

시그널 연습문제

□ 문제5 ~ 7



메모리 매핑의 개념

□ 메모리 매핑

- 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

□ 메모리 매핑과 기존 방식의 비교

▪ 기존 방식

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

▪ 메모리매핑 함수 사용

```
fd = open(...);  
addr = mmap((caddr_t)0, len, (PROT_READ|PROT_WRITE), MAP_PRIVATE, fd,  
offset);
```

read 함수를 사용하지
않고도 데이터 접근 가
능

메모리 매핑 함수

□ 메모리 매핑: mmap(2)

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

- fildes가 가리키는 파일에서 off로 지정한 오프셋부터 len크기만큼 데이터를 읽어 addr이 가리키는 메모리 공간에 매핑
- prot : 보호모드
 - PROT_READ : 매핑된 파일을 읽기만 함
 - PROT_WRITE : 매핑된 파일에 쓰기 허용
 - PROT_EXEC : 매핑된 파일을 실행가능
 - 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
 - MAP_SHARED : 다른 사용자와 데이터의 변경 내용공유
 - MAP_PRIVATE : 데이터의 변경 내용 공유 안함
 - MAP_FIXED : 매핑할 주소를 정확히 지정(권장 안함)
 - MAP_NORESERVE : 매핑된 데이터를 복사해 놓기 위한 스왑영역 할당 안함
 - MAP_ANON : 익명의 메모리 영역 주소를 리턴
 - MAP_ALIGN : 메모리 정렬 지정
 - MAP_TEXT : 매핑된 메모리 영역을 명령을 실행하는 영역으로 사용
 - MAP_INITDATA : 초기 데이터 영역으로 사용



```
...
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
```

명령행 인자로 매핑할
파일명 입력

(다음 쪽)

[예제 8-1] mmap 함수 사용하기(2)

ex8_1.c

```
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 }
```

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
```

메모리 매핑 해제 함수

□ 메모리 매핑 해제: munmap(2)

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

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

- **addr0이 가리키는 영역에 len 크기만큼 할당해 매핑한 메모리 해제**
- **해제한 메모리에 접근하면 SIGSEGV 또는 SIGBUS 시그널 발생**

[예제 8-2] munmap 함수 사용하기

ex8_2.c

```
...
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) {
```

[예제 8-2] munmap 함수 사용하기(2) – test2.c

ex8_2.c

```
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)

메모리 매핑의 보호모드 변경

□ 보호모드 변경: mprotect(2)

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

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

- mmap 함수로 메모리 매핑을 수행할 때 초기값을 설정한 보호모드를 mprotect 함수로 변경 가능
- prot에 지정한 보호모드로 변경
 - PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE



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

□ 경로명을 사용한 파일 크기 확장: truncate(3)

```
#include <unistd.h>

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

- path에 지정한 파일의 크기를 length로 지정한 크기로 변경

□ 파일 기술자를 사용한 파일 크기 확장: ftruncate(3)

```
#include <unistd.h>

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

- 일반 파일과 공유메모리에만 사용가능
- 이 함수로 디렉토리에 접근하거나 쓰기 권한이 없는 파일에 접근하면 오류 발생



```
...
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))
17 == -1) {
18         perror("open");
19         exit(1);
20     }
21
22     if (ftruncate(fd, (off_t) length) == -1) {
23         perror("ftruncate");
24         exit(1);
25     }
```

메모리의 페이지 크기정보 검색

빈 파일의 크기 증가



[예제 8-3] ftruncate 함수 사용하기(2)

ex8_3.c

```
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 }
```

메모리 매핑

매핑한 메모리에 데이터 쓰기

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



매핑된 메모리 동기화

□ 매핑된 메모리 동기화

- 매핑된 메모리의 내용과 백업 내용이 일치하도록 동기화 필요

□ 매핑된 메모리 동기화: `msync(3)`

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

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

- `addr`로 시작하는 메모리 영역에서 `len` 길이만큼의 내용을 백업저장장치에 기록
- `flags` : 함수의 동작 지시
 - `MS_ASYNC` : 비동기 쓰기 작업
 - `MS_SYNC` : 쓰기 작업을 완료할 때까지 `msync` 함수는 리턴 안함
 - `MS_INVALIDATE` : 메모리에 복사되어 있는 내용을 무효화



```
...
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     }
```

파일의 상세 정보 검색



[예제 8-4] msync 함수 사용하기(2)

ex8_4.c

```
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     printf("-----\n");
39     addr[0] = 'D';
40     printf("%s", addr);
41
42     msync(addr, statbuf.st_size, MS_SYNC);
43
44     return 0;
45 }
```

메모리 매핑

매핑된 내용 출력

매핑된 내용 수정

수정된 내용 동기화

```
# cat mmap.dat
HANBIT
BOOK
# ex8_4.out mmap.dat
HANBIT
BOOK
-----
DANBIT
BOOK
# cat mmap.dat
DANBIT
BOOK
```

□ 메모리 매핑을 이용한 데이터 교환

- 부모 프로세스와 자식 프로세스가 메모리 매핑을 사용하여 데이터 교환 가능

```
...
09  int main(int argc, char *argv[]) {
10      int fd;
11      pid_t pid;
12      caddr_t addr;
13      struct stat statbuf;
14
15      if (argc != 2) {
16          fprintf(stderr, "Usage : %s filename\n", argv[0]);
17          exit(1);
18      }
19
20      if (stat(argv[1], &statbuf) == -1) {
21          perror("stat");
22          exit(1);
23      }
24
```

```
25     if ((fd = open(argv[1], O_RDWR)) == -1) {
26         perror("open");
27         exit(1);
28     }
29
30     addr = mmap(NULL, statbuf.st_size, PROT_READ|PROT_WRITE,
31                MAP_SHARED, fd, (off_t)0);
32     if (addr == MAP_FAILED) {
33         perror("mmap");
34         exit(1);
35     }
36     close(fd);
37
38     switch (pid = fork()) {
39         case -1 : /* fork failed */
40             perror("fork");
41             exit(1);
42             break;
```

메모리 매핑

fork 함수로 자식 프로세스 생성




```
43     case 0 :    /* child process */
44         printf("1. Child Process : addr=%s", addr);
45         sleep(1);
46         addr[0] = 'x';
47         printf("2. Child Process : addr=%s", addr);
48         sleep(2);
49         printf("3. Child Process : addr=%s", addr);
50         break;
51     default : /* parent process */
52         printf("1. Parent process : addr=%s", addr);
53         sleep(2);
54         printf("2. Parent process : addr=%s", addr);
55         addr[1] = 'y';
56         printf("3. Parent process : addr=%s", addr);
57         break;
58     }
59
60     return 0;
61 }
```

자식 프로세스가 매핑된 내용 수정

부모 프로세스가
매핑된 내용 수정

```
# cat mmap.dat
HANBIT BOOK
# ex8_5.out mmap.dat
1. Child Process : addr=HANBIT BOOK
1. Parent process : addr=HANBIT BOOK
2. Child Process : addr=xANBIT BOOK
2. Parent process : addr=xANBIT BOOK
3. Parent process : addr=xyNBIT BOOK
3. Child Process : addr=xyNBIT BOOK
# cat mmap.dat
xyNBIT BOOK
#
```

