Part 02. 리눅스 시스템 프로그래밍

Chapter 06. 메모리 매핑 I/O

진행 내용

Chapter 06_01 메모리 매핑 I/O 개요

리눅스 커널은 애플리케이션이 파일을 메모리에 매핑할 수 있는 인터페이스(mmap)를 제공한다.
(메모리 주소와 파일의 워드 사이 일대일 대응)

이를 활용하면 개발자는 메모리에 상주하는 데이터 처럼(예를 들어 배열처럼) 메모리에 직접 쓰거나 읽음으로써 빠르게 파일에 직접 접근할 수 있다.

mmap 장점
- 파일 내용을 접근할 때 시스템 호출(read, write 등)을 사용하지 않게 되므로 성능향상의 효과가 있다.
- 여러 개의 프로세스가 동일한 파일을 메모리에 매핑한다면 데이터는 모든 프로세스 간에 공유된다.
- 간단한 포인터 조작만으로 메모리 매핑 영역을 탐색 가능하다. (lseek 등 불필요)

mmap 주의점
- 메모리 매핑 공간을 공유해서 사용할 경우 크리티컬 섹션 보호에 주의해야 한다.
- 메모리 매핑은 항상 페이지 크기의 정수배로 이루어진다.
  (예를 들어 페이지 크기가 4KB(4,096Byte)이고 8Byte 파일을 매핑할 경우 4,088Byte가 낭비된다.

Chapter 06_02 mmap

```
#include <sys/mman.h>

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

파일 디스크립터 fd가 가리키는 파일의 offset 위치에서 len 바이트만큼 메모리에 매핑합니다.
addr이 NULL이면 커널은 매핑을 생성 할 주소를 선택합니다. (일반적인 방법)
addr이 NULL이 아닌 경우 커널은 매핑을 배치 할 위치에 대한 힌트로 이를 사용합니다.
Linux에서는 매핑이 가까운 페이지 경계에서 생성됩니다. 새 매핑의 주소는 호출 결과로 반환됩니다.
접근 권한은 prot에 지정하고, 추가적인 동작은 flags에 명시합니다.

prot
PROT_NONE : 접근이 불가능한 페이지(거의 사용되지 않음)
PROT_READ : 읽기가 가능한 페이지
PROT_WRITE : 쓰기가 가능한 페이지
PROT_EXEC : 실행이 가능한 페이지

메모리 보호 정책은 파일 디스크립터 모드와 충돌하면 안됩니다.
(예를 들어 파일은 읽기전용으로 열었는데 prot에 PROT_WRITE 플래그를 지정하면 안됩니다.)

flags
MAP_FIXED : 원하는 메모리 시작 번지를 지정할 때 사용, 커널이 해당 주소를 확보하지 못하면 호출 실패.
MAP_PRIVATE : 매핑이 공유되지 않음을 명시, 처음 생성시에만 파일 내용을 메모리에 복사,
이 후 매핑된 내용에 변경이 발생하더라도 실제 파일이나 다른 프로세스에 반영하지 않는다.
MAP_SHARED : 같은 파일을 매핑한 모든 프로세스와 공유한다.
매핑된 페이지에 쓰기를 하면 실제 파일에도 동일한 내용을 기록한다.

Chapter 06_02 mmap

참고사항
파일 디스크립터를 매핑하면 해당 파일의 참조카운터 증가
파일을 매핑한 후에 파일 디스크립터를 닫아도 매핑된 주소에 접근 가능
파일의 매핑을 해제하거나 프로세스가 종료될 때 감소

```
void *p;

p = mmap(0, len, PROT_READ, MAP_SHARED, fd, 0);
if (p == MAP_FAILED)
        perror("mmap error: ");
```

mmap()이 성공하면 매핑된 주소를 리턴한다.
실패하면 MAP_FAILED를 리턴하고, errno을 적절한 값으로 설정한다.

errno
EACCES : 파일 디스크립터가 prot이나 flags와 출돌을 일으키는 모드로 열렸다.
EAGAIN : 파일이 잠긴 상태
EBADF : 파일디스크립터가 유효하지 않다.
EINVAL : addr, len, off 중 하나 이상의 인자가 유효하지 않다.
ENFILE : 시스템 열린 파일 개수 제한
ENODEV : 파일시스템이 메모리 매핑을 지원하지 않음.
ENOMEM : 사용 가능한 메모리가 부족하다.
EOVERFLOW : addr+len 값이 주소 공간의 크기를 초과
EPERM : PROT_EXEC 설정, 파일 시스템이 noexec 모드로 마운트

Chapter 06_03 munmap

```
#include <sys/mman.h>

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

addr에서 시작해서 len 바이트만큼 사이 프로세스 주소 공간 내에 존재하는 매핑을 해제한다.
일반적으로 mmap()의 반환값과 mmap()을 실행할 때 사용했던 len 값을 munmap() 인자로 전달한다.

```
if (munmap(addr, len) == -1)
        perror("munmap error: ");
```

munmap()은 성공 시 0을 리턴하고, 실패하면 -1을 리턴하고 errno을 적절한 값으로 설정한다.

errno
EINVAL : 주어진 인자가 유효하지 않다.

Chapter 06_04 mremap

```
#define _GNU_SOURCE

#include <sys/mman.h>

void * mremap (void *addr, size_t old_size, size_t new_size, unsigned long flags);
```

mremap()은 리눅스 전용으로 주어진 메모리 매핑 영역의 크기를 조절한다.
mremap()은 addr에서 시작해서 old_size 만큼 매핑된 영역을 new_size 만큼의 크기로 변경한다.

flags는 0이거나 MREMAP_MAYMOVE
MREMAP_MAYMOVE : 크기 변경 요청을 수행할 때 필요하다면 매핑의 위치를 이동해도 좋다.

errno
EAGAIN : 파일이 잠긴 상태
EFAULT : 유효하지 않은 주소공간이거나 페이지 매핑 시 문제 발생
FINVAL : 인자가 유효하지 않다.
ENOMEM : 프로세스의 주소 공간에 메모리가 충분하지 않다.

Chapter 06_05 msync

#include <sys/mman.h>

int msync (void *addr, size_t len, int flags);

msync()는 mmap()으로 매핑된 파일에 대한 변경 내용을 디스크에 기록하여 파일과 매핑을 동기화한다.
일반적으로 addr, len 인자는 mmap()에서 반환한 값과 len 인자를 사용한다.

msync()를 호출하지 않으면 매핑이 해제되기 전까지 매핑된 메모리에 쓰여진 내용이 디스크로 반영된다는 보장이 없다.
여러 프로세스가 mmap()을 통해 공유하는 경우 msync()를 수행 후 munmap()을 수행하는 것이 좋다.

flags
MS_SYNC : 디스크에 모든 페이지를 기록하기 전까지 msync()는 리턴하지 않는다.
MS_AYNC : 비동기 방식, 갱신 작업은 예약되고 msync()는 즉시 리턴한다.
MS_INVAILDATE : 매핑의 캐시 복사본을 모두 무효화한다.

```
if (msync(addr, len, MS_ASYNC) == -1)
        perror("msync error: ");
```

msync() 호출이 성공하면 0 리턴, 실패 시 -1 리턴 errno 설정

errno
EINVAL : 인자가 유효하지 않다.
ENOMEM : 주어진 메모리 영역이 매핑되지 않았다.

Chapter 06_06 페이징 개요

페이징(paging) 기법이란 프로세스의 주소 공간을 특정 사이즈의 페이지 단위로 나누어
물리 메모리에 불연속적으로 저장하는 기법을 말한다.

페이지(page)는 MMU(Memory Management Unit)에서 사용하는 최소 단위이다.

주소공간을 페이지 단위로 나누고, 실제 메모리 공간은 페이지 크기와 같은 프레임으로 나누어 사용한다.
어떤 프로세스가 현재 참조하고 있는 페이지가 메모리 상에 존재한다면 그 프로세스는 수행될 수 있다.
반대로 메모리 상에 없다면(page fault) 해당 페이지를 디스크로부터 읽어와서 페이지 프레임의 한 블록에 저장한다.

페이지의 크기는 하드웨어에 의해 정의된다.
일반적으로 x86/amd64 에서는 4KB(4,096Byte), ia64 에서는 8KB(8,192Byte) 크기를 가진다.

sysconf() 함수를 통해 페이지 사이즈를 확인할 수 있다.

```
#include <unistd.h>

long sysconf (int name);
```

sysconf()는 설정 항목 name의 값을 리턴하거나 name이 유효하지 않을 경우 -1을 리턴한다.
페이지 크기를 바이트 단위로 _SC_PAGESIZE로 정의한다.

```
long page_size = sysconf(_SC_PAGESIZE);
```

Chapter 06_07 실습

```
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
        int fd;
        struct stat sb;
        off_t len;
        char *p;

        if (argc < 2) {
                printf("Usage: %s <file>\n", argv[0]);
                return -1;
        }

        fd = open(argv[1], O_RDONLY);
        if (fd == -1) {
                perror("open error: ");
                return -1;
        }
```

```
        if (fstat(fd, &sb) == -1) {
                perror("fstat error: ");
                return -1;
        }

        if (!S_ISREG(sb.st_mode)) {
                printf("file %s is not a file\n", argv[1]);
                return 1;
        }

        p = mmap(0, sb.st_size, PROT_READ, MAP_SHARED, fd, 0);
        if (p == MAP_FAILED) {
                perror("mmap error: ");
                return -1;
        }

        if (close(fd) == -1) {
                perror("close error: ");
                return -1;
        }

        for (len = 0; len < sb.st_size; len++) {
                putchar(p[len]);
        }
```

Chapter 06_07 실습

```
        if (munmap(p, sb.st_size) == -1) {
                perror("munmap error: ");
                return -1;
        }

        return 0;
}
```

```
[root@localhost ch13]# gcc -g mmap_example.c -o mmap_example

[root@localhost ch13]# ./mmap_example
Usage: ./mmap_example <file>

[root@localhost ch13]# cat test.file
Hello World!

[root@localhost ch13]# ./mmap_example test.file
Hello World!
```