Linux System Programming

by ProgCoach4U

파일 다루기 - advanced

디렉터리의 파일 다루기

```
DIR *opendir(const char *name);
```

파라미터
- name: 파일 경로

반환값
- 성공 시 열린 디렉터리 스트림 포인터
- 실패 시 NULL 포인터

디렉터리의 파일 다루기

```
struct dirent *readdir(DIR *dirp);
```

파라미터
- dirp: 디렉터리 스트림 포인터

반환값
- 성공 시 디렉터리 엔트리 포인터
- 실패 시 NULL 포인터

```
struct dirent {
    ino_t          d_ino;
    off_t          d_off;     /* XXX
*/
    unsigned short d_reclen;
    unsigned char  d_type;
    char           d_name[256];
};
```

디렉터리의 파일 다루기

Value	Meaning	Value	Meaning
DT_BLK	block device	DT_LNK	symbolic link
DT_CHR	character device	DT_REG	regular file
DT_DIR	directory	DT_SOCK	unix-domain socket
DT_FIFO	named pipe	DT_UNKNOWN	N/A

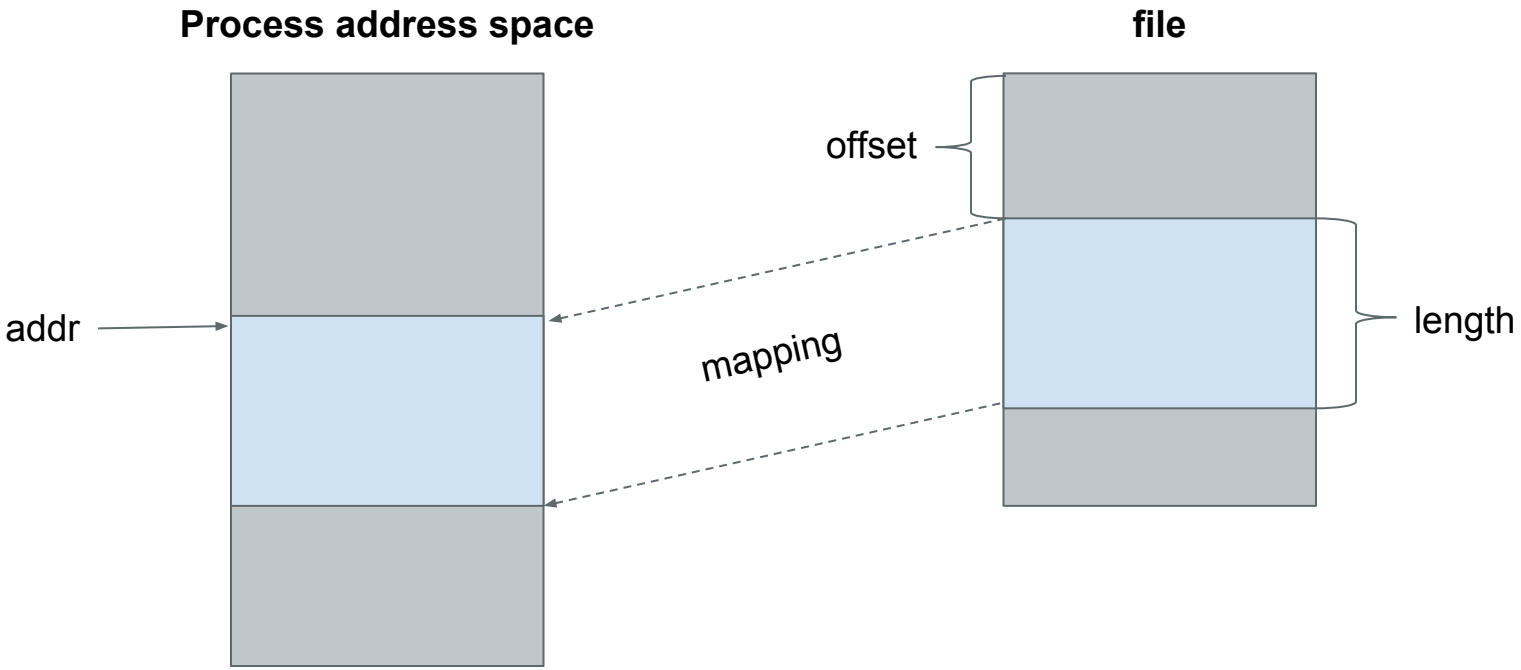
디렉터리의 파일 다루기

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

파라미터
- dirp: 디렉터리 스트림 포인터

반환값
- 성공 시 0
- 실패 시 -1

파일을 메모리에 매핑



파일을 메모리에 매핑

```
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

파라미터
- addr: mapping 될 address
- length: mapping 할 길이
- prot
  - PROT_EXEC
  - PROT_READ
  - PROT_WRITE
  - PROT_NONE
- flags
  - MAP_SHARED
  - MAP_PRIVATE
  - MAP_FIXED
- fd
- offset: 반드시 page size의 배수

반환값
- 성공 시 mapping 된 주소
- 실패 시 MAP_FAILED

파일을 메모리에 매핑

```
int munmap(void *addr, size_t length);
```

파라미터
- addr: mapped address
- length: mapped 길이

반환값
- 성공 시 0
- 실패 시 -1

파일을 이용한 동기화

Race condition(경쟁 상태)

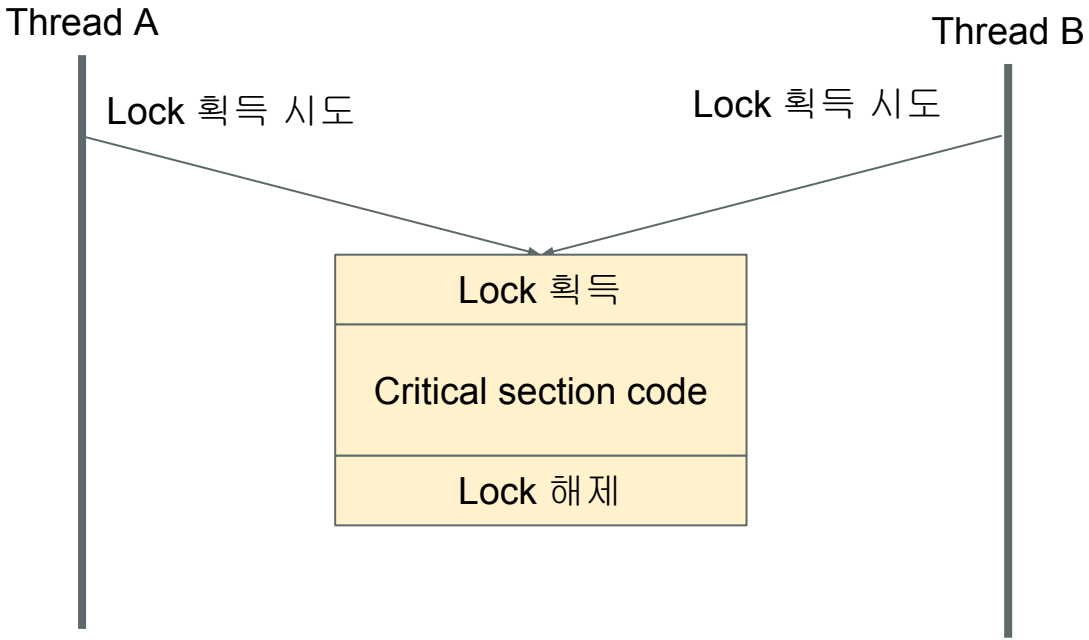
- 둘 이상의 Process/Thread가 동시에 어떤 작업을 수행할 때, 타이밍 등에 의해 의도치 않은 결과가 나올 수 있는 상태

Critical Section(임계 영역)

- 둘 이상의 Process/Thread가 동시에 접근하면 안되는 공유 데이터를 접근하는 코드 영역
- 즉, Race condition을 발생시킬 수 있는 코드 영역

파일을 이용한 동기화

Race condition을 해결하기 위한 방법: Lock mechanism



파일을 이용한 동기화

```
int flock(int fd, int operation);
```

파라미터
- fd: 파일 디스크립터
- operation
  - LOCK_SH: shared lock 걸기
  - LOCK_EX: exclusive lock 걸기
  - LOCK_UN: lock 풀기
  - LOCK_NB: non-block. 다른 값과 ORing하여 사용

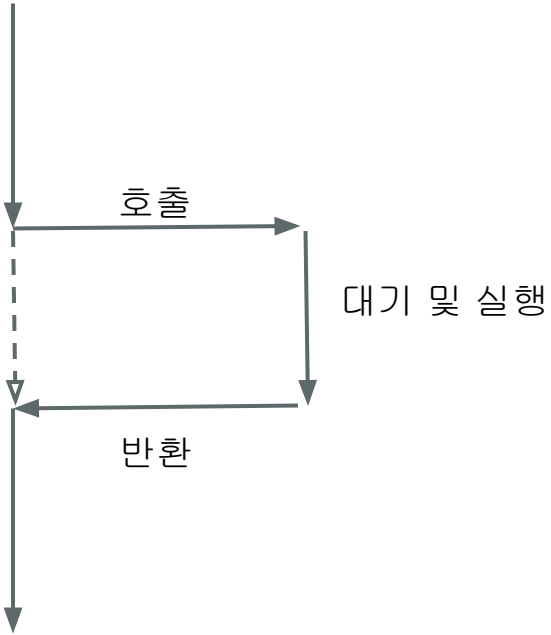
반환값
- 성공 시 0
- 실패 시 -1

파일을 이용한 동기화

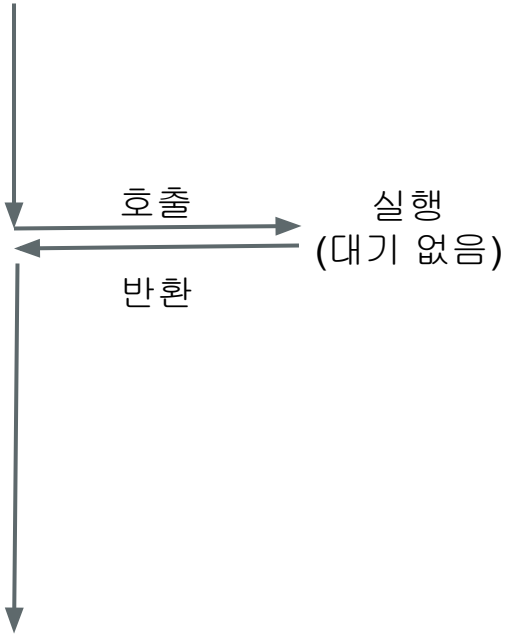
	shared lock 획득 시도	exclusive lock 획득 시도
lock 없음	즉시 성공	즉시 성공
shared locked 상태	즉시 성공	모든 shared lock이 풀릴 때까지 대기
exclusive locked 상태	exclusive lock이 풀릴 때까지 대기	exclusive lock이 풀릴 때까지 대가

Blocking vs. Non-blocking

Blocking operation



Non-Blocking operation



Blocking vs. Non-blocking

Non-blocking 모드 설정 방법

1. open() 시 O_NONBLOCK 설정
2. open() 이후 fcntl()을 이용해 O_NONBLOCK 설정/해제

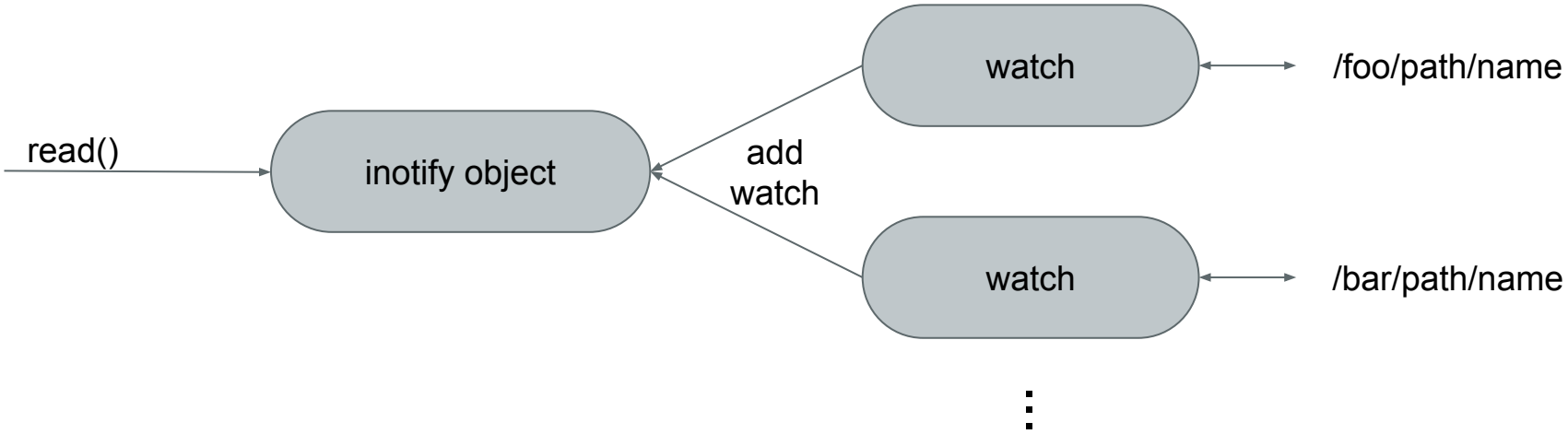
```
int fcntl(int fd, int cmd, ... /* arg */ );
```

파라미터
- fd: 변경하고자 하는 file descriptor
- cmd
  - F_GETFL
  - F_SETFL

반환값
- 성공 시 0
- 실패 시 -1

파일 이벤트 - inotify

inotify: 특정 파일/디렉토리에서 발생하는 이벤트를 감시하는 기능



파일 이벤트 - inotify

```
int inotify_init(void);
int inotify_init1(int flags);
```

파라미터
- flags
  - IN_NONBLOCK: set non-blocking mode
  - IN_CLOEXEC: set close-on-exec

반환값
- 성공 시 inotify 객체(file descriptor)
- 실패 시 -1

파일 이벤트 - inotify

```
int inotify_add_watch(int fd, const char *pathname, uint32_t mask);
```

파라미터
- fd: inotify 객체
- pathname: 감시 대상 경로
- mask: 어떤 이벤트에 대해 어떻게 감시할 것인지 명시

반환값
- 성공 시 watch descriptor
- 실패 시 -1

파일 이벤트 - inotify

add watch 시 & event read 시 mask에 설정 가능

Mask	Meaning	Mask	Meaning
IN_ACCESS	파일 엑세스	IN_DELETE	삭제됨
IN_ATTRIB	메타데이터 변경	IN_DELETE_SELF	감시중인 경로 자체가 삭제됨
IN_CLOSE_WRITE	쓰기 권한 파일이 닫힘	IN_MODIFY	파일 변경
IN_CLOSE_NOWRITE	쓰기외 권한 파일이 닫힘	IN_MOVE_SELF	감시중인 경로가 이동
IN_CREATE	생성됨	IN_MOVED_FROM	이름 변경 시 예전 이름
IN_OPEN	열림	IN_MOVED_TO	이름 변경 시 새로운 이름

파일 이벤트 - inotify

add watch 시 mask에 설정 가능

Mask	Meaning
IN_DONT_FOLLOW	symbolic link인 경우 추적 하지 않도록 함
IN_EXCL_UNLINK	WD에서 unlink된 파일에 대해 이벤트를 발생하지 않도록 함
IN_MASK_ADD	이미 추가된 경로인 경우 기존 mask에 ORing
IN_ONESHOT	이벤트 1회 발생 후 자동으로 WD를 삭제함
IN_ONLYDIR	감시 경로가 디렉토리인 경우만 허용

event read 시 mask에 설정 가능

Mask	Meaning
IN_IGNORED	감시가 제거되었거나 감시 경로가 제거된 경우
IN_ISDIR	디렉토리엔 경우
IN_Q_OVERFLOW	event queue가 overflow
IN_UNMOUNT	감시 경로가 unmounted

파일 이벤트 - inotify

```
int inotify_rm_watch(int fd, int wd);
```

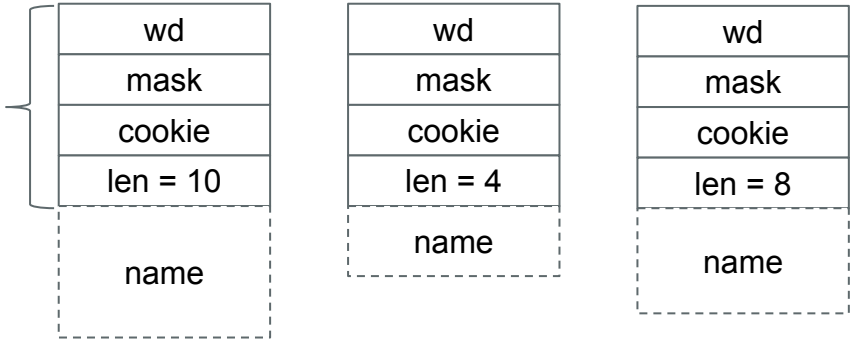
파라미터
- fd: inotify 객체
- wd: 삭제할 watch descriptor

반환값
- 성공 시 0
- 실패 시 -1

파일 이벤트 - inotify

```
struct inotify_event {
    int      wd;       /* Watch descriptor */
    uint32_t mask;     /* Mask describing event */
    uint32_t cookie;   /* Unique cookie associating
                          related events (for rename(2)) */
    uint32_t len;      /* Size of name field */
    char     name[];   /* Optional null-terminated name */
};
```

sizeof(struct inotify_event)



I/O Multiplexing

한 application에서 여러 fd에 대해 read를 하려면?

1. polling with non-blocking mode
2. multi-thread
3. I/O Multiplexing

I/O Multiplexing

2. multi-thread model

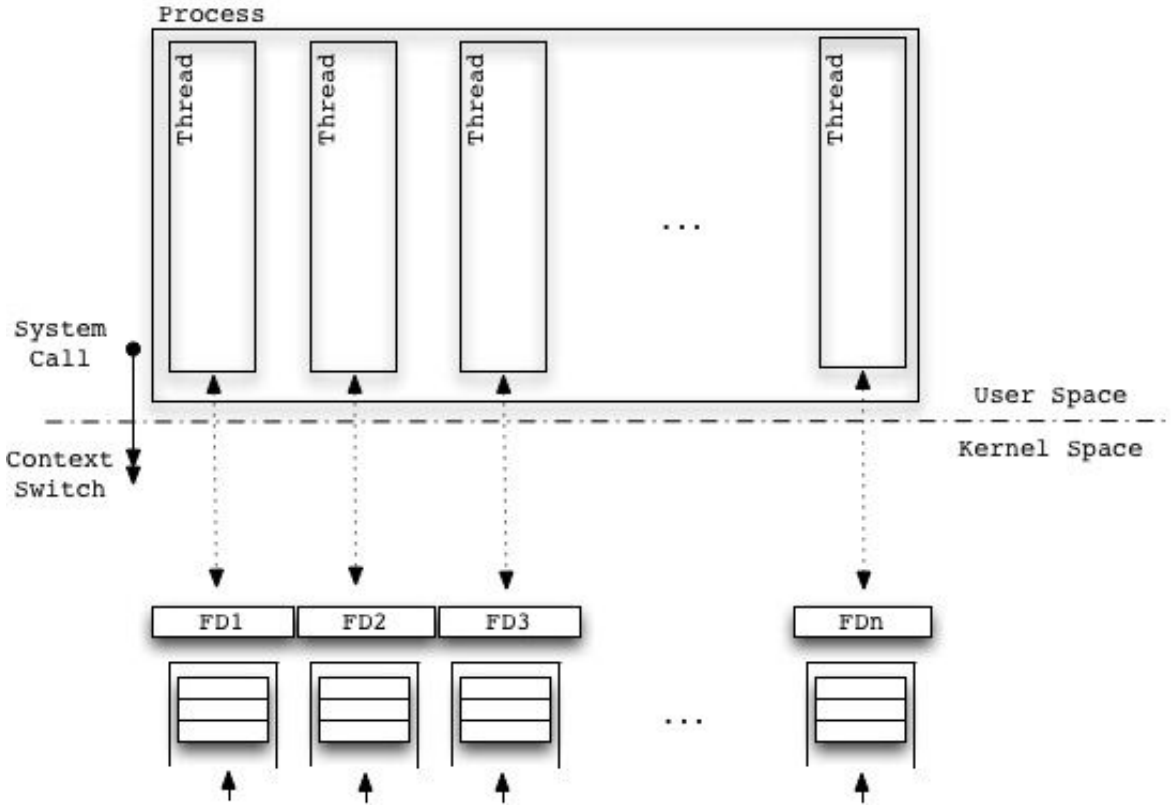


image source:
https://thetechsolo.wordpress.com/2016/02/29/scalable-io-events-vs-multithreading-based/

I/O Multiplexing

3. I/O Multiplexing

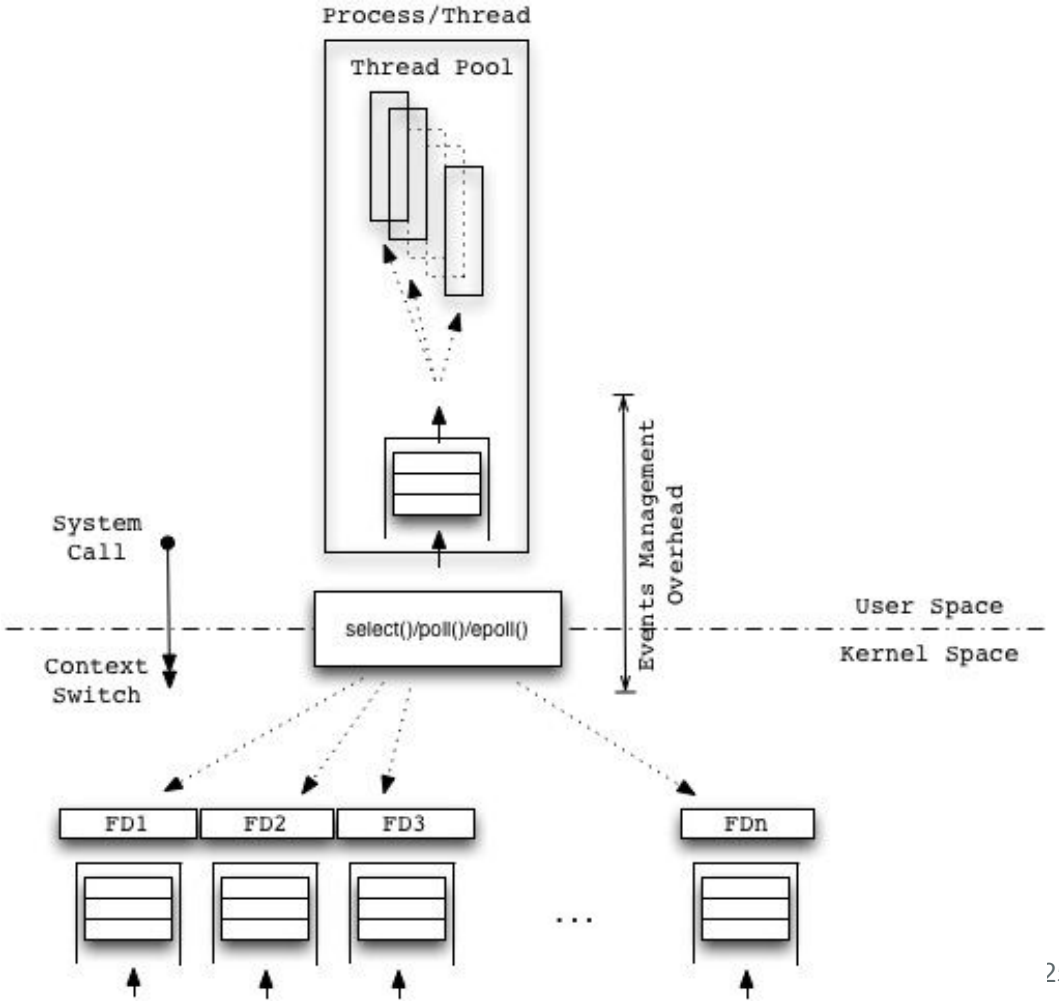
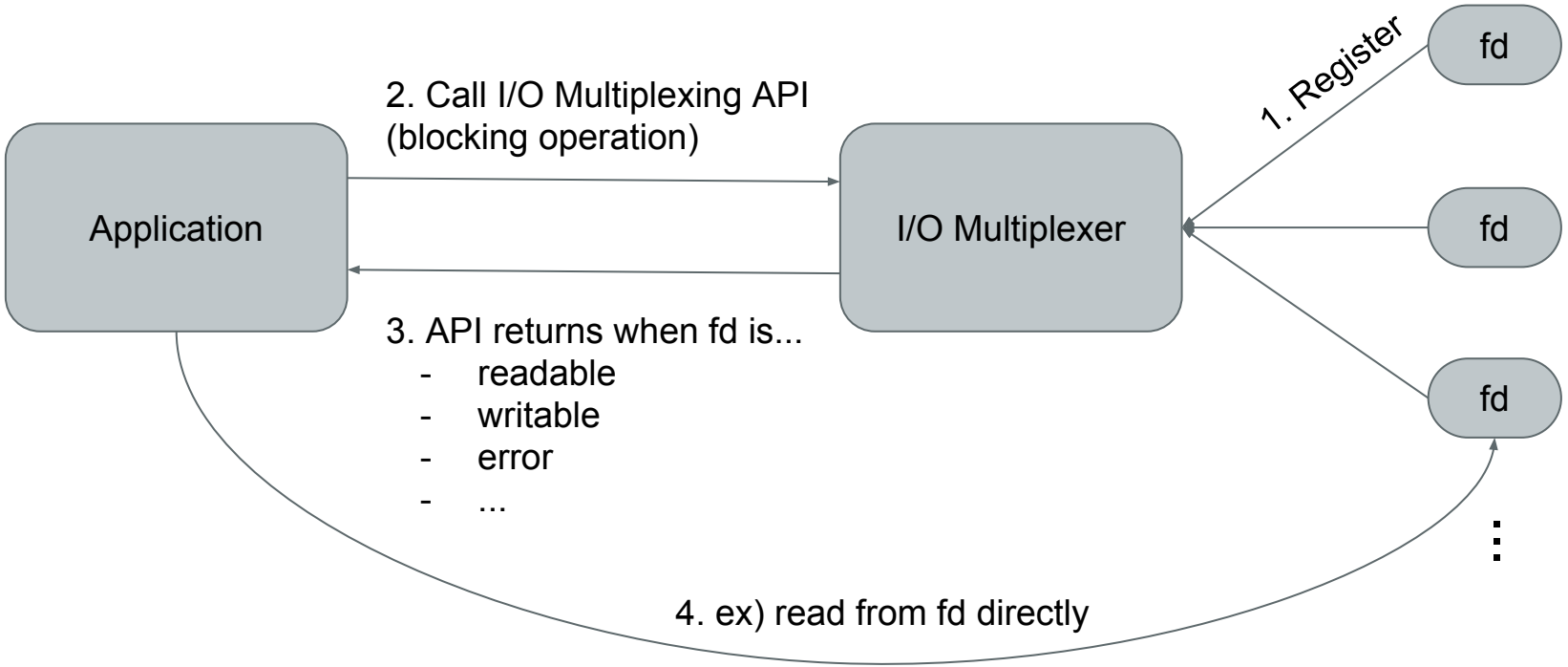
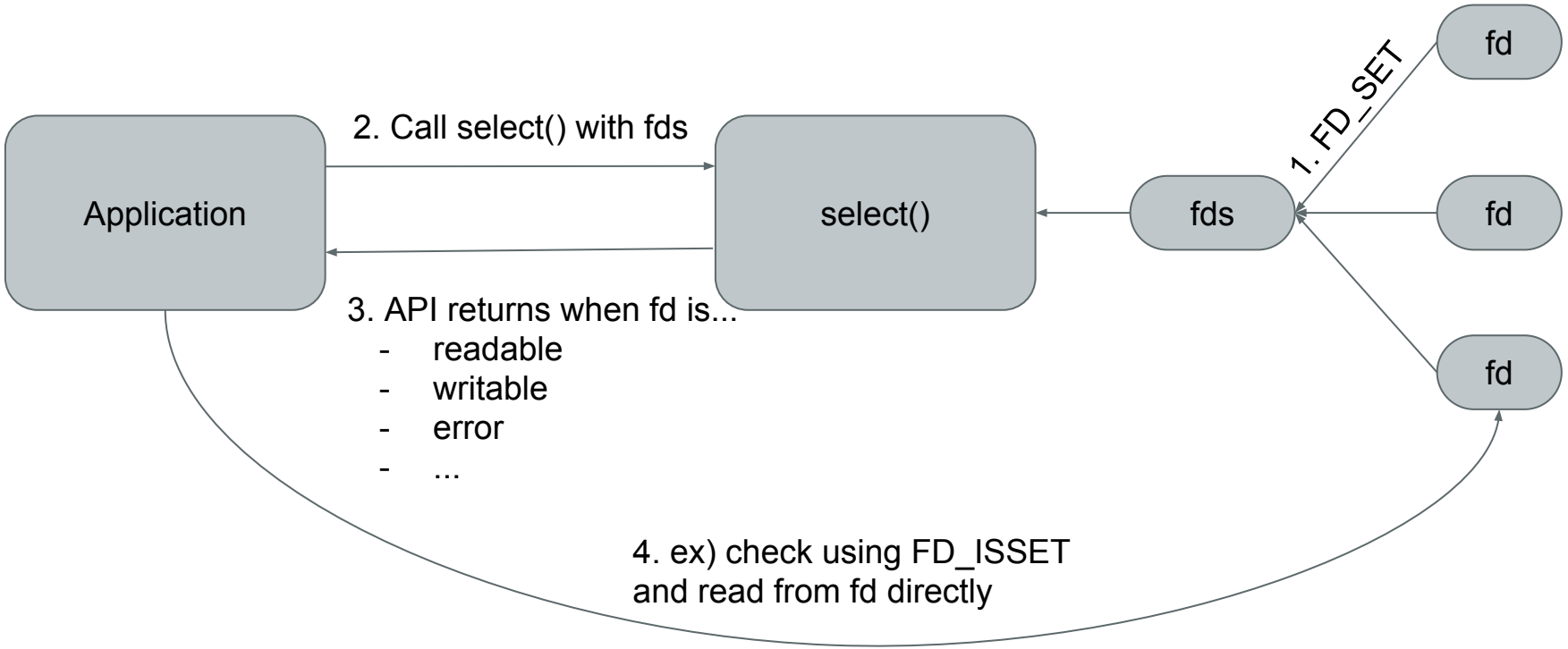


image source:
https://thetechsolo.wordpress.com/2016/02/29/scalable-io-events-vs-multithreading-based/

I/O Multiplexing



I/O Multiplexing - select()



I/O Multiplexing - select()

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

파라미터
- nfds: fdset에 포함된 fd 중 가장 큰값 + 1
- readfds: read가 가능한 시점을 감시하기 위한 fd set, NULL 입력 가능
- writefds: write가 가능한 시점을 감시하기 위한 fd set, NULL 입력 가능
- exceptfds: exception이 가능한 시점을 감시하기 위한 fd set, NULL 입력 가능
- timeout:
  - non NULL: 최대 block timeout 값 지정
  - NULL: 무한 blocking

반환값
- 양수: event가 발생한 fd의 개수
- 0: timeout 발생
- -1: 에러 발생

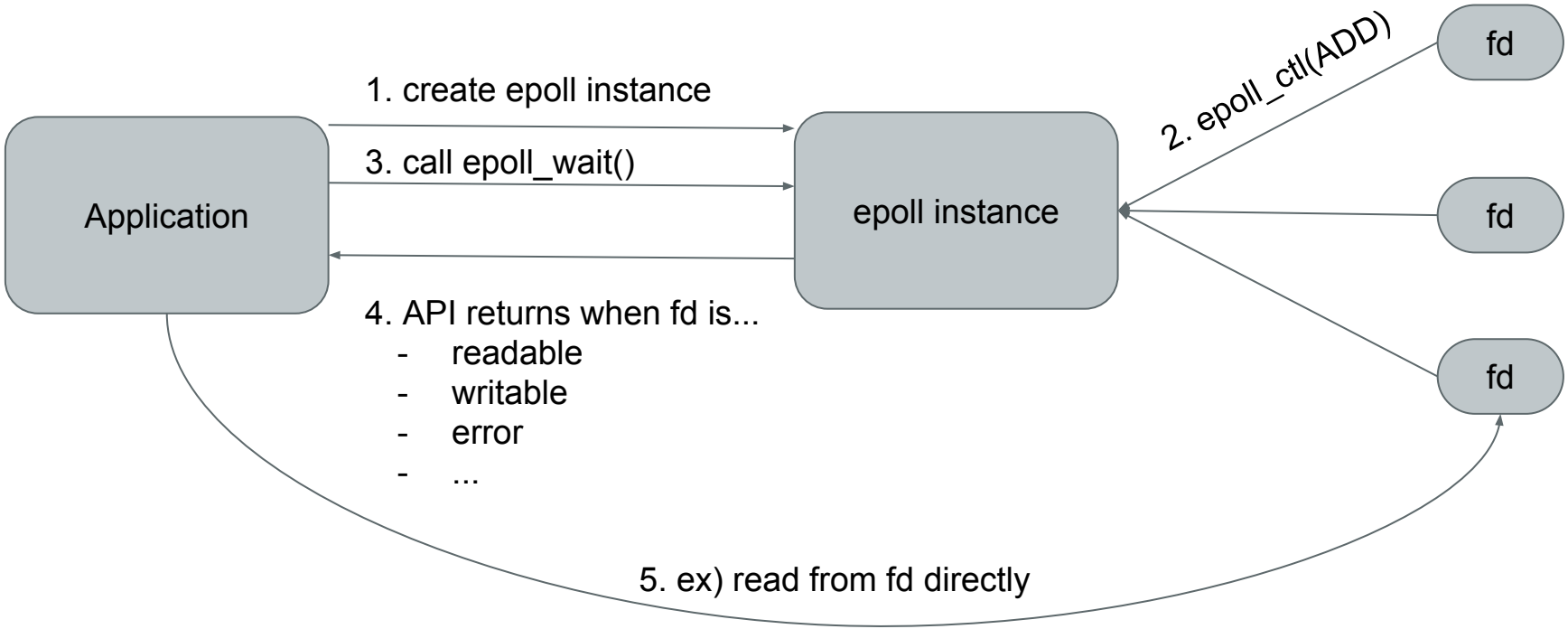
I/O Multiplexing - select()

```
struct timeval {
    long    tv_sec;         /* seconds */
    long    tv_usec;        /* microseconds */
};

void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);

(example)
fd_set rfds;
FD_ZERO(&rfds);
FD_SET(0, &rfds);
```

I/O Multiplexing - epoll



I/O Multiplexing - epoll

```
int epoll_create(int size);
int epoll_create1(int flags);
```

파라미터
- size: 무시되지만 반드시 양수이어야 함
- flags
  - EPOLL_CLOEXEC: close-on-exec

반환값
- 성공시 epoll instance(non-negative)
- 실패시 -1

I/O Multiplexing - epoll

```
int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
```

파라미터
- epfd: epoll instance
- op: 수행할 동작
  - EPOLL_CTL_ADD
  - EPOLL_CTL_MOD
  - EPOLL_CTL_DEL
- fd: 감시할 fd
- event: 감시 이벤트 명시

반환값
- 성공시 0
- 실패시 -1

I/O Multiplexing - epoll

```
int epoll_wait(int epfd, struct epoll_event *events,
               int maxevents, int timeout);
```

파라미터
- epfd: epoll instance
- events: 발생한 이벤트 정보
- maxevents: events 변수의 최대 엔트리 개수
- timeout: 최대 blocking timeout(milliseconds 단위)

반환값
- 성공시 0
- 실패시 -1

I/O Multiplexing - epoll

```
typedef union epoll_data {
    void    *ptr;
    int      fd;
    uint32_t u32;
    uint64_t u64;
} epoll_data_t;

struct epoll_event {
    uint32_t     events;    /* Epoll events */
    epoll_data_t data;      /* User data variable */
};
```

I/O Multiplexing - epoll

epoll event mask	epoll_ctl	epoll_wait	Description
EPOLLIN	O	O	read 가능
EPOLLOUT	O	O	write 가능
EPOLLRDHUP	O	O	연결 종료, 상대방 소켓 셧다운
EPOLLPRI	O	O	TCP socket에서 OOB 데이터 수신 등 예외 상황
EPOLLERR	X	O	에러가 발생한 상황
EPOLLHUP	X	O	장애가 발생한 상황
EPOLLET	O	X	이벤트 감지를 엣지 트리거(Edge trigger)로 설정
EPOLLONESHOT	O	X	해당 fd에서는 단 한번의 이벤트만 발생

I/O Multiplexing - select() vs. epoll

	select()	epoll
이벤트 발생 fd 탐색	모든 fd에 대해 발생 여부 체크	이벤트 발생한 fd를 직접 알려줌
감시 대상 fd 설정	select() 호출 시마다 설정해야 함	재설정 필요 없음
이식성	좋다. 대부분의 OS에서 지원	Linux only
성능	나쁘다	좋다

감사합니다.