10주차 2차시 메모리 매핑

【학습목표】

1. 메모리 매핑의 기본 개념을 설명할 수 있다.
2. 메모리 매핑 함수를 사용할 수 있다.

학습내용1 : 메모리 매핑

1. 메모리 매핑의 개념

파일을 프로세스의 메모리에 매핑
프로세스에 전달할 데이터를 저장한 파일을 직접 프로세스의 가상 주소 공간으로 매핑
read, write 함수를 사용하지 않고도 프로그램 내부에서 정의한 변수를 사용해 파일에서 데이터를 읽거나 쓸 수 있음

2. 메모리 매핑과 기존 방식의 비교

* 기존 방식
파일을 읽고 필요에 따라 파일의 오프셋을 이동시키고 read() 함수 호출해 데이터를 buf로 읽어와서 작업

```
fd = open(...)
lseek(fd, offset, whencd);
read(fd, buf, len);
```

* 메모리매핑 함수 사용
파일을 열고 열린 파일의 내용을 mmap() 함수를 사용해 메모리에 매핑하고, 이후의 address가 가리키는 메모리 영역의 데이터를 대상으로 수행,
매번 read() 함수로 데이터를 읽어올 필요가 없다.

```
#include <sys/mman.h>
fd = open(...)
addr = mmap( (caddr_t)0, len, (PROT_READ|PROT_WRITE), MAP_PRIVATE, fd, offset);
```

학습내용2 : 메모리 매핑 함수

1. 메모리 매핑: mmap(2)

fildes가 가리키는 파일에서 off로 지정한 오프셋부터 len크기만큼 데이터를 읽어 addr가 가리키는 메모리 공간에 매핑
성공 시 : 매핑된 메모리 시작 주소 리턴
실패 시 : 상수 MAP_FAILED 리턴

```
#include <sys/mman.h>
void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);
```

addr : 매핑할 메모리 주소 혹은 NULL
len : 메모리 공간의 크기
prot : 보호 모드
flages : 매핑된 데이터의 처리 방법을 지정하는 상수
fildes : 파일 기술자
off : 파일 오프셋

* 시스템 호출의 오류 처리방법
① PROT_READ : 매핑된 파일을 읽기만 함
② PROT_WRITE : 매핑된 파일에 쓰기 허용
③ PROT_EXEC : 매핑된 파일을 실행가능
④ PROT_NONE : 매핑된 파일에 접근 불가
prot에 PROT_WRITE를 지정하려면 flags에 MAP_PRIVATE를 지정하고, 파일을 쓰기 가능 상태로 열어야함

* flags : 매핑된 데이터를 처리하기 위한 정보 저장
① MAP_SHARED : 다른 사용자와 데이터의 변경 내용공유
② MAP_PRIVATE : 데이터의 변경 내용 공유 안함
③ MAP_FIXED : 매핑할 주소를 정확히 지정(권장 안함)
④ MAP_NORESERVE : 매핑된 데이터를 복사해 놓기 위한 스왑영역 할당 안함
⑤ MAP_ANON : 익명의 메모리 영역 주소를 리턴
⑥ MAP_ALIGN : 메모리 정렬 지정
⑦ MAP_TEXT : 매핑된 메모리 영역을 명령을 실행하는 영역으로 사용
⑧ MAP_INITDATA : 초기 데이터 영역으로 사용

* mmap 함수 사용하기(1)

```
...
08  int main(int argc, char *argv[]) {
09      int fd;
10      caddr_t addr;
11      struct stat statbuf;
12
13      if (argc != 2) {
14          fprintf(stderr, "Usage : %s filename\n", argv[0]);
15          exit(1);
16      }
17
18      if (stat(argv[1], &statbuf) == -1) {
19          perror("stat");
20          exit(1);
21      }
22
23      if ((fd = open(argv[1], O_RDWR)) == -1) {
24          perror("open");
25          exit(1);
26      }
27
28      addr = mmap(NULL, statbuf.st_size, PROT_READ|PROT_WRITE,
29                  MAP_SHARED, fd, (off_t)0);
30      if (addr == MAP_FAILED) {
31          perror("mmap");
32          exit(1);
33      }
34      close(fd);
35
36      printf("%s", addr);
37
38      return 0;
39  }
```

명령행 인자로 매핑할 파일명 입력

파일 내용을 메모리에 매핑

매핑한 파일내용 출력

```
# cat mmap.dat
HANBIT
BOOK
# ex8_1.out
Usage : ex8_1.out filename
# ex8_1.out mmap.dat
HANBIT
BOOK
```

2. 메모리 매핑 해제 함수

* 메모리 매핑 해제: munmap(2)
addr이 가리키는 영역에 len 크기만큼 할당해 매핑한 메모리 해제
해제한 메모리에 접근하면 SIGSEGV 또는 SIGBUS 시그널 발생

```
#include <sys/man.h>
int munmap(void *addr, size_t len);
```

addr : 매핑된 메모리 시작 주소
len : 메모리 영역의 크기

* munmap 함수 사용하기

```
...
08  int main(int argc, char *argv[]) {
09      int fd;
10      caddr_t addr;
11      struct stat statbuf;
12
13      if (argc != 2) {
14          fprintf(stderr, "Usage : %s filename\n", argv[0]);
15          exit(1);
16      }
17
18      if (stat(argv[1], &statbuf) == -1) {
19          perror("stat");
20          exit(1);
21  }
22
23      if ((fd = open(argv[1], O_RDWR)) == -1) {
24          perror("open");
25          exit(1);
26      }
27
28      addr = mmap(NULL, statbuf.st_size, PROT_READ|PROT_WRITE,
29                  MAP_SHARED, fd, (off_t)0);
30      if (addr == MAP_FAILED) {
31          perror("mmap");
32          exit(1);
33      }
34      close(fd);
35
36      printf("%s", addr);
37
38      if (munmap(addr, statbuf.st_size) == -1) {
39          perror("munmap");
40          exit(1);
41      }
42
43      printf("%s", addr);
44
45      return 0;
46  }
```

파일 내용을 메모리에 매핑

메모리 매핑 해제

매핑이 해제된 메모리에 접근

```
# ex8_2.out mmap.dat
HANBIT
BOOK
세그멘테이션 결함(Segmentation Fault)(코어 덤프)
```

3. 메모리 매핑의 보호모드 변경

* 보호모드 변경: mprotect(2)
mmap 함수로 메모리 매핑을 수행할 때 초기값을 설정한 보호모드를 mprotect 함수로 변경 가능
prot에 지정한 보호모드로 변경

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

int mprotect(void *addr, size_t len, int prot);
```

addr : 매핑된 메모리의 시작 주소
prot : 보호 모드
len : 메모리 영역의 크기

4. 파일의 크기 확장 함수

* 파일의 크기와 메모리 매핑
존재하지 않거나 크기가 0인 파일은 메모리 매핑할 수 없음
빈 파일 생성시 파일의 크기를 확장한 후 메모리 매핑을 해야함
truncate(), ftruncate()

* 경로명을 사용한 파일 크기 확장: truncate(3)
path에 지정한 파일의 크기를 length로 지정한 크기로 변경

```
#include <unistd.h>

int truncate(const char *path, off_t length);
```

path : 크기를 변경할 파일의 경로
length : 변경하려는 크기

* 파일 기술자를 사용한 파일 크기 확장: ftruncate(3)
일반 파일과 공유메모리에만 사용가능
이 함수로 디렉토리에 접근하거나 쓰기 권한이 없는 파일에 접근하면 오류 발생

```
#include <unistd.h>

int ftruncate(int fildes, off_t length);
```

fildes : 크기를 변경할 파일의 파일 기술자
length : 변경하려는 크기

* ftruncate 함수 사용하기

```
...
09  int main(void) {
10      int fd, pagesize, length;
11      caddr_t addr;
12
13      pagesize = sysconf(_SC_PAGESIZE);
14      length = 1 * pagesize;
15
16      if ((fd = open("m.dat", O_RDWR | O_CREAT | O_TRUNC, 0666))
== -1) {
17          perror("open");
18          exit(1);
19      }
20
21      if (ftruncate(fd, (off_t) length) == -1) {
22          perror("ftruncate");
23          exit(1);
24      }
25
26      addr = mmap(NULL, length, PROT_READ|PROT_WRITE, MAP_SHARED,
                    fd, (off_t)0);
27      if (addr == MAP_FAILED) {
28          perror("mmap");
29          exit(1);
30      }
31
32      close(fd);
33
34      strcpy(addr, "Ftruncate Test\n");
35
36      return 0;
37  }
```

메모리의 페이지 크기정보 검색

빈 파일의 크기 증가

메모리 매핑

매핑한 메모리에 데이터 쓰기

```
# ls m.dat
m.dat: 해당 파일이나 디렉토리가 없음
# ex8_3.out
# cat m.dat
ftruncate Test
```

【학습정리】

1. 메모리 매핑의 개념
- 파일을 프로세스의 메모리에 매핑
- 프로세스에 전달할 데이터를 저장한 파일을 직접 프로세스의 가상 주소 공간으로 매핑
- read, write 함수를 사용하지 않고도 프로그램 내부에서 정의한 변수를 사용해 파일에서 데이터를 읽거나 쓸 수 있음

2. 메모리 매핑 함수 사용
- 파일을 열고 열린 파일의 내용을 mmap() 함수를 사용해 메모리에 매핑하고, 이후의 address가 가리키는 메모리 영역의 데이터를 대상으로 수행
- 매번 read() 함수로 데이터를 읽어올 필요가 없다.

```
#include <sys/mman.h>
fd = open(...)
addr = mmap( (caddr_t)0, len, (PROT_READ|PROT_WRITE), MAP_PRIVATE, fd, offset);
```

3. 메모리 매핑 함수

기능	함수원형
메모리 매핑	void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);
메모리 매핑 해제	int munmap(void *addr, size_t len);
파일 크기 조정	int truncate(const char *path, off_t length); int ftruncate(int fildes, off_t length);
매핑된 메모리 동기화	int msync(void *addr, size_t len, int flages);