**강의명: 유닉스 시스템**

**실습 번호: 3**

**실습 제목: 파일 입출력(File Input/Output)**

**학생 이름: 황귀훈**

**학번: 201710885**

**1. 파일 생성**

**1.1**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <errno.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main(int argc, char \*argv[])**

**{**

**int fd, length, i, n;**

**if ((fd = open(argv[1], O\_CREAT | O\_WRONLY | O\_EXCL, 0644)) == -1) {**

**perror("open");**

**exit(1);**

**}**

**length = atoi(argv[3]);**

**for(i = 0; i < length; i++) {**

**n = write(fd, argv[2], 1);**

**if(n != 1) {**

**perror("write");**

**exit(1);**

**}**

**}**

**close(fd);**

**exit(0);**

**}**

실습 1번 파일 생성을 하기위해 작성한 코드는 위와 같다. System call 함수를 사용하기 위해 그 함수의 prototype과 macro들을 저장하고 있는 헤더파일들을 include 해줘야 하는데 위 코드에서 사용하는 헤더파일들의 정보는 다음과 같다.

<unistd.h> =>POSIX운영체제 API에 대한 엑세스를 제공하는 헤더파일이다. 위 코드에서는 write(), close()함수를 사용하기 위해 include했다.

<fcntl.h> => 위 코드에서 open()함수를 사용하고 open()함수를 사용할 때 flags를 사용하기 위해 include했다. Flags는 파일을 어떤 용도로 open할 지를 결정한다.

<errno.h> => perror()의 error list에 대한 매크로가 저장 되어있다.

<stdio.h> => perror() 함수가 정의되어 있다.

<stdlib.h> => atoi(), exit() 함수가 정의되어 있다.

(명령어 “./mycreate out1 H 512” 에서 argv[0]은 ./mycreate, argv[1]은 out1, argv[2]은 H, argv[3]은 512이다. 이 때 512는 string)

실습 1번 파일 생성은 argv[1]에 주어지는 파일 이름으로 새 파일을 생성하는데 만약 생성할 수 없는 파일이거나, 같은 이름의 파일이 존재하면 오류 메시지를 출력하고 종료한다. 이 파일에 argv[2]에 주어지는 문자를 argv[3]에 주어지는 정수 개수만큼 출력하는 프로그램을 작성해야 한다. 따라서 먼저 System call open() 함수를 사용하여 파일을 열거나 생성하고 file descriptor를 넘겨 줘야한다. 생성되는 파일 명은 argv[1]으로 넘어오는 값이기 때문에 open()함수의 첫 번째 argument에 argv[1]을 넣어주었고, 새 파일을 생성하고 write하며 이미 존재하는 파일을 생성할 때 오류 메시지를 출력해야 하기 때문에 두 번째 argument인 flag에는 O\_CREAT | O\_WRONLY | O\_EXCL를 사용했다. (‘|’는 bitwise or 이다.) open()함수의 세 번째 argument로 mode에는 0644를 주었는데 앞에서 부터 두 번째 자리의 ‘6’은 user는 read, write를 할 수 있음을 의미하고 세 번째 자리의 ‘4’는 같은 group 사람이 read할 수 있음을 의미하며 마지막 자리의 ‘4’는 user도 group도 아닌 다른 사람이 read 할 수 있음을 의미한다. 시스템 호출의 경우 return value를 꼭 체크하고 문제가 있는 경우 에러 메시지를 출력해야 하기 때문에 if문을 사용하여 만약 open()의 결과 반환되는 fd 값이 -1 일 경우(문제가 있는 경우) perror()함수를 통해 오류 메시지를 출력하고 프로그램을 종료했다. 프로그램을 종료할 때 오류로 종료할 경우 exit(1)을 사용한다. 성공적으로 종료할 경우 exit(0)를 사용하고 exit(0)이 아닌 경우는 모두 오류로 종료한 경우이다. 이 때 문제가 있는 경우는 생성할 수 없는 파일이거나 같은 이름의 파일이 이미 존재할 경우이다.

파일을 열었으면 그 다음으로 이 파일에 argv[2]에 주어지는 문자를 argv[3]에 주어지는 정수 개수만큼 출력해야 하기 때문에 for문을 사용했는데 먼저 atoi() 함수를 사용하여 argv[3]에 주어지는 string값을 정수 값으로 변환하여 반복 횟수를 설정했다. argv[2]의 문자를 출력하기 위해 System call write() 함수를 사용했고 첫 번째 argument로 출력할 파일의 fd값을 주고 두 번째 argument로는 출력할 문자 argv[2]을 주고, 세 번째 argument로 1을 줬다. 이 때 1은 출력할 데이터의 byte값이다. (char 값을 출력하므로 1 byte) write() 함수 역시 경우 return value를 꼭 체크하고 문제가 있는 경우 에러 메시지를 출력해야 하기 때문에 if문을 사용하여 체크했는데, write()함수는 성공 시 write한 byte수 1을 return 하기 때문에 return값이 1이 아닌 경우 perror() 함수를 통해 오류 메시지를 출력하고 exit(1) 함수를 사용하여 종료하게 했다. exit(0)이 아닌 경우는 모두 오류로 종료한 경우이다.

마지막으로 파일을 열고 출력하는 과정이 끝나면 더 이상 사용하지 않는 파일을 닫아야 하기 때문에 close()함수에 file descriptor를 전달하여 파일을 닫고, exit(0)을 사용하여 성공적으로 종료하게 하였다.

**1.2**

테이블이(가) 표시된 사진

자동 생성된 설명

"./mycreate out1 H 512"로 수행한 후 “cat out1”을 수행하여 확인한 결과 argv[2]의 H 값이 argv[3]의 512개 출력되어 파일 “out1”이 잘 생성됨을 확인했다. 파일 “out1”의 문자수를 새는 “wc -m out1” 명령어를 수행하면 512가 나온다.

**1.3**



명령줄 "./mycreate /xyz X 512" 수행 결과이다. 학생은 root directory밑에 파일을 생성할 권한이 없기 때문에 오류 메시지가 나온다.

**1.4**



명령 줄 "./mycreate out1 H 512"을 한 번 더 수행한 결과이다. 이미 존재하는 이름의 파일을 생성하려고 했기 때문에 오류 메시지가 나온다.

**2. 파일 추가**

**2.1**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <errno.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main(int argc, char \*argv[])**

**{**

**int fd, length, i, n;**

**if ((fd = open(argv[1], O\_APPEND | O\_WRONLY)) == -1) {**

**perror("open");**

**exit(1);**

**}**

**length = atoi(argv[3]);**

**for(i = 0; i < length; i++) {**

**n = write(fd, argv[2], 1);**

**if(n != 1) {**

**perror("write");**

**exit(1);**

**}**

**}**

**close(fd);**

**exit(0);**

**}**

헤더 파일은 1.1에서 설명한 것과 같다. 실습 2번 파일 추가는 argv[1]에 주어지는 파일 이름을 가지는 파일을 여는데 만약 이 파일이 존재하지 않으면 오류 메시지를 출력하고 종료한다. 그리고 이 파일에 argv[2]에 주어지는 한 문자를 argv[3]에 주어지는 정수 개 개수만큼 추가(append)하는 프로그램이다. 따라서 먼저 System call open() 함수를 사용하여 파일을 열거나 생성하고 file descriptor를 넘겨 줘야한다. 여는 파일 명은 argv[1]으로 넘어오는 값이기 때문에 open()함수의 첫 번째 argument에 argv[1]을 넣어주었고, append(파일 끝에 write)해야 하기 때문에 두 번째 argument인 flag에는 O\_APPEND | O\_WRONLY를 사용했다. (‘|’는 bitwise or 이다.) 이미 만들어진 파일을 열기 때문에 세 번째 argument로 mode는 주지 않았다. 시스템 호출의 경우 return value를 꼭 체크하고 문제가 있는 경우 에러 메시지를 출력해야 하기 때문에 if문을 사용하여 만약 open()의 결과 반환되는 fd 값이 -1 일 경우(문제가 있는 경우) perror()함수를 통해 오류 메시지를 출력하고 프로그램을 종료했다. 프로그램을 종료할 때 오류로 종료할 경우 exit(1)을 사용한다. 성공적으로 종료할 경우 exit(0)를 사용하고 exit(0)이 아닌 경우는 모두 오류로 종료한 경우이다. 이 때 문제가 있는 경우는 argv[1]에 주어지는 이름의 파일이 존재하지 않아 파일을 열 수 없는 경우이다.

파일을 열었으면 그 다음으로 append해야 하는데, 이 파일에 argv[2]에 주어지는 문자를 argv[3]에 주어지는 정수 개수만큼 출력해야 하기 때문에 for문을 사용했는데 먼저 atoi() 함수를 사용하여 argv[3]에 주어지는 string값을 정수 값으로 변환하여 반복 횟수를 설정했다. argv[2]의 문자를 출력하기 위해 System call write() 함수를 사용했고 첫 번째 argument로 출력할 파일의 fd값을 주고 두 번째 argument로는 출력할 문자 argv[2]을 주고, 세 번째 argument로 1을 줬다. 이 때 1은 출력할 데이터의 byte값이다. (char 값을 출력하므로 1 byte) write() 함수 역시 경우 return value를 꼭 체크하고 문제가 있는 경우 에러 메시지를 출력해야 하기 때문에 if문을 사용하여 체크했는데, write()함수는 성공 시 write한 byte수 1을 return 하기 때문에 return값이 1이 아닌 경우 perror() 함수를 통해 오류 메시지를 출력하고 exit(1) 함수를 사용하여 종료하게 했다. exit(0)이 아닌 경우는 모두 오류로 종료한 경우이다.

마지막으로 파일을 열고 출력하는 과정이 끝나면 더 이상 사용하지 않는 파일을 닫아야 하기 때문에 close()함수에 file descriptor를 전달하여 파일을 닫고, exit(0)을 사용하여 성공적으로 종료하게 하였다.

**2.2**

테이블이(가) 표시된 사진

자동 생성된 설명

명령 줄 "./myappend out1 X 512"로 수행한 후 “cat out1”을 통해 확인한 결과이다. 실습 1번에서 적은 H 512개 뒤로 X 512개가 append 되었다. “wc -m out1” 결과 1024가 출력된다.

**2.3**



현재 directory에 ‘nosuchfile’ 이라는 파일이 없어서 에러 메시지가 출력 되었다.

**3. 파일 보기**

**3.1**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <errno.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define BUFFSIZE 4096**

**char buf[BUFFSIZE];**

**int main(int argc, char \*argv[])**

**{**

**int fd, n;**

**if((fd = open(argv[1], O\_RDONLY)) == -1) {**

**perror("open");**

**exit(1);**

**}**

**while((n = read(fd, buf, BUFFSIZE)) >0) {**

**if(write(STDOUT\_FILENO, buf, n) != n) {**

**perror("write");**

**exit(1);**

**}**

**}**

**if(n<0) {**

**perror("read");**

**exit(1);**

**}**

**close(fd);**

**exit(0);**

**}**

헤더 파일은 1.1에서 설명한 것과 같고 추가적으로 실습 3번에서 read()함수와 STDOUT\_FILENO 매크로를 사용하는데 이 함수와 매크로는 <unistd.h>에 저장되어 있다.

실습 3번 파일 보기는 argv[1]에 주어지는 파일 이름을 가지는 파일을 읽어서 그 내용을 표준 출력으로 출력하는 프로그램이다. 따라서 먼저 System call open() 함수를 사용하여 파일을 열거나 생성하고 file descriptor를 넘겨 줘야한다. 여는 파일 명은 argv[1]으로 넘어오는 값이기 때문에 open()함수의 첫 번째 argument에 argv[1]을 넣어주었고, 파일을 읽기만 할 것이므로 두 번째 argument인 flag에는 O\_RDONLY를 사용했다. 이미 만들어진 파일을 열기 때문에 세 번째 argument로 mode는 주지 않았다. 시스템 호출의 경우 return value를 꼭 체크하고 문제가 있는 경우 에러 메시지를 출력해야 하기 때문에 if문을 사용하여 만약 open()의 결과 반환되는 fd 값이 -1 일 경우(문제가 있는 경우) perror()함수를 통해 오류 메시지를 출력하고 프로그램을 종료했다. 프로그램을 종료할 때 오류로 종료할 경우 exit(1)을 사용한다. 성공적으로 종료할 경우 exit(0)를 사용하고 exit(0)이 아닌 경우는 모두 오류로 종료한 경우이다. 이 때 문제가 있는 경우는 argv[1]에 주어지는 이름의 파일이 존재하지 않아 파일을 열 수 없는 경우이다.

파일을 열었으면 다음으로 읽어야 하기 때문에 System call read() 함수를 사용했다. 첫 번째 argument fd는 읽을 파일의 file descriptor이고, 두 번째 세 번째 argument는 각각 입, 출력할 data 저장장소와 그 장소의 변경가능한 size를 의미한다. 위 코드에서는 buf라는 저장 장소에 BUFFSIZE(=4096) 만큼의 공간을 할당했다. read() 시스템 호출의 return 값은 세 번째 인수에서 요구한 byte 수 보다 작을 수 있고 이 값이 0이면 파일의 끝까지 읽었음을 의마하고 음수이면 read 오류를 의미한다. 따라서 while문을 사용하여 파일을 읽는데 만약 return value 값인 n이 0보다 작으면 오류 메시지를 출력하고 exit(1)을 사용하여 오류로 종료하도록 했다.

n이 0보다 클 동안 파일을 읽고 이 읽은 것을 표준 출력으로 출력해야 하기 때문에 while 문 밑에 if문을 사용하여 write() 함수를 사용했다. write() 함수의 첫 번째 argument는 write할 파일의 file descriptor인데 실습 3에서는 표준 출력에 출력을 해야 한다. 표준 출력은 매크로 STDOUT\_FILENO이다. 두 번째 argument로 buf를 주었는데 이것은 이전에 파일에서 읽은 data들이 저장된 공간이다. 마지막으로 세 번째 argument로 write할 byte수 n을 주었다. write의 return

value는 성공 시 write한 byte수 즉 n이기 때문에 만약 write()함수에 오류가 발생하면 (return 값이 n이 아니면) perror()함수를 통해 에러 메시지를 출력하고 exit(1)을 사용하여 오류로 종료하도록 했다.

성공적으로 파일을 열어서 읽고 표준 출력으로 출력을 마치면 더 이상 사용하지 않는 파일을 닫아야 하기 때문에 close()함수에 file descriptor를 전달하여 파일을 닫고, exit(0)을 사용하여 성공적으로 종료하게 하였다.

**3.2**

테이블이(가) 표시된 사진

자동 생성된 설명

명령 줄 "./mycat out1"으로 수행한 결과 2.2에서 명령 줄 “cat out1”한 결과와 같음을 알 수 있다.

(tera term 창의 크기가 달라져서 2.2의 결과와 사진은 다르지만 H :512개 X:512개로 내용이 같다.)

**3.3**



현재 directory에 ‘nosuchfile’이라는 이름의 파일이 존재하지 않아 오류 메시지가 출력 되었다.

**4. 파일 복사**

**4.1**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <errno.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define BUFFSIZE 4096**

**char buf[BUFFSIZE];**

**int main(int argc, char \*argv[])**

**{**

**int r\_fd, w\_fd, n;**

**if((r\_fd = open(argv[1], O\_RDONLY)) == -1) {**

**perror("open");**

**exit(1);**

**}**

**if((w\_fd = open(argv[2], O\_CREAT | O\_WRONLY | O\_EXCL, 0644)) == -1) {**

**perror("open");**

**exit(1);**

**}**

**while((n = read(r\_fd, buf, BUFFSIZE)) >0) {**

**if(write(w\_fd, buf, n) != n) {**

**perror("write");**

**exit(1);**

**}**

**}**

**if (n<0) {**

**perror("read");**

**exit(1);**

**}**

**close(r\_fd);**

**close(w\_fd);**

**exit(0);**

**}**

헤더 파일은 1.1에서 설명한 것과 같고 추가적으로 실습 4번에서 read()를 사용하는데 이 함수는 <unistd.h>에 저장되어 있다.

실습 4번 파일 복사는 argv[1]에 주어지는 파일을 argv[2]에 주어지는 파일로 복사하는 프로그램이고, 만약 파일 argv[2]가 이미 존재하면 오류 메시지 출력 후 종료한다. 이 프로그램은 argv[1]에 주어지는 파일을 열어서 읽고 argv[2]에 주어지는 파일에 복사(write) 해야 하기 때문에 파일을 두개 열어야 한다. 따라서 file descriptor 변수도 두개 필요하기 때문에 argv[1]로 주어진 read할 file descriptor 변수를 r\_fd, argv[2]로 주어진 write할 file descriptor 변수를 w\_fd로 설정하였다.

첫 번째로 여는 파일을 System call open() 함수를 사용하여 복사할(read할) 파일을 열었는데 여는 파일 이름은 argv[1]으로 넘어오는 값이기 때문에 open()함수의 첫 번째 argument에 argv[1]을 넣어주었고, read만 하기 때문에 두 번째 argument인 flag에는 O\_RDONLY를 사용했다. read만 하기 때문에 세 번째 argument로 mode는 주지 않았다.

그리고 두 번째로 여는 파일을 System call open() 함수를 사용하여 복사될(write할) 파일을 열었는데 이 때에는 첫 번째 argument로 argv[2]를 주었고 새 파일을 생성하고 write하며 이미 존재하는 파일을 생성할 때 오류 메시지를 출력해야 하기 때문에 두 번째 argument인 flag에는 O\_CREAT | O\_WRONLY | O\_EXCL를 사용했다. (‘|’는 bitwise or이다.) open()함수의 세 번째 argument로 mode에는 0644를 주었는데 앞에서부터 두 번째 자리의 ‘6’은 user는 read, write를 할 수 있음을 의미하고 세 번째 자리의 ‘4’는 같은 group 사람이 read할 수 있음을 의미하며 마지막 자리의 ‘4’는 user도 group도 아닌 다른 사람이 read 할 수 있음을 의미한다.

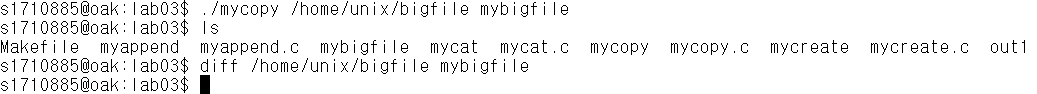
시스템 호출의 경우 return value를 꼭 체크하고 문제가 있는 경우 에러 메시지를 출력해야 하기 때문에 if문을 사용하여 만약 open()의 결과 반환되는 fd 값이 -1 일 경우(문제가 있는 경우) perror()함수를 통해 오류 메시지를 출력하고 프로그램을 종료했다. 프로그램을 종료할 때 오류로 종료할 경우 exit(1)을 사용한다. 성공적으로 종료할 경우 exit(0)를 사용하고 exit(0)이 아닌 경우는 모두 오류로 종료한 경우이다. 이 때 문제가 있는 경우는 argv[1]에 주어지는 이름의 파일이 존재하지 않아 파일을 열 수 없는 경우 이거나 argv[2]에 주어지는 이름의 파일이 이미 존재 할 경우이다.

파일을 열었으면 다음으로 읽어야 하기 때문에 System call read() 함수를 사용했다. read()함수의첫 번째 argument r\_fd는 읽을 파일의 file descriptor이고, 두 번째 세 번째 argument는 각각 입, 출력할 data 저장장소와 그 장소의 변경가능한 size를 의미한다. 위 코드에서는 buf라는 저장 장소에 BUFFSIZE(=4096) 만큼의 공간을 할당했다. read() 시스템 호출의 return 값은 세 번째 인수에서 요구한 byte 수 보다 작을 수 있고 이 값이 0이면 파일의 끝까지 읽었음을 의마하고 음수이면 read 오류를 의미한다. 따라서 while문을 사용하여 파일을 읽는데 만약 return value 값인 n이 0보다 작으면 오류 메시지를 출력하고 exit(1)을 사용하여 오류로 종료하도록 했다.

n이 0보다 클 동안 파일을 읽고 이 읽은 것을 복사될 파일로 출력해야 하기 때문에 while 문 밑에 if문을 사용하여 write() 함수를 사용했다. write() 함수의 첫 번째 argument는 write할 파일의 file descriptor인데 실습 4에서는 argv[2]로 넘어온 이름의 파일로 출력을 해야 한다. 위 코드에서 argv[2]로 넘어온 복사될 파일의 file descriptor는 w\_fd이다. 두 번째 argument로 buf를 주었는데 이것은 이전에 파일(r\_fd)에서 읽은 data들이 저장된 공간이다. 마지막으로 세 번째 argument로 write할 byte수 n을 주었다. write의 return value는 성공 시 write한 byte수 즉 n이기 때문에 만약 write()함수에 오류가 발생하면 (return 값이 n이 아니면) perror()함수를 통해 에러 메시지를 출력하고 exit(1)을 사용하여 오류로 종료하도록 했다.

성공적으로 파일을 열어서 읽고 표준 출력으로 출력을 마치면 더 이상 사용하지 않는 파일들을 닫아야 하기 때문에 close()함수에 file descriptor(r\_fd, w\_fd)를 전달하여 파일들을 닫고, exit(0)을 사용하여 성공적으로 종료하게 하였다.

**4.2**



명령 줄 "./mycopy /home/unix/bigfile mybigfile"을 수행한 후, 복사가 바르게 잘 되었는지 확인하는 명령 줄 "diff /home/unix/bigfile mybigfile"을 수행한 결과 아무것도 나오지 않았는데 "diff /home/unix/bigfile mybigfile"는 두 파일 /home/unix/bigfile과 mybigfile의 내용을 비교하여 차이가 있으면 어떤 차이가 있는지 출력하고 만약 차이가 없다면 아무것도 출력되지 않는 명령어이다. Copy가 잘 돼 차이가 없기 때문에 아무것도 출력되지 않은 것이 맞다. (파일 복사가 잘 되었다.)

**4.3**



명령 줄 "./mycopy /home/unix/bigfile mybigfile"을 한 번 더 수행하면 이미 현재 directory에 ‘mybigfile’ 이라는 이름의 파일이 존재하기 때문에 오류 메시지가 출력된다.

**5. 입출력 효율성**

**5.1**

|  |  |  |  |
| --- | --- | --- | --- |
| 버퍼 크기 | real | user | sys |
| 128 | 0m28.982s | 0m7.968s | 0m21.012s |
| 1024 | 0m4.472s | 0m0.968s | 0m3.500s |
| 4096 | 0m1.733s | 0m0.308s | 0m1.424s |
| 65536 | 0m1.206s | 0m0.012s | 0m1.196s |
| 524388 | 0m1.161s | 0m0.004s | 0m1.156s |

**5.2**

real: wall clock time으로 명령 수행의 시작부터 끝까지 걸린 총 시간이다. (사람이 느끼는 시간)

user: user CPU time으로 프로그램이 cpu를 사용한 시간을 의미한다.

sys: system CPU time으로 kernel이 cpu를 사용한 시간을 의미한다.

**5.3**

5.1의 결과 표를 보면 shell의 time 기능을 사용한 명령어의 결과로 나온 BUFFSIZE별 3개의 시간인 real, user 및 sys값이 달라짐을 알 수 있는데, 버퍼 크기가 증가함에 따라 time 값들이 감소하는 것을 볼 수 있다. 버퍼 크기가 4096이전일 때는 단축되는 시간의 양이 크지만 버퍼 크기가 4096을 넘어서고는 단축되는 시간 폭이 작기 때문에 입, 출력 버퍼의 크기를 4096으로 설정하는 것이 합리적 일 것이다. 수업 시간에도 버퍼 크기가 disk의 block size인 4096일때가 좋은 효율을 나타낸다고 했고 사용하는 메모리는 작을수록 좋기 때문에 버퍼 크기는 4096으로 설정하는 것이 좋을 것 같다.

**5.4**

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 “ls” 명령어를 통해 현재 directory에 mybigfile이 존재하고 있음을 확인했고, “rm mybigfile” 명령어를 통해 mybigfile을 제거하였다. 그 다음 다시 “ls” 명령어를 통해 확인 한 결과 현재 directory안에 mybigfile이 사라진 것을 확인하였다.

끝.