Part 02. 리눅스 시스템 프로그래밍

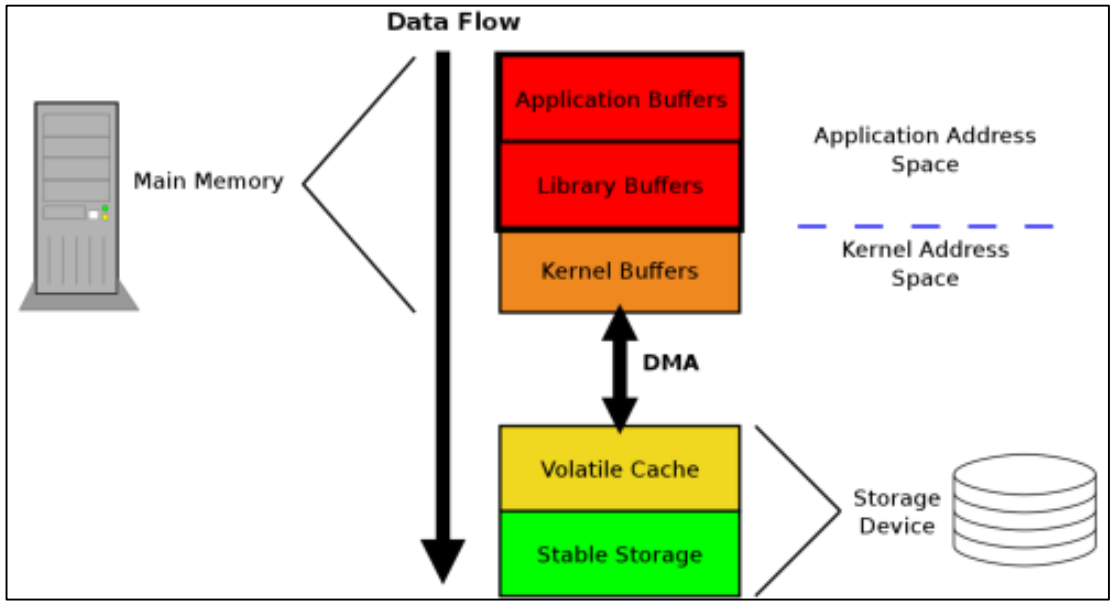
Chapter 03. 파일 I/O(3)

진행 순서

Chapter 03_01 동기화된 I/O (Synchronized I/O)

I/O 동기화란?
기본적으로 리눅스 시스템은 메모리에 보존된 버퍼 캐시로 부터 읽고, 쓰기 행위 시 버퍼 캐시로 씁니다.
그리고 버퍼가 가득 차거나 동기화 기능을 호출하여 버퍼 캐시를 플러시 하기 전까지 실제로 데이터를 디스크(저장장치)에 전송하지 않습니다. (I/O Buffering)
이렇게 하면 필요한 경우 디스크에 기록하는 비교적 느린 기계적 프로세스를 줄임으로써 성능이 향상됩니다.

그럼에도 불구하고 응용 프로그램이 데이터가 디스크(저장장치)에 기록되는 시기를 제어하려는 경우가 있습니다.
이러한 용도로 리눅스 커널은 동기화된 작업을 위해 몇 가지 옵션을 제공합니다.



I/O Buffering
(https://lwn.net/Articles/457667/)

Chapter 03_02 fsync()

```
#include <unistd.h>

int fsync (int fd);
```

데이터가 디스크에 도달했는지 확인하는 가장 간단한 방법은 fsync() 시스템 호출을 사용하는 것입니다.

fsync()를 호출하면 파일 디스크립터 fd에 의해 매핑된 파일과 연관된 모든 더티 데이터가 디스크에 기록됩니다.
파일 디스크립터 fd는 쓰기 가능하도록 열려 있어야 합니다.
하드 드라이브가 데이터와 메타 데이터가 디스크에 있다고 말할 때까지 반환되지 않습니다.

성공시 0을 리턴, 실패시 -1 리턴 (errno)

EBADF
- 주어진 파일 디스크립터가 유효하지 않습니다.

EINVAL
- 지정된 파일 디스크립터가 동기화를 지원하지 않는 오브젝트에 매핑됩니다.

EIO
- 동기화중에 하위 레벨 I/O 에러가 발생했습니다. (실제 I/O 오류)

Chapter 03_03 fdatasync()

```
#include <unistd.h>

int fdatasync (int fd);
```

이 시스템 호출은 데이터만 플러시 한다는 점을 제외하고 fsync()와 동일한 작업을 수행합니다.
이 호출은 메타 데이터가 디스크와 동기화 되었음을 보장하지 않으므로 잠재적으로 더 빠릅니다.

fsync / fdatasync 모두 파일을 포함하는 업데이트 된 디렉토리 항목이 디스크와 동기화되도록 보장하지 않습니다.
디렉토리 항목에 대한 업데이트도 디스크에 커밋되도록 할려면 디렉토리 자체에 대해 열린 파일 디스크립터에서도 fsync를 호출해야 합니다.

Chapter 03_04 sync()

```
#include <unistd.h>

void sync (void);
```

sync() 시스템 호출은 모든 버퍼를 디스크에 동기화하기 위해 제공됩니다.
이 함수에는 매개 변수가 없으면 반환 값이 없습니다.
항상 성공하고, 모든 버퍼 (데이터 및 메타 데이터 모두)는 디스크에 기록됩니다.

Chapter 03_05 open() 동기화 플래그(O_SYNC)

O_SYNC 플래그는 open()에 전달되어 파일은 모든 I/O가 동기화 되어야 함을 나타냅니다.

```
fd = open (file, O_WRONLY | O_SYNC);
```

write() 호출은 일반적으로 동기화되지 않습니다.
O_SYNC 플래그는 write() 호출이 동기화된 I/O를 수행하도록 합니다.

O_SYNC는 쓰기 작업에 대해 사용자 및 커널 시간을 더 소모합니다.
또한 작성중인 파일의 크기에 따라 I/O 대기 시간이 발생하기 때문에 경과 시간을 증가시킬 수 있습니다.
꼭 필요할 때가 아니면 O_SYNC 플래그는 사용하지 않습니다.

일반적으로 쓰기 작업이 디스크에 닿는 것을 보장해야 하는 응용 프로그램을 fsync() 또는 fdatasync()를 사용합니다. 덜 자주 호출될 수 있기 때문에(즉, 특정 작업이 완료된 후에만 호출) O_SYNC 보다 비용이 적게 드는 경향이 있습니다.

Chapter 03_06 디렉토리 열기 (opendir)

디렉토리의 내용을 읽으려면 파일 디스크립터와 마찬가지로 디렉토리를 open하여
디렉토리 스트림을 생성해야 합니다.

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

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

opendir()은 성공 시 디렉토리 스트림 포인터를 정상적으로 리턴하며, 실패 시 NULL 포인터를 리턴합니다.

디렉토리 스트림은 열린 디렉토리를 나타내는 파일 디스크립터, 일부 메타 데이터, 디렉토리 내용을 담을 수 있는 버퍼를 가리키는 포인터에 지나지 않습니다.
디렉토리 스트림 뒤에 있는 파일 디스크립터는 dirfd()를 통해 얻을 수 있습니다.

```
int dirfd (DIR *dir);
```

dirfd()는 성공 시 파일 디스크립터를 정상적으로 리턴하며, 실패 시 -1을 리턴합니다.

Chapter 03_07 디렉토리 읽기 (readdir)

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

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

readdir()이 성공적으로 호출되면 dirent 구조체 정보가 리턴됩니다.
<dirent.h>에 정의되어 있는 dirent 구조체 정보는 다음과 같습니다.

```
struct dirent {
        ino_t d_ino;                    /* inode 번호 */
        off_t d_off;                    /* 다음 dirent의 오프셋 */
        unsigned short d_reclen;        /* 해당 레코드의 길이 */
        unsigned char d_type;           /* 파일 타입 */
        char d_name[256];               /* 파일 이름 */
};
```

응용 프로그램은 readdir ()을 계속 호출하여 검색중인 파일을 찾을 때까지 또는 전체 디렉토리를 읽을 때까지 (readdir ()이 NULL을 반환 할 때까지) 디렉토리의 각 파일을 가져옵니다.
실패하면 readdir ()도 NULL을 반환합니다.
오류와 모든 파일을 읽은 것을 구별하려면 응용 프로그램은 리턴 값과 errno를 모두 확인해야 합니다.
(EBADF - dir이 유효하지 않음)

Chapter 03_08 디렉토리 닫기 (closedir)

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

int closedir (DIR *dir);
```

파일과 마찬가지로 opendir()로 열린 디렉토리 스트림은 closedir()을 통해 닫아야 합니다.
성공 시 디렉토리 스트림이 닫히고 0을 리턴합니다. 실패 시 -1을 리턴합니다.

Chapter 03_09 디렉토리 실습

프로그램의 인자로 디렉토리 경로와 파일명을 받고 해당 디렉토리안에 해당 파일이 존재하는지 여부를 판별한다.

```
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <stdbool.h>
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[])
{
        struct dirent *entry;
        DIR *dir;
        bool found = false;

        if (argc < 3) {
                printf("Usage: %s <dir> <file>\n", argv[0]);
                return -1;
        }

        dir = opendir(argv[1]);
        if (dir == NULL) {
                perror("opendir error: ");
                return -1;
        }
```

```
        errno = 0;
        while ((entry = readdir(dir)) != NULL) {
                if (strcmp(entry->d_name, argv[2]) == 0) {
                        printf("found! file(%s) exist\n", argv[2]);
                        found = true;
                        break;
                }
        }

        if (errno) {
                perror("readdir error: ");
                closedir(dir);
                return -1;
        }

        if (!found)
                printf("file(%s) doesn't exist!\n", argv[2]);

        closedir(dir);
        return 0;
}
```

Chapter 03_09 디렉토리 실습 (결과)

```
[root@localhost ch8]# gcc -g directory_example.c -o directory_example

[root@localhost ch8]# ./directory_example
Usage: ./directory_example <dir> <file>

[root@localhost ch8]# ./directory_example ./ directory_example.c
found! file(directory_example.c) exist

[root@localhost ch8]# ./directory_example ./ directory_example.h
file(directory_example.h) doesn't exist!
```