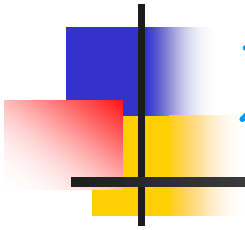
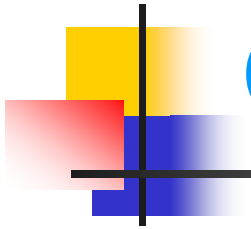
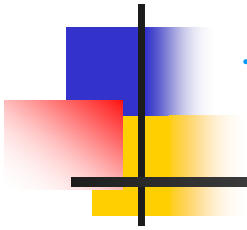


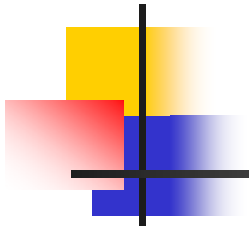
제12장 고급 입출력



Contents



12.1 Nonblocking I/O



Nonblocking I/O

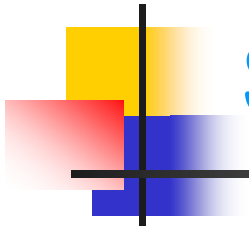
- Blocking I/O
  - I/O 작업 완료를 기다리며 영원히 리턴 안 할 수 있다 (block forever)
  - 예)

    getchar(): 엔터키를 입력할 때까지 영원히 기다린다

- Nonblocking I/O
  - I/O 작업이 당장 완료될 수 없으면 바로 에러를 리턴한다
  - 예)

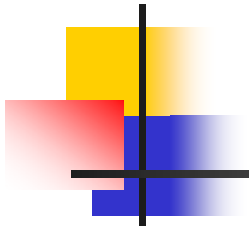
    이미 입력된 키가 있으면 그 값을 리턴하고,

    아니면 기다리지 않고 즉시 에러를 리턴한다



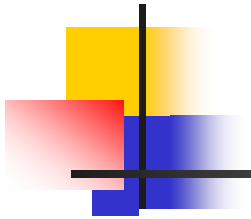
Slow system calls

- Slow system calls which can block forever
  - Reads from pipes, terminals, network, if data isn't present
  - Writes to above if they don't handle reading promptly
  - Opens of files block until some condition occurs
  - Open of a terminal device attached to a modem
  - Open of a FIFO for writing only when there is no other process for reading the FIFO
  - Reads and writes of files that have mandatory record locking enabled
  - Certain ioctl operations
  - Some of IPC functions



Nonblocking I/O 설정 방법

- open() 으로 파일을 열 때 O_NONBLOCK 플래그
  - open("a.out",O_RDONLY | O_NONBLOCK);
- open file에 fcntl()로 O_NONBLOCK 플래그 설정
  - int flag;
    flag = fcntl(fd, F_GETFL, 0);
    flag |= O_NONBLOCK;
    fcntl(fd, F_SETFL, flag);
- O_NONBLOCK 플래그 설정 취소
  - int flag;
    flag = fcntl(fd, F_GETFL, 0);
    flag &= ~O_NONBLOCK;
    fcntl(fd, F_SETFL, flag);

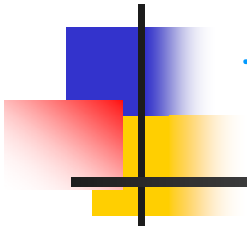


예제: nonblockw.c

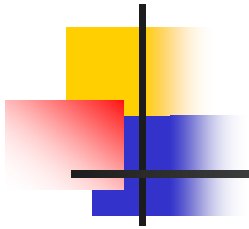
```
#include  <fcntl.h>

void set_fl(int fd, int flags) {
  int    val;

  if ( (val = fcntl(fd, F_GETFL, 0)) < 0)
    perror("fcntl F_GETFL error");
  val |= flags;    /* turn on flags */
  if (fcntl(fd, F_SETFL, val) < 0)
    perror("fcntl F_SETFL error");
}

void clr_fl(int fd, int flags)  {
  int    val;

  if ( (val = fcntl(fd, F_GETFL, 0)) < 0)
    perror("fcntl F_GETFL error");
  val &= ~flags;    /* turn flags off */
  if (fcntl(fd, F_SETFL, val) < 0)
    perror("fcntl F_SETFL error");
}
```

예제: nonblockw.c

```
#include  <sys/types.h>  /* nonblockw.c */
#include  <errno.h>
#include  <fcntl.h>
char  buf[100000];
int main(void) {
  int    ntowrite, nwrite; char  *ptr;

  ntowrite = read(STDIN_FILENO, buf, sizeof(buf));
  fprintf(stderr, "read %d bytes₩n", ntowrite);
  set_fl(STDOUT_FILENO, O_NONBLOCK);  /* set nonblocking */
  for (ptr = buf; ntowrite > 0; ) {
    errno = 0;
    nwrite = write(STDOUT_FILENO, ptr, ntowrite);
    fprintf(stderr, "nwrite = %d, errno = %d₩n", nwrite, errno);
    if (nwrite > 0) {
      ptr += nwrite;
      ntowrite -= nwrite;
    }
  }
  clr_fl(STDOUT_FILENO, O_NONBLOCK);  /* clear nonblocking */
  exit(0);
}
```



12.2 Record locking



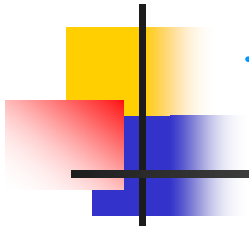
Record Locking

- 필요한 이유
  - 하나의 파일에 두 프로세스가 동시에 쓴다면 어떤 내용은 덮어 쓰여질 수 있다
- 방법
  - 쓰여지는 내용이 파일의 뒤에 덧붙여지는 것이라면 open() 에서 O_APPEND 모드로 열면 된다
  - 한 프로세스가 파일의 영역을 읽거나 수정할 때 다른 프로세스의 접근을 제한하기 위해서 그 영역에 lock 을 걸어야 한다

Lock 사이의 호환성

		Request for read lock	Request for write lock
Region currently has	No locks	OK	OK
	One or more read locks	OK	denied
	One write lock	denied	denied

- Types of Locks
  - F_RDLCK : Shared Read Lock
  - F_WRLCK : Exclusive Write Lock

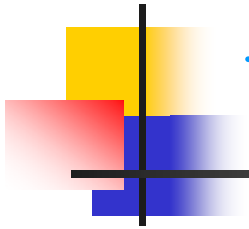


fcntl()

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

int fcntl(int filedes, int cmd, struct flock *flockptr);

                returns : depends on cmd if OK, -1 on error
```

- *cmd* : F_GETLK, F_SETLK, F_SETLKW

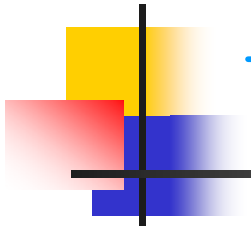
```
struct flock {
    short l_type;   /* F_RDLCK, F_WRLCK, F_UNLCK */
    off_t l_start;  /* 로킹 영역의 시작 지점 */
    short l_whence; /* SEEK_SET, SEEK_CUR, SEEK_END */
    off_t l_len;    /* 로킹 영역의 길이 (0이면 파일 끝까지) */
    pid_t l_pid;    /* 프로세스의 PID */
};
```



fcntl()

- F_GETLK
  - *flockptr* 에 설정한 lock을 걸 수 있는지, 즉 이미 lock 이 걸려 있는지 검사
  - 걸 수 없다면 이미 걸려있는 lock 정보 *flockptr* 가 채워짐
  - 걸 수 있다면 *flockptr* 의 l_type 만 F_UNLCK으로 변경됨

- F_SETLK and l_type == F_RDLCK or F_WRLCK
  - *flockptr* 에 설정한 lock 을 건다
  - 실패하면 즉시 -1을 리턴한다 (errno = EACCESS or EAGAIN)

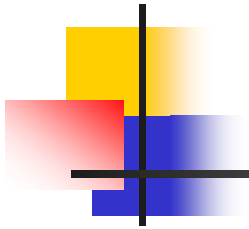


fcntl()

- F_SETLK and l_type == F_UNLCK
  - *flockptr* 에 설정한 lock 을 해제한다
- F_SETLKW
  - F_SETLK 의 blocking version
  - 프로세스는 lock 이 가능할 때까지 sleep 상태가 된다
  - This sleep is interrupted if a signal is caught

예:report.c

```
#include <fcntl.h> #include <errno.h>
#include "emp.h“                    #define MAX 10
int main(argc, argv)
int argc; char *argv[];
{
        struct flock lock;
        struct emp record;
        int fd, open(), fcntl(), sum=0, try=0;
        if ((fd=open(argv[1], O_RDONLY))== -1) {
                perror(argv[1]);
                exit(1);
        }
        lock.l_type = F_RDLCK;
        lock.l_whence = 0;
        lock.l_start = 0L;
        lock.l_len = 0L;
        while (fcntl(fd,F_SETLK, &lock) == -1) {
                if (errno == EAGAIN) {
                    if (try++ < MAX) {
                        sleep(1);                continue;
                    }
```

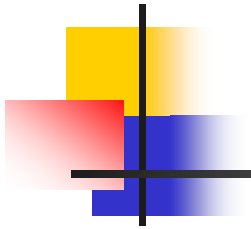


예:report.c

```
                printf("%s busy -- try later₩n", argv[1]);
                 exit(2);
            }
            perror(argv[1]); exit(3);

        }
         sum = 0;
         while (read(fd, (char *) & record, sizeof(record)) >0) {
                printf("Employee: %s, Salary: %d₩n", record.name,
              record.salary);
                sum += record.salary;
         }
         printf("₩nTotal salary: %d=₩n", sum);

         lock.l_type = F_UNLCK;
         fcntl(fd, F_SETLK, &lock);
         close(fd);
}
```

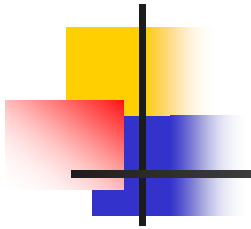
예: update.c

```
#include <fcntl.h>              #include "emp.h"
main(argc, argv)
int argc; char *argv[];
{
        struct flock lock;
        struct emp record;
        int fd, open(), pid, getpid(), recnum;
        long position;

        if ((fd=open(argv[1], O_RDWR))== -1) {
                perror(argv[1]);
                exit(1);
        }

        pid = getpid();
        for(;;) {
                printf("₩n Enter record number; ");
                scanf("%d", &recnum);
                if (recnum <0) break;
                position = recnum * sizeof(record);
```



예: update.c

```
lock.l_type = F_WRLCK;
lock.l_whence = 0;
lock.l_start = position;
lock.l_len = sizeof(record);
if (fcntl(fd,F_SETLKW, &lock) == -1) {
        perror(argv[1]);
        exit(2);
}

lseek(fd, position,0);
if (read(fd, (char *) & record, sizeof(record)) == 0) {
        printf("record %d not found₩n", recnum);
        lock.l_type = F_UNLCK;
        fcntl(fd, F_SETLK, &lock);
        continue;
}
printf("Employee: %s, salray: %d₩n", record.name, record.salary);
record.pid = pid;
printf("Enter new salary: ");
scanf("%d", &record.salary);
```

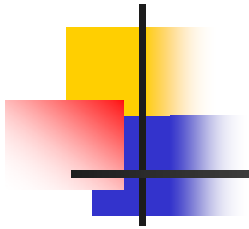


예: update.c

```
    lseek(fd, position, 0);
    write(fd, (char *) &record, sizeof(record));
    lock.l_type = F_UNLCK;
    fcntl(fd, F_SETLK, &lock);
    }
    close(fd);
}
```

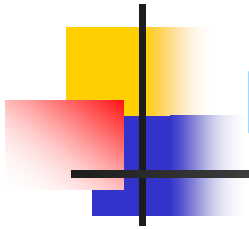
Lock 요청 및 해제

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

int
lock_reg(int fd, int cmd, int type, off_t offset, int whence, off_t len)
{
  struct flock  lock;

  lock.l_type = type;        /* F_RDLCK, F_WRLCK, F_UNLCK */
  lock.l_start = offset;     /* byte offset, relative to l_whence */
  lock.l_whence = whence; /* SEEK_SET, SEEK_CUR, SEEK_END */
  lock.l_len = len;          /* #bytes (0 means to EOF) */

  return( fcntl(fd, cmd, &lock) );
}
```



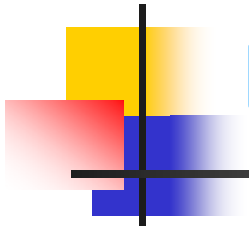
Lock 요청 및 해제

```
#define read_lock(fd, offset, whence, len) ₩
        lock_reg(fd, F_SETLK, F_RDLCK, offset, whence, len)
#define readw_lock(fd, offset, whence, len) ₩
        lock_reg(fd, F_SETLKW, F_RDLCK, offset, whence, len)
#define write_lock(fd, offset, whence, len) ₩
        lock_reg(fd, F_SETLK, F_WRLCK, offset, whence, len)
#define writew_lock(fd, offset, whence, len) ₩
        lock_reg(fd, F_SETLKW, F_WRLCK, offset, whence, len)
#define un_lock(fd, offset, whence, len) ₩
        lock_reg(fd, F_SETLK, F_UNLCK, offset, whence, len)
```



Lock 시험 예

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

pid_t lock_test(int fd, int type, off_t offset, int whence, off_t len)
{
  struct flock  lock;

  lock.l_type = type;             /* F_RDLCK or F_WRLCK */
  lock.l_start = offset;          /* byte offset, relative to l_whence */
  lock.l_whence = whence;  /* SEEK_SET, SEEK_CUR, SEEK_END */
  lock.l_len = len;               /* #bytes (0 means to EOF) */

  if (fcntl(fd, F_GETLK, &lock) < 0)
    perror("fcntl error");

  if (lock.l_type == F_UNLCK)
    return(0);           /* false, region is not locked by another proc */
  return(lock.l_pid);    /* true, return pid of lock owner */
}
```

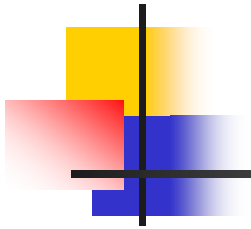


Lock 상속 및 해제

- Lock 해제
  - 프로세스가 종료하면 그 프로세스가 설정한 모든 lock 은 해제된다
  - fd가 닫히면 이와 관련된 모든 lock 은 해제된다.

- 다음과 같은 경우들에도 모든 lock 은 해제된다
  - fd1 = open("a", ...);
    read_lock(fd1, ...);
    fd2 = dup(fd1);
    close(fd2)

  - fd1 = open("a", ...);
    read_lock(fd1, ...);
    fd2 = open("a",...);
    close(fd2)



Lock 상속 및 해제

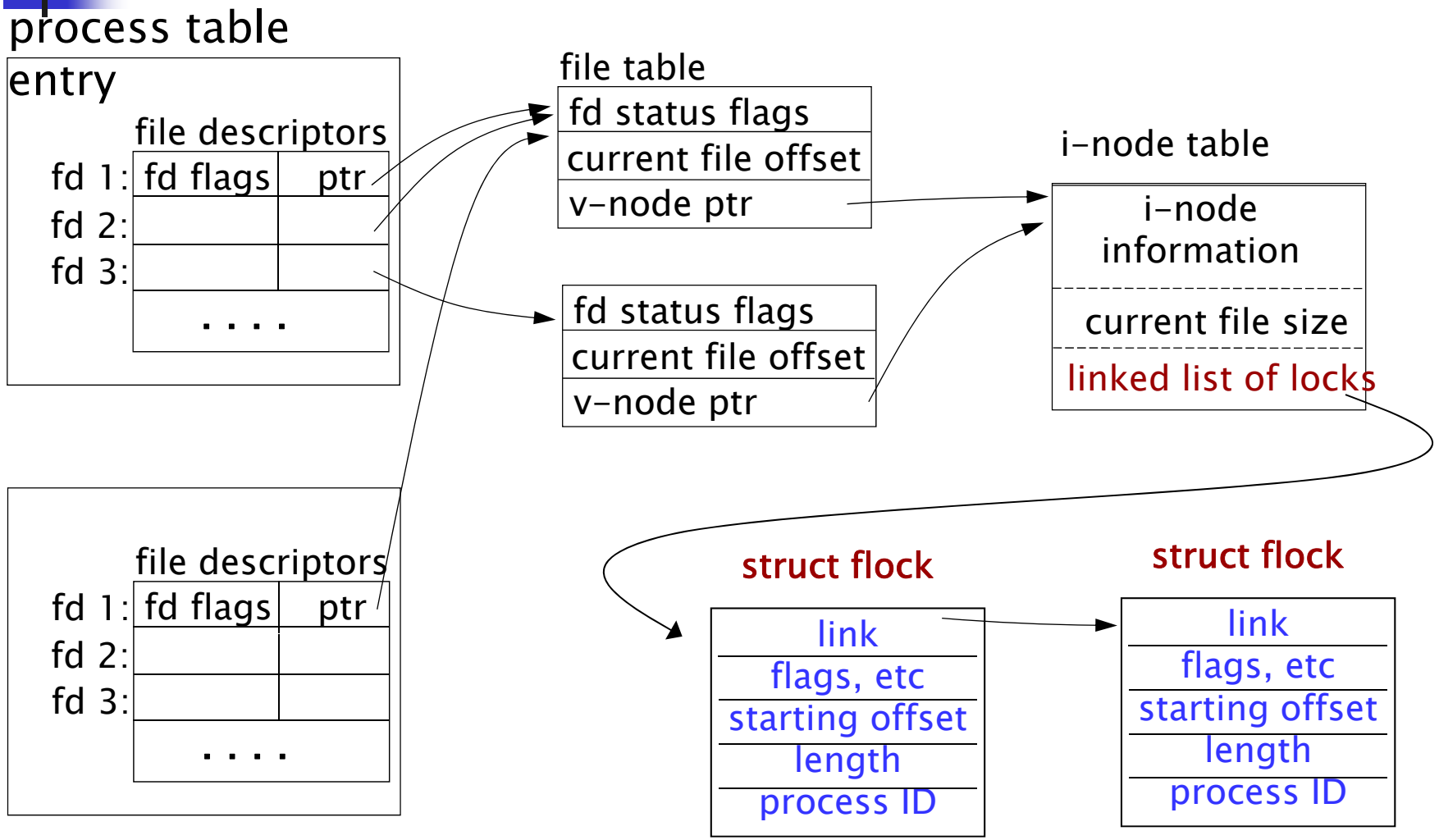
- fork() 에 의한 자식 프로세스에게 lock 은 상속되지 않는다
- exec() 의 경우에는 상속된다

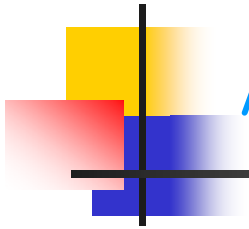
커널 구현

```
fd1 = open("a", ... );
write_lock(fd1, 0, ... );     /* parent write locks byte 0 */
if (fork() > 0) {             /* parent */
    fd2 = dup(fd1);
    fd3 = open("a", ... );
    pause();
    close(fd3);               /* is the write lock released? */
} else {                      /* child */
    read_lock(fd1, ... );      /* child read locks byte 1 */
    pause();
}
```

- 위 실행 결과는 다음 슬라이드와 같음
- lock 은 file descriptor 가 아니고 file 에 연관되어 있다
- 따라서 fd1, fd2, fd3 중 어느 것이라도 close 하면 lock 은 해제된다

커널 구현


process table
entry
file descriptors
fd 1:
fd 2:
fd 3:
fd flags
ptr
. . . .
file table
fd status flags
current file offset
v-node ptr
fd status flags
current file offset
v-node ptr
i-node table
i-node information
current file size
linked list of locks
file descriptors
fd 1:
fd 2:
fd 3:
fd flags
ptr
. . . .
struct flock
link
flags, etc
starting offset
length
process ID
struct flock
link
flags, etc
starting offset
length
process ID



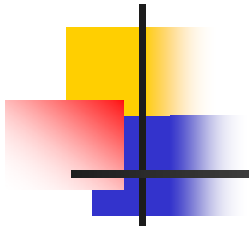
Advisory vs. Mandatory Locking

- Advisory Locking
  - lock 을 걸 수 있고 검사할 수 있지만 강제되지는 않는다
  - 즉 locking 규칙을 무시하고 읽고 쓰기 가능
  - 따라서 모든 프로세스들이 자발적으로 규칙을 준수해야 함

- Mandatory Locking
  - 커널이 locking 규칙을 강제하므로 무시하고 읽고 쓰기 불가능
  - 커널이 모든 입출력 호출을 감시해야 하므로 시스템 부하 증가
  - 설정 방법

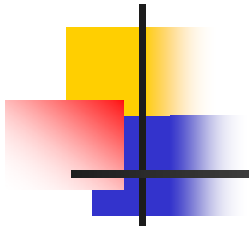
    set-group-ID 비트를 켜고 group-execute 비트를 끔

    -rw-r-lr-- 1 lsj 5 Jul 15 12:11 lockfile
  - $ chmod 2644 lockfile



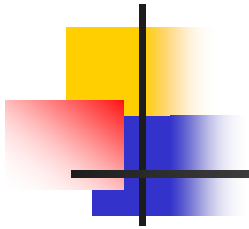
Mandatory Locking

	Blocking descriptor, tries to		Non Blocking descriptor, tries to	
	read	write	read	write
Read lock exists on region	OK	blocks	OK	EAGAIN
Write lock exists on region	blocks	blocks	EAGAIN	EAGAIN



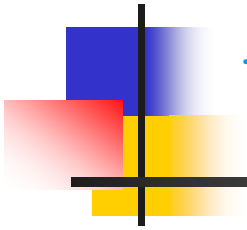
예: file_lock.c

```
#include <stdio.h>
#include <fcntl.h>
int main(int argc, char **argv) {
    static struct flock lock;
    int fd, ret;

    if (argc < 2) {
        fprintf(stderr, "사용법: %s 파일₩n", argv[0]);
        exit(1);
    }

    fd = open(argv[1], O_WRONLY);
    if(fd == -1) {
        printf("파일 열기 실패 ₩n");
        exit(1);
    }
```

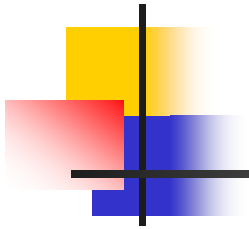


예: file_lock.c

```
    lock.l_type = F_WRLCK;
    lock.l_start = 0;
    lock.l_whence = SEEK_SET;
    lock.l_len = 0;
    lock.l_pid = getpid();

    ret = fcntl(fd, F_SETLKW, &lock);
    if(ret == 0) {    // 파일 잠금 성공하면
        while (1) {
            scanf("%c", NULL);
        }
    }
}
```



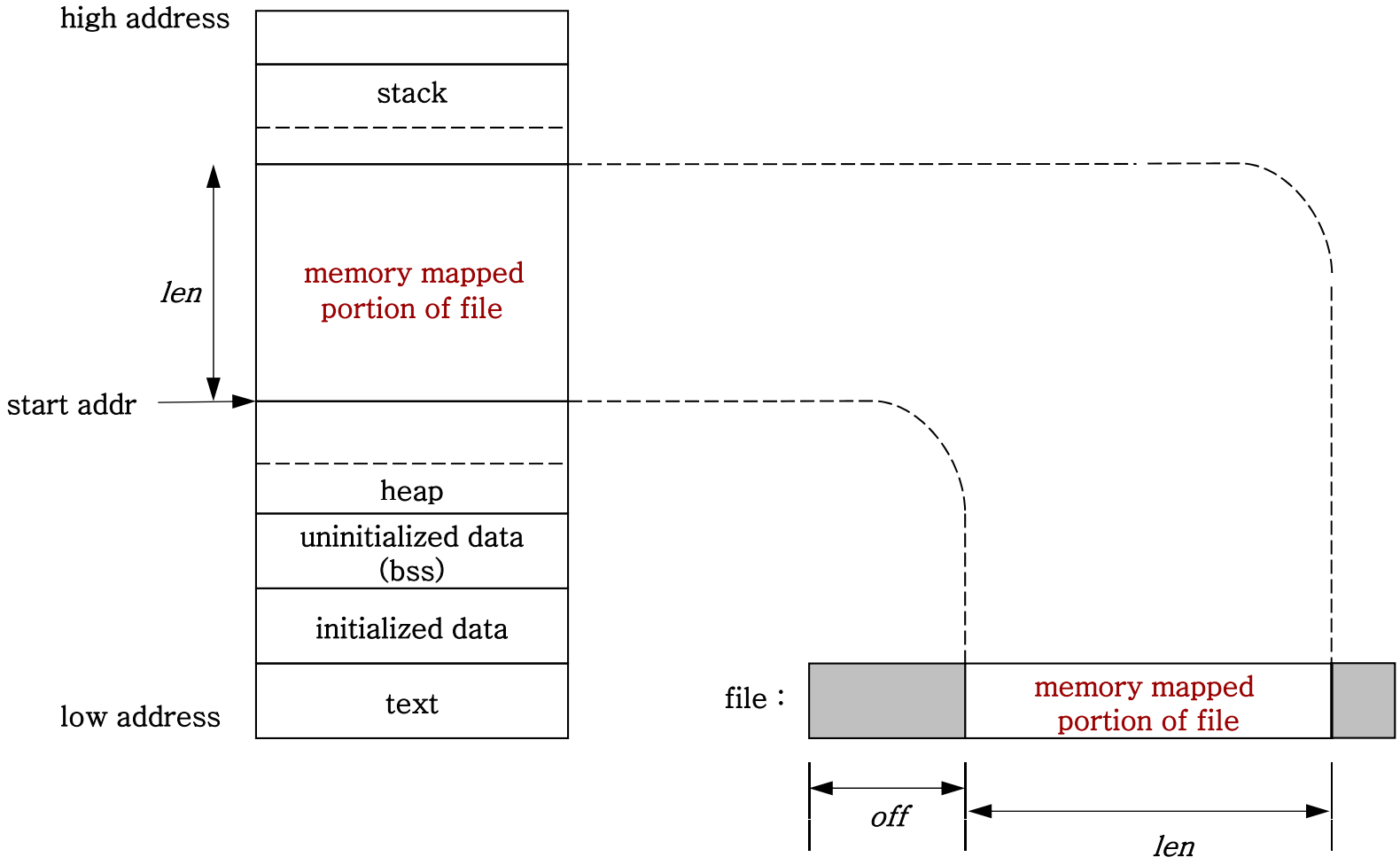
12.3 Memory Mapped I/O

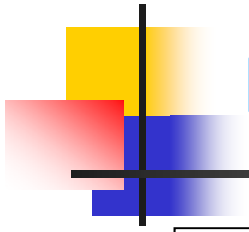


Memory Mapped I/O

- 디스크의 파일에 메모리 주소를 부여한다
  - 주소에서 읽는 것은 파일의 데이터를 읽는다
  - 주소에 쓰면 파일에 저장된다
- 메모리인 것처럼 포인터와 배열을 사용하여 파일의 데이터를 다룰 수 있게 된다
- 성능 향상을 기대할 수 있다
  - 파일 I/O처럼 시스템 버퍼와 라이브러리 버퍼 사이에 데이터가 복사될 필요 없다
- 알고리즘 단순화
  - 마치 메모리처럼 배열과 포인터 사용 가능

예: memory mapped file

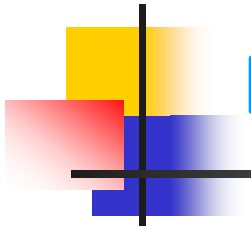




mmap()

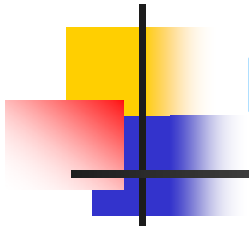
```
#include <sys/types.h>
#include <sys/mman.h>

caddr_t mmap(caddr_t addr, size_t len, int prot, int flag,
                                int filedes, off_t off);

        Returns : starting address of mapped region if OK, -1 on error
```

- 디스크 파일에 메모리 주소를 부여한다
- *addr* : 부여될 메모리 시작 주소
  - addr 가 NULL 이면 시스템이 적당한 시작 주소를 선택한다
  - virtual memory page의 배수이어야 한다
- *len* : 메모리 크기



mmap()

- *prot* : specify the protection of the mapped region.
  - PROT_READ : the region can be read.
  - PROT_WRITE : the region can be written.
  - PROT_EXEC : the region can be executed.
  - PROT_NONE : the region cannot be accessed (not in 4.3+BSD).
  - The protection specified for a region has to match the **open** mode of the file.



mmap()

- *flag* : mapped region의 특성
  - MAP_FIXED
    - 반환 주소가 *addr*와 같아야 한다.
    - MAP_FIXED 가 지정되지 않고 *addr* 가 0이 아니면, 커널은 *addr* 를 단지 힌트로만 사용한다.
  - MAP_SHARED
    - 부여된 주소에 쓰면 파일에 저장된다
    - MAP_SHARED 나 MAP_PRIVATE 중의 하나가 반드시 지정되어야 한다
  - MAP_PRIVATE
    - 부여된 주소에 쓰면 파일의 복사본이 만들어지고
    - 이후부터는 복사본을 읽고 쓰게 된다

예: memcpy이용한 파일 복사

```
#include  <sys/types.h>  /* mcopy.c */
#include  <sys/stat.h>
#include  <sys/mman.h>  /* mmap() */
#include  <fcntl.h>

int main(int argc, char *argv[]) {
  int       fdin, fdout;
  char     *src, *dst;
  struct stat  statbuf;

  if (argc != 3) err_quit("usage: a.out <fromfile> <tofile>");
  if ( (fdin = open(argv[1], O_RDONLY)) < 0)
          perror("can't open %s for reading", argv[1]);
  if ( (fdout = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, FILE_MODE)) < 0)
          perror("can't creat %s for writing", argv[1]);
  if (fstat(fdin, &statbuf) < 0)  /* need size of input file */
          perror("fstat error");
```

예: memcpy이용한 파일 복사

```
    /* set size of output file */
    if (lseek(fdout, statbuf.st_size - 1, SEEK_SET) == -1)
            perror("lseek error");
    if (write(fdout, "", 1) != 1)
            perror("write error");
    if ( (src = mmap(0, statbuf.st_size, PROT_READ,
           MAP_FILE | MAP_SHARED, fdin, 0)) == (caddr_t) -1)
            perror("mmap error for input");
    if ( (dst = mmap(0, statbuf.st_size, PROT_READ | PROT_WRITE,
           MAP_FILE | MAP_SHARED, fdout, 0)) == (caddr_t) -1)
            perror("mmap error for output");

    memcpy(dst, src, statbuf.st_size);  /* does the file copy */
    exit(0);
}
```