10

메모리 매핑 (Memory mapped File)

고급 프로그래밍

학습목표

- 메모리 매핑을 이용한 IPC 기법 이해
  - 메모리 맵드 I/O
  - 메모리 맵드 파일
- 메모리 매핑 함수를 사용해 프로그램 작성
  - 메모리 매핑의 개념
  - 메모리 매핑 함수
  - 메모리 매핑 해제 함수
  - 메모리 매핑의 보호모드 변경
  - 파일의 크기 확장
  - 매핑된 메모리 동기화
  - 데이터 교환하기

Conventional v.s. Memory-mapped 파일 I/O

- In traditional file I/O involving read,
  - data is copied from the **disk to a kernel buffer**,
  - then the **kernel buffer is copied into the process's heap space** for use.
- In memory-mapped file I/O,
  - data is copied **from the disk straight into the process's address space**, into the segment where the file is mapped.
  - The file is then accessed by references to memory locations with pointers.
- mmap는 프로세스의 주소공간을 파일에 대응
  - 파일은 O/S 전역적 자원으로 타 프로세스와 공유 가능
- Inter-Process Communication (IPC)
  - Shared memory (Memory mapped file기법을 이용)
  - Message queue
  - Shared file
  - Pipe

메모리 매핑

메모리 맵 파일(Memory mapped file, MMF, 메모리 사상 파일)

- 운영 체제에서 파일을 다루는 방법 중 하나, 표준 I/O device와 다름!
- 물리 디스크 파일, 장치, 공유 메모리 객체와 같이 운영 체제에서 파일로 다루는 모든 대상에 대해서 사용 가능
- 메모리 맵 파일을 통해 프로세스의 가상 메모리 주소 공간에 파일을 매핑한 뒤 가상 메모리 주소에 직접 접근하는 것으로 파일 읽기/쓰기를 대신

메모리 맵드 파일의 용도

- 프로세스를 실행할 때 실행 파일의 각 세그먼트를 메모리에 사상하기 위해 메모리 맵 파일을 이용
- 하나의 프로세스가 동시에 여러 번 동작할 때 실행 코드와 같은 읽기 전용 세그먼트를 공유할 수 있어 중복된 내용만큼의 메모리를 절약
- 메모리 맵 파일이 가지는 지연 적재 특징 덕분에 실행 파일에서 사용하지 않는 부분이 있을 때도 메모리를 최적화하여 사용 가능
- 파일의 내용이 프로그래밍 언어에서 직접 다룰 수 있는 구조체와 같은 형식을 가지고 작성되어 있으면, 메모리 맵 파일을 이용해 자료에 바로 접근하는 방법을 사용 가능
- 프로세스 간 통신으로 메모리 맵 파일은 공유 메모리를 사용

Memory-mapped 파일 - 특징

- 생성된 메모리 맵을 포인터를 이용하여 쉽게 사용 가능
- 파일로 연결하여 사용 시 메모리-파일 사이의 동기화 간편
- IPC(프로세스간 통신)로 활용 가능 – 동기화 문제
- 대용량의 데이터 사용 시 성능이 향상 (No faster but efficient!)

* 주의점

- 메모리 맵은 바로 파일을 처리하는 게 아니라 가상 메모리로 활용되는 페이지에 맵핑하는 방식
  - 파일과 해당 메모리 맵이 된 페이지가 다른 공간
  - 커널에 의해 여유 시간에 동기화(둘의 데이터가 같아지는..)가 될 때까지 서로 다른 데이터를 가질 수 있음
  - 동기화에 대한 주의 필요
- 개발자가 직접 커널에 동기화를 명령 할 수 있는 함수
  - msync() – fsync()>
- IPC로 사용 할 때에도 프로세스간 동기화에 대한 주의 필요

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 가능

* 메모리 맵 생성 함수 - mmap()

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

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

- *void *addr*
  - 할당 받기 원하는 메모리 주소. 보통 0을 써서 커널이 적합한 공간을 임의로 할당해 주소를 받을 수 있고, 직접 입력하여 사용해도 된다. 하지만 직접 입력하는 경우 해당 시스템의 페이지 (배수)단위로 주소 값을 설정해줘야 한다.
  - NULL로 입력하고 매핑된 메모리 시작 주소를 반환 받는 형식도 가능
- *size_t length*
  - 메모리 맵을 할 크기. 바이트 단위로 설정한다.
- *int prot*
  - 메모리 보호 메커니즘 - 플래그 형식이므로 비트 연산으로 복수 속성으로 지정 가능
  - + PROT_EXEC: 해당 페이지 실행 가능
  - + PROT_READ: 해당 페이지 읽기 가능
  - + PROT_WRITE: 해당 페이지 쓰기 가능
  - + PROT_NONE: 해당 페이지 접근 불가
  - => 매핑할 파일 디스크립터와 속성이 같아야 함
  - OR 연산자로 중복 가능
  - prot에 PROT_WRITE를 지정하려면 flags에 MAP_PRIVATE를 지정하고, 파일을 쓰기 가능 상태로 열어야 함

mmap()

- *– int flags*
  - + MAP_SHARED; 공유 메모리 맵 방식.
  - + MAP_PRIVATE; 복사 메모리 맵 방식.
  - + MAP_FIXED ; 메모리 시작 번지 지정 시 사용.
  - 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
  - =>MAP_SHARED/MAP_PRIVATE 둘 중에 반드시 하나는 지정
  - OR 연산자로 중복 가능
- *– int fd*
  - 메모리 맵 방식을 사용할 파일 디스크립터.(파일 혹은 디바이스)
- *– ott_t offset*
  - 해당 파일 디스크립터에서 메모리 맵을 시작할 오프셋 값.
- *+ return value*
  - 메모리 맵핑된 가상 메모리 시작 주소. 실패 시 MAP_FAILED가 발생하고 errno에 해당 상황에 대한 값이 설정된다.

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
(다음 쪽)
```

명령행 인자로 매핑할 파일명 입력

mmap 함수 사용하기(2)

```
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

열린 파일을 닫아도 매핑된 주소를 통해 파일 내용 접근 가능

매핑한 파일내용 출력

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

메모리 맵 해제 - munmap()

- 메모리 맵을 사용하고 자원을 해제 할 때 사용

```
#include <sys/mman.h>
int munmap(void *start, size_t length);
```

- *void *start*
  - 메모리 맵핑이 시작된 주소. mmap()의 반환 값을 넣으면 된다.
- *size_t length*
  - 메모리 맵을 한 길이. mmap() 사용시 size_t length 인자와 크기를 주면 된다.
- *+ return value*
  - 성공 0, 실패 -1
- addr이 가리키는 영역에 len 크기만큼 할당해 매핑한 메모리 해제
- 해제한 메모리에 접근하면 SIGSEGV 또는 SIGBUS 시그널 발생

메모리 매핑 해제 함수

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      }
```

munmap 함수 사용하기(2)

```
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

파일의 크기 확장/축소 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

ftruncate 함수 사용하기(1)

```
...
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
```

메모리의 페이지 크기정보 검색

빈 파일의 크기 증가

ftruncate 함수 사용하기(2)

```
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  }
```

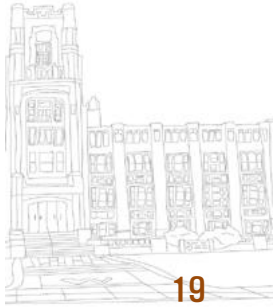
메모리 매핑

매핑한 메모리에 데이터 쓰기

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

메모리 맵과 파일의 동기화 – msync()

- 매핑된 메모리의 내용과 백업 내용을 일치시키는 명령 수행
- 메모리 맵에 데이터를 갱신해도 바로 파일과 동기화가 이루어지는 것이 아님
- 커널이 여유 있을 때 동기화를 수행
- 개발자가 직접 동기화를 보장하고 싶을 때 사용
- munmap()을 하여 메모리 맵을 해제 할 때에도 동기화를 해주면 데이터가 보장 됨
- 매핑된 메모리 공간 v.s. 백업 공간의 동기화
  - MAP_SHARED mode: 백업 저장장치로 file
  - MAP_PRIVATE mode: 백업 저장장치로 swap space
  - msync () 사용
- wait ()와 시그널 함수 등과 조합으로 사용



msync()

int msync(void *start, size_t length, int flags);

- *void *start*
  - mmap()를 통해 리턴 받은 메모리 맵의 시작 주소.
- *size_t length*
  - 동기화를 할 길이. 시작 주소로 부터 길이를 지정
- *int flags*
  - + MS_ASYNC
    - 동기화(Memory->File)하라는 명령만 내리고 결과에 관계 없이 바로 리턴
    - 동기화 여부 알 수 없음
  - + MS_SYNC
    - 동기화(Memory->File)가 될 때까지 블럭 상태로 대기
  - + MS_INVALIDATE
    - 현재 메모리 맵을 무효화하고 파일의 데이터로 갱신. 즉 File->Memory

전형적 사용 예제

```
#define MMAP_FILENAME "test_mmap"
#define MMAP_SIZE 64
...
int main()
{
   int fd;
   char *p_mmap; //메모리 맵으로 사용할 데이터 타입
  ...
   fd = open(MMAP_FILENAME, O_RDWR|O_CREAT, 0664);
  ...
   p_mmap = (char*) mmap ((void*)0, MMAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
   //공유 메모리 방식을 사용한다. 읽기/쓰기 가능
  ...
   //memcpy(); 등으로 메모리 맵 데이터 갱신.
   // memset(); 으로 메모리 데이터 초기화
  ...
   msync(p_mmap, MAP_SIZE, MS_SYNC);
  ...
   munmap(p_mmap);
   return 0;
}
```

msync 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      }
```

파일의 상세 정보 검색

msync 함수 사용하기(2)

```
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
```

데이터 교환하기(1)

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
```

데이터 교환하기(2)

```
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

데이터 교환하기(3)

```
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
#
```

기타 메모리 관련 함수

- 메모리 영역을 단순히 이진 데이터로 복사
- 동일 영역, 즉 자기자신을 자기 자신에 복사는 불가
- 유사한 함수로 memmove() – 동일영역 복사 가능

void *memcpy(void *dest, const void *src, size_t n);

- #include <string.h>
- **void *dest – 복사될 메모리의 포인터**
- **void *src – 복사할 메모리 포인터**
- **size_t n – 복사할 바이트 수**

```
#include <stdio.h>
#include <string.h>

int main (void)
{
   char *ptr_sour =  "Korea Polytechnic University";
   char *ptr_dest;

   ptr_dest = (char *)malloc( 29);
   memcpy (ptr_dest, ptr_sour, strlen (ptr_sour)+1); // NULL까지 포함하기 위해 +1
   printf("%s₩n", ptr_dest);
   free (ptr_dest);

 return 0;
}
```

기타 메모리 관련 함수

- memset():
  - 메모리 데이터 초기화 – malloc() 이나 calloc() 에서 할당 받은 메모리를 특정 값으로 초기화
  - 보통 어떤 작업을 하기 전에 NULL로 초기화할 때 많이 사용
  - 데이터를 읽어 들이거나 어떤 함수를 호출 후 메모리에 입력된 값을 처리하는 경우 미리 메모리를 초기화를 하는 것이 디버깅에 유리

- memchr() : 메모리의 특정한 블록에 문자 기록
- memcmp() : 두 메모리 블록을 비교

실습:

- 교재 359쪽
- 연습문제 4, 5
- 연습문제 6