lect9 HW

시스템프로그래밍 1분반 12180626 성시열

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

```
#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(;;);
}
~
~
```

```
[12180626@linuxer1 ~]$ vi ex0.c
[12180626@linuxer1 ~]$ gcc -o ex0 ex0.c
```

```
[12180626@linuxer1 ~]$ ex0&
[1] 23332
[12180626@linuxer1 ~]$ PID : 23332 PPID : 23135
ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
12180626 23135 23134  0 01:10 pts/4    00:00:00 -bash
12180626 23332 23135 88 01:16 pts/4    00:00:05 ex0
12180626 23336 23135  0 01:16 pts/4    00:00:00 ps -f
```

컴파일 후 ex0&로 실행해주었는데 이 때 '&'는 프로세스를 백그라운드로 실행하여 프로그램을 돌리면서 다른 명령어를 실행할 수 있게 한다.
ex0 프로그램을 돌리면서 ps –f을 통해 현재 진행중인 프로세스를 확인할 수 있었다.
실행결과 PID는 23332, PPID는 23135임을 알 수 있었고,
ps –f를 통해 ex0이 무한루프가 계속 돌아가고 있음을 확인하였다.

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?

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

void main(){
        int x;
        x = fork();
        printf("hello\n");
        for(;;);
}
~
~
```

```
[12180626@linuxer1 ~]$ vi ex1.c
[12180626@linuxer1 ~]$ gcc -o ex1 ex1.c
```

```
[12180626@linuxer1 ~]$ ex1&
[1] 4758
[12180626@linuxer1 ~]$ hello
hello
ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
12180626  4346  4345  0 09:12 pts/17   00:00:00 -bash
12180626  4758  4346 48 09:16 pts/17   00:00:02 ex1
12180626  4759  4758 20 09:16 pts/17   00:00:01 ex1
12180626  4767  4346  0 09:16 pts/17   00:00:00 ps -f
```

```
[12180626@linuxer1 ~]$ kill 4758 4759
[12180626@linuxer1 ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
12180626  4346  4345  0 09:12 pts/17   00:00:00 -bash
12180626  5219  4346  0 09:25 pts/17   00:00:00 ps -f
```

fork()를 통해 프로세스를 복제해준다. 복제한, 복제된 파일은 x=fork(); 다음 라인부터 실행된다. 컴파일 후 ex1파일 실행 중 ps –f를 보기 위해 &를 붙여준다. 실행하면 hello가 두 번 출력됨을 알 수 있다. ps –f를 통해 ex1이 두 개가 실행되고 있고, PPID를 통해 4759의 부모 프로세스가 4758임을 알 수 있다.
kill을 통해 두 프로세스를 중지시키고, ps –f를 통해 값이 잘 중지되었는지 확인하였다.

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.

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.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(;;);
}
~
~
```

```
[12180626@linuxer1 ~]$ vi ex1.c
[12180626@linuxer1 ~]$ gcc -o ex1 ex1.c
[12180626@linuxer1 ~]$ ex1&
[1] 5637
[12180626@linuxer1 ~]$ hello, my pid is 5637
and my parent pid is 4346
hello, my pid is 5638
and my parent pid is 5637
```

```
ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
12180626  4346  4345  0 09:12 pts/17   00:00:00 -bash
12180626  5637  4346 42 09:31 pts/17   00:01:03 ex1
12180626  5638  5637 36 09:31 pts/17   00:00:54 ex1
12180626  5773  4346  0 09:33 pts/17   00:00:00 ps -f
```

먼저 실행된 프로세스가 5637이다.

두 번째 실행된 프로세스가 5638이고 이 프로세스의 부모 PID는 5637이다.
ps –f를 통해 확인하면 –bash → ex1 → ex1.child / -bash → ps –f의 부모자식 관계가 있음을 알 수 있다.

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

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main()
        int x;
        x = fork();
        printf("hello: %d\n", x);
}
```

```
[12180626@linuxer1 ~]$ vi ex2.c
[12180626@linuxer1 ~]$ gcc -o ex2 ex2.c
[12180626@linuxer1 ~]$ ex2
hello: 6599
hello: 0
```

fork()의 return 값인 x를 출력하였다. 자식에서는 return값이 0이고, 부모에서는 return 값이 자식의 PID이다. 즉, 부모 → 자식 순으로 실행되었으며, 자식 PID는 6599임을 알 수 있다.

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

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main(){
        int x;
        x = fork();
        printf("hello: %d\n", x);
        for(;;);
}
```

```
[12180626@linuxer1 ~]$ cp ex2.c ex3.c
[12180626@linuxer1 ~]$ vi ex3.c
[12180626@linuxer1 ~]$ gcc -o ex3 ex3.c
```

```
[12180626@linuxer1 ~]$ vi ex3.c
[12180626@linuxer1 ~]$ ex3&
[1] 7360
[12180626@linuxer1 ~]$ hello: 7361
hello: 0
ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Jun15 ?        00:00:10 /usr/lib/systemd/systemd --switc
root         2     0  0 Jun15 ?        00:00:00 [kthreadd]
```

ex2.c와 코드가 유사하므로 cp를 통해 값을 복사한 뒤, ex3.c 코드를 수정하였다.
이후 컴파일한 뒤 ex3&를 통해 실행한다. 프로세스 실행되는 도중에 ps –ef를 통해 다른 사용자들의 프로세스도 확인한다.

```
root         1     0  0 Jun15 ?        00:00:10 /usr/lib/systemd/systemd --switc
root      1020     1  0 Jun15 ?        00:00:21 /usr/sbin/sshd -D
root      7183  1020  0 10:04 ?        00:00:00 sshd: 12180626 [priv]
12180626  7186  7183  0 10:04 ?        00:00:00 sshd: 12180626@pts/17
12180626  7187  7186  0 10:04 pts/17   00:00:00 -bash
12180626  7360  7187 25 10:05 pts/17   00:00:01 ex3
12180626  7361  7360 11 10:05 pts/17   00:00:00 ex3
12180626  7362  7187  0 10:05 pts/17   00:00:00 ps -ef
```

위 실행코드를 통해 7361 ex1 부모임을 알 수 있었다. 결과값으로 부모, 부모의 부모 등을 파악할 수 있다.
... → -bash → ex3 → ex3.child
... → -bash→ ps –ef

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

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main(){
        int x;
        x = fork();
        if(x==0){
                printf("hello: %d\n", x);
        }
        else{
                printf("hi: %d\n", x);
        }
}
```

```
[12180626@linuxer1 ~]$ ex4
hi: 8350
hello: 0
```

출력값에 hi: 8285는 fork()의 return 값이 0이 아니라는 뜻으로 부모 프로세스가 먼저 실행됨을 알 수 있다. 이후 hello: 0이 나온 것으로 보아 자식 프로세스가 실행되었음을 알 수 있다.

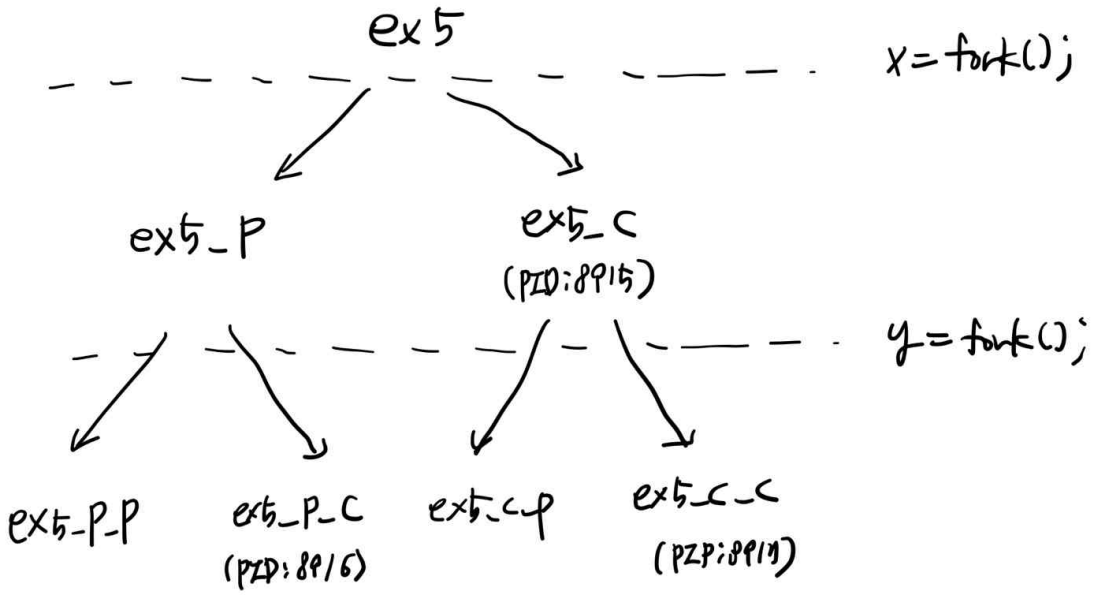
6) Try below (ex5.c). How many hellos do you see? Explain why you have that many hellos. Draw the process tree.

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

void main()
{
        int x, y;
        x = fork();
        y = fork();
        printf("hello: %d %d\n", x, y);
}
```

```
[12180626@linuxer1 ~]$ vi ex6.c
[12180626@linuxer1 ~]$ gcc -o ex6 ex6.c
[12180626@linuxer1 ~]$ ex6
hello: 8451 8452
hello: 8451 0
hello: 0 8453
hello: 0 0
```

먼저 x=fork();를 통해 ex5가 복제된다. 복제된 프로그램은 x=fork(); 이후의 코드이므로 각각의 복제된 프로그램에서 y=fork();가 실행되어 총 4개의 프로그램이 존재하게 된다.
x=fork();를 통해 복제된 프로그램을 ex5_p / ex5_c라고 하고 각각의 프로그램에서 복제된 프로그램을 모두 고려하면 ex5_p_p / ex5_p_c / ex5_c_p / ex5_c_c라고 할 수 있다.
첫 fork()를 통해 ex5_p은 자식 PID인 8915을 return하고 ex5_c은 0을 return 한다.
이후 ex5_p_p는 ex5_p_c의 PID인 8916를, ex5_p_c는 0을 return한다.
이후 ex5_p_p는 ex5_c_c의 PID인 8917를, ex5_c_c는 0을 return한다.



tree로 표현하면 다음과 같다.

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

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

void main(){
        int x, y;
        x = fork();
        printf("hello: %d\n", x);
        y = fork();
        printf("hello: %d\n", y);
}
```

```
[12180626@linuxer1 ~]$ vi ex6.c
[12180626@linuxer1 ~]$ gcc -o ex6 ex6.c
[12180626@linuxer1 ~]$ ex6
hello: 9114
hello: 0
hello: 9115
hello: 0
hello: 9116
hello: 0
```

x = fork();를 통해 ex6를 복제하였다. 복제된 파일은 x값을 출력하는데 이 때 부모가 먼저 실행되어 자식 PID를 출력한다. 이후 자식이 실행되어 0을 출력한다.
이후에 y = fork();를 통해 복제된 각각의 프로그램에서 복제를 한 번 더 진행한다.
두 번의 fork()를 통해 총 4개의 process가 존재한다.
부모로부터 복제된 자식 PID는 9115이므로 9115를 출력하고 자식 PID는 0을 출력한다.
자식으로부터 복제된 자식 PID는 9116이므로 9115를 출력하고 자식 PID는 0을 출력한다.

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();   //프로그램을 복제한다.
        if (x==0){ //x==0일 때는 자식일 때를 의미한다.
            int k;
            for(k=0;k<10;k++){   //프로그램 복제를 반복하면서
                                 자식일 경우에만 이 내부 for문을 실행한다.
                printf("%d-th child running %d-th iteration₩n", i, k);
                    //몇번째 복제에서 몇 번째 for문을 실행하는지 표시한다.
                fflush(stdout);          // 버퍼에 쌓아둔뒤 실행하지 않고 즉시 실행
                sleep(1);                // 1초간 지연
            }
            exit(0);    // child exits after 10 iterations
        }
    }
    // now parent

    printf("parent exits₩n");
}
```

```
[12180626@linuxer1 ~]$ vi ex7.c
[12180626@linuxer1 ~]$ gcc -o ex7 ex7.c
[12180626@linuxer1 ~]$ ex7
0-th child running 0-th iteration
1-th child running 0-th iteration
2-th child running 0-th iteration
3-th child running 0-th iteration
parent exits
```

```
[12180626@linuxer1 ~]$ 4-th child running 0-th iteration
0-th child running 1-th iteration
2-th child running 1-th iteration
3-th child running 1-th iteration
1-th child running 1-th iteration
4-th child running 1-th iteration
0-th child running 2-th iteration
2-th child running 2-th iteration
3-th child running 2-th iteration
1-th child running 2-th iteration
4-th child running 2-th iteration
0-th child running 3-th iteration
2-th child running 3-th iteration
3-th child running 3-th iteration
1-th child running 3-th iteration
4-th child running 3-th iteration
0-th child running 4-th iteration
3-th child running 4-th iteration
2-th child running 4-th iteration
1-th child running 4-th iteration
4-th child running 4-th iteration
0-th child running 5-th iteration
3-th child running 5-th iteration
2-th child running 5-th iteration
1-th child running 5-th iteration
4-th child running 5-th iteration
0-th child running 6-th iteration
3-th child running 6-th iteration
2-th child running 6-th iteration
1-th child running 6-th iteration
4-th child running 6-th iteration
0-th child running 7-th iteration
3-th child running 7-th iteration
2-th child running 7-th iteration
1-th child running 7-th iteration
4-th child running 7-th iteration
0-th child running 8-th iteration
3-th child running 8-th iteration
2-th child running 8-th iteration
1-th child running 8-th iteration
4-th child running 8-th iteration
0-th child running 9-th iteration
3-th child running 9-th iteration
2-th child running 9-th iteration
1-th child running 9-th iteration
4-th child running 9-th iteration
```

결과를 분석해보면 iteration은 항상 0부터 9의 순서대로 출력되지만 child의 running은 실행시마다 순서가 달라지는 것을 확인할 수 있었다. 이는 scheduler의 복잡한 계산에 의해 어느 것이 더 시급한 프로그램인가를 조사 후 실행시키기 때문임을 알 수 있다.
이것은 매번 달라지기 때문에 랜덤하다고 봐도 무방할 것이다.
ex5가 종료되는 순간 shell도 실행 대상이므로 shell도 scheduler의 계산에 의해 정해진 순서에서 실행된다.
이는 부모가 끝나기 전에 shell만 wait인 상태이고 나머지는 모든 프로세스가 scheduler의 대상이 되기 때문이다.

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.

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

void main(){
        int x, i;
        for(i=0; i<5; i++){
                x = fork();
                if(x==0){
                        int k;
                        for(k=0; k<10; k++){
                                printf("%d-th child running %d-th iteration\n", i, k);
                                fflush(stdout);
                                sleep(1);
                        }

                }
        }
        //now parent
        printf("parent exits\n");
}
~
~
```

```
[12180626@linuxer1 ~]$ vi ex7.c
[12180626@linuxer1 ~]$ gcc -o ex7 ex7.c
[12180626@linuxer1 ~]$ ex7
0-th child running 0-th iteration
1-th child running 0-th iteration
2-th child running 0-th iteration
parent exits
4-th child running 0-th iteration
[12180626@linuxer1 ~]$ 3-th child running 0-th iteration
0-th child running 1-th iteration
1-th child running 1-th iteration
2-th child running 1-th iteration
4-th child running 1-th iteration
3-th child running 1-th iteration
0-th child running 2-th iteration
1-th child running 2-th iteration
2-th child running 2-th iteration
4-th child running 2-th iteration
3-th child running 2-th iteration
0-th child running 3-th iteration
1-th child running 3-th iteration
2-th child running 3-th iteration
4-th child running 3-th iteration
3-th child running 3-th iteration
1-th child running 4-th iteration
0-th child running 4-th iteration
2-th child running 4-th iteration
4-th child running 4-th iteration
3-th child running 4-th iteration
1-th child running 5-th iteration
0-th child running 5-th iteration
2-th child running 5-th iteration
4-th child running 5-th iteration
3-th child running 5-th iteration
1-th child running 6-th iteration
0-th child running 6-th iteration
```

이전 코드에서 exit(0)을 제거한 뒤 컴파일하여 실행시켜보았다. 실행결과 계속해서 자식이 생성된다. fork()를 통해 복제되는 횟수는 2의 n승만큼의 프로세스를 만들게 된다.
즉 for문을 통해 (i는 0~4) 5번 반복하므로 2의 5승인 32개의 프로세스가 생성될 것이다.
확인을 위해 코드에 for(;;)를 추가한 뒤 ex7&로 실행하고, 실행되고 있는 프로세스를 확인하

기 위해 ps –f를 사용한다.

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

void main(){
        int x, i;
        for(i=0; i<5; i++){
                x = fork();
                if(x==0){
                        int k;
                        for(k=0; k<10; k++){
                                printf("%d-th child running %d-th iteration\n", i, k);
                                fflush(stdout);
                                sleep(1);
                        }

                }
        }
        //the parent
        printf("parent exits\n");
        for(;;);
}
~
```

```
4-th child running 9-th iteration
parent exits
ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
12180626  7187  7186  0 10:04 pts/17   00:00:00 -bash
12180626 11406  7187  5 11:32 pts/17   00:00:03 ex7
12180626 11407 11406  1 11:32 pts/17   00:00:01 ex7
12180626 11408 11406  1 11:32 pts/17   00:00:01 ex7
12180626 11409 11406  1 11:32 pts/17   00:00:01 ex7
12180626 11410 11406  1 11:32 pts/17   00:00:01 ex7
12180626 11411 11406  1 11:32 pts/17   00:00:01 ex7
12180626 11413 11407  1 11:32 pts/17   00:00:00 ex7
12180626 11414 11408  1 11:32 pts/17   00:00:00 ex7
12180626 11415 11409  1 11:32 pts/17   00:00:01 ex7
12180626 11416 11408  1 11:32 pts/17   00:00:00 ex7
12180626 11417 11409  1 11:32 pts/17   00:00:01 ex7
12180626 11418 11408  1 11:32 pts/17   00:00:00 ex7
12180626 11419 11407  1 11:32 pts/17   00:00:01 ex7
12180626 11420 11407  1 11:32 pts/17   00:00:00 ex7
12180626 11421 11407  1 11:32 pts/17   00:00:01 ex7
12180626 11422 11410  1 11:32 pts/17   00:00:00 ex7
12180626 11427 11413  0 11:32 pts/17   00:00:00 ex7
12180626 11428 11413  0 11:32 pts/17   00:00:00 ex7
12180626 11429 11413  0 11:32 pts/17   00:00:00 ex7
12180626 11430 11420  0 11:32 pts/17   00:00:00 ex7
12180626 11431 11415  0 11:32 pts/17   00:00:00 ex7
12180626 11432 11419  0 11:32 pts/17   00:00:00 ex7
12180626 11433 11419  0 11:32 pts/17   00:00:00 ex7
12180626 11434 11416  0 11:32 pts/17   00:00:00 ex7
12180626 11435 11414  0 11:32 pts/17   00:00:00 ex7
12180626 11436 11414  0 11:32 pts/17   00:00:00 ex7
12180626 11445 11428  0 11:32 pts/17   00:00:00 ex7
12180626 11446 11427  0 11:32 pts/17   00:00:00 ex7
12180626 11447 11427  0 11:32 pts/17   00:00:00 ex7
12180626 11448 11432  0 11:32 pts/17   00:00:00 ex7
12180626 11449 11435  0 11:32 pts/17   00:00:00 ex7
12180626 11450 11446  0 11:32 pts/17   00:00:00 ex7
12180626 11464  7187  0 11:33 pts/17   00:00:00 ps -f
```

실행중인 프로세스가 32개임을 확인하였다.