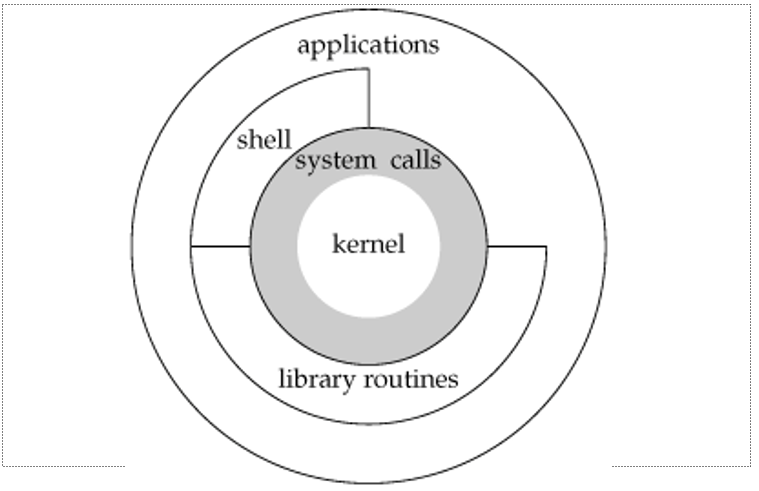
1. Unix

1973년 최초로 고급언어(C언어)를 이용하여 만들어진 운영체제.

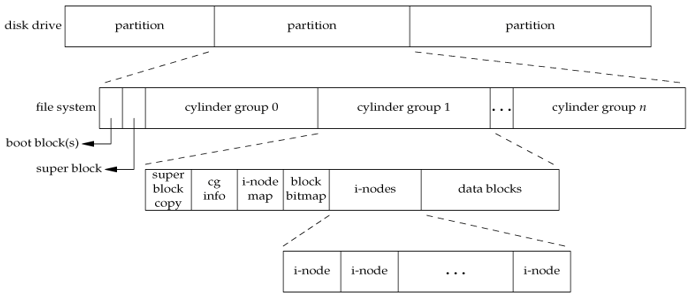
1. Unix structure



1. File system

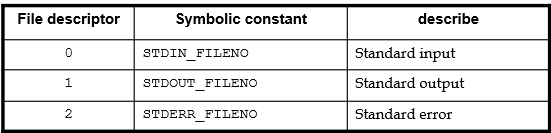
Unix에서의 file system은 hierarchical arrangement로 계층적 구조를 가진다. 이 계층의 최상위 노드를 root라고 한다.

1. UNIX 파일 시스템



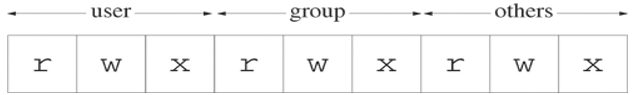
1. Block: 보통 UNIX에서 514byte \* 2의 배수 크기로 정의된다.
2. Boot block: OS를 로드하고 초기화(bootstrap) 할 때 필요한 코드
3. Super block: 파일 시스템 자체의 속성과 메타데이터가 저장된 block
4. i-node: 모든 파일이 하나씩 가지고 있는 identifier. 0,1이 i-node인 파일은 없고 root의 i-node는 2이다.
5. File type
6. Regular file: 일반적인 파일.
7. Directory file: 폴더라고 불리는 파일
8. Character special and block special file: Terminals (character special), disks (block special)
9. FIFO (pipe): IPC를 위한 pipe
10. Socket: process들 사이의 network communication을 위한 file.
11. File descriptor

파일을 열면 kernel에서 파일에 접근하기 위하여 사용하는 것. 양의 정수만을 값으로 가지며 표준 입출력과 표준 에러에 대하여 0,1,2 값이 할당되어 있다.



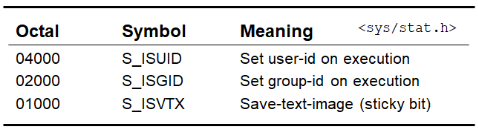
(primitive system data type: <sys/type.h>헤더에 있는 시스템에 따라 다르게 지정될 수 있는 data type. ex) ssize\_t: 64bit 컴퓨터에서는 int 자료형)

1. Permission
2. Owner: file을 create한 user를 보통 일컫는 말. Superuser(root)와 owner만 소유권(파일의 권한을 바꿀 수 있는 권리) 변경 가능.
3. uid: process를 시작한 user의 id
4. gid: group id. /etc/grroup에 정의되어 있음.
5. Ruid: real user id. Process의 실제 user id. 실제 본인의 id(root외에는 바꿀 수 없음)
6. Euid: effect user id. 현재 휘두를 수 있는 id. Euid로 권한을 받을 수 있음.
7. Permission:



1. Extra permission for executable file

실행 프로그램이 포함된 경우에만 관련.

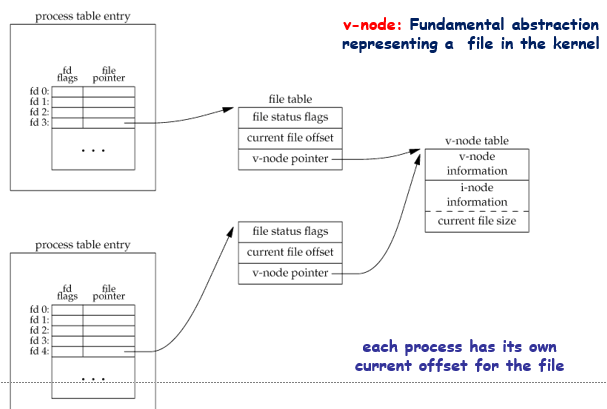


S\_ISUID가 on되어있으면 실행 파일이 시작되면 euid는 그 프로그램의 owner가 됨.

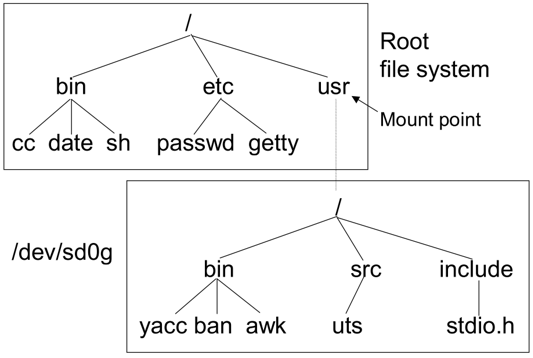
1. File share

서로 다른 process에서 파일을 공유하기 위해서는 아래 3가지 데이터 structure를 사용하여 같은 파일의 v-node를 얻을 수 있게 한다.

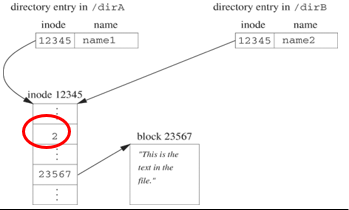
1. Process table entry: process에서 열린 파일의 file descriptor와 file table pointer 정보가 있는 table.
2. File table: 해당 파일에 대한 상태 플래그, 읽었던 곳까지의 offset, v-node pointer등의 정보를 가지고 있는 table
3. v-node structure: i-node를 포함한 파일의 크기 등의 정보를 가지고 있음.



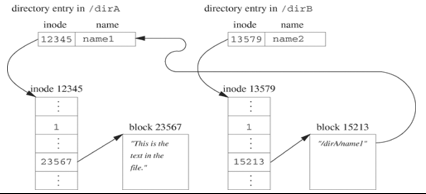
1. Redirection: 표준 입출력의 file descriptor에 다른 파일을 포인터를 씌워 표준 입출력 대상을 바꾸는 작업.
2. Standard I/O
3. Error handling – errno: system call은 동작할 때 error가 발생하면 -1이 발생하지 않는 다면 0이 return되는데 <errno.h>헤더 안의 errno라는 변수에는 이때 발생한 error code가 저장되어 어떤 error가 발생했는지 알 수 있다.
4. Mount-on: mount를 하게 되면 mount point가 mount하게 되는 파일 시스템의 루트 파일 시스템으로 설정되어 그 파일 시스템의 루트 디렉토리가 된다.



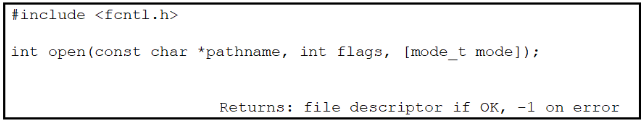
1. Link
2. Hard link: 이름이 다른 같은 파일을 생성하는 것. 원본 파일이 삭제되어도 pointer만 삭제. 한 쪽에서 파일이 바뀌면 같이 바뀜. 같은 i-node사용



1. Symbolic link: 바로가기와 같은 것. Pointer만 추가하기 때문에 원본 파일이 삭제되면 같이 삭제된다. 다른 i-node를 사용한다. (다른 파일인 것)



1. System call 함수들
2. Open: 파일을 열거나 생성.



1. Flags:

O\_RDONLY: Open for reading only #0

O\_WRONLY: Open for writing only #1

O\_RDWR: Open for reading and writing #2

1. optional flags:

O\_APPEND: 기존 데이터 보존하고, 뒤에 이어서 저장

O\_CREAT: 만약 파일이 존재하지 않다면 생성하라

O\_EXCL: O\_CREAT도 지정되어 있고 파일이 이미 있는 경우 오류를 생성하라

O\_TRUNC: 파일이 있는 경우 기존 데이터 삭제 (파일 길이 0 됨)

O\_NONBLOCK: 비차단 파일 열기

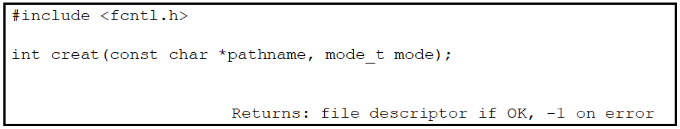
1. mode:

O\_CREAT flag 쓸 때만 사용

File security permission

1. Create

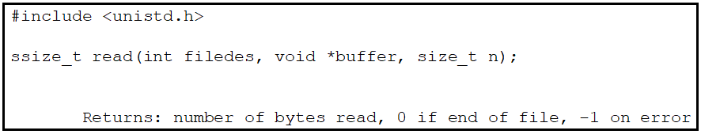
파일을 만드는 다른 방법. 파일이 이미 있는 경우 두번째 인자가 무시됨. Open과 달리 file descriptor를 반환하기 전에 항상 기존 파일의 데이터를 삭제함. 항상 writing only로 파일 open할 때만 사용가능. 이때 생성된 파일의 owner는 euid.



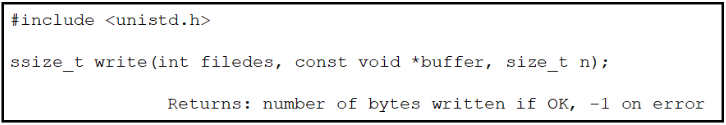
1. Close: 열려 있는 파일 닫기. Process가 정상 종료되면 알아서 닫힘.



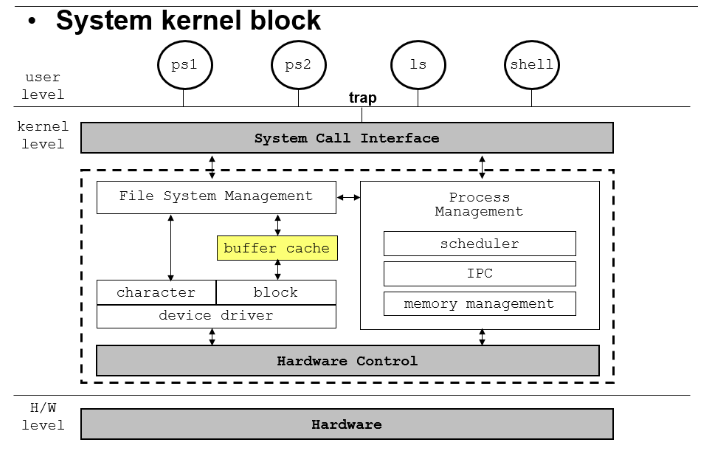
1. Read: 파일을 읽으면 현재 파일 위치에서 메모리로 바이트가 복사된 다음 파일 위치를 업데이트 함. 인자는 차례대로 file descriptor, 메모리 버퍼, 읽을 바이트 수.



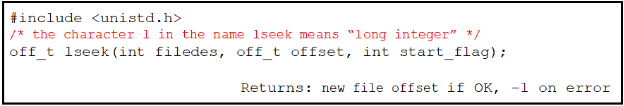
1. Write: 파일을 쓰면 메모리에서 현재 파일 위치로 바이트가 복사된 다음 현재 파일 위치를 업데이트 함.



Write system call은 write를 할 때마다 디스크에 데이터를 넣는 것이 아니라 커널의 버퍼 캐쉬에 넣어 두고 한번에 디스크로 옮기기 때문에 굉장히 빠름. 아래 그림의 buffer cache가 그 부분. 파일 시스템의 메타데이터와 관련된 블록들을 저장하는 캐쉬.

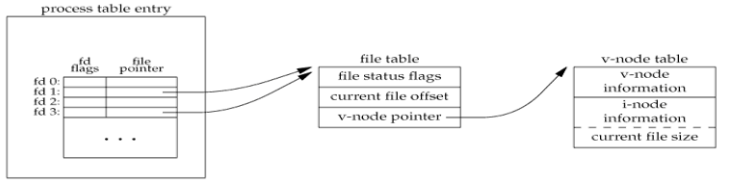


1. Lseek: 열린 파일의 offset을 명시적으로 설정. 인자의 start\_flag는 시작위치, offset은 start\_flg로부터 떨어진 바이트 수.

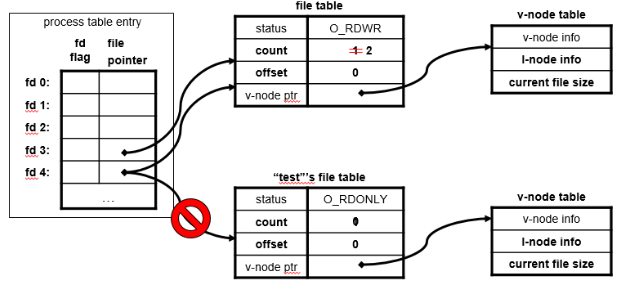


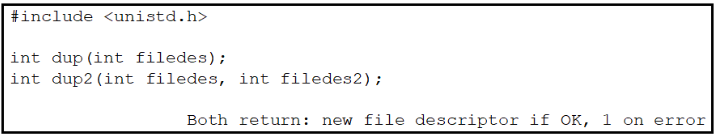
1. Dup, Dup2

Dup: 새로운 file descriptor를 만들어 원래 가리키던 file table을 똑같이 가리킴.

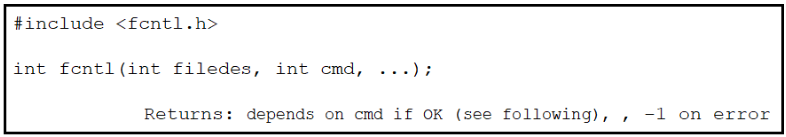


Dup2: 기존의 file descriptor가 다른 file table을 가리키게 함.





1. Fcntl: 이미 열려 있는 file의 속성을 바꿈.



1. cmd: 수행할 특정 함수를 선택한다.

F\_DUPFD : 기존의 디스크립터를 복제

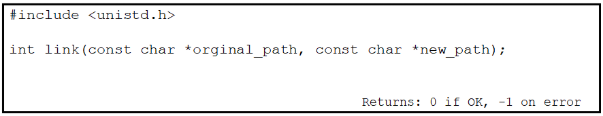
F\_GETFD or F\_SETFD : Get/set file descriptor flags

F\_GETFL or F\_SETFL : Get/set file status flags

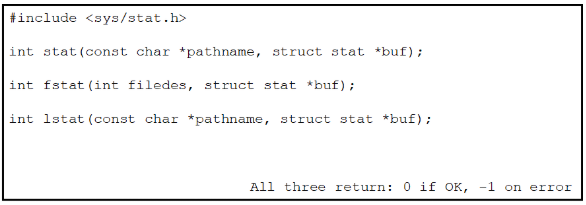
F\_GETOWN or F\_SETOWN : Get/set asynchronous I/O ownership

F\_GETLK, F\_SETLK, or F\_SETLKW : Get/set record locks

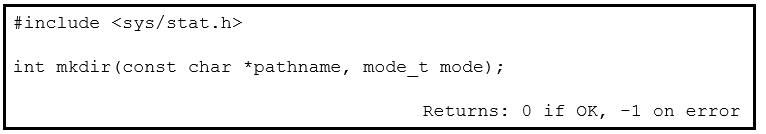
1. Umask: 파일을 O\_CREAT 모드를 설정하고 열 때 파일의 mode가 지정된 대로만이 아닌 특정 권한을 불허할 수 있는 방법.
2. Access: 접근 권한이 있는지 확인.
3. Chmod: permission 변경. Ownership(owner, superuser만 가짐)이 있어야 가능.
4. Chown: 파일의 uid나 gid를 변경.
5. Link: hard link. 새로운 directory entry를 만들고 link count를 증가시킨다.



1. Unlink: 존재하던 directory entry를 지운다. Link name만 지우고 link count를 감소시킨다. Directory의 쓰기, 실행 권한이 필요함.
2. Symlink: symbolic link.
3. Readlink: sympath를 열고 버퍼에 있는 파일 내용을 읽은 후 sympath를 닫는다. 원본파일이 삭제되었다면 symbolic link에서는 내용은 볼 수 없다.
4. Stat: 파일의 정보 얻기.



1. stat: pathname에 지정한 파일의 정보를 검색해 buf로 지정한 구조체에 저장
2. fstat: 현재 열려 있는 파일의 파일 디스크립터를 인자로 받아 정보를 검색한 후 buf로 지정한 구조체에 저장
3. lstat: symbolic link에 대한 정보 저장
4. Directory
5. Path: 상대경로(current working directory에서 출발했을 때의 경로)와 절대 경로(root에서 출발했을 때의 경로)가 존재.
6. Working directory: process가 동작하는 directory
7. System call 함수들
8. Mkdir: directory 생성.

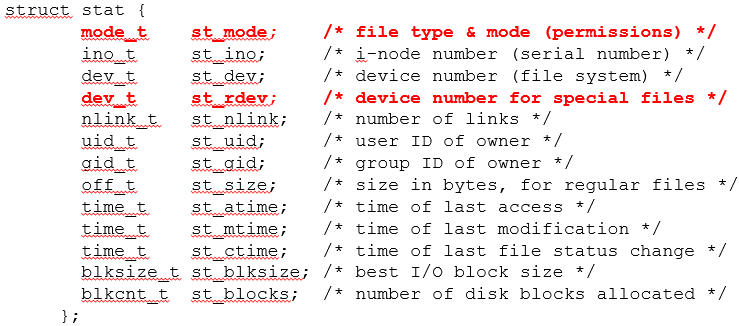


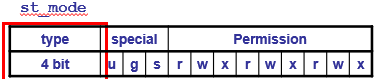
1. Ftw(3): directory나 file을 탐색하기 위한 함수.
2. Sync, Fsync: 동기식 입출력을 위한 함수.
3. Device

유닉스에서 device는 모두 major, minor 번호를 할당 받음. major번호는 device의 타입을 의미하고 minor번호는 특정 device의 instance를 의미한다.

1. Block device file: disk, magnetic tape등의 장치를 말하며 블록이나 섹터 등 정해진 단위로 데이터가 전송됨.
2. Character device file: 키보드, 마우스, 모니터 등의 장치가 있으며 byte단위로 데이터를 전송.
3. Stat

파일에 관한 정보들을 담는 구조체. St\_mode로 파일 타입과 권한을 알 수 있음.





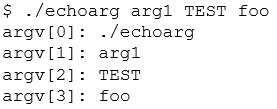
1. Process
2. Main function

커널은 main함수를 호출하기 전에 특별한 start-up루틴을 호출함. start-up루틴으로 커널로부터 command line arguments와 환경변수를 받음.

1. Command line argument

프로그램 실행 시 exec를 호출한 프로세스는 command line argument들을 새 프로그램으로 넘길 수 있음.

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



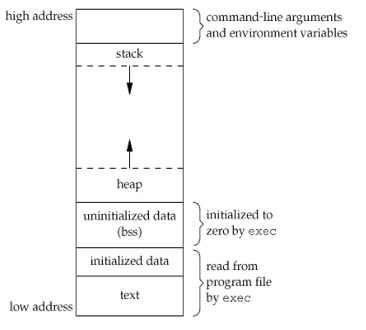
1. Environment list (환경변수 리스트)

시스템의 환경 변수들을 말하는 것. HOME, PATH 같은 것들. Global 변수로 정의되어 있기 때문에 환경변수들에 접근을 쉽게 하기 위해 getenc, putenv등의 함수가 지원된다.

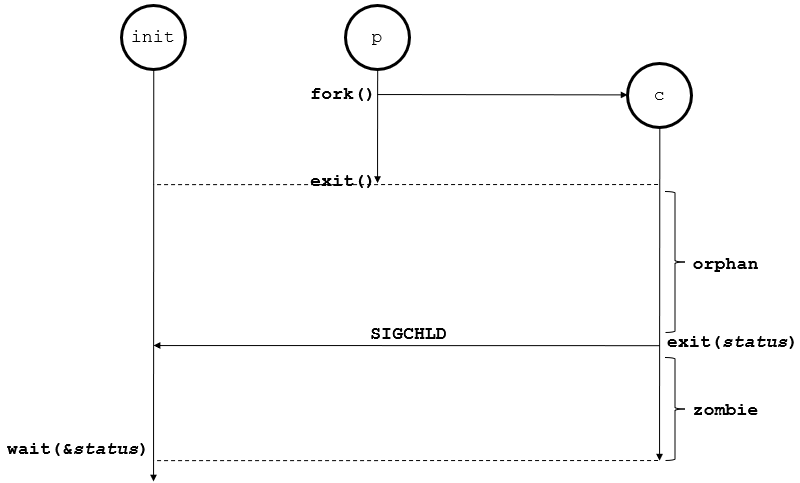
1. Memory Layout in C program

C 프로그램은 크게 text, global, stack, heap 영역으로 메모리가 나뉜다.

1. Code: 말 그대로 프로그램의 코드들이 저장되는 영역.
2. Global: Global 영역은 초기화 여부에 따라 initialized data segment, uninitialized data segment(bss: block started by symbol)로 나뉘고 initialized data segment만 compile된 실행 모듈에 포함된다. 초기화되지 않는 global 변수들은 kernel에 의해 메모리가 할당되고 초기화 된다.
3. Stack: 지역변수들과 함수 호출 시 돌아갈 주소등이 저장되는 영역.
4. Heap: 동적 메모리들이 할당되는 영역.



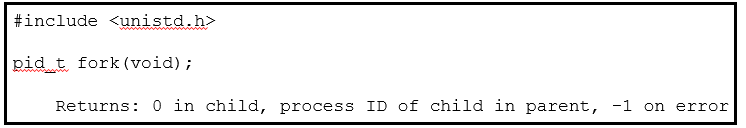
1. Process termination
2. Normal termination: 최종적으로 exit함수를 호출하게 되는 경우.
3. Abnormal termination: exit에 의해서 종료되지 않는 경우. 주로 시그널과 관련됨.
4. Zombie process와 orphan process
5. Zombie process: 부모가 먼저 죽은 process
6. Orphan process: zombie process에서 종료후에 부모의 상위 process가 수거해줄 때까지 기다리는 process



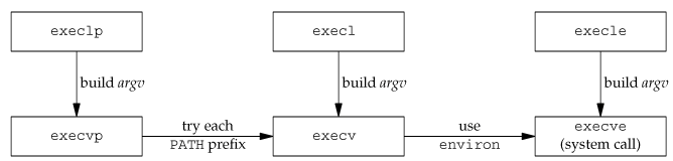
1. System call 함수들
2. Getpid: 호출한 process의 pid를 return.

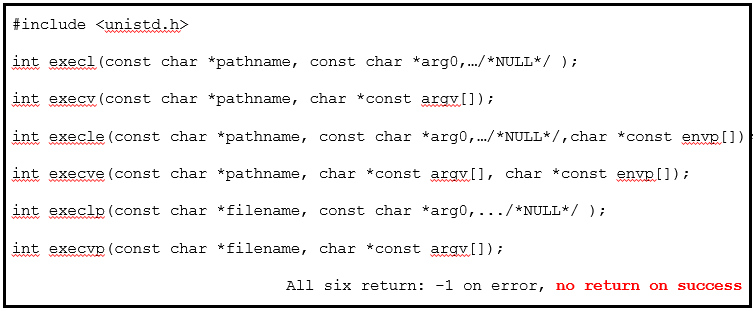
(Pid 0은 스케쥴러 프로세스(swapper), pid 1은 init process: 최상위 프로세스)

1. Getppid: 호출한 process의 부모 process의 pid를 return.
2. Fork: 자식 process 생성.

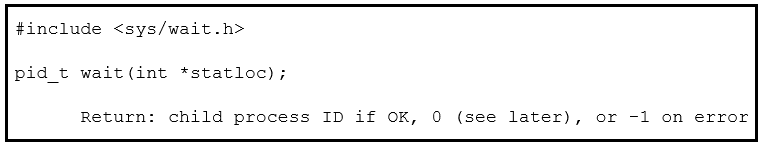


1. Exec: 자식 process로 새로운 process 실행.

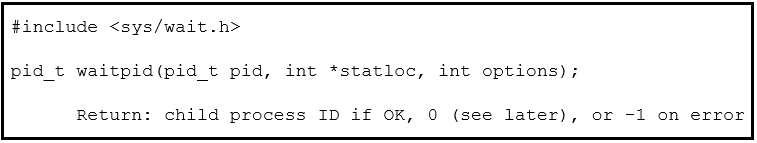




1. Wait: 자식 process의 종료까지 기다림.



1. Waitpid: 자식 프로세스 종료까지 기다림.



1. Pid

Pid==-1: 아무 자식이나 종료 때까지 기다림.

Pid>0: pid가 인자 pid인 자식 프로세스의 종료까지 기다림.

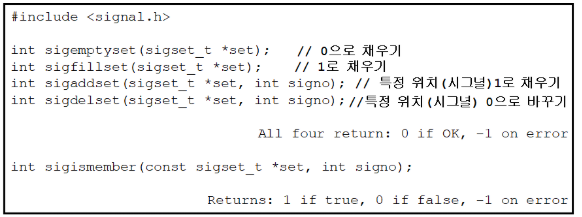
Pid==0: group id가 같은 자식 프로세스의 종료까지 기다림.

Pid<0: pid의 절대값과 동일한 group id를 가진 자식 프로세스의 종료까지 기다림.

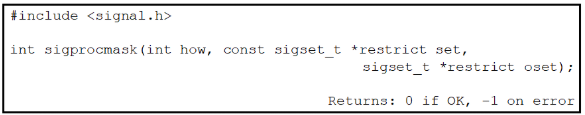
1. Signal

Software interrupt. 에러처리와 다름. 보통은 실행되고 있는 프로세스를 종료할 때 사용. Signal 번호는 <signal.h>에 정의 되어있고, process는 signal을 받으면 signal을 처리하기 위한 액션(ignore, signal handler, default action 중 하나)을 취해야함.

1. Signal handling
2. Ignore: signal을 무시하는 것. SIGKELL, SIGSTOP은 무시할 수 없음.
3. User-defined action: signal handler를 사용하여 action을 정의하는 것. SIGKELL, SIGSTOP은 못함.
4. Default action: 모든 signal은 원래 default action이 있음. 그것을 취하는 것.
5. Signal mask: process attribute로써 이 목록에 있는 signal은 block된다. 이 mask를 만들고 자식 process를 만들면 mask도 함께 상속됨.
6. System call 함수들
7. Signal



1. Sigaction: 특정 시그널과 관련된 액션을 조사, 수정.
2. Sigsetjump, Sigtlongjmp: 시그널 handler에서 분기할 대 항상 사용.
3. Sigprocmask: process 시그널 마스크를 변경.



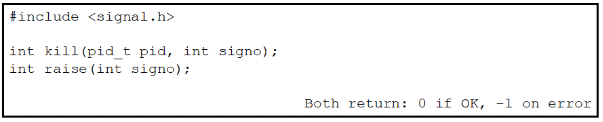
1. How:

SIG\_BLOCK: add a collection of signals to those currently blocked

SIG\_UNBLOCK: delete a collection of signals from those currently blocked

SIG\_SETMASK: set the collection of signals being blocked to the specified set

1. Kill, Raise: kill은 프로세스들에게 시그널을 보내고 raise는 자기 자신한테 보냄.



1. Pid

pid > 0: pid가 pid인 process가 대상.

pid == 0: 보내는 process와 group id가 같은 process가 대상

pid < 0: -pid가 포함된 모든 그룹의 process들이 대상.

pid ==-1: pid의 절대값과 group id가 같은 process가 대상.

1. Pause: 시그널이 catch될 때까지 일시 중단.
2. PIPE

process간의 통신을 위한 메커니즘. 커널에서 생성되는 통신 파일과 같음.

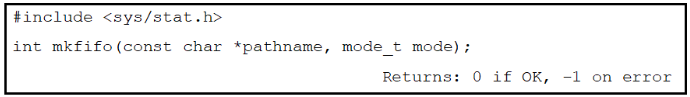
FIFO(first in first out)에 기반한 통신을 위한 특수파일로 lseek을 이용해 파일을 랜덤하게 접근할 수 없고 pipe파일에 접근하기 위해 읽기, 쓰기 용 file descriptor가 따로 존재하고 pipe는 부모, 자식 process간의 통신만을 위해 사용됨.

Fork를 하면 pipe에 대한 file descriptor를 공유하면서 pipe를 사용하게 되는데 이때 두 process가 동시에 pipe를 사용하면 혼란이 올 수 있으므로 보통 process는 하나의 pipe file descriptor만(read나 write) 사용한다.

1. I/O관련 Blocking
2. Read: pipe, terminal device, network socket에서 읽을 데이터가 없는데 read하는 경우.
3. Write: pipe, socket 같은 파일에 데이터를 쓸 공간이 없는데 write한 경우.
4. Open: read용 FIFO가 open된 경우(write FIFO가 open될 때까지 block됨).
5. Blocking을 막는 방법
6. Opentl O\_NONBLOCK 옵션을 주어 open한다.
7. Open된 파일을 fcntl을 이용하여 속성을 변경한다.
8. System call 함수들
9. Pipe



1. Mkfifo: 주어진 pathname을 FIFPO 파일로 만듦. Mode는 permission mask.



1. IPC
2. Record locking

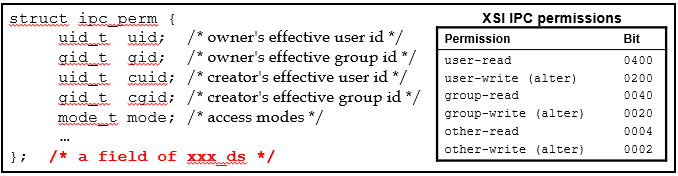
여러 프로세스가 같은 데이터에 접근하는 일을 막기 위한 방법 중 하나. Lock을 기록하기. Lock은 fork시에는 상속되지 않고 exec같은 new program에 접근할 때만 상속된다.

1. Advisory locking: process들간의 약속. 특정 파일 사용시 process간에 약속을 해놓고 파일을 사용시 locking을 검사하는 방법. (강제성은 없기 때문에 process들 간의 협력이 필요하다.)
2. Mandatory locking: kernel에서 lock들을 관리하게 하는 방법. 한 process가 lock을 하고 있을 때 다른 process가 lock된 부분에 read나 write를 하게 되면 kernel이 이 process를 sleep상태로 변경한 후 unlock되었을 때 sleep된 process가 깨어나게 한다.
3. Deadlock

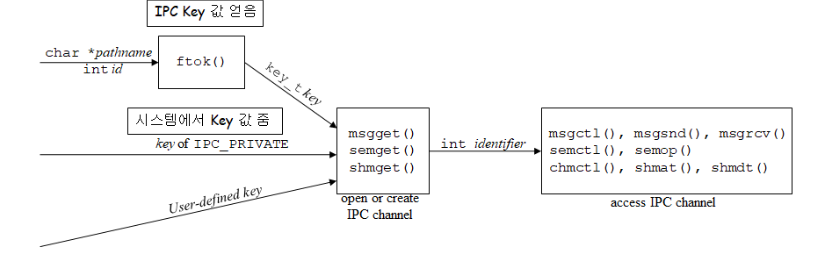
유닉스에서는 fcntl system call로 deadlock을 detect할 수 있고 예방이 가능하다.

1. Permission structure

IPC 객체가 만들어지면 시스템에서 IPC facility의 status 구조체를 만든다. Msgctl, semctl, shmctl을 호출해서 udi, gid, mode를 수정할 수 있다.



1. Identifier, Key
2. Key: IPC 객체의 외부 이름. IPC 객체가 만들어질 때마다(msgget, semget, shmget) 지정됨.
3. Identifier: IPC 객체의 내부 이름(모든 process 공유 가능). 음이 아닌 정수, get연산의 결과로 얻을 수 있음.
4. IPC key로 IPC identifier를 생성하는 방법



1. Message Queue

메세지의 linked list를 커널에 저장하고 message queue identifier로 식별. 메시지 큐는 커널 메모리에서 관리하는 것이므로 process가 종료되어도 사라지지 않음.

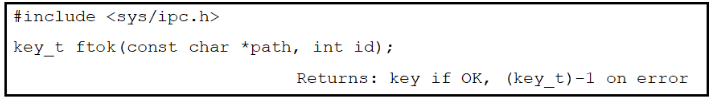
1. Semaphore

Wait과 signal 값을 가지는 정수 변수. set으로 관리되며 semaphore set의 원소가 semaphore.

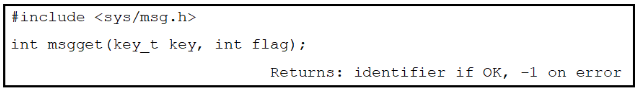
1. Shared memory

여러 process들이 같은 메모리 세그먼트에서 읽고 쓰는 방식. 공유 메모리 접근 동기화를 위해 semaphore가 사용되고 이는 사용자가 직접 공유 메모리에 접근 제어를 해야함.

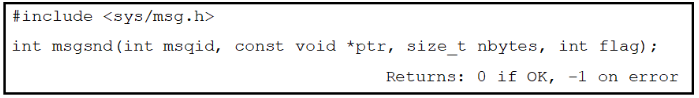
1. System call 함수들
2. Ftok: path와 id를 key\_t 값으로 변환(IPC key 값 얻음).



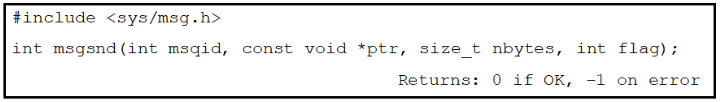
1. Msgget: key 파라미터와 연결된 메시지 큐 identifier를 리턴



1. Msgsnd: 큐에 메시지를 넣음.



1. Msgrcv: 지정된 버퍼(ptr)에 메시지 받음.

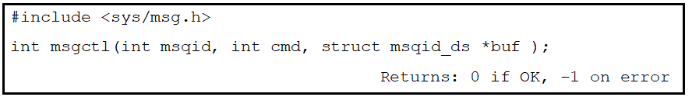


1. flag: IPC\_NOWAIT, MSG\_NOERROR

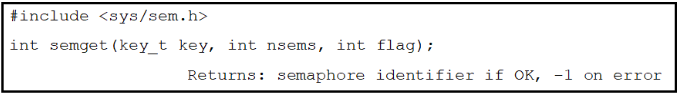
만약 반환된 메시지가 nbytes보다 크고, flag의 MSG\_NOERROR bit가 set 되어 있다면, nbytes 이후의 메시지는 잘린다. (버려짐)

만약 메시지가 너무 크고, flag 값이 지정되지 않았다면, E2BIG의 에러가 대신 리턴된다.

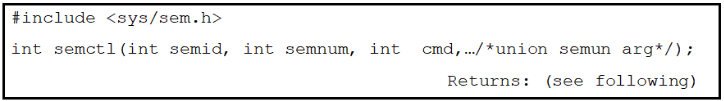
1. Msgctl: 지정된 메시지 큐의 permission을 바꾸거나 할당 해제.



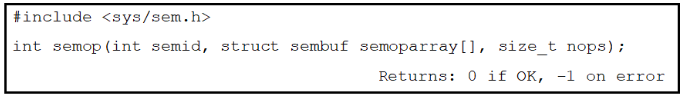
1. Semget: semaphore set을 생성



1. Semctl: semaphore집합의 요소들을 사용 전에 초기화.



1. Semop: 식별자 semid와 연관된 semaphore 집합에 대해 사용자 정의 semaphore 연산의 집합을 수행.



1. sem\_num: 다루고자 하는 세마포어 번호
2. sem\_op: 수행할 동작 (양수, 음수, 0)

음수: 세마포어로 값으로부터 그 값을 뺀다.

양수: 세마포어로 값에 그 값을 더한다

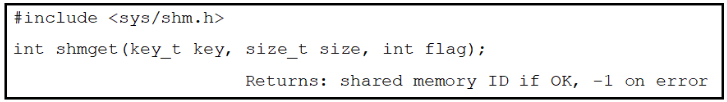
0: 호출한 프로세스는 세마포어의 값이 0이 될 때까지 기다림.

1. sem\_flg: 동작 플래그

SEM\_NOWAIT: 세마포어 호출 즉시 실행하지 못했을 경우(세마포어 값이 0이 되면 원래 wait을 하는데), 기다리지 않고 실패로 바로 리턴함.

SEM\_UNDO: 프로세스가 종료되면 시스템에서 세마포어 설정을 원래 상태로 되돌림. 세마포어 할당 해제 안 하고 프로그램 종료하는 것 예방

1. Shmget: shard memory 공간 확보. Size는 메모리 세그먼트의 최소 크기.



1. Shmat: 지정된 공유 메모리 세그먼트를 호출 process의 주소 공간에 첨부하고 자기 메모리처럼 사용할 수 있게 함.

