 링크          */
#define SYMTYPE         '2'         /* 예비용        */
#define CHRTYPE         '3'         /* 문자형 특수   */
#define BLKTYPE         '4'         /* 블럭형 특수   */
#define DIRTYPE         '5'         /* 디렉토리      */
#define FIFOTYPE        '6'         /* FIFO형 특수   */
#define CONTTYPE        '7'         /* 예비용        */

/* 모드 필드에 사용되는 비트- 8진수 값 */
#define TSUID           04000       /* 실행시 UID 설정 */
#define TSUID           02000       /* 실행시 GID 설정 */
#define TSVTX           01000       /* 예비용 */
                                    /* 파일 접근 허용 */
```

```
#define TUREAD      00400       /* 소유자에 의한 읽기 */
#define TUWRITE     00200       /* 소유자에 의한 쓰기 */
#define TUEXEC      00100       /* 소유자에 의한 실행/검색*/
#define TGREAD      00040       /* 그룹에 의한 읽기 */
#define TGWRITE     00020       /* 그룹에 의한 쓰기 */
#define TGEXEC      00010       /* 그룹에 의한 실행/탐색 */
#define TOREAD      00004       /* 기타에 의한 읽기 */
#define TOWRITE     00002       /* 기타에 의한 쓰기 */
#define TOEXEC      00001       /* 기타에 의한 실행/검색 */
```

모든 문자들은 ASCII(American Standard Code for Inform*ation* Interchange) 코드로 기술된다. 구현사이의 최대의 이식성을 위해서는 이름은 **POSIX 파일명용 문자집합** [3.3]에 표현된 문자들로부터 선택해야 하며 이 문자들은 패리티가 없는 8 비트 문자들이다. 만약 POSIX 파일명용 문자 집합 이외의 확장된 문자집합이 사용되고 두 개의 다른 시스템에 대한 포맷 읽기 유틸리티와 포맷 생성 유틸리티가 같은 확장된 문자집합을 사용한다면, 파일명이 유지된다. 그러나 포맷 읽기 유틸리티는 이 표준에서 이전에 기술된 함수를 통해 접근할 수 없는 지역 시스템에 대한 파일명은 결코 생성할 수 없다.(*open*()[6.3.1], *stat*()[6.6.2], *chdir*()[6.2.1], *fcntl*()[7.5.2], *opendir*()[6.1.2] 참고). 잘못된 파일명을 만드는 매체에서 파일명이 발견되면, 파일로부터의 데이타가 파일의 계층구조상에서 저장되는지와 저장된다면 어떤 이름하에 저장되는지를 구현시 정의하게 된다. 포맷 읽기 유틸리티는 파일이 무시됨을 표시하는 에러를 만드는 한 이들 파일을 무시할 것이다.

헤더 블럭내의 각 필드는 연속적이다. 즉, 문자채움(padding)이 사용되지 않는다. 저장 매체상에서 각 문자들은 연속적으로 저장된다.

magic, u*name*, g*name*등과 같은 필드는 null로 끝나는 문자열이다. 배열내에 있는 문자들이 마지막 문자를 포함하여 모두 null이 아닌 문자를 포함하고 있는 경우를 제외하고는 *name*, link*name*, prefix등과 같은 필드는 null로 끝나는 문자열이다. version필드는 문자 “00”를 포함한 두 바이트이다. *type*flag는 한 문자를 갖는다. 모든 다른 필드들은 0으로 시작하는 8진수 ASCII 문자를 갖는다. 각 수치 필드는 하나 이상의 공란이나 null문자로 끝난다.

*name*과 prefix 필드는 파일의 경로명을 나타낸다. 파일의 계층적 관계는 경로 접두어로서 경로명을, 접미어로 /(사선)과 파일명을 명시함으로써 유지된다. 만약prefix가 null이 아닌 문자를 포함하면, prefix, 사선, *name*들은 수정이나 새로운 문자의 추가 없이 끝에 덧붙여져 새로운 경로명을 만든다. 이 방식으로 최대 256 문자들의 경로명이 지원될 수 있다. 만약 경로명이 지원되는 공간에 적합하지 않으면, 포맷 생성 유틸리티는 사용자에게 알려주며 파일, 헤더, 데이타의 일부분을 매체에 저장하려는 어떠한 시도도 포맷 생성 유틸리티에 의해 만들어지지 않는다.

다음에서 기술되는 link*name* 필드는 경로명을 만들어내는데 prefix를 사용하지 않는다. 하나의 link*name*은 99 문자로 제한된다. 만약 그 이름이 제공되는 공간에 적합하지 않으면, 포맷 생성 유틸리티는 사용자에게 에러를 알려주며, 유틸리티는 그 매체상에 link를 저장하는 시도를 하지 않는다.

mode 필드는 파일의 접근허용을 명시하는 9 비트와 UID 설정, GID 설정과 VTX 모드를 명시하는 3 비트를 제공한다. 이 비트들의 값은 이미 정의되었다.이들 모드 비트들의 하나를 설정하는데 적절한 권한이 요구되고, 저장소로부터 파일을 복원하려는 사용자가 적절한 권한을 가지고 있지 못할 때, 사용자가 적절한 권한을 가지고 있지 못한 모드 비트들은 무시된다. 저장 포맷에 있어서의 몇가지 모드 비트들은 이 표준의 다른 곳에서는 언급되지 않는다. 만약 구현이 이 비트들을 지원하지 않으면 그들은 무시된다.

uid 필드와 *gid* 필드는 각각 파일의 사용자와 그룹의 사용자 ID와 그룹 ID이다.

size 필드는 바이트 단위의 파일 크기이다. 만약 *type*flag 필드가 파일이 LNKTYPE 형이나 SYMTYPE 형이라는 것을 명시하기위해 설정되어 있다면, *size* 필드는 0으로 명시되어야만 한다. *type*flag 필드가 파일이 DIRTYPE 형인 것을 명시하기위해 설정되어 있으면, *size* 필드는 그 레코드 형의 정의하에서 기술된 것으로 해석된다. *type*flag 필드가 CHARTYPE, BLKTYPE, FIFOTYPE 등으로 설정되어 있으면, *size* 필드의 의미는 구현시 정의되며, 어느 데이타 블럭도 그 매체에 저장되지 않는다. *type*flag 필드가 어느 다른 값으로 설정되어 있으면, 헤더 다음에 기록된 블럭의 수는 나눗셈의 소수부분은 무시하는 (*size*=511)/512이 된다.

mtime 필드는 파일이 저장되었을 때의 파일의 수정 시간이다. 그것은 *stat*() 함수로부터 취한 수정시간의 8 진수 값으로 ASCII로 표현된다.

chksum 필드는 헤더블럭내의 모든 바이트들의 단순합의 8 진수 값을 ASCII로 표현한 형태이다. 이 헤더에서 각각의 바이트(8비트)는 부호없는 값으로 취급된다. 이 값들은 초기에 0이었던 부호없는 정수에 더해지는데, 이 정수는 표시 단위가 17 비트 미만이어서는 안된다. chksum을 계산할 때, checksum 필드는 그것이 모두 공란인 것으로 취급된다.

*type*flag 필드는 기록 보존 파일의 형을 명시한다. 만약 특정 구현이 그 형을 인식하지 못하거나, 사용자가 그 형을 만들 적절한 권한을 가지고 있지 못한 경우에 만일 파일 형이 블럭이 데이타 매체에 쓰여지도록 하는 *size* 필드의 의미를 갖도록 정의되어 있다면 그 파일은 정규 파일인 것처럼 추출된다. (*size*에 대한 이전의 서술을 참고하라) 만약 보통 파일로에의 변환이 일어나면, 포맷 읽기 유틸리티는 변환이 일어났음을 알리는 에러를 발생시킨다.

ASCII 숫자 '0' : 정규 파일을 나타낸다. 후진(backward) 호환성을 위해서는 이진수 0('\0')의 *typeflag* 값은 기록보관소에서 파일을 꺼낼때 정규 파일을 의미하는 것으로 인식되어야 한다. 저장파일 포맷이 버전으로 기록된 저장들은 *typeflag* 값으로 ASCII '0' 값을 갖는 정규 파일을 생성한다.

ASCII 숫자 '1' : 어떤 파일이 형에는 상관없이 이미 저장되어 있는 다른 파일에 링크되어 있는 것을 의미한다. 그러한 파일들은 같은 디바이스와 파일 일련번호를 갖는 것으로 각 파일에 의해 인식된다. 링크되어 있는 곳에 대한 이름은 null로 끝나는 *linkname* 필드에서 명시된다.

ASCII 숫자 '2' 는 디바이스나 파일의 일련 번호가 다른 파일(형은 아무형이나 됨)에 대한 링크(link)를 표현하도록 예약되어 있다.

ASCII 숫자 '3'과'4' : 각각 문자형 특수 파일과 블럭형 특수 파일들을 나타낸다. 이 경우에, *devmajor*와 *devminor* 필드는 구현시 정의되는 디바이스를 정의하는 정보를 가진다. 구현은 디바이스 명세들을 그들 자신의 지역 명세로 변환하거나 엔트리(entry)를 무시할 수 있다.

ASCII 숫자 '5' : 디렉토리나 서브_디렉토리(sub-directory)를 명시한다. 디스크 할당이 디렉토리 단위로 행해지는 시스템에서는 *size* 필드는 디렉토리가 가질수 있는 최대 바이트 수를 포함한다. 이 값은 가장 가까운 디스크 블럭할당 단위로 반올림될 수 있다. *size* 필드가 0 이라면, 이는 그러한 한계을 가지지 않는 것을 나타낸다. 이러한 방식으로 한계를 제공하지않는 시스템은 *size* 필드를 무시해야 한다.

ASCII 숫자 '6' : FIFO형 특수 파일을 명시한다. FIFO 파일을 기록 보존(archiving)하는 것은 이 파일의 존재여부를 기록하는 것이지 그것의 내용을 기록하는 것은 아니다.

ASCII 숫자 '7' : 구현시 어떠한 고성능 속성을 부여하는 파일을 표현하도록 예약되었다. 그러한 확장이 없는 구현은 이 파일을 정규 파일('0'형)으로 취급해야 한다.

ASCII 문자 'A'부터'Z' :필요에 따라 구현할 수 있도록 예약되었다. 모든 다른 값은 미래에 표준의 개정에 있어서의 명시를 위해 예약되었다.

magic 필드는 이 기록 보존 파일이 기록 보존 파일 포맷으로 출력되어졌다는 것을 나타낸다. 만약 이 필드가 TMGIC을 포함한다면, u*name*와 g*name* 필드는 각각 그 파일에대한 소유자와 그룹의 ASCII 표현을 포함해야 한다(필요시 맞게 하기위해 잘라 낼 수 있다). 이 파일이 특별 권한과 보호 유지 기능을 갖는 유틸리티 버전에 의해 복구될 때, 패스워드 파일과 그룹 파일이 이러한 이름들을 위해 스캔되어진다. 발견된다면 이 파일들에 포함되어 있는 사용자 ID와 그룹 ID가 *uid*와 *gid* 필드에 포함되어 있는 값들보다 자주 사용된다. 헤더의 암호화는 다른 기계간에 이식성이 있도록 설계된다.

11.1.1.1 참고

<graph> [10.2.1], **<pwd.h>** [10.22], **<sys/stat.h>** [6.6.1], *stat*() [6.6.2], **<unistd.h>** [3.10]

11.1.2 확장된 cpio 포맷

바이트 단위로 행해지는 cpio 기록 보존 파일 포맷은 일련의 엔트리인데 각각의 엔트리는 파일을 기술하는 헤더, 파일명, 파일의 내용으로 구성된다.

기록 보존 파일은 바이트 단위로 된 고정된 크기의 블럭들로 기록되어진다. 이 블럭화는 물리적인 입출력을 보다 더 효율적으로 하기 위해서만 사용되어진다. 블럭의 마지막 그룹은 항상 가득 차 있다.

바이트 단위로 된 cpio 기록 보존 파일 포맷을 위해서는 각각의 엔트리 정보는 표 11-2에 의해 나타내는 순서로 되어있어야 한다.

<표 11-2 > 바이트 단위 cpio기록 보존 파일 엔트리

필드명	헤더 길이(바이트단위)	해석
c_magic	6	8진수
c_dev	6	8진수
c_ino	6	8진수
c_mode	6	8진수
c_uid	6	8진수
c_gid	6	8진수
c_nlink	6	8진수
c_mtime	11	8진수
c_namesize	6	8진수
c_filesize	11	8진수

파일명	파일명 길 이	해 석
c_name	*c_namesize*	경로명스트링

파일명	파일자료 길 이	해 석
c_filedata	*c_fileseze*	자 료

11.1.2.1 헤더

기록 보존 파일에 있는 각 파일에 대해 전에 정의된 것과 같은 헤더가 쓰여져야한다. 헤더 필드에 있는 정보는 8진수로 해석되어지는 ASCII 문자의 스트링으로 쓰여진다. 8진수는 ASCII '0'을 그 숫자의 가장 중요자리(MSB)끝에 추가함으로써 필요한 길이로 확장되어진다. 결과는 가장 중요자리부터 시작해 바이트 스트림으로 쓰여진다. 그 필드들은 다음과 같이 해석되어진다.

(1) c_magic은 MAGIC(070707)에 의해 정의된 매직 바이트(magic byte)를 포함함으로써 기록 보존 파일을 이동가능한 기록 보존 파일로 인식한다.

(2) *c_dev*와 c_ino는 기록 보존 파일 안에서 파일을 유일하게 나타내는 값을 포함한다. (즉 *c_dev*와 *c_ino* 값들이 같은 파일에 대한 링크가 아닌한 어떤 파일도 같은 값의 *c_dev*, *c_ino* 쌍을 가질수 없다.) 이 값은 구현시 정의된 방식에 따라 결정된다.
(3) *c_mode*는 아래의 표에서 정의된 것과 같이 파일 타입과 접근 허용을 포함한다.
(4) *c_uid*는 소유자의 사용자 ID를 포함한다.
(5) *c_gid*는 그룹의 그룹 ID를 포함한다.
(6) *c_nlink*는 기록 보존 파일이 만들어지는 때, 파일을 참고하는 링크 수를 포함한다.
(7) *c_rdev*는 문자형 특수 파일이나 블럭형 특수 파일에 대한 구현시 정의되는 정보를 포함한다.
(8) *c_mtime*은 기록 보존 파일이 만들어질 때, 파일을 수정한 마지막 시간을 포함한다.
(9) *c_namesize*는 종료 널(null) 바이트를 포함하여, 경로명의 길이를 포함한다.
(10) *c_filesize*는 바이트단위로 파일의 길이를 포함한다. 이것은 헤더 구조 다음에 오는 자료 부분의 길이이다.

11.1.2.2 파일명

*c_name*은 파일의 경로명을 포함한다. 바이트 단위로 된 이 필드의 길이는 *c_namesize*의 값이다. 파일명이 잘못된 경로명을 만드는 매체에서 발견되어진다면, 구현시 파일의 데이타가 파일 계층화로 저장되는지의 여부와 파일이 무슨 이름으로 저장되는 지를 정의한다.

모든 문자는 ASCII로 표현되어진다. 구현간의 이식성을 최대로 하기위해, 이름은 POSIX **파일명용 문자 집합**[3.3]에 표현된 문자들로부터 선택되어 패리티가 없는 8 비트 문자로 표현된다. 만약 POSIX 파일명용 문자 집합 이외의 확장 문자집합을 사용되고 두 개의 다른 시스템에 포맷 읽기 유틸리티와 포맷 생성 유틸리티가 같은 확장 문자집합을 사용한다면 파일명은 유지되어야 한다.

그러나, 포맷읽기 유틸리티는 이 표준에서 전에 서술된 함수를 통하여 접근될 수 없는 파일명을 지역 시스템에 만들 수 없다. *open*() [6.3.1], *stat*() [6.6.2], *chdir*() [6.2.1], *fcntl*() [7.5.2], *opendir*() [6.1.2]를 참고하라. 만약 파일명이 잘못된 파일명을 만드는 매체에서 발견된다면, 파일의 데이타가 지역 파일 시스템에 저장되는지의 여부와 파일이 무슨 이름으로 저장되는지가 구현시 정의되어야 한다. 포맷읽기 유틸리티는 파일이 무시되어진다는 것을 나타내는 에러를 만드는 한, 이런 파일을 무시하도록 선택할 수 있다.

11.1.2.3 파일 데이타

c_name 다음에, *c_filesize* 바이트의 데이타가 있다. 그러한 데이타는 파일에 따라 다른 방식으로 해석된다. 만약 *c_filesize*가 0 이라면, 어떠한 자료도 *c_filedata*에 포함되어서는 안된다.

11.1.2.4 특수 엔트리

FIFO형 특수파일들,디렉토리와 trailer는 *c_filesize* 값을 0 으로 한채 기록된다. 다른 특수 파일에 대해, *c_filesize*는 구현시 정의된다. 기록 보존 파일에 있는 다음 파일 엔트리를 위한 헤더는 앞선 파일 엔트리의 마지막 바이트 다음에 바로 쓰여져야 한다. 파일명 "TRAILER!!!" 를 나타내는 헤더는 기록 보존 파일의 끝을 나타낸다. 그러한 헤더 다음에 오는 기록 보존 파일의 마지막 블럭의 내용은 정의되어지지 않는다.

11.1.2.5 cpio 값

cpio 기록 보존 파일 포맷에 의해 필요한 값은 표 11-3에 기술되어진다. C_ISDIR, C_ISFIFO와 C_ISREG는 IEEE Std 1003.1-1988 을 수용하는 시스템에서 지원되어야 한다. 전에 정의된 기타 값들은 기존 시스템과의 호환성을 위해 예약되어 있다. 추가 파일타입이 지원되어진다. 그러나 그러한 파일은 이식성있는 시스템에 이송하고자 하는 기록 보존 파일에는 쓰여지지 않는다.

C_ISVTX, C_ISCTG, C_ISLNK, C_ISSOK는 기존의 구현과 호환성을 유지하기 위해 이 표준에 의해 예약되어있다.

기존 보존 파일로 부터 복구할 때에는 :

(1) 만약 사용자가 지정된 형의 파일을 만들기위한 적절한 권한을 가지고 있지 않다면, 포맷 해석 유틸리티는 엔트리를 무시하고, 표준 에러 출력장치에 에러를 내보내야 한다.

(2) 단지 정규 파일들만이 복구될 자료를 가진다. 정규 파일이라고 간주할 때 사용자에 의해 포맷 읽기 유틸리티에 부과된 어떠한 선택 기준도 만족시킨다면 그러한 자료는 복구된다.

(3) 만약 사용자가 특정 모드 플래그를 설정할 적절한 권한을 가지고 있지 않다면, 플래그는 무시되어 진다. 기록 보존 파일 포맷에서의 일부 모드 플래그는 이 표준이외에서는 언급이 없다. 만약 구현이 이들 플래그를 지원하지 않는다면 이들 플래그는 무시된다.

표 11-3 cpio c_*mode* 필드

파일접근 허용(file permission)

항 목	값	설 명
C_IRUSR	000400	소유자의 읽기
C_IWUSR	000200	소유자의 쓰기
C_IXUSR	000100	소유자의 실행
C_IRGRP	000040	그룹의 읽기
C_IWGRP	000020	그룹의 쓰기
C_IXGRP	000010	그룹의 실행
C_IROTH	000004	기타의 읽기
C_IWOTH	000002	기타의 쓰기
C_IXOTH	000001	기타의 실행
C_ISUID	004000	*uid* 설정
C_ISGID	002000	*gid* 설정
C_ISVTX	001000	예 비 용

파일 타입

항 목	값	설 명
C_ISDIR	040000	디렉토리
C_ISFIFO	010000	FIFO
C_ISREG	0100000	정규 파일
C_ISBLK	060000	블럭형 특수파일
C_ISCHR	020000	문자형 특수파일
C_ISCTG	0110000	예 비 용
C_ISLNK	0120000	예 비 용
C_ISSOCK	0140000	예 비 용

11.1.2.6 참고

< graph > [10.2.1], **< pwd.h >** [10.2.2], **< sys/stat.h >** [6.6.1], *chmod*() [6.6.4], *link*() [6.3.4], *mkdir*() [6.4.1], *read*() [6.4.1, *stat*() [6.6.2]

11.1.3 다중 볼륨

기록 보존/상호 호환 파일 포맷에 의해 표현된 자료가 한 파일 이상에 존재하는 것이 가능하다.

포맷은 바이트의 스트림(*stream*)으로 간주된다. 파일의 끝(end-of-file)(또는 end-of-media)조건은 논리적 바이트 스트림의 어느 두 바이트사이에서도 일어날 수 있

다. 이 조건이 일어나면, 파일 끝 다음에 오는 바이트는 다음 파일에서 첫 번째 바이트가 된다. 포멧 읽기 유틸리티가 다음 파일로 무슨 파일을 읽을 것인가를 결정하는데 이는 구현시 정의되는 방법에 따른다.

보 칙

1. 이 표준에서 정하지 아니한 사항에 대하여는 “전산망기술기준에 관한 규칙”의 관계 규정을 적용한다.

부 칙

1. 이 표준은 1994년 3월 1일 부터 시행한다.