**LAB4 20113315 이형준**

**1.**[**강의노트**](http://advsys.cs.kookmin.ac.kr/lab?pli=1) **p24,25의 wait1.c 프로그램을 실행시키고 출력되는 각 라인에 대해서 설명하시오.**

**<Wait1.c>**

#include"prexit.c"

#include<sys/types.h>

#include<sys/wait.h>

#include"ourhdr.h"

void err\_sys(const char\* x)

{

perror(x);

exit(1);

}

int main(void){

pid\_t pid;

int status;

//fork()함수를 실행하여 그대로 복사를 한다.

//child일 경우 0을 return한다.

//parent일 경우 child의 pid를 반환한다.

if( (pid=fork()) < 0)

err\_sys("fork error");

//내가 child일 경우에 실행되는 문장이다.

else if (pid == 0){

//child의 process상태를 확인하기 위해 sleep을 넣어준다

sleep(2);

// 정상종료하고 7을 반환한다.

exit(7);

}

//아래 코드부터는 child가 종료되었으므로, parent만 해당된다.

//wait함수를 실행하고,status값이 저장되는데,이 wait의리턴값과 child 의pid가 다르면 에러.

if (wait(&status) != pid) /\* wait for child \*/

err\_sys("wait error");

//parent의 process상태를 확인하기 위해 sleep을 넣어준다.

sleep(2);

//현재 status를 출력합니다.(정상종료, child의 리턴값 7을 출력한다.)

pr\_exit(status); /\* and print its status \*/

//fork()합니다

if ( (pid = fork()) < 0)

err\_sys("fork error");

//child일 때,신호를 생성해서 비정상적인 종료를 만들어냅니다.

else if (pid == 0){ /\* child \*/

sleep(2);

abort(); /\* generates SIGABRT \*/

}

if (wait(&status) != pid) /\* wait for child \*/

err\_sys("wait error");

//parent의 process상태를 확인하기 위해 sleep을 넣어준다.

sleep(2);

//현재 status출력. (비정상종료, return 6)

pr\_exit(status); /\* and print its status \*/

if ( (pid = fork()) < 0)

err\_sys("fork error");

else if (pid == 0){ /\* child \*/

sleep(2);

//나누기 0 으로 인한 시그널. 비정상종료.

status /= 0; /\* divide by 0 generates SIGFPE \*/

}

if (wait(&status) != pid) /\* wait for child \*/

err\_sys("wait error");

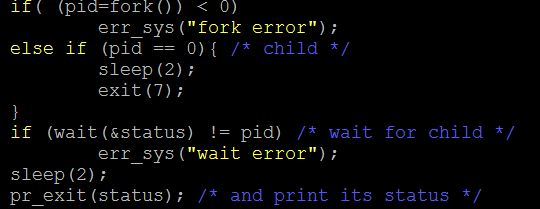
sleep(2);

pr\_exit(status); /\* and print its status \*/

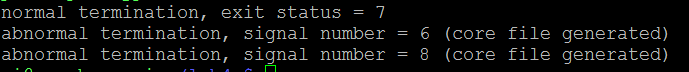
//전체종료

exit(0);

}



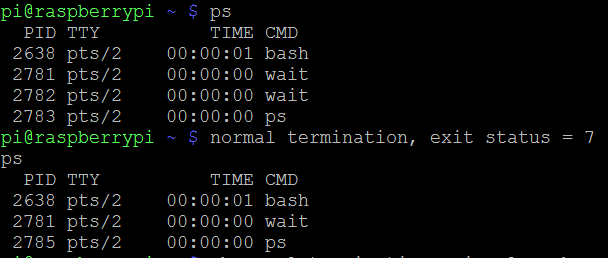
* 컴파일 결과



* Ps로 상태확인

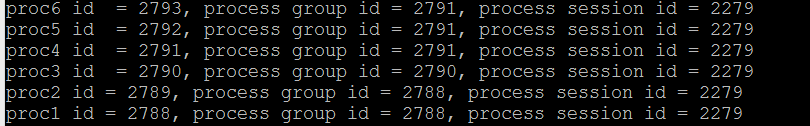
현재 parent와 child의 프로세스를 실시간으로 확인 가능합니다.





**2. proc1, proc2, proc3, ... loop을 통해 sleep()을 여러번 실행하여 충분히 길게 실행하는 프로그램으로 작성하고 이를 이용해서 강의노트 p39, p42의 상황을 만드시오.**

**1) process group, session 등을 관찰하시오.**



$ ./proc1 | ./proc 2 &

$ ./proc3 &

$ ./proc4 | ./proc5 | ./proc6

파이프를 통해 같이 실행시킨 프로세스들은 같은 그룹으로 묶이게 됩니다..

셸이 세션의 리더가 되고 셸을 통해서 실행시킨 1,2,3,4,5,6 은 같은 세션id를 갖게 됩니다..

**2) process group을 변경하고 관찰해 보시오.**

우선 fornt에서 실행중이던 proc4, proc5, proc6은 ^c 를 이용하여 종료가 가능합니다.

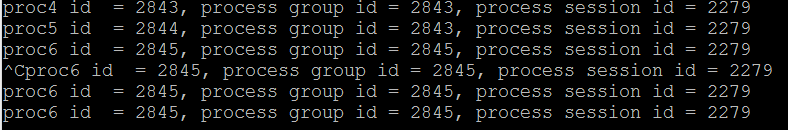


그다음, Setpgid(0,0) 을 사용해서, proc6의 pid를 process group id로 변경시켰습니다..

그리고 나서 ^c를 입력하였더니, proc6을 제외한 proc4,proc5가 종료되었습니다.

별도의 그룹으로 독립하도록 수전한 proc6은 , ^c가 적용되지 않았습니다.

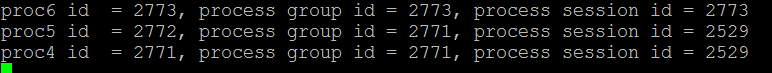
**(Ctrl+C** : 해당 프로세스에 SIGINT가 전달이 되어서 프로세스가 종료가 된다)



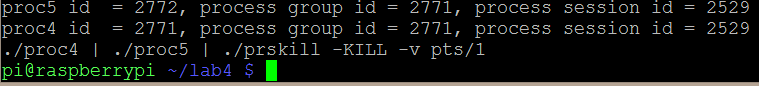
**3) process session을 변경하고 관찰해 보시오.**



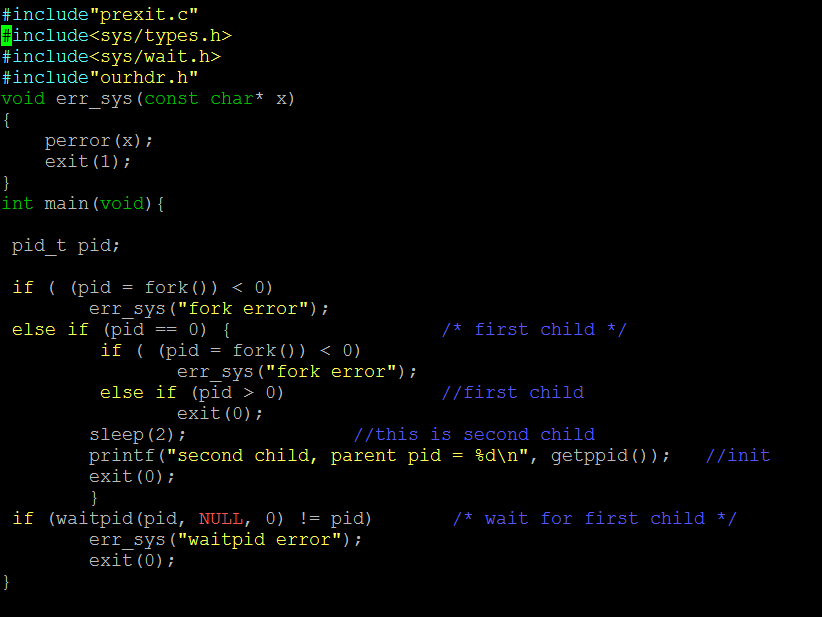
프로세스의 그룹의 리더가 아닌 proc5의 세션을 변경하였습니다.



변경된 결과는 다음과 같습니다. proc6의 세션 id가 변경된 것을 볼 수 있습니다.



**3 .** [**강의노트**](http://advsys.cs.kookmin.ac.kr/lab?pli=1)**p28,29의 fork2.c 프로그램을 실행시키고 관찰하시오**



parent에서 first child를 fork하였습니다.

First child는 second child를 fork합니다. 그리고 정상종료 합니다.

Second child의 parent인 first child 가 종료되었으므로, init의 child가 됩니다.

Second child의 getpid는 init 인 1이 출력됩니다.

그리고 parent는 waitpid로 first child의 종료를 기다리게 됩니다.

