Spring 2019 Operating Systems
Lab 05: UNIX Error Catch

0. 과제 수행 전 유의사항
프로그래밍은 자기 손으로 작성해야 한다. 아이디어를 서로 공유하고 도움을 받는 것은 좋으나, 다른 사람의 코드를 그대로 가져다 쓰는 것은 자신에게도 손해이고 다른 사람에게 피해를 끼치는 결과를 낳게되므로 허용하지 않는다. 과제 체크 시 두 사람의 코드가 유사하면, 두 사람 모두에게 과제 점수를 부여할 수 없으니 유의바란다.

1. 실습의 목표
오류처리는 프로그래머의 기본자질 중에 하나이다. 본 실습에서는 UNIX 계열의 운영체제에서 오류의 내용을 찾아볼 수 있는 방법에 대하여 배운다.

2. errno
대부분의 라이브러리 함수는 오류가 발생하면 errno 라는 변수에 양의 정수로 된 오류번호를 저장한다. 따라서, 프로그래머는 errno 변수에 저장된 내용을 살펴보면 어떤 오류인지를 파악할 수 있다.

```
errno 변수
기능
        오류 번호를 저장한다
기본형
        extern int errno;
헤더 파일
        <errno.h>
```

[errno_1.c]

```
 1
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <sys/types.h>
 5 #include <sys/stat.h>
 6 #include <fcntl.h>
 7 #include <errno.h>
 8
 9 void
10 main(void)
11 {
12     int fd;
13
14     if ((fd = open("nodata", O_RDONLY)) == -1) {
15         printf("errno = %d\n", errno);
16         exit(1);
17     }
18
19     exit(0);
20 }
21
```

이 코드를 실행하면 "errno = 2" 라는 출력결과를 볼 수 있을 것이다.

14번 라인을 보면 fopen 함수가 아닌 open 함수를 사용하였다. open 함수는 시스템에서 사용하는 함수이다. 기본적으로 POSIX를 만족하는 표준 라이브러리이다. 따라서, 어떤 운영체제건 POSIX를 지원하는 운영체제라면 open 함수를 이용할 수 있다.

일반적으로 위와 같은 코드에서 오류가 발생했을 때, fopen 함수는 NULL을 반환하고 open 함수는 -1을 반환한다. 이는 포인터를 반환하는 함수와 정수값을 반환하는 함수의 차이이다.

함수 호출에 오류가 발생하면 errno 변수에 0이 아닌 양의 정수값이 설정되고, 일반적으로 0은 오류가 없음을 의미하므로 함수 호출전에 errno 변수에 0을 초기화 하는 것이 바람직하다.

과제: POSIX 란 무엇인지 정리하시오.

[errno_2.c]

```c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <errno.h>

void
main(void)
{
    double y;

    errno = 0;
    y = sqrt(-1);

    if (errno != 0) {
        printf("errno = %d\n", errno);
        exit(1);
    }
    exit(0);
}

```

이 코드는 errno = 33 이라는 결과를 출력한다.

이처럼 라이브러리 함수 호출에 오류가 발생하면 errno에 0이 아닌 양의 정수가 설정되고, 이러한 오류 관련 정수값을 이해하기 쉽도록 <errno.h> 헤더 파일에 133개의 매크로를 정의해놓았다.

[errno_3.c]

```c

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

void
main(void)
{
    int fd;

    if ((fd = open("nodata", O_RDONLY)) == -1) {
        if (errno == ENOENT) {
            printf("nodata is not exist.\n");
        } else {
            printf("unexpected error: errno = %d\n", errno);
        }
        exit(1);
    }
    exit(0);
}

```

과제: 다음 코드를 실행해보고 코드를 분석하시오. 참고로 ENOENT는 "그와 같은 파일 혹은 디렉토리가 없다" 라는 뜻이다.

3. assert

assert는 어떤 조건이 거짓이면 오류 메시지를 출력하고 프로그램의 현재 상태를 코어덤프 한 후 프로그램을 종료시키는 함수로 사용법은 다음과 같다.

```
assert 함수
기능
        expression이 거짓이면 오류 메시지 출력, 코어 덤프하고 프로그램을 종료시킨다.
기본형
        void assert(int expression);
        expression: 참과 거짓을 판별하는 수식
반환값
        없음
헤더 파일
        <assert.h>
```

[assert_1.c]

```
 1
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <assert.h>
 5
 6 void
 7 main(void)
 8 {
 9     FILE *fp;
10
11     fp = fopen("yesdata", "r");
12
13     assert(fp);
14     printf("yesdata exists.\n");
15     fclose(fp);
16
17     fp = fopen("nodata", "r");
18     assert(fp);
19     printf("nodata exists\n");
20     fclose(fp);
21
22     exit(0);
23 }
24
```

11번 라인에서 사용하는 yesdata 라는 빈 파일을 하나 만들고 테스트한다.

11번 라인에서 fp는 거짓(NULL)이 아니므로 다음 문장이 실행되나, 17번 라인의 fp는 거짓이므로 실행 결과에서 오류메시지를 출력한 후 코어 덤프한 후 종료하게 된다.

프로그램을 개발할 때, assert 함수를 사용해 테스트와 디버깅을 하면 효율적이다. 하지만, 프로그램 개발이 완료되면 assert 함수의 호출 부분을 모두 지워야하기 때문에 여간 번거로운일이 아닐 수 없다. 이를 위한 다음 예제를 살펴본다.

[assert_2.c]

```
 1
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 #define NDEBUG
 6 #include <assert.h>
 7
 8 void
 9 main(void)
10 {
11     FILE *fp;
12
13     fp = fopen("yesdata", "r");
14     assert(fp);
15     printf("yesdata exists\n");
16     fclose(fp);
17
```

```
18      fp = fopen("nodata", "r");
19      assert(fp);
20      printf("nodata exists\n");
21      fclose(fp);
22
23      exit(0);
24  }
25
```

위 코드는 1번과 동일하나, 5번 라인의 NDEBUG 라는 선언을 함으로써 assert 함수를 실행하지 않고 지나칠 수 있다. 따라서, nodata 파일을 만들지 않았음에도 exists 문장이 출력된다.

과제: 코어 덤프가 무엇인지 조사해보시오.

4. abort

abort는 프로그램의 현재 상태를 코어 덤프하고 프로그램을 비정상적으로 종료시키는데, 엄밀히 말하면 abort 함수를 호출한 프로그램은 SIGABRT 시그널을 받아 종료되는 것이다. 시그널에 대해서는 추후 살펴본다.

abort 함수
기능
        코어 덤프하고 프로그램을 종료시킨다.
기본형
        void abort(void);
반환값
        없음
헤더파일
        <stdlib.h>

[abort_1.c]

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void
6  main(void)
7  {
8      abort();
9
10     printf("not run\n");
11 }
12
```

위 코드는 abort 함수에 의해 코어 덤프하고 비정상적으로 프로그램이 종료된다. 따라서 10번 라인은 실행되지 않는다.

[abort_2.c]

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void my_assert(int);
6
7  void
8  main(void)
9  {
10     my_assert(7 == 7);
11     printf("yes\n");
12
```

```
13      my_assert(3 == 5);
14      printf("no\n");
15  }
16
17  void
18  my_assert(int expression)
19  {
20      if (!(expression)) {
21          printf("%s:%d:Assertion failed\n", __FILE__, __LINE__);
22          abort();
23      }
24  }
25
```

이전 절인 3번 assert 함수에서는 오류메시지를 출력하고, 코어 덤프하고 프로그램을 종료한다고 하였다. 이는 내부적으로 abort 함수를 호출하는 것이다.

즉, abort 함수를 이용하여 assert 함수를 구현할 수 있다. [assert_2.c] 의 예제가 바로 그런 것이다.

21번 라인에서 __FILE__은 파일의 이름을, __LINE__은 현재 라인번호를 의미하는 매크로이다.

5. strerror

errno 변수를 이용하면 오류가 발생한 정수값을 얻을 수 있지만 그 정수값의 의미를 명확히 모르면 원인이 무엇인지 알기 쉽지않다. 이 때, 오류의 원인을 정수값이 아니라 풀어서 설명해주면 좋겠다. 그것이 strerror 함수이다.

```
strerror 함수
기능
        오류 번호를 설명하는 문자열을 반환한다
기본형
        char *strerror(int errnum);
        errnum: 오류번호
반환값
        성공: 오류 번호를 설명하는 문자열
        실패: unknown error 메시지
헤더파일
        <string.h>
```

[strerror_1.c]

```
 1
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4  #include <string.h>
 5  #include <errno.h>
 6
 7  void
 8  main(void)
 9  {
10      FILE *fp;
11
12      if ((fp = fopen("nodata", "r")) == NULL) {
13          fprintf(stderr, "ERROR: %s\n", strerror(errno));
14          exit(1);
15      }
16
17      exit(0);
18  }
19
```

13번 라인의 strerror 함수는 오류번호인 errno를 전달받아 원인에 대한 설명인 "No such file or directory"를 반환한다. 여기서 strerr 는 표준 오류인 모니터를 의미한다.

결론적으로, strerror 함수를 이용하면 각 정수값에 대한 설명을 직접 확인할 수 있고, 여기서 0은 "success" 를 의미한다.

[strerror_2.c]

```
 1
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <errno.h>
 6
 7 void
 8 main(void)
 9 {
10     int i;
11
12     for (i = 0; i < 134; i++) {
13         printf("[%3d] %s\n", i, strerror(i));
14     }
15     exit(0);
16 }
17
```

위 예제는 오류번호에 해당하는 설명을 모두 출력해준다. 세대를 거듭하면서 오류의 갯수는 133개가 되었다.

6. perror

strerror 함수는 오류 번호를 통해 오류의 원인을 알아내야 했다. 하지만, 프로그래머들은 사용하기 더 편리한 함수를 만들었다. 그것이 perror 함수이며 오류가 발생했을 때 시스템이 오류 메시지를 자동으로 출력한다.

perror 함수
기능
        오류 메시지를 출력한다
기본형
        void perror(const char *s);
        s: 오류 발생 시 오류 메시지 앞부분에 출력할 문자열
반환값
        없음
헤더 파일
        <stdio.h>

[perror_1.c]

```
 1
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 void
 6 main(void)
 7 {
 8     FILE *fp;
 9
10     if ((fp = fopen("nodata", "r")) == NULL) {
11         perror("ERROR");
12         exit(1);
13     }
14     exit(0);
15 }
16
```

위 코드는 11번 라인을 제외하고는 앞서 작성한 코드와 다를 것이 없다. 하지만, 실행화면에서 볼 수 있듯이 시스템이 오류에 맞는 메시지를 자동으로 출력한다.

하지만, perror와 strerror 함수는 10번 라인처럼 사용자가 별도의 체크문을 만들어 줘야하는 약간의 불편함이 존재한다.

7. 제출

위 1~6에 해당하는 모든 코드를 작성하고 테스트한 결과를 각각 캡쳐한다음 워드파일로 정리한다. 중간중간 "과제" 라고 적힌 부분은 해당하는 출력결과의 아래부분에 작성해서 제출한다.