LAB08
시스템 프로그래밍_류은경**Prof**

2021115744_권구태

1. (waitdemo1.c 수정)
- 명령행 인자로 자식 프로세스의 수 n을 입력받음
- 부모는 n개의 자식 프로세스를 생성
- 각 자식은 1~5초 사이의 무작위 시간 동안 sleep한 뒤, 종료 코드로 그 시간을 반환
- 부모는 모든 자식이 종료할 때까지 wait, 자식이 exit될 때마다 자식의 PID와 status 코드를 출력

```
$ ./ex1 2
자식 프로세스 1001: 3초 동안 sleep
자식 프로세스 1002: 1초 동안 sleep
부모: 자식 1002 종료, 종료코드 1
부모: 자식 1001 종료, 종료코드 3
모든 자식 종료 완료

$ ./ex1 3
자식 프로세스 12345: 4초 동안 sleep
자식 프로세스 12346: 2초 동안 sleep
자식 프로세스 12347: 3초 동안 sleep
부모: 자식 12346 종료, 종료코드 2
부모: 자식 12347 종료, 종료코드 3
부모: 자식 12345 종료, 종료코드 4
모든 자식 종료 완료

$
```

[코드 리뷰**]**

```c
int forkNum = atoi(arvg[1]);
int forkNumCp1 = forkNum;
int forkNumCp2 = forkNum;
int randNUM[MAX] = {0};
while (forkNumCp1--)
{
    randNUM[forkNumCp1] = (rand() % 5) + 1;
}

while (forkNum--)
{
    if ( (newpid = fork()) == -1 )
        perror("fork");
    else if ( newpid == 0 ) {
```

```
            child_code(randNUM[forkNum]);
        }
    }
    while (forkNumCp2--)
    {
        if ( newpid != 0 )
            parent_code(newpid);
    }
```

터미널 명령어 단계에서 두번째 인자의 fork 갯수를 입력 받은 문자열을 atoi 함수를 이용하여 정수로 변환. 자식프로세스, 부모프로세스, 랜덤값 생성을 위한(자식 프로세스 fork 하면서 생성하면 rand값이 생성되지 않고, 같은 값으로 모든 자식 프로세스에게 적용됨. 프로세스 복사로 인하여.) forkNumCp를 추가 선언.
이후, forkNum 값에 맞게 랜덤값 생성, 프로세스 복제함. 그리고 자식 프로세스는 모두 출력, 이후 자식 프로세스의 출력이 모두 끝나면 (with sleep of child) 부모 프로세스의 wait가 풀리면서(kernel 단에서 풀어줌. 각각의 자식 프로세스에 맞게) 부모 프로세스 출력.

```
    wait_rv = wait(&child_status);
```

이때 child_status의 주소값을 넘겨 child_status에 상위 8bit은 자식 프로세스의 exit value가 들어있기 때문에

```
high_8 = child_status >> 8;
```

비트 이동을 하여 출력함. 이때 당연히 자식 프로세스에서 자식 프로세스 종료 시 exit(sleep값)을 넣어줘야 원하는 출력값을 얻을 수 있음.

이후 모든 while문이 끝나고 나면 모든 자식 종료 완료를 print함.

[실행결과]

```
kwongutae@kwongutae-Inspiron-15-5510:~/Documents/경북대학교/강의/시스템프로그래밍_류은경/src/HW/10주차$ ./waitdemo 4
자식프로세스 49100: 3초 동안 sleep
자식프로세스 49099: 4초 동안 sleep
자식프로세스 49101: 2초 동안 sleep
자식프로세스 49102: 4초 동안 sleep
부모: 자식 49102 종료, 종료코드: 2
부모: 자식 49102 종료, 종료코드: 3
부모: 자식 49102 종료, 종료코드: 4
부모: 자식 49102 종료, 종료코드: 4
모든 자식 종료 완료
kwongutae@kwongutae-Inspiron-15-5510:~/Documents/경북대학교/강의/시스템프로그래밍_류은경/src/HW/10주차$ ./waitdemo 6
자식프로세스 49235: 4초 동안 sleep
자식프로세스 49236: 3초 동안 sleep
자식프로세스 49237: 2초 동안 sleep
자식프로세스 49238: 4초 동안 sleep
자식프로세스 49239: 2초 동안 sleep
자식프로세스 49240: 4초 동안 sleep
부모: 자식 49240 종료, 종료코드: 2
부모: 자식 49240 종료, 종료코드: 2
부모: 자식 49240 종료, 종료코드: 3
부모: 자식 49240 종료, 종료코드: 4
부모: 자식 49240 종료, 종료코드: 4
부모: 자식 49240 종료, 종료코드: 4
모든 자식 종료 완료
kwongutae@kwongutae-Inspiron-15-5510:~/Documents/경북대학교/강의/시스템프로그래밍_류은경/src/HW/10주차$
```

- 8.4, 8.5 각 실행 결과 분석 요약

8.4 `fork` *and file descriptors* Consider this code:

```
main()
{
        int   fd;
        int   pid;
        char  msg1[] = "Testing 1 2 3 ..\n";
        char  msg2[] = "Hello, hello\n";

        if ( (fd = creat("testfile", 0644)) == -1 )
            return 0;
        if ( write(fd, msg1, strlen(msg1)) == -1 )
            return 0;

        if ( (pid = fork())  == -1 )
            return 0;
        if ( write(fd, msg2, strlen(msg2)) == -1 )
            return 0;
        close(fd);
        return 1;
}
```

Test this program. After the call to `fork`, both processes have a file descriptor set to the same current position in the output file. How many messages appear in the file? What does the number of lines tell you about file descriptors and connections to files?

[코드 분석]
testfile이라는 파일을 읽기쓰기 전용으로 생성하고, msg1을 해당 파일에 작성한다. 그리고 프로세스를 복제한다. 프로세스는 복제되면서 파일 디스크립터는 복제되지만, 프로세스 내의 파일 entry는 공유한다. 따라서 이후 또 write를 하게되면 두 프로세스에서 모두 write가 일어나지만 파일 entry를 공유하기 때문에 msg2가 프로세스 실행 순서에 따라 덮어 써지거나 엉망이 될 수 있다.

8.5 *fork and standard I/O* Consider this code:

```
#include <stdio.h>

main()
{
        FILE *fp;
        int   pid;
        char  msg1[] = "Testing 1 2 3 ..\n";
        char  msg2[] = "Hello, hello\n";
        if ( (fp = fopen("testfile2", "w")) == NULL )
                return 0;
        fprintf(fp, "%s", msg1);

        if ( (pid = fork())  == -1 )
                return 0;
        fprintf(fp, "%s", msg2);
        fclose(fp);
        return 1;
}
```

Test this program. How many messages appear in the file? Explain the results. Compare this program to the output of `forkdemo1.c` in the text.

[코드 리뷰**]**

부모와 자식 프로세스가 같은 FILE 구조체를 공유한다.
따라서 같은 파일을 가리키게 되고(fp)가 따라서 msg2가 프로세스의 실행 순서에 따라 덮어써지는 현상이 발생한다.
또 중요한 점은 fopen의 쓰기 연산은 버퍼에 저장되고 일정 시점에만 실제로 파일에 쓴다. 그러나 fork 이후 부모 혹은 자식 프로세스 중 하나의 프로세스가 버퍼에 쓰기를 했는데, 해당 데이터가 파일로 이동하기 전에 다른 프로세스의 쓰기가 일어나면 데이터 손실이 일어날 수 있어서. 해당 코드처럼 짜면 안된다.