lec

process: 메모리에 탑재된 프로그램 = 프로세스 descriptor

실제로는 하나씩 차례대로 실행되고 있는 것 입니다.

* fork: duplicate a process
* exec: transform a process into another //변경
* exit: stop the process
* wait: wait for the exit of the child process
* getpid: get process ID
* getppid: get parent process ID

fork – 생명 복제와 비슷

1) copy body

2) copy process descriptor

3) adjust child's process (복제된 proc) descriptor

4) return 0 to child

5) return child's pid to parent (원 proc)

**\* u = undo**

**\* ctrl +r(R)? = redo**

**\* gg=G 줄정렬**

fork를 루프 안에 사용할 때는 언제나 주의해야 합니다. 특히 무한 루프안에 fork를 넣는 일은 이 클래스에서는 피하기 바랍니다. fork는 기하급수적으로 프로세스를 생성하므로 적절히 제어하지 못하면 프로세스 과다로 시스템을 다운시킬 수 있습니다. 따라서 fork를 루프안에 넣을 때는 유한 루프를 사용하기 바랍니다 ( 실제 상용 프로그램에서는 무한루프 안에 fork가 들어갑니다. 이 클래스에서만 유한루프를 사용하라는 의미 )

hw

0) Try ex0 below. Who is the parent of ex0?

ex0.c:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void main(){

int x,y;

x=getpid();

y=getppid();

printf("PID:%d PPID:%d\n", x, y);

for(;;); // to make this process alive

}

$ gcc –o ex0 ex0.c

Run ex0 with &. "&" puts process in the background so that you can issue next command to the shell.

$ ex0&

.........

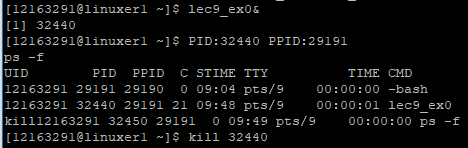
Now confirm with "ps –f".

$ ps –f

.........

Now kill ex0.

$ kill xxxx(the pid of ex0)



lec9\_ex0의 PID는 32440, PPID는 29191 입니다.

kill PID를 해줍니다.



이미 kill된 상태에서는 kill하면 위와같이 결과가 출력됩니다. – 사라졌습니다.

1) Try ex1 below. Why do we have two hello's? What are the PID of ex1 and ex1's child? Who is the parent of ex1?

ex1.c:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void main(){

int x;

x=fork();

printf(“hello\n”);

for(;;);

}

$ gcc –o ex1 ex1.c

$ ex1&

hello

hello

$ ps -f

............

$ kill xxxx(pid of ex1) yyyy(pid of ex1's child)

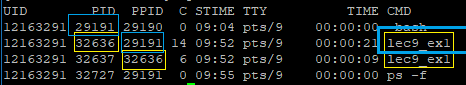
+ \*fork – 생명복제 같은 mechnism

\*schedule에 따라, 프로그램이 실행되기 때문에, ex1이 두개로 복제된 상태인 것입니다.

따라서, 똑같은 프로그램이 2번 실행되게 됩니다.

이 떄, 복제된 자식 process는 rtn값이 0입니다.

그러므로, if( …=0) 문을 사용하여 두 process간의 schedule을 조절할 수 있습니다.



두개의 process가 있습니다.

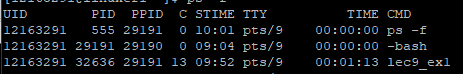
하나는 parent pid가 -bash를 가리키고 있고, (파랑 박스 부분)

다른 하나는 parent pid가 같은 프로세스를 가리키고 있습니다 (노랑 박스 부분)

따라서, child \_process는 두개의 lec9\_ex1중 아래에 있는 process입니다.



child \_proc PID를 Kill해주었습니다.



 !

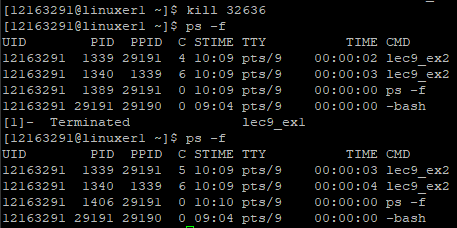
출력 결과에 "defunct"가 표시되는데, 이는 프로세스가 작업을 완료했거나 손상되었거나 종료되었지만 하위 프로세스가 여전히 실행 중이거나 이러한 상위 프로세스가 하위 프로세스를 모니터링하고 있음을 의미합니다.

즉, lec9\_ex1 을 실행하면



여전히 hello는 두번 출력되게 됩니다.

+



kill parent의 PID – 깔끔히 종료됨

2) Modify ex1.c such that it prints its own pid and the parent pid. Confirm the result with "ps -ef". Who is the parent of the parent of ex1? Who is the parent of the parent of the parent of ex1? Follow the parent link until you reach PID 0 and show all of them.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void main(){

int x;

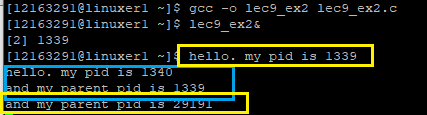
x=fork();

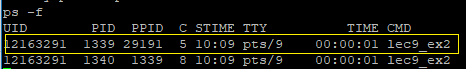
printf("hello. my pid is %d\n", getpid());

printf("and my parent pid is %d\n", getppid());

for(;;);

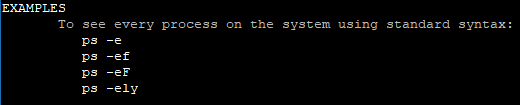
}





parent PID는 1339입니다.

즉, 노란부분이 부모 부분이고, 파란 부분이 자식 부분입니다.



man ps를 통해 모든 프로세스 id를 확인할 수 있는 옵션을 확인해 보았습니다.

ps -ef를 통해 모든 프로세스의 PID를 확인해 보았습니다.

1339의 PPID인 29191을 찾아야합니다.











위에서 아래 순서대로, PPID 정보를 이용하여 상위 process로 나아가는 과정입니다.

3) Try below (ex2.c). Which hello is displayed by the parent and which hello is by the child?

void main(){

int x;

x=fork();

printf(“hello: %d\n”, x);

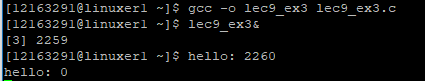
}

$ gcc –o ex2 ex2.c

$ ex2

hello: 22644

hello: 0



x 의 rtn값이 0인 hello가 child입니다.

즉, 아래 hello가 child 입니다.

위의 결과로 알 수 있는 사실은, 별다른 if문과 같은 조건이 없다면,

fork() 사용시에는

parent가 먼저, 다음에 child가 실행되는 것을 알 수 있습니다.

4) Try below (ex3.c) and show all ancestor processes of ex3 (parent, parent of parent, etc).

void main(){

int x;

x=fork();

printf(“hello: %d\n”, x);

for(;;); // make the parent and child alive

}

$ gcc –o ex3 ex3.c

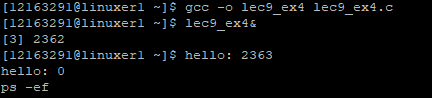
$ ex3 &

hello: 22644

hello: 0

$ ps -ef

……..





부모의 PID는 2362이고, 29191 쪽으로 나아가야 합니다.







위에서 아래 순서대로, PPID 정보를 이용하여 상위 process로 나아가는 과정을 나타낸 캡쳐본 입니다.

5) Try below (ex4.c). Which message was displayed by the parent and which one by the child?

void main(){

int x;

x=fork();

if (x==0){

printf(“hello: %d\n”, x);

}else{

printf(“hi: %d \n”, x);

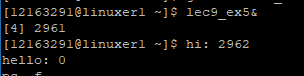
}

}

$ gcc –o ex4 ex4.c

$ ex4

………



hi가 먼저 출력되고, hello가 나중에 출력됩니다.

hi가 부모이고 hello가 자식입니다. ( hello의 rtn 값이 0 이므로 child proc 입니다.)

6) Try below (ex5.c). How many hellos do you see? Explain why you have that many hellos. Draw the process tree.

void main(){

int x,y;

x=fork();

y=fork();

printf(“hello: %d %d\n”, x, y);

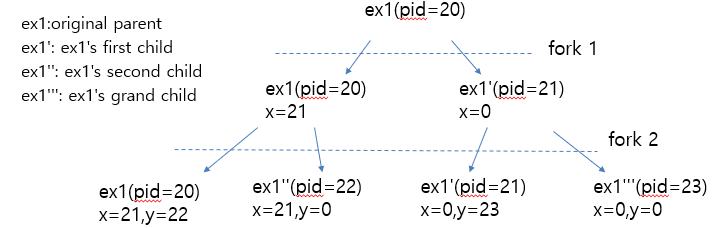
}

$ gcc –o ex5 ex5.c

$ ex5

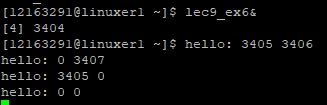
……………

위의 코드를 실행하게 되면,

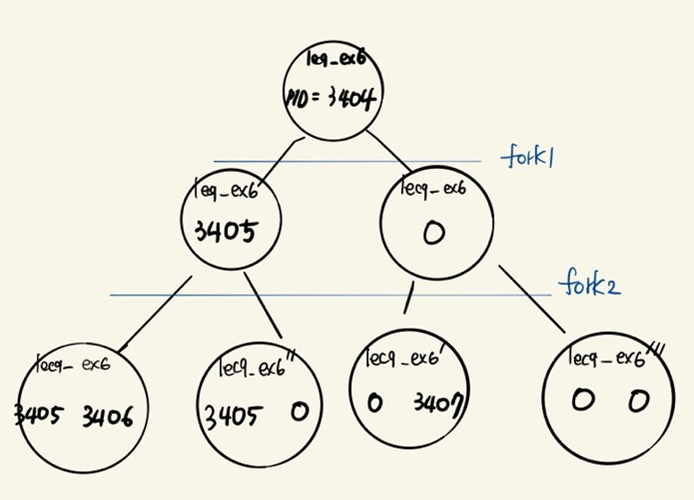


위의 그림과 같이 프로세스가 4개로 복제됩니다.

즉, fork()를 사용하게 되면, 2^( fork()의 개수) 만큼 프로세스의 개수가 증가 하게 되는 것을 알 수 있습니다.



이를 위의 트리 구조와 같이 나타내면 다음과 같습니다.



7) Try below (ex6.c). How many hellos do you see? Explain why you have that many hellos.

void main(){

int x,y;

x=fork();

printf(“hello: %d\n”, x);

y=fork();

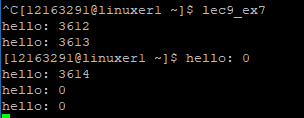
printf(“hello: %d\n”, y);

}

$ gcc –o ex6 ex6.c

$ ex6

……….

 &없 버전

위에서부터 차례대로, (3612 …)

첫 fork함수에서 만들어지는 child와

두번째 fork함수에서 만들어지는 CHILD와

parent의 x값 hello 입니다. = 0

두번째 fork함수에서 만들어지는 chlid’ 와

parent의 y값 hello 입니다. = 0

child의 y값 hello 입니다. = 0

위 의 결과들이 나오는 것은 확실하지만, schedule은 랜덤입니다.

8) Try below (ex7.c). When you run ex7, how many processes run at the same time? Which process finishes first and which process finishes last? Show the finishing order of the processes. Run ex7 again and compare the finishing order with that of the first run.

#include <stdio.h>

#include <stdlib.h>

void main(){

int x, i;

for(i=0;i<5;i++){

x=fork();

if (x==0){ // child

int k;

for(k=0;k<10;k++){

printf("%d-th child running %d-th iteration “, i, k);

fflush(stdout); // to make printf work immediately

sleep(1); // sleep 1 second

}

exit(0); // child exits after 10 iterations

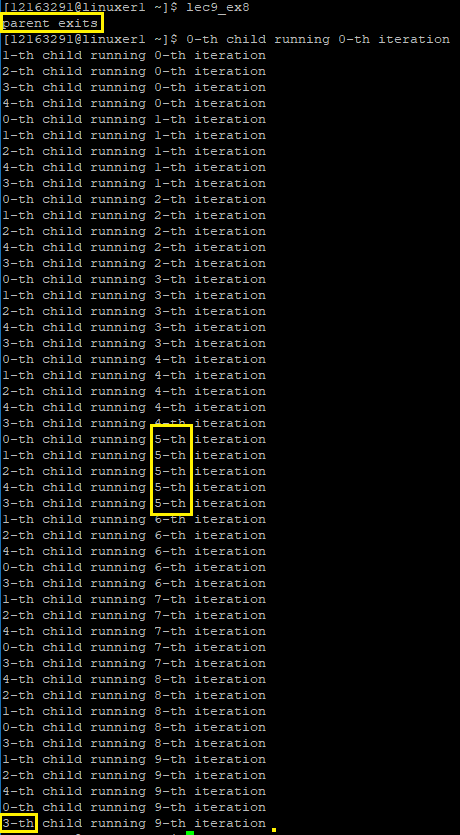
}

}

// now parent

printf(“parent exits\n”);

}

 &없

한번에 5번의 child 이터레이션을 합니다.

즉 동시에 실행되는 proc개수는 6개입니다. ( 1+5 )

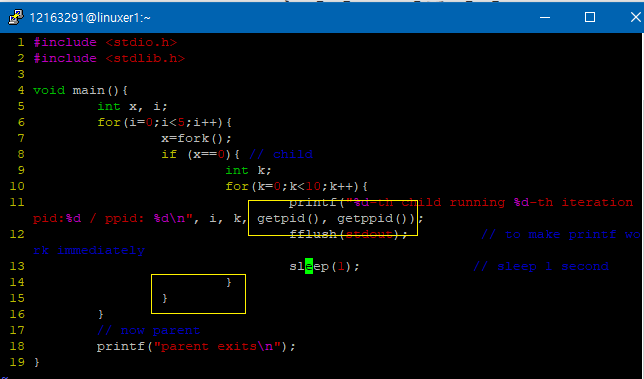
가장 마지막에 끝나는 proc는 3번째 child입니다.

앞서 설명하였듯 process 실행 schedule은 random이므로 lec9\_ex8을 실행시킬 때마다 process가 실행되는 순서는 항상 다른 것을 확인할 수 있습니다.

9) If you delete "exit(0)" in ex7.c, how many processes will be created?

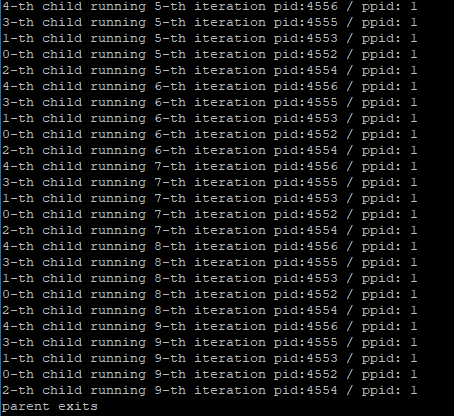
Confirm your answer by modifying the code such that each process displays its own pid.

: exit(0)을 삭제할 경우, -> 9-th iteration을 계속 반복합니다.

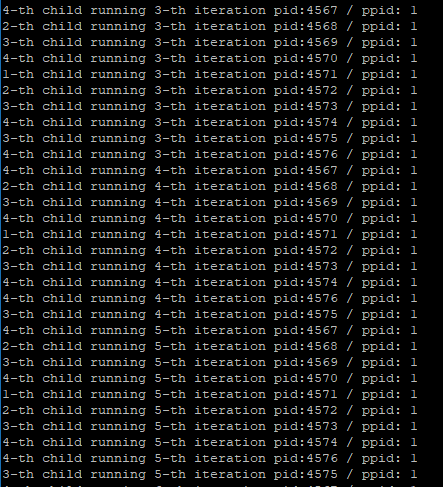


자신의 pid와, ppid를 출력하게 하였고, exit(0)을 없애주었습니다

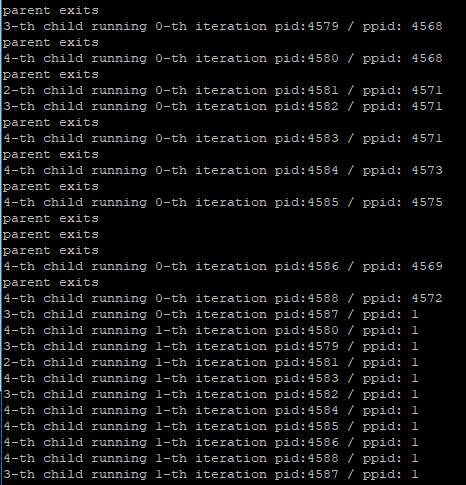
(ppid는 1로 일관된 결과가 나오는 것을 확인할 수 있었습니다.)



iteration 당 5번

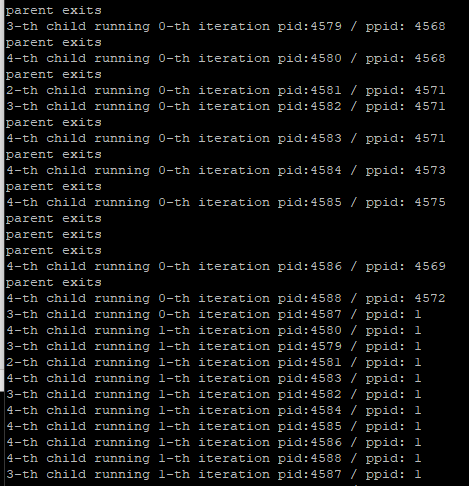


iteration 당 10번



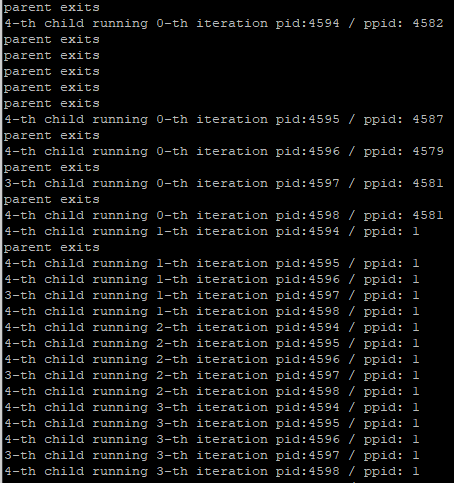
또 iteration당 10번

….

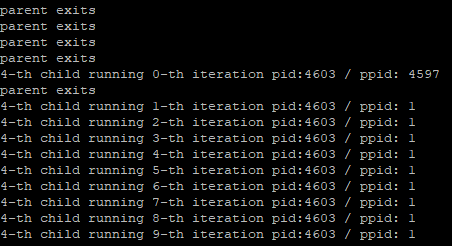


또 10번씩 반복

…



이번엔 5번 반복



마지막 부분

출력 결과 앞부분은 8)의 결과와 비슷하지만 출력되는 시간이 더 길어졌습니다.

exit 함수가 없어 child를 종료시키지 않았기에 child의 수가 늘어났습니다

도중에 child의 PPID가 1로 바뀐 이유는, parent가 종료되었기 때문에 가장 상위의 PPID를 출력하였기 때문이라 생각됩니다.

child의 이름이 같고 iteration 수도 같지만 PID가 다른 경우가 발생하는데, 이는 exit()를 삭제하여 child를 종료시키지 않았기 때문에 발생합니다.