7. System Call

Hyunchan, Park

http://oslab.jbnu.ac.kr

Division of Computer Science and Engineering

Jeonbuk National University

학습 내용

- System Call
- Error Handling

개인 과제 7: 실습 (1/2)

- 실습 과제
  - 실습 내용에서 등장한 프로그램들을 작성하고, 그 결과를 확인할 것
    - 결과 성공 후, cat 으로 파일의 내용을 출력할 것
    - 슬라이드에 캡처된 프로그램들은 모두 포함되어야 함
  - 제출 방법
    - Old LMS, 과제 7
    - Xshell 로그 파일 1개 제출 (뒤의 8. File 내용과 결합하여 제출)
    - 파일 명: 학번.txt
- 제출 기한
  - 10/26 (월) 23:59 (지각 감점: 5%p / 12H, 1주 이후 제출 불가)

System Call

시스템 프로그래밍

- System Call: OS가 제공하는 기능들을 사용하는 것
  - 하드웨어를 제어하거나,
  - 다른 프로세스와의 통신을 수행하거나,
  - 시스템 정보에 접근, 수정하거나,
  - 시스템을 제어하는 기능 등

- 대표적인 시스템 콜
  - 화면 출력: printf() <- C 라이브러리. 내부에서 write() system call 사용
  - 파일 제어: open(), close(), read(), write()
  - 동적 메모리 할당: malloc() <- C 라이브러리. 내부에서 brk(), mmap() 사용
  - 네트워크 통신: socket(), send(), receive()

System Call vs. Library function

- 시스템 호출
  - OS Kernel이 제공하는 서비스를 이용해 프로그램을 작성할 수 있도록 제공되는 프로그래밍 인터페이스
  - 기본적인 형태는 C 언어의 함수 형태로 제공

리턴값 = 시스템호출명(인자, ...);

    - C Library 가 이러한 형태로 편리하게 이용할 수 있게 제공해주는 것

- 라이브러리 함수
  - 라이브러리 : 미리 컴파일된 함수들을 묶어서 제공하는 특수한 형태의 파일
    - /lib, /usr/lib에 위치하며 lib*.a 또는 lib*.so 형태로 제공
  - 자주 사용하는 기능을 독립적으로 분리하여 구현해둠으로써 프로그램의 개발과 디버깅을 쉽게하고 컴파일을 좀 더 빠르게 할 수 있다
  - OS 커널과 무관하게 단순한 C 코드를 수행
    - 예) strcpy() : 문자열 복사를 위한 연산을 수행. 편의를 위한 라이브러리 함수

System Call vs. Library function

- 두 가지 시스템 콜 호출 방법
  - 직접 시스템콜 호출: 본래 시스템콜은 특수한 방식(trap)을 통해 호출하여야 함
    - 이유1: 시스템콜의 코드가 일반 사용자가 접근할 수 없는 OS 커널 영역에 존재하기 때문
    - 이유2: 시스템콜은 다수 존재하고, 번호로 구분하기 때문에 직관적이지 않음
    - 굳이 하려면? : syscall() 함수 이용

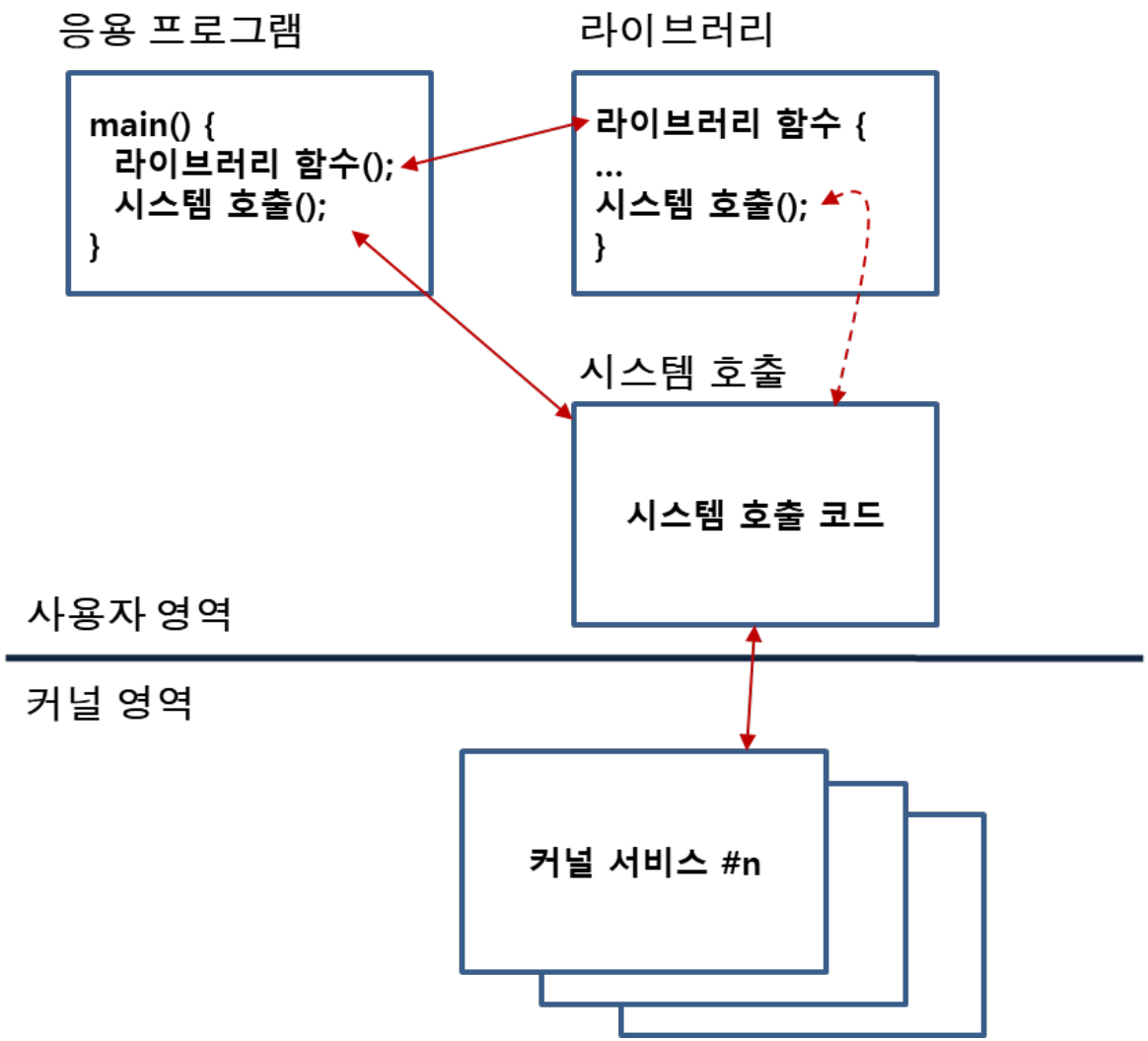
```
NAME
       syscall - indirect system call

SYNOPSIS
       #include <unistd.h>
       #include <sys/syscall.h>   /* For SYS_xxx definitions */

       long syscall(long number, ...);
```

  - **라이브러리 함수 사용**
    - C 라이브러리는 시스템콜을 간편히 사용할 수 있게 도와주는 여러 함수 제공
    - 예) printf() : 화면 출력을 위해서는 본래 write() 시스템 콜을 사용해야 함
    - 일반적으로 거의 모든 경우에 라이브러리 함수를 통해 시스템콜을 이용함

System Call vs. Library function



Syscall() 의 사용

```
ubuntu@41983:~/hw1$ vi syscall.c
ubuntu@41983:~/hw1$ gcc -o syscall syscall.c
ubuntu@41983:~/hw1$ ./syscall & ps
[1] 779962
My pid: 779962
My pid: 779962 by syscall()
    PID TTY          TIME CMD
 659370 pts/0    00:00:00 bash
 779962 pts/0    00:00:00 syscall
 779963 pts/0    00:00:00 ps
ubuntu@41983:~/hw1$
```

```c
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <stdio.h>

int main (void) {

        printf("My pid: %d\n", getpid());
        printf("My pid: %d by syscall()\n", (int) syscall(39));

        sleep(1);

        return 0;
}
```

```
NAME
       getpid, getppid - get process identification

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

       pid_t getpid(void);
       pid_t getppid(void);
```

- getpid()
  - 현재 프로세스의 PID를 반환
  - C 라이브러리 함수
  - 이 정보는 OS가 관리하기 때문에, 시스템콜을 통해 수행됨
  - 이때 sys_getpid 라는 이름의 시스템콜을 이용하며, 해당 시스템콜 번호는 39

(참고) Man page 에서의 구분

- 시스템 호출 : man 페이지가 섹션 2에 속함

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

NAME
       getpid, getppid - get process identification

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

       pid_t getpid(void);
       pid_t getppid(void);
```

- 라이브러리 함수 : man 페이지가 섹션 3에 속함

```
STRCPY(3)                                                Linux Programmer's Manual

NAME
       strcpy, strncpy - copy a string

SYNOPSIS
       #include <string.h>

       char *strcpy(char *dest, const char *src);

       char *strncpy(char *dest, const char *src, size_t n);
```

(참고) Man page 에서의 섹션 구분

man page sections

Each man page comes in sections. The table below shows the section numbers of the manual followed by the types of pages they contain:

- **Section # 1** : User command (executable programs or shell commands)
- **Section # 2** : System calls (functions provided by the kernel)
- **Section # 3** : Library calls (functions within program libraries)
- **Section # 4** : Special files (usually found in /dev)
- **Section # 5** : File formats and conventions eg /etc/passwd
- **Section # 6** : Games
- **Section # 7** : Miscellaneous (including macro packages and conventions),
- **Section # 8** : System administration commands (usually only for root)
- **Section # 9** : Kernel routines [Non standard]

(참고) Man page 에서의 섹션 구분

- 같은 이름의 page 가 존재하는 경우, section 으로 구분해서 확인할 수 있음
  - $ man <section> <page>
- $ man 1 printf

```
PRINTF(1)                                                User Commands

NAME
       printf - format and print data

SYNOPSIS
       printf FORMAT [ARGUMENT]...
       printf OPTION
```

- $ man 3 printf

```
PRINTF(3)                                        Linux Programmer's Manual

NAME
       printf,  fprintf,  dprintf, sprintf, snprintf, vprintf, vfprintf, vdpri
       sion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);
```

Error Handling

시스템 호출의 오류 처리방법

- 결과값: 성공하면 0을 리턴, 실패하면 -1을 리턴
- 실패 시, 전역변수 errno에 오류 코드 저장
  - Extern 을 이용해 C 라이브러리와 사용자 프로그램이 전역 변수를 공유할 수 있음
  - Extern: 해당 소스 파일의 외부에서 선언한 변수를 인용해서 사용하는 것
- 오류 코드의 확인: errno 유틸리티 사용

```
ubuntu@41983:~/hw1$ vi errno.c
ubuntu@41983:~/hw1$ gcc -o errno errno.c
ubuntu@41983:~/hw1$ ./errno
errno=2
ubuntu@41983:~/hw1$ errno 2

Command 'errno' not found, but can be installed with:

sudo apt install moreutils

ubuntu@41983:~/hw1$ sudo apt install moreutils
Reading package lists... Done
Building dependency tree
```

```
Setting up libtimedate-perl (2.3200-1) ...
Setting up moreutils (0.63-1) ...
Processing triggers for man-db (2.9.1-1) ...
ubuntu@41983:~/hw1$ errno 2
ENOENT 2 No such file or directory
ubuntu@41983:~/hw1$
```

```c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

extern int errno;

int main(void) {
    if (access("unix.txt", F_OK) == -1) {
        printf("errno=%d\n", errno);
    }
    return 0;
}
```

라이브러리 함수의 오류 처리방법

- 오류가 발생하면 NULL을 리턴, 함수의 리턴값이 int 형이면 -1 리턴
- errno 변수에 오류 코드 저장

```
ubuntu@41983:~/hw1$ vi errno_lib.c
ubuntu@41983:~/hw1$ gcc -o errno_lib errno_lib.c
ubuntu@41983:~/hw1$ ./errno_lib
errno=2
ubuntu@41983:~/hw1$
```

```
NAME
       fopen, fdopen, freopen - stream open functions

SYNOPSIS
       #include <stdio.h>

       FILE *fopen(const char *pathname, const char *mode);

       FILE *fdopen(int fd, const char *mode);

       FILE *freopen(const char *pathname, const char *mode, FILE *stream);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       fdopen(): _POSIX_C_SOURCE
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>

extern int errno;

int main(void) {
    FILE *fp;

    if ((fp = fopen("unix.txt","r")) == NULL) {
        printf("errno=%d\n", errno);
        return 1;
    }
    fclose(fp);

    return 0;
}
```

- fopen()
  - File stream 을 여는 C 라이브러리 함수
  - 해당 파일이 없으면 에러가 나고, NULL 을 리턴
  - 다음 강의에서 더 배워봅시다!

보다 편리한 오류 처리

- 오류 메시지 출력 : perror(3)
- Errno 에 따라 에러 메시지를 출력함

```
PERROR(3)                                          Linux Programmer's Manual

NAME
       perror - print a system error message

SYNOPSIS
       #include <stdio.h>

       void perror(const char *s);
```

```
ubuntu@41983:~/hw1$ vi perror.c
ubuntu@41983:~/hw1$ gcc -o perror perror.c
ubuntu@41983:~/hw1$ ./perror
my message: No such file or directory
```

```c
#include <unistd.h>
#include <stdio.h>

int main(void) {
    if (access("unix.txt", F_OK) == -1) {
        perror("my message");
        return 1;
    }
    return 0;
}
```