Systems Programming

Spring 2023

4 주차

- Review & Summary
  - 01-linking
  - 02-relocation
  - 03-exceptions
  - 04-signals
  - 05-io
- NO session/class on 4/4 화
- Q&A

중간고사 1

- 범위
  - 01-linking
  - 02-relocation
  - 03-exceptions
  - 04-signals
  - 05-io

- 4월 6일 목요일 오후 2시 ~

- 장소
  - 302동 105호
  - 302동 209호
  - 개인별 시험장소 배정 공지 예정 (eTL, ~4/2)

4월 수업

- 자율학습 & Q&A Session 연장
  - maybe, ~5/11 (before second midterm)
- 이론: 자율학습 (pdf 강의자료 + audio)
  - course materials will be uploaded
  - Schedule / coverage will be announced later through eTL
- 매주 목요일 Q&A Session 계속 진행 (302동 105호)
  - Q&A form will be closed until midterm 1
  - Will be open again after midterm 1

05-io

Questions

2.

7p에서 closing an already closed file은 threaded program에서 매우 위험할 수 있다고 되어있는데 5장 끝까지 봐도 딱히 내용이 나오는게 없어 질문드립니다. 또 이후 13p에서도 Buffered RIO routine들은 thread-safe하다는 말이 나오는데 이 또한 정확히 숙지하고 있어야 하는지 궁금합니다. (찾아보니 13-concurrency쯤에 thread에 관한 내용이 나오는데 이부분까지도 필요한지 모르겠습니다)

+ closing an already closed file 예시를 생각해봤다가 fork한 이후 child에서 close하는 경우를 생각하게 되었으나 이는 refcnt를 그냥 하나 줄이는 것으로 해결될 것 같아 제 생각이 맞는지 여쭤보고 싶습니다.

```
void function f ()
{
    int fd = open("file A");

    // do something

    close(fd);

    // do some other things

    close(fd);
}
```

Questions

2.

7p에서 closing an already closed file은 threaded program에서 매우 위험할 수 있다고 되어있는데 5장 끝까지 봐도 딱히 내용이 나오는게 없어 질문드립니다. 또 이후 13p에서도 Buffered RIO routine들은 thread-safe하다는 말이 나오는데 이 또한 정확히 숙지하고 있어야 하는지 궁금합니다. (찾아보니 13-concurrency쯤에 thread에 관한 내용이 나오는데 이부분까지도 필요한지 모르겠습니다)

+ closing an already closed file 예시를 생각해봤다가 fork한 이후 child에서 close하는 경우를 생각하게 되었으나 이는 refcnt를 그냥 하나 줄이는 것으로 해결될 것 같아 제 생각이 맞는지 여쭤보고 싶습니다.

< thread A >

```
int fd = open("file A");

// do something

close(fd);

// do some other things

close(fd);
```

< thread B >

```
int fd = open("file B");

// do something

close(fd);

// do some other things

close(fd);
```

Questions

2.

7p에서 closing an already closed file은 threaded program에서 매우 위험할 수 있다고 되어있는데 5장 끝까지 봐도 딱히 내용이 나오는게 없어 질문드립니다. 또 이후 13p에서도 Buffered RIO routine들은 thread-safe하다는 말이 나오는데 이 또한 정확히 숙지하고 있어야 하는지 궁금합니다. (찾아보니 13-concurrency쯤에 thread에 관한 내용이 나오는데 이부분까지도 필요한지 모르겠습니다)

+ closing an already closed file 예시를 생각해봤다가 fork한 이후 child에서 close하는 경우를 생각하게 되었으나 이는 refcnt를 그냥 하나 줄이는 것으로 해결될 것 같아 제 생각이 맞는지 여쭤보고 싶습니다.

```
void function f ()
{
    int fd = open("file A");

    // do something

    if (close(fd) == -1) {
        perror("close");
        exit(1);
    }
    fd = -1;

    // do some other things

    close(fd);          // will return error!
}
```

Personal Recommendation

Questions

2.

7p에서 closing an already closed file은 threaded program에서 매우 위험할 수 있다고 되어있는데 5장 끝까지 봐도 딱히 내용이 나오는게 없어 질문드립니다. 또 이후 13p에서도 Buffered RIO routine들은 thread-safe하다는 말이 나오는데 이 또한 정확히 숙지하고 있어야 하는지 궁금합니다. (찾아보니 13-concurrency쯤에 thread에 관한 내용이 나오는데 이부분까지도 필요한지 모르겠습니다)

+ closing an already closed file 예시를 생각해봤다가 fork한 이후 child에서 close하는 경우를 생각하게 되었으나 이는 refcnt를 그냥 하나 줄이는 것으로 해결될 것 같아 제 생각이 맞는지 여쭤보고 싶습니다.

12.7.1 Thread Safety

When we program with threads, we must be careful to write functions that have a property called thread safety. A function is said to be *thread-safe* if and only if it will always produce correct results when called repeatedly from multiple concurrent threads. If a function is not thread-safe, then we say it is *thread-unsafe*.

We can identify four (nondisjoint) classes of thread-unsafe functions:

Questions

1. RIO에서 read하던 중 100바이트를 읽고 싶었는데 80바이트 후에 EOF를 만났다고 가정하면, 80을 리턴하는 건가요? 아니면 0을 리턴하는 건가요? 코드상으로는 80이라고 생각했는데 주석에 0을 리턴한다고 되어있어서 혼란스럽습니다.
3.
14p와 15p에 나오는 세부사항이 달라 질문드립니다.
14p에서는 "Return: num. bytes transferred if OK, 0 on EOF (rio_readn only), -1 on error"라고 되어있어서 EOF일경우 0을 돌려준다고 되어있는데 15p에서는 EOF가 들어왔을 때 return 0을 하는 것이 아니라 마치 short count처럼 지금까지 들어온 입력받은 데이터의 크기를 return하도록 되어있는데 어떤 것이 맞다고 봐야할까요?

The RIO Package

- **RIO is a set of wrappers that provide efficient and robust I/O in apps, such as network programs that are subject to short counts**

10.4 Robust Reading and Writing with the Rio Package

In this section, we will develop an I/O package, called the Rio (Robust I/O) package, that handles these short counts for you automatically. The Rio package provides convenient, robust, and efficient I/O in applications such as network programs that are subject to short counts. Rio provides two different kinds of functions:

Questions

- **Download from http://csapp.cs.cmu.edu/public/code.html**
  - → `src/csapp.c` and `include/csapp.h`

```
/**********************************************************************
 * The Rio package - robust I/O functions
 **********************************************************************/
/*
 * rio_readn - robustly read n bytes (unbuffered)
 */
/* $begin rio_readn */
ssize_t rio_readn(int fd, void *usrbuf, size_t n)
{
    size_t nleft = n;
    ssize_t nread;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nread = read(fd, bufp, nleft)) < 0) {
            if (errno == EINTR) /* Interrupted by sig handler return */
                nread = 0;      /* and call read() again */
            else
                return -1;      /* errno set by read() */
        }
        else if (nread == 0)
            break;              /* EOF */
        nleft -= nread;
        bufp += nread;
    }
    return (n - nleft);         /* return >= 0 */
}
/* $end rio_readn */
```

Questions

1. RIO에서 read하던 중 100바이트를 읽고 싶었는데 80바이트 후에 EOF를 만났다고 가정하면, 80을 리턴하는 건가요? 아니면 0을 리턴하는 건가요? 코드상으로는 80이라고 생각했는데 주석에 0을 리턴한다고 되어있어서 혼란스럽습니다.

3.
14p와 15p에 나오는 세부사항이 달라 질문드립니다.
14p에서는 "Return: num. bytes transferred if OK, 0 on EOF (rio_readn only), -1 on error"라고 되어있어서 EOF일경우 0을 돌려준다고 되어있는데 15p에서는 EOF가 들어왔을 때 return 0을 하는 것이 아니라 마치 short count처럼 지금까지 들어온 입력받은 데이터의 크기를 return하도록 되어있는데 어떤 것이 맞다고 봐야할까요?

```
#include "csapp.h"

ssize_t rio_readn(int fd, void *usrbuf, size_t n);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
```

Returns: number of bytes transferred if OK, 0 on EOF (`rio_readn` only), −1 on error

The `rio_readn` function transfers up to `n` bytes from the current file position of descriptor `fd` to memory location `usrbuf`. Similarly, the `rio_writen` function transfers `n` bytes from location `usrbuf` to descriptor `fd`. The `rio_readn` function can only return a short count if it encounters EOF. The `rio_writen` function never returns a short count. Calls to `rio_readn` and `rio_writen` can be interleaved arbitrarily on the same descriptor.

Questions

2. rio_readn과 rio_readnb가 인자가 똑같으면 둘 다 한 번에 n 바이트씩 읽어오는 것 같은데, 굳이 buffer가 있다는 점의 차이를 잘 모르겠습니다.

```
/*****************************************************************
 * The Rio package - robust I/O functions
 *****************************************************************/
/*
 * rio_readn - robustly read n bytes (unbuffered)
 */
/* $begin rio_readn */
ssize_t rio_readn(int fd, void *usrbuf, size_t n)
{
    size_t nleft = n;
    ssize_t nread;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nread = read(fd, bufp, nleft)) < 0) {
            if (errno == EINTR) /* Interrupted by sig handler return */
                nread = 0;      /* and call read() again */
            else
                return -1;      /* errno set by read() */
        }
        else if (nread == 0)
            break;              /* EOF */
        nleft -= nread;
        bufp += nread;
    }
    return (n - nleft);         /* return >= 0 */
}
/* $end rio_readn */
```

```
/*
 * rio_readnb - Robustly read n bytes (buffered)
 */
/* $begin rio_readnb */
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n)
{
    size_t nleft = n;
    ssize_t nread;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nread = rio_read(rp, bufp, nleft)) < 0)
            return -1;          /* errno set by read() */
        else if (nread == 0)
            break;              /* EOF */
        nleft -= nread;
        bufp += nread;
    }
    return (n - nleft);         /* return >= 0 */
}
/* $end rio_readnb */
```

Questions

2. rio_readn과 rio_readnb가 인자가 똑같으면 둘 다 한 번에 n 바이트씩 읽어오는 것 같은데, 굳이 buffer가 있다는 점의 차이를 잘 모르겠습니다.

```
/*
 * rio_readnb - Robustly read n bytes (buffered)
 */
/* $begin rio_readnb */
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n)
{
    size_t nleft = n;
    ssize_t nread;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nread = rio_read(rp, bufp, nleft)) < 0)
            return -1;          /* errno set by read() */
        else if (nread == 0)
            break;              /* EOF */
        nleft -= nread;
        bufp += nread;
    }
    return (n - nleft);         /* return >= 0 */
}
/* $end rio_readnb */
```

```
/*
 * rio_read - This is a wrapper for the Unix read() function that
 *    transfers min(n, rio_cnt) bytes from an internal buffer to a user
 *    buffer, where n is the number of bytes requested by the user and
 *    rio_cnt is the number of unread bytes in the internal buffer. On
 *    entry, rio_read() refills the internal buffer via a call to
 *    read() if the internal buffer is empty.
 */
/* $begin rio_read */
static ssize_t rio_read(rio_t *rp, char *usrbuf, size_t n)
{
    int cnt;

    while (rp->rio_cnt <= 0) {  /* Refill if buf is empty */
        rp->rio_cnt = read(rp->rio_fd, rp->rio_buf,
                           sizeof(rp->rio_buf));
        if (rp->rio_cnt < 0) {
            if (errno != EINTR) /* Interrupted by sig handler return */
                return -1;
        }
        else if (rp->rio_cnt == 0)  /* EOF */
            return 0;
        else
            rp->rio_bufptr = rp->rio_buf; /* Reset buffer ptr */
    }

    /* Copy min(n, rp->rio_cnt) bytes from internal buf to user buf */
    cnt = n;
    if (rp->rio_cnt < n)
        cnt = rp->rio_cnt;
    memcpy(usrbuf, rp->rio_bufptr, cnt);
    rp->rio_bufptr += cnt;
    rp->rio_cnt -= cnt;
    return cnt;
}
/* $end rio_read */
```

Questions

3. PPT 24쪽에서 dev/kmem의 type이 왜 other인가요? directory 아닌지 궁금합니다.

Carnegie Mellon

Unix Files

- A Unix *file* is a sequence of *m* bytes:
  - $B_0, B_1, \ldots, B_k, \ldots, B_{m-1}$
- All I/O devices are represented as files:
  - `/dev/sda2` (`/usr` disk partition)
  - `/dev/tty2` (terminal)
- Even the kernel is represented as a file:
  - `/dev/kmem` (kernel memory image)
  - `/proc` (kernel data structures)

3

kmem(4) - Linux man page

Name

mem, kmem, port - system memory, kernel memory and system ports

Description

mem is a character device file that is an image of the main memory of the computer. It may be used, for example, to examine (and even patch) the system.

Byte addresses in **mem** are interpreted as physical memory addresses. References to nonexistent locations cause errors to be returned.

Examining and patching is likely to lead to unexpected results when read-only or write-only bits are present.

It is typically created by:

```
mknod -m 660 /dev/mem c 1 1
chown root:kmem /dev/mem
```

The file **kmem** is the same as **mem**, except that the kernel virtual memory rather than physical memory is accessed.

It is typically created by:

```
mknod -m 640 /dev/kmem c 1 2
chown root:kmem /dev/kmem
```

port is similar to **mem**, but the I/O ports are accessed.

It is typically created by:

```
mknod -m 660 /dev/port c 1 4
chown root:mem /dev/port
```

Files

/dev/mem
/dev/kmem
/dev/port

https://man7.org/linux/man-pages/man4/kmem.4.html

Questions

1.
6p에서 "Elegant mapping of files to devices allows kernel to export simple interface called Unix I/O" 라고 되어있는데, 이 문장의 의미가 Unix io가 실제 물리적인 device(disk 등등)와 각 device에 대응되는 file이랑 연결시켜준다는 의미로 이해하면 맞는지 궁금합니다.
(예를 들어 disk랑 block special files를 연결해주는게 unix io라던가...)

- The /dev/ directory consists of files that represents devices that are attached to the local system

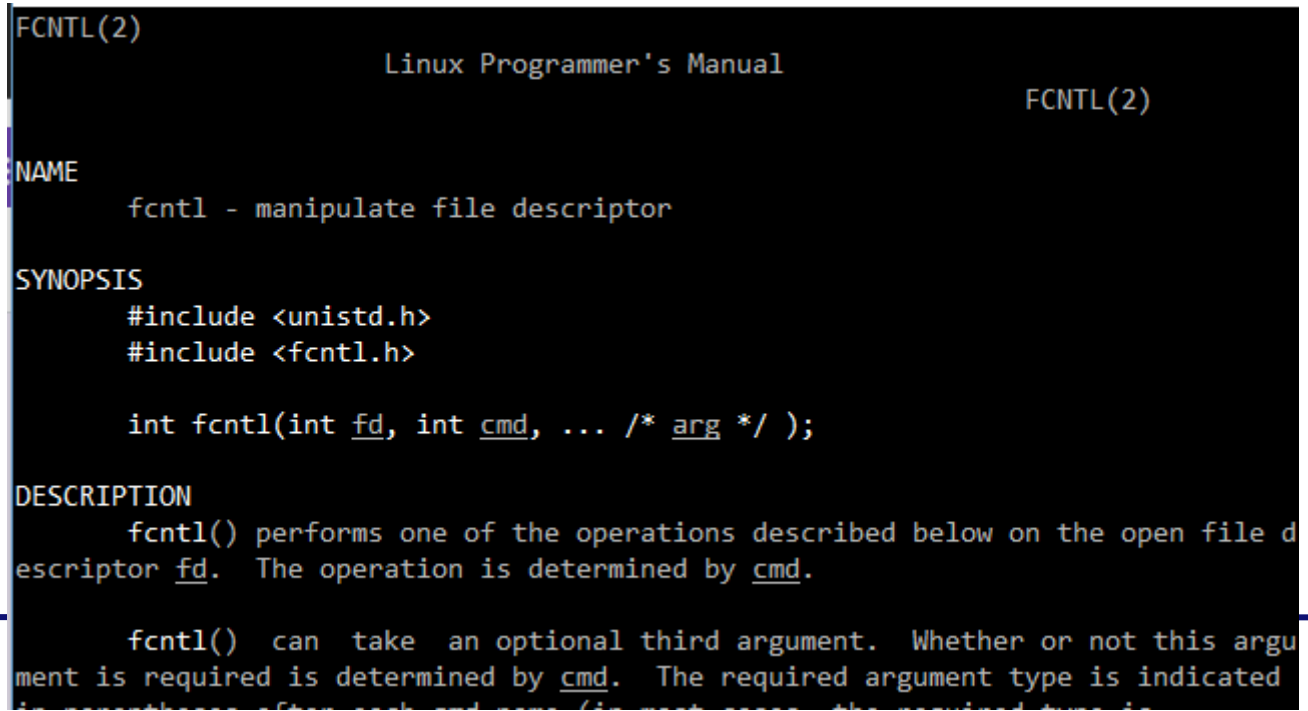
- For more details, please refer to following articles
  - https://tldp.org/LDP/sag/html/dev-fs.html
  - https://www.baeldung.com/linux/dev-directory

Questions

4.
28p에서 "situation unchanged by exec functions"라고 되어있는 부분이 exec function들은 fork랑 다르게 descriptor table/open file table/ v-node table을 손대지 않고 새로운 프로세스로 대체되는 것이다 정도로 이해하면 맞는지 여쭤보고 싶습니다.

How Processes Share Files: Fork()

- **A child process inherits its parent's open files**
  - Note: situation unchanged by **`exec`** functions (use **`fcntl`** to change)



```
FCNTL(2)                    Linux Programmer's Manual                    FCNTL(2)

NAME
       fcntl - manipulate file descriptor

SYNOPSIS
       #include <unistd.h>
       #include <fcntl.h>

       int fcntl(int fd, int cmd, ... /* arg */ );

DESCRIPTION
       fcntl() performs one of the operations described below on the open file d
escriptor fd.  The operation is determined by cmd.

       fcntl()  can  take  an optional third argument.  Whether or not this argu
ment is required is determined by cmd.  The required argument type is indicated
```

Questions

6.
31p에서 "Happens in child executing shell code, before exec"라고 되어있는데 정확히 무슨뜻인지 이해하지 못해 여쭤보고 싶습니다.

I/O Redirection

- **Question: How does a shell implement I/O redirection?**

```
unix> ls > foo.txt
```

```
fork();
/* open file */
execve("bin/ls", ... );
```

- **Step #1: open file to which stdout should be redirected**
  - Happens in child executing shell code, before **exec**

Questions

5.
33/34/35p에 있는 fun with file descriptors 답을 알려주실 수 있으실까요?

Questions

7.
강의에서는 설명하지 않은 46p내용이 정확히 무엇을 설명하려고 나온 내용인지 모르겠습니다. binary file이라는 것이 존재하며 다루는 함수들이 따로 존재한다 이정도로만 이해하고 넘어가면 될까요?

Aside: Working with Binary Files

- **Binary File Examples**
  - Object code, Images (JPEG, GIF),
- **Functions you shouldn't use on binary files**
  - Line-oriented I/O such as **`fgets, scanf, printf, rio_readlineb`**
    - Different systems interpret **`0x0A ('\n')`** (newline) differently:
      - Linux and Mac OS X: **`LF(0x0a) ['\n']`**
      - HTTP servers & Windoes: **`CR+LF(0x0d 0x0a) ['\r\n']`**
    - Use **`rio_readn`** or **`rio_readnb`** instead
  - String functions
    - **`strlen, strcpy`**
    - Interprets byte value 0 (end of string) as special

Questions

- 02-relocation.pdf (p.14) 에서 value 가 왜 -4 가 아닌지?
  - 수행시의 PC 값을 기준으로 보정 값이 0 임을 의미하는 예제임

Carnegie Mellon

Before Relocation (.text) main.o

```
Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   b8 00 00 00 00          mov    $0x0,%eax
   9:   e8 00 00 00 00          callq  e <main+0xe>
                        a: R_386_PC32   <swap>
   e:   b8 00 00 00 00          mov    $0x0,%eax
  13:   5d                      pop    %rbp
  14:   c3                      retq
```

After Relocation (.text) main

```
00000000004004ed <main>:
  4004ed:       55                      push   %rbp
  4004ee:       48 89 e5                mov    %rsp,%rbp
  4004f1:       b8 00 00 00 00          mov    $0x0,%eax
  4004f6:       e8 07 00 00 00          callq  400502 <swap>
  4004fb:       b8 00 00 00 00          mov    $0x0,%eax
  400500:       5d                      pop    %rbp
  400501:       c3                      retq
```

14