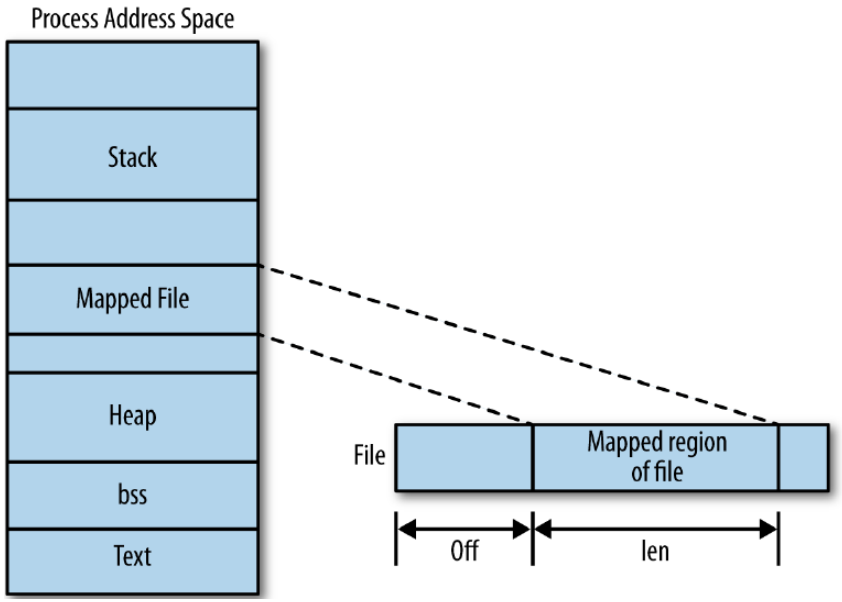
Chapter 8

메모리 매핑

Memory Mapping

Mapping Files into Memory

- **File을 프로세스의 가상 메모리 공간으로 매핑**
  - File I/O system call 사용하지 않고, 접근 가능



<Image from "Linux system programming", Robert Love>

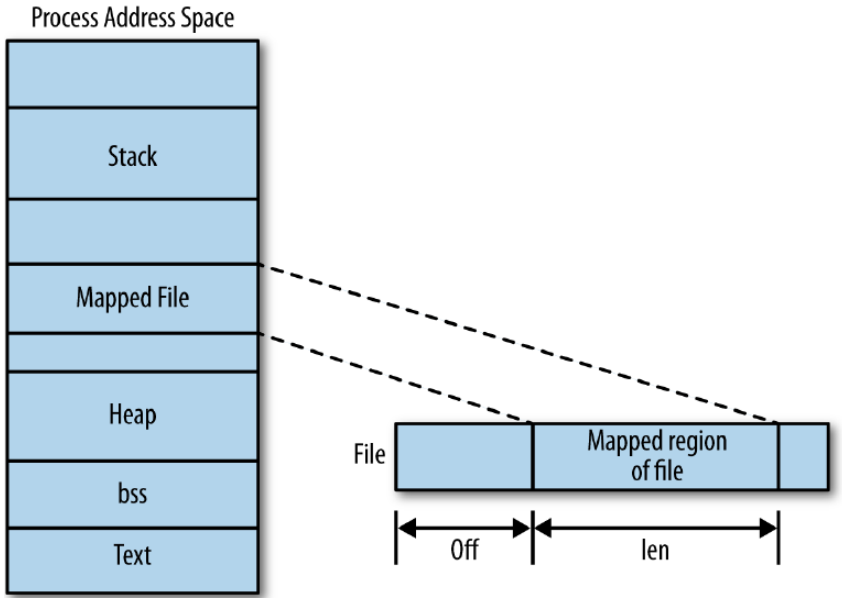
Mapping a File to Memory

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

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

$ man -s 2 mmap

- **addr**
  - 매핑할 메모리 주소 hint (일반적으로 NULL 사용)
- **length**
  - 매핑 할 길이 (단위 : bytes)
- **prot**
  - Memory protection mode
- **flags**
  - 매핑 형태와 동작 방식 지정
- **fd**
  - 매핑할 파일의 file descriptor
- **offset**
  - 매핑을 시작 지점을 지정하는 offset
- Return : **할당된 메모리 주소**



<Image from "Linux system programming", Robert Love>

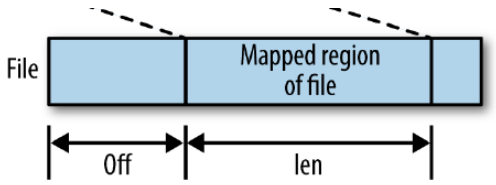
Mapping a File to Memory

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

$ man -s 2 mmap

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

- **length & offset**
  - 파일의 시작에서 offset 지점 부터 length 만큼 매핑
  - Offset은 반드시 page size의 배수 여야 함
- **Mapping은 |Page| 단위로 이루어짐**
  - System page size 얻기
    - sysconf(_SC_PAGESIZE)
    - getpagesize(2) for Linux



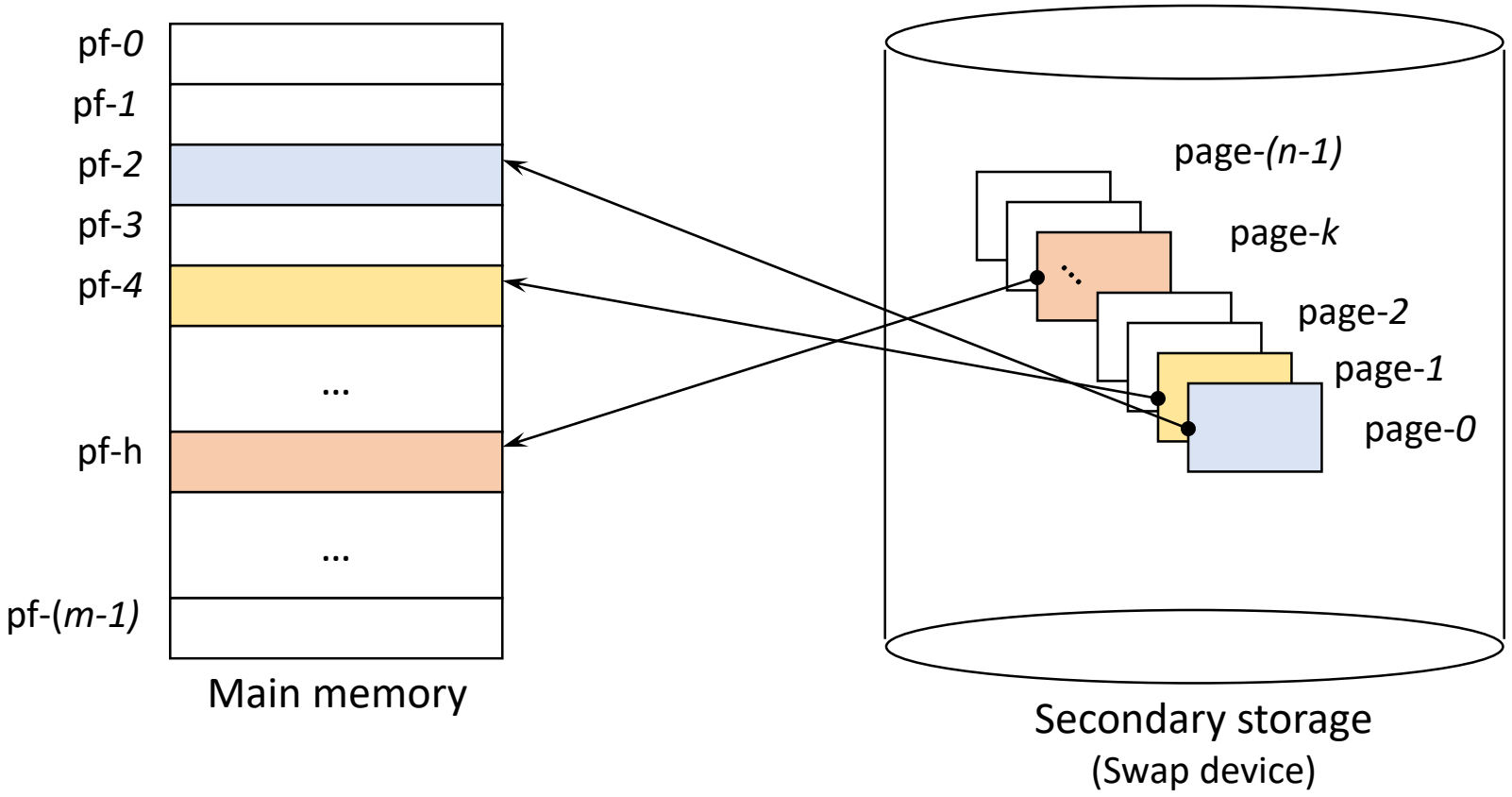
<Image from “Linux system programming”, Robert Love>

Paging System

- 프로그램을 같은 크기의 블록으로 분할 (Pages)

- Terminologies
  - Page
    - 프로그램의 분할된 block
  - Page frame
    - 메모리의 분할 영역
    - Page와 같은 크기로 분할

Paging System



Mapping a File to Memory

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

$ man -s 2 mmap

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

- **Memory protection (prot)**
  - PROT_NONE : 접근할 수 없음
  - PROT_READ
  - PROT_WRITE
  - PROT_EXEC

  OR(|)로 조합 가능

- **FD의 open mode와 충돌이 나면 안됨**

Mapping a File to Memory

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

$ man -s 2 mmap

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

- **Flags**
  - MAP_FIXED : Mapping 할 주소를 지정 (addr)
    - 해당 영역에 다른 mapping이 있으면, 그것을 해제 후 mapping
  - MAP_PRIVATE : 변경 된 내용이 공유 및 반영되지 않는다
    - Mapping 된 메모리의 사본에 작업
  - MAP_SHARED : Mapping 된 내용이 공유 된다
    - Mapping 에 대해 write 시, 파일에 반영 됨

Mapping a File to Memory

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

$ man -s 2 mmap

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

- Return
  - Success : **Mapping address**
  - Fail : **MAP_FAILD**

- **관련 Signal**
  - **SIGBUS : mapping region is invalid**
  - **SIGSEGV : try to write to a read-only mapping region**

Removing a Mapping

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

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

$ man -s 2 munmap

- **addr** : Mapping 시작 주소
- **length** : Mapping 길이
- Return
  - **0** : Success, **-1** : fail

Mapping a File to Memory

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

#define CHECK_MMAP_SUCCESS(_addr)\
if(_addr == MAP_FAILED) {\
    perror("mmap");\
    exit(1);\
}
#define printAddrs(msg) \
{printf("%s\n", msg); printf("addr1 = %s", addr1); printf("addr2 = %s", addr2);}

int main(int argc, char *argv[]) {
     int fd;
     caddr_t addr1, addr2;
     char fileName[255] = "input.dat";

     if (argc > 1) strcpy(fileName, argv[1]);

     if ((fd = open(fileName, O_RDWR)) == -1) {
          perror("open");
          exit(1);
     }
```

Mapping a File to Memory

```
    int pageSize = getpagesize();

    addr1 = mmap(NULL, pageSize, PROT_READ | PROT_WRITE, MAP_SHARED, fd, (off_t)0);
    CHECK_MMAP_SUCCESS(addr1);

    addr2 = mmap(NULL, pageSize, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, (off_t)0);
    CHECK_MMAP_SUCCESS(addr2);

    close(fd);
    printf("%s", addr1);

    addr1[0] = '1';printAddrs("After addr1[0]=1");
    addr2[0] = '2'; printAddrs("After addr2[0]=2"); // CoW
    addr1[0] = '3'; printAddrs("After addr1[0]=3");

    execlp("cat", "cat", fileName, NULL);

    return 0;
}
```

여러 조합으로
테스트 해보기

Copy-on-Write (CoW)

```
addr1[0] = '1';printAddrs("After addr1[0]=1");
addr2[0] = '2'; printAddrs("After addr2[0]=2"); // CoW
addr1[0] = '3'; printAddrs("After addr1[0]=3");
```

- **Write 시, 복사본 생성**
  - 그전까지 복사본 생성을 미룬다
  - 불필요한 복사를 피해서 시스템의 부하를 줄이는 기법
- **fork() 시에도 적용 됨**

*** 시스템에 따라 다를 수 있음**

```
0
After addr1[0]=1
addr1 = 1
addr2 = 1
After addr2[0]=2
addr1 = 1
addr2 = 2
After addr1[0]=3
addr1 = 3
addr2 = 2
3
```

Changing File Size

$ man -s 2 truncate

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

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

- **파일의 크기를 변경 한다**
  - 늘어난 부분은 0으로 초기화 된다
- **파일에 대한 쓰기 권한 필요**

- **Mapping해서 사용할 파일의 크기가 작을 때 사용 가능**

Outline

- Mapping a File to Memory
- **Synchronizing a File with a Mapping**
- **IPC with a Memory Mapped File**

Synchronizing a File with a Mapping

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

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

$ man -s 2 msync

- **강제로 Page write-back 수행**
  - Memory의 내용과 파일 내용 동기화
  - fflush() 유사 함
- **flags**
  - MS_SYNC : write-back이 끝날 때 까지 대기
  - MS_ASYNC : write-back을 비동기적으로 수행
    - 기다리지 않고 바로 return
  - MS_INVALIDATE : 메모리에 변경된 내용 무효화

Outline

- Mapping a File to Memory
- Synchronizing a File with a Mapping
- **Inter-Process Communication (IPC) through a Memory Mapped File**

IPC with Memory Mapping

- 같은 Memory mapping region을 공유함으로써 프로세스들 사이의 통신(대화) 가능

- 하지만, 프로세스들 간의 동기화를 잘 고려해서 사용해야 함

- Inter-Process Communication 방법들
  - Pipe → Chap 9
  - Message Queue, Shared Memory, Semaphore → Chap 10

IPC with Memory Mapping

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

#define CHECK_MMAP_SUCCESS(_addr)\
if(_addr == MAP_FAILED) { perror("mmap"); exit(1); }


int main(int argc, char *argv[]) {
     int fd;
     caddr_t addr;
     char fileName[255] = "input.dat";

     if (argc > 1) strcpy(fileName, argv[1]);

     if ((fd = open(fileName, O_RDWR)) == -1) {
          perror("open");
          exit(1);
     }
```

IPC with Memory Mapping

```
        int pageSize = getpagesize();

        addr = mmap(NULL, pageSize, PROT_READ | PROT_WRITE, MAP_SHARED, fd, (off_t)0);
        CHECK_MMAP_SUCCESS(addr);
        close(fd);

        switch (pid = fork()) {
        case -1:  /* fork failed */
              perror("fork");
              exit(1);
              break;
        case 0:   /* child process */
              printf("1. Child Process : addr=%s", addr);
              sleep(1);
              addr[0] = 'x';
              printf("2. Child Process : addr=%s", addr);
              sleep(2);
              printf("3. Child Process : addr=%s", addr);
              break;
        default:   /* parent process */
              printf("1. Parent process : addr=%s", addr);
              sleep(2);
              printf("2. Parent process : addr=%s", addr);
              addr[1] = 'y';
              printf("3. Parent process : addr=%s", addr);
              break;
        }

        return 0;
}
```

1. Parent process : addr=Hello!
1. Child Process : addr=Hello!
2. Child Process : addr=xello!
2. Parent process : addr=xello!
3. Parent process : addr=xyllo!
3. Child Process : addr=xyllo!

Producer-Consummer

- **Producer.c**

```
...

int main(int argc, char *argv[]) {
      int fd;
      char* addr;
      char fileName[255] = "map.dat";

      if (argc > 1) strcpy(fileName, argv[1]);
      if ((fd = open(fileName, O_RDWR)) == -1) { perror("open"); exit(1); }

      addr = mmap(NULL, getpagesize(), PROT_READ | PROT_WRITE, MAP_SHARED, fd, (off_t)0);
      CHECK_MMAP_SUCCESS(addr);
      close(fd);

      char c = 'a';
      while (1) {
            sleep(2);
            addr[0] = c++;
            if (c > 'z') c = 'a';
      }

      return 0;
}
```

Producer-Consummer

- **consummer.c**

```
...

int main(int argc, char *argv[]) {
      int fd;
      char* addr;
      char fileName[255] = "map.dat";

      if (argc > 1) strcpy(fileName, argv[1]);
      if ((fd = open(fileName, O_RDWR)) == -1) { perror("open"); exit(1); }

      addr = mmap(NULL, getpagesize(), PROT_READ | PROT_WRITE, MAP_SHARED, fd, (off_t)0);
      CHECK_MMAP_SUCCESS(addr);
      close(fd);

      while (1) {
            sleep(2);
            printf("%c\n", addr[0]);
      }

      return 0;
}
```

Summary

- **Mapping a File to Memory**
- **Synchronizing a File with a Mapping**
- **Inter-Process Communication (IPC) through a Memory Mapped File**