상명대학교 컴퓨터과학부 "EA0014: 유닉스프로그래밍" 실습 지침서

- 실습 번호 : lab-06

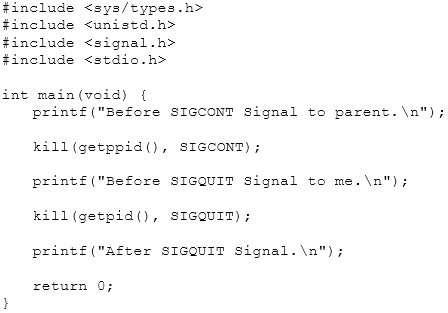
- 실습 디렉토리 : ~/unix/lab-06 이름 : 임현

- 실습 날짜 : 2017년 5월 16일 분반 : 1분반

- 실습 제목 : Signal and mmap

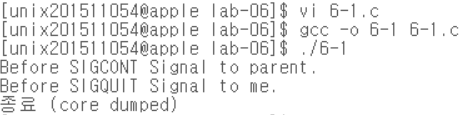
- 실습 내용 : 학번 : 201511054

1. (6-1.c, 6-1) 다음과 같이 파일 6-1.c를 편집하고 6-1로 컴파일하여 수행하시오.



수행 결과를 적고, 결과를 설명하시오.

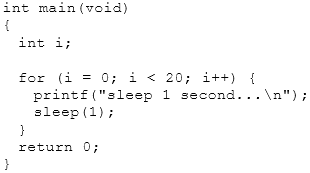
수행 결과는 아래와 같고,



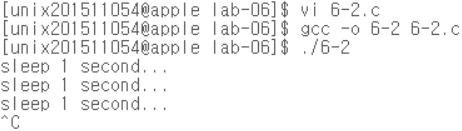
결과를 설명하자면, ‘kill(getpid(), SIGQUIT);’로 인해 코어 덤프가 발생됩니다.

그 이유는 부모 프로세스에 보내는 시그널인 SIGCONT는 기본 처리가 무시이므로 특별한 처리를 하지 않지만, 자신에게 보내는 시그널인 SIGQUIT는 코어 덤프를 발생시키고, 프로세스를 종료 시키므로, printf(“After SIGQUIT Signal.\n”,); 출력이 되지 않고, 프로세스가 종료됩니다.

2. (6-2.c, 6-2) 아래 프로그램을 파일 6-2.c에 입력하고 컴파일하여 6-2를 만드시오.



컴파일된 프로그램을 수행시킨 후 프로그램이 종료하기 전에 키 보드에서 '^C'를 입력해보시오. 이때 어떤 현상이 일어나는 지를 관찰하여 기록하고 그 이유를 설명하시오.

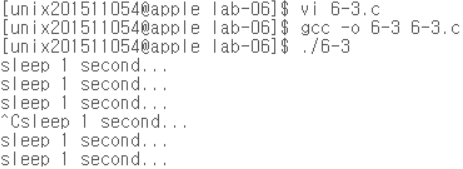


유닉스 신호에서 ‘Ctrl + C’(인터럽트)는 INT 신호 (SIGINT)를 내보냅니다. 기본적으로 프로세스를 종료하는 역할을 합니다.

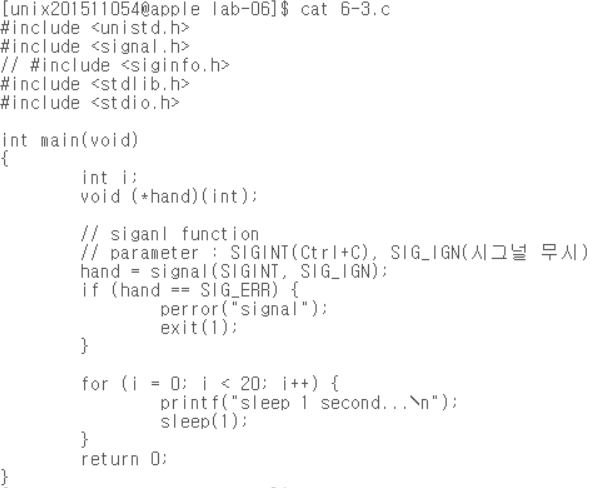
그 때문에 20초 동안 sleep이 지속되어야 하는 프로세스인 6-2에서 인터럽트 신호를 받고 바로 프로세스를 종료시켰습니다.

3. (6-3.c, 6-3) "6-2.c"를 수정하여 이 프로그램이 수행되는 도중 키보드에서 '^C'를 입력하더라도 이를 무시하고 계속 수행되게 하는 프로그램 “6-3.c”를 만드시오. 프로그램을 컴파일 후 수행시켜 원하는 동작이 일어나는 지를 살펴보고 그 결과와 작동 원리를 적으시오.

‘^C’를 입력하더라도 이를 무시하고 계속 수행되게 하는 프로그램은 아래와 같습니다.

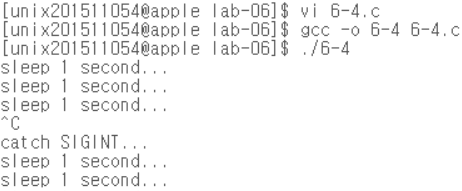


작동 원리는 시그널 핸들러 함수(void (\*signal(int sig, void (\*disp)(int)))(int);))를 사용하여 disp(시그널 핸들러의 함수명) 자리에 ‘SIG\_IGN’을 적어줌으로써 시그널을 무시하도록 지정해주었습니다.

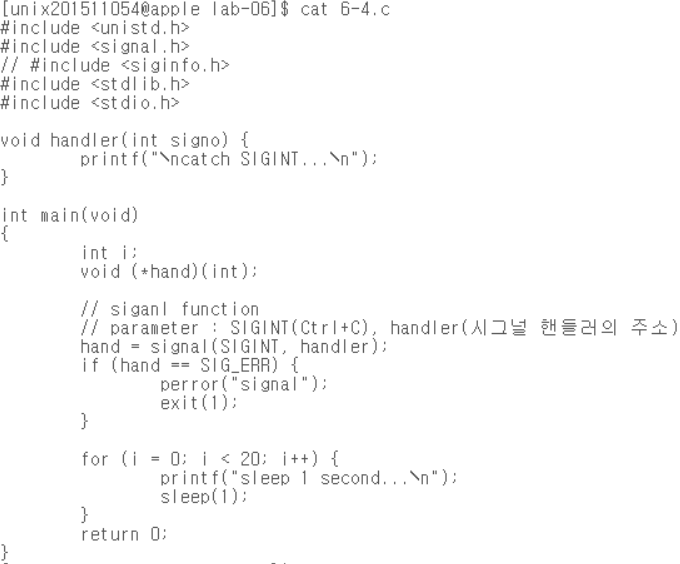


4. (6-4.c, 6-4) "6-3.c"를 수정하여 이 프로그램이 수행되는 도중 키 보드에서 '^C'를 입력하면 표준 출력에 "catch SIGINT..."을 출력하고 프로그램을 계속 수행시키는 프로그램 “6-4.c”를 작성하시오. 프로그램을 컴파일 후 수행시켜 원하는 동작이 일어나는 지를 살펴보고 그 결과와 작동 원리를 적으시오.

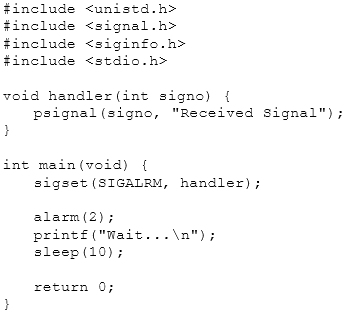
‘^C’를 입력하면 “catch SIGINT”를 출력하고 계속 수행시키는 프로그램은 아래와 같습니다.



작동 원리는 시그널 핸들러 함수(void (\*signal(int sig, void (\*disp)(int)))(int);))를 사용하여 disp(시그널 핸들러의 함수명) 자리에 ‘handler’(시그널 핸들러의 주소)를 적어줌으로써 기능을 수행하였습니다..

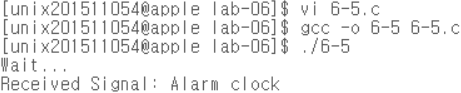


5. (6-5.c, 6-5) 다음과 같은 프로그램 “6-5.c”를 만들고 컴파일하여 실행하시오.



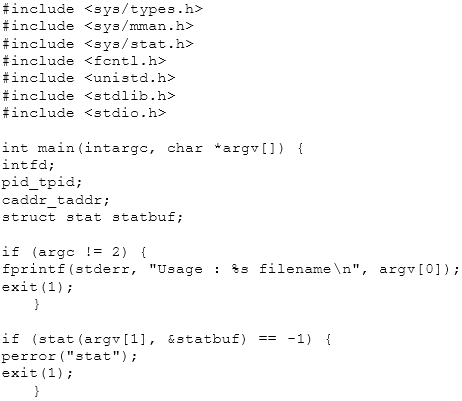
실행 결과를 적고 왜 그런 결과가 나오는지를 설명하시오.

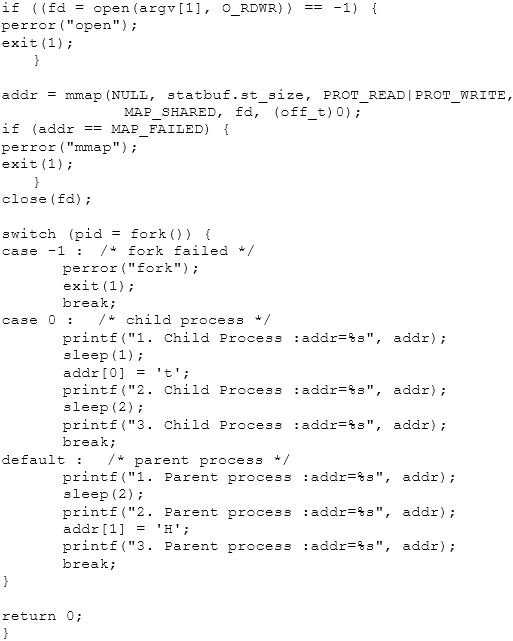
실행 결과는 아래와 같고,



아래와 같이 나오는 이유는 10초 동안 sleep하고 있는 중에 2초가 지나면 SIGALRM 시그널이 발생해 시그널 핸들러가 호출됩니다. 그래서 실행 결과를 보면 프로그램이 종료되기 전에(즉, 10초가 지나기 전에) 시그널 핸들러를 호출(2초)한 사실을 알 수 있습니다.

6. (6-6.c, 6-6) 다음과 같이 파일 6-6.c를 편집하고 6-6으로 컴파일하여 수행하시오.





$ cat test.txt

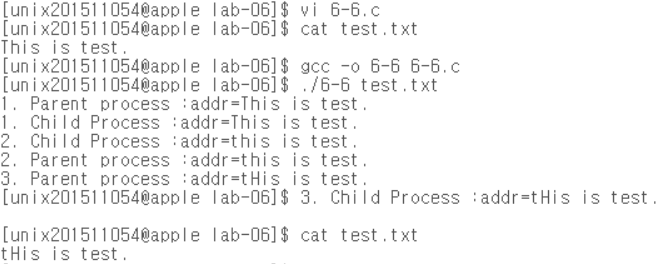
This is test.

와 같은 test.txt파일로

$ ./6-6 test.txt

하여 실행 결과를 적고, 설명하시오.

실행 결과는 아래와 같고,



설명하자면 우선 stat함수로 파일의 속성 정보를 검색해 파일의 크기를 알아내고,

open 함수로 메모리에 매핑할 데이터 파일을 연 후,

mmap함수로 파일을 메모리에 매핑합니다.

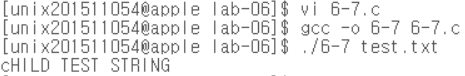
그리고 fork 함수를 실행해 자식 프로세스를 생성하고, 자식 프로세스는 부모 프로세스를 상속하므로 매핑된 메모리의 시작 주소인 addr을 공유합니다.

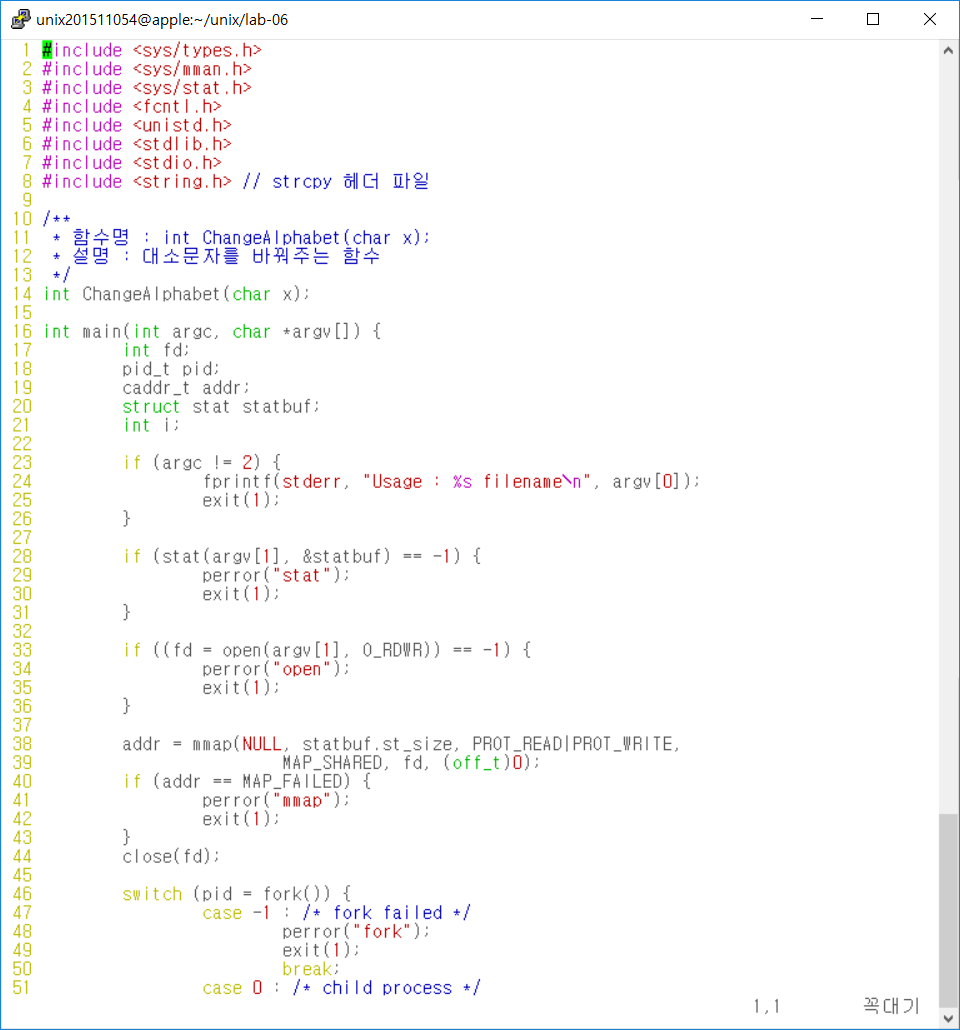
그 후 자식 프로세스가 addr[0]의 내용을 ‘t’로 변경하고 출력 내용을 보면 자식 프로세스가 변경한 값을 부모 프로세스로 읽을 수 있음을 확인할 수 있습니다.

반대로 부모 프로세스는 addr[1]의 내용을 ‘H’로 변경합니다.

그래서 변경 내용을 부모 프로세스와 자식 프로세스가 각각 읽고 있음을 알 수 있습니다.

7. (6-7.c, 6-7) 위의 6-6.c를 바탕으로 “test.txt”라는 파일을 위한 메모리 맵 영역을 할당하고 child 프로세스에서 해당 영역에 “Child test string”을 써 넣고 parent 프로세스는 해당 영역을 읽어 대문자는 소문자로 소문자는 대문자로 바꾸어 stdout으로 출력하는 프로그램 6-7.c를 작성하시오. 이 프로그램을 컴파일하여 수행하고 결과를 보이시오.







끝.