강의명: 유닉스시스템

실습 번호: 3

실습 제목: 파일 입출력(File Input/Output)

학생 이름: 김준호

학번: 201710860

1 파일 생성

1.1

실습 내용을 살펴보면 새 파일을 생성하되 만약 생성할 수 없는 파일이거나 같은 이름의 파일이 존재하면 오류 메세지를 출력하고 종료해야한다. 또 만약 파일이 열리면 2번째 인자의 문자를 세 번째 인자만큼 출력해야한다.

따라서 먼저 파일을 open 해야하고, validation후, write을 해야한다.

먼저 system call open 함수의 prototype은 아래와 같다

Text

Description automatically generated

이들은 file descriptor를 return한다.

open함수는 fcntl.h에 정의돼 있기 때문에 이를 include한다.

실습에서 파일을 생성하려면 해당 파일에 permission이 필요하므로 두 함수 중

int open(const char \*pathname, int flags, mode\_t mode);

위 제시된 함수를 사용할 것이다. 그렇지 않으면 에러가 발생한다.

open의 인자로는 첫째로 filename, 어떻게 사용할지에 대한 flags 그리고 permission에 해당하는 mode가 있다.

프로그램을 실행할 때 첫 번째 인자로 파일 이름을 준다. 따라서 pathname에는 argv[1] 와 같이 인자를 전달할 수 있다.

두 번째 인자로는 flags를 줘야하는데 이는 fcntl.h에 정의돼있다.

파일을 열고, 쓰고 (write), 이미 같은 이름으로 만들어진 파일이 있다면 error를 생성해야한다. 이들은 flag로 각각 아래와 같이 정의 돼있다.

O\_CREAT : 새 파일 생성, 추가 인수 필요

O\_WRONLY : write only

O\_EXCL : O\_CREAT과 같이 사용되면 존재하는 파일을 또 생성할 때 error 생성

세 번째 인자로는 가장 기본적인 permission인 644를 준다.

6은 owner, 즉 파일을 생성하는 user의 permission이며, 가운데 4는 group에 대한 4이다. 끝에 4는 그 외의 user들에 대한 permission이다.

각각의 permission에 대한 값은 아래와 같이 정의된다,

read는 4, write은 2, execute는 1이다. permission 값은 이 값들의 합으로 정의된다.

6(=4 + 2)은 read, write 할 수 있다는 뜻이며, 4는 read만 할 수 있게 한다는 뜻이다.

mode의 값은 보통 8진수로 주어지기 때문에 앞에 0644와 같이 인자로 전달한다.

따라서 코드를 아래와 같이 작성하였다.

Text

Description automatically generated

먼저 위에서 설명한 바와 같이 fd = open(argv[1], O\_CREAT | O\_WRONLY | O\_EXCL, 0644) 와 같이 파일을 open한다. 이때 open함수는 에러가 발생했을 때 -1을 return하므로 이를 검사한다.

만약 open의 return값이 -1이라면 에러가 발생한 것이므로 에러를 출력하고 종료한다.

그리고 write system call을 사용하여 인자로 주어지는 문자들을 일정 횟수 반복하여 출력해야한다.

먼저 세 번째 인자로 반복 횟수가 주어지므로 반복문으로 해당 횟수 만큼 출력을 반복하면 된다. 인자로 주어지는 값들은 string이므로 이를 정수형으로 변환해줘야한다.

이를 위해atoi 함수를 사용할 것이고, 이는 stdlib.h 에 정의돼 있다.

코드는 아래와 같다.



그리고 write system call로 두 번째 인자로 주어지는 문자를 반복해서 출력해야한다.

write의 prototype은 아래와 같다.

A picture containing text

Description automatically generated

첫 번째 인자 fd로는 위에서 받아온 fd를 넘겨줄 것이며, buf로는 write할 문자가 저장된 주소인 argv[2]를 줄 것이다. 그리고 문자가 주어지므로 count는 1로 한다.

따라서 코드는 아래와 같이 작성하였다.

Text

Description automatically generated

write함수는 return 값으로 write한 byte 수를 return한다. 문자는 1 byte 이므로 만약 1 byte을 return하지 않으면 무언가 문제가 생긴 것이므로 이를 validation하며 반복문을 수행하였다.

마지막으로 fd를 close하고 정상적으로 종료하였기 때문에 exit(0); 을 호출한다.

Text

Description automatically generated with medium confidence

perror 함수는 stdio.h 에 정의 돼있다. 그리고 perror는 내부적으로 errno.h에 정의돼있는 값들을 사용한다.

1.2

Text

Description automatically generated with medium confidence

잘 출력되었다.

1.3

Graphical user interface, text, application

Description automatically generated

root directory에는 파일을 생성할 권한이 없으므로 Permssion denied error가 발생하였다.

1.4

Graphical user interface, application

Description automatically generated

파일이 이미 존재하므로 File exists error가 발생하였다.

2 파일 추가

2.1

1.1 의 설명과 마찬가지로 file을 open하고, 거기에 append하면 된다.

따라서 먼저 open을 하는데 flag는 다음과 같다.

O\_WRONLY : write only

O\_APPEND : Append

이미 존재하는 파일을 여는 것이므로 mode는 사용하지 않는다.

따라서 아래의 open함수를 사용한다.

int open(const char \*pathname, int flags);

flags, open을 사용하기 위한 fcntl.h를 include하고 그 외에 perror, printf를 위해 stdio.h, exit을 위해 stdlib.h, perror가 내부적으로 사용할 errno.h를 include한다.

open 함수는 파일을 열지 못했을 때 -1을 return하므로 만약 open의 return값이 -1이라면 에러 코드를 출력하고 종료한다.

실제 코드는 아래와 같이 작성하였다.

Text

Description automatically generated

문자를 append할 파일을 열었다면 반복적으로 write system call을 사용해 문자를 출력해주면 된다.

write system call의 prototype은 아래와 같다.

Text

Description automatically generated with low confidence

먼저 buffer로 쓰일 buf는 문자를 하나씩 읽으면 1byte로도 충분하므로 char 변수 c를 선언해주어 이 변수의 주소를 인자로 줄 것이고, count는 1로 할 것이다.

write가 성공적으로 완료되면 해당 파일의 읽기/쓰기 위치인 file offset이 write한 byte 수 만큼 증가한다.

메인 함수의 세 번째 인자가 반복 횟수, 두 번째 인자가 출력할 문자이고, write가 제대로 되었는지 확인하기 위해 write함수의 return값이 1인지 조사하여 1이 아니라면 에러를 출력하도록 작성하였다.

실제 코드는 아래와 같이 작성하였다.

Text

Description automatically generated

fd를 close하고 프로그램을 종료한다.

2.2

Text

Description automatically generated

처음에 입력하였던 H 512개 뒤에 X 512개가 “append”됐다.

2.3

A black background with white text

Description automatically generated with medium confidence

해당 파일이 없다는 에러를 출력하고 종료된다.

3 파일 보기

3.1

1, 2 실습에서 설명한 것같이 아래 함수들을 위해 “:” 다음의 라이브러리들을 include했다.

open, flags, fd : fcntl.h

exit : stdlib.h

printf, perror : stdio.h

close : unistd.h

perror의 error list: errno.h

3번 실습에서는 cat 함수를 작성하는 것이므로 read system call을 사용해야 할 것이다.

read system call은 unistd.h 에 정의돼있다. 이를 모두 포함하여 아래와 같이 include문들을 작성하였다.

Text, chat or text message

Description automatically generated

3번 실습에서는 read만 할 것이므로 open system call의 flag로 O\_RDONLY만을 주면 된다.

그리고 예외를 처리하는 구문을 넣고 open 함수로 부터 file descriptor를 받아온다.

Text

Description automatically generated

이제 read를 해야한다. read system call의 prototype은 아래와 같다.

Graphical user interface, text, application

Description automatically generated

첫 번째 인자로 file descriptor, 두 번째 인자로는 읽어온 값을 저장할 버퍼, 세 번째 인자는 몇 개씩 읽을 것인지에 대한 count 값이다.

따라서 위 코드에서 9번째 줄에 char 형 변수 c를 선언했고 이 변수의 메모리를 버퍼로 사용할 것이다. 문자를 하나씩 읽으므로 count는 1이다.

실제 코드 작성은 아래와 같이 하였다.

Text

Description automatically generated

read 는 수행에 성공하면 read한 byte 수를 return한다. 이때 count는 1이므로 항상 1을 반환하여야 성공한 것이다. 만일 그렇지 않다면 두 가지 경우가 있다.

하나는 파일의 끝에 도달하여 0을 return 한 경우와 오류 시 -1을 return 한 경우이다.

각각 while문에서 탈출, 혹은 error 문구를 출력하고 종료되게 프로그램을 작성했다.

문자를 잘 읽어왔다면 그대로 출력한다.

A picture containing text

Description automatically generated

fd 를 close하고 정상 종료한다.

3.2

Text

Description automatically generated

화면으로 출력되는 것을 보니 표준 출력으로 잘 출력되고 있다.

3.3

Graphical user interface, text, application

Description automatically generated

에러가 출력되고 종료하였다.

4 파일 복사

4.1

write, close : unistd.h

FD MACRO, flags : fcntl.h

error list for perror : errno.h

printf, perror : stdio.h

exit : stdlib.h

함수 : 정의된 헤더 파일

위와 같이 필요하므로 모두 아래와 같이 include 해준다.

Text

Description automatically generated

큰 파일을 옮겨야 하므로 지금까지는 버퍼의 크기를 1로 했지만 이번에는 ext4 파일 시스템의 disk block size인 4K, 즉 4096으로 버퍼 크기를 사용할 것이다.

따라서 아래와 같이 정의하였다.



파일을 읽고, 쓰고를 동시에 해야하므로 file descriptor가 2개가 필요하다.

따라서 실습 1, 3에서 했던 코드를 거의 그대로 사용하였다.

코드는 아래와 같다.

Text

Description automatically generated

fd\_read는 읽을 파일을 여는 file descriptor, fd\_write는 새로 write할 file의 file descriptor이다. 읽는 것의 flags는 read only의

O\_RDINLY

write할 file descriptor의 flags는 새로 만들고, 쓰고, 혹시 이미 파일이 있다면 에러를 출력해야하기 때문에 아래와 같다.

O\_CREAT | O\_WRONLY | O\_EXCL

mode도 실습 1과 마찬가지로 0644로 주었다.

그리고 에러 상황에 대하여 에러를 출력하고 종료할 코드도 넣었다.

그리고 while문에서 더 이상 읽을 문자열이 없을 때까지 읽고 읽은 버퍼를 가져다 그대로 write하는 코드를 작성하였다.

Text

Description automatically generated

n은 write와 read의 return 값을 확인하여 에러 출력과 종료를 하기 위함이다.

c 는 버퍼를 선언하였고 크기는 BUFSIZE == 4096 이다.

fd를 close하고 함수를 정상 종료한다.

4.2

Graphical user interface, text

Description automatically generated

diff는 만약 두 파일 간에 다른 부분이 있다면 이를 출력해주는 명령어이다.

mycopy, diff 아무것도 출력하지 않았으므로 mycopy는 정상 종료, diff에서는 두 파일은 완전히 똑같다는 결론을 얻을 수 있다.

4.3

Text

Description automatically generated with medium confidence

이미 앞서 mybigfile을 만들었기 때문에 에러를 출력하고 종료했다.

5 입출력 효율성

5.1

Code

실습 4의 코드에 아래 코드를 추가하여 버퍼의 크기를 출력하였다.

Text

Description automatically generated

BUFSIZE == 128

Text

Description automatically generated

BUFSIZE == 1024

Text

Description automatically generated

BUFSIZE == 4096

Text

Description automatically generated

BUFSIZE == 65536

Text

Description automatically generated

BUFSIZE == 524288

Text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| 버퍼 크기 | Real | User | Sys |
| 128 | 29.478s | 7.828s | 21.648s |
| 1024 | 4.526s | 1.076s | 3.448s |
| 4096 | 1.766s | 0.224s | 1.532s |
| 65536 | 1.202s | 0.040s | 1.164s |
| 524388 | 1.178s | 0.004s | 1.172s |

5.2

real:

wall clock time을 의미하며 process의 시작부터 끝까지 수행하는데 걸린 시간이다.

user:

user CPU time을 의미하며 process가 processor를 점유한 시간을 의미한다.

sys:

system CPU time을 의미하며 process를 실행하기 위해 실행된 kernel이 수행된 시간.

5.3

program의performance는 빠를 수록 좋고 사용하는 memory는 작을 수록 좋다.

BUFSIZE가 128 에서 4096까지는 그래도 memory를 더 사용할 수록 performance가 좋아진다고 볼 수 있지만 4096 이후로는 거의 차이가 없음을 알 수 있다.

또한 stat -f . 명령어로 현재 사용중인 block size를 알아보니 아래와 같았다.

Text

Description automatically generated

system에서도 disk로 부터 4096byte씩 data를 가져오니 user program에서도 딱 그만큼씩 가져오는게 잘 아껴쓰는게 아닐까 생각한다.

4096byte

5.4

Text

Description automatically generated

먼저 mybigfile이 실습 디렉토리에 있는 것을 확인했고 이를 삭제하였다.

제대로 삭제되었는지 확인하기 위해 ls -hl 명령어로 total 용량과 각 file들의 용량을 확인하였다. total 용량은 88KB이다.

그리고 /home/unix/bigfile의 용량도 확인해보니 1.6GB 였다.

실습 디렉토리의 용량이 88KB이니 잘 지워진 것으로 확인된다.

끝.

ps.

글자 깨짐에 대하여

맥의 한글 인코딩과 윈도우에서의 한글 인코딩이 달라 발생하는 것으로 알고 있습니다!

$ convmv -r -f utf8 -t utf8 --nfc --notest <dir>

와 같이 사용하여 해결한다고 해결할 수 있다고 합니다.

제 맥에서 위 명령어로 인코딩을 다시하여 보내드립니다!