



Chapter 3. Process (Part 2)

3.3 Operations on Processes

In UNIX-like OS

- 새로운 프로세스는 `fork()` 를 통해 생성
- child 프로세스의 구성
 - ↳ parent 프로세스의 주소 공간을 그대로 복사하여 child에 집어넣음
- 두 개의 프로세스(child, parent)는 계속 실행됨
 - ↳ child는 `fork()` 시스템 콜 이후의 명령어를 실행
- 두 프로세스의 구분
 - ↳ `fork()` 이후에, return 코드가 zero 면 child process
 - ↳ return nonzero 면 parent process

`fork()` 예제

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid = fork(); // child process 를 만들
    printf("Hello, Process! %d\n", pid);

    return 0;
}
```

- 실행 결과

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

→ ~ cd Desktop
→ Desktop g++ os.cpp
→ Desktop ./a.out
Hello, Process! 4101
Hello, Process! 0
→ Desktop
```

- 설명

1. parent process

- a. fork() > parent의 주소 공간 복사 > child process가 생성
- b. 이후 parent의 print문이 수행 > "Hello, Process! 4101 "
- c. return 0 > parent process가 종료

2. child process

- a. fork()를 한 부분 부터 실행
- b. child의 print문 수행 > "Hello, Process! 0 "
- c. return 0 > child process 종료

fork() - wait()

- parent의 주소 공간을 복제 후, 계속 실행
- child run 하는 동안, parent process에서 wait()를 호출
 - ↳ 생성한 child를 책임지기 위해 wait()를 호출
- parent process는 wait queue로 이동 > child process가 종료될 때 까지 대기

fork() - wait() 예제

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

int main(){
    pid_t pid;
    pid = fork(); // child 생성
    if (pid > 0 )
        wait(NULL);
    printf("Hello, Process! %d\n", pid);

    return 0;
}
```

- 실행 결과

```

    TERMINAL    PROBLEMS    OUTPUT    DEBUG CONSOLE
→ Desktop gcc os.c
→ Desktop ./a.out
Hello, Process! 0
Hello, Process! 5481
→ Desktop █

```

- 설명

1. parent → fork() > child 생성
2. parent → if문 > 부모의 pid는 0보다 큼 > wait() 호출 > child가 종료될 때 까지 대기
3. child → fork() 이후 부터 실행 > print문 수행 > "Hello, Process! 0"
4. child → return 0 만나 종료
5. parent → child가 종료되었으므로 다시 실행 시작 > print 문 수행 > "Hello, Process! 5481"
6. parent → return 0 만나 종료

fork() - wait() 예제 2

```

int value = 5; //전역 변수

int main(){
    pid_t pid;
    pid = fork();
    if (pid == 0){ //child 인 경우
        value += 15;
        return 0;
    }
    else if (pid > 0){ //parent 인 경우
        wait(NULL);
        printf("Parent: value = %d\n", value);
    }
}

```

- 실행 결과

```

    TERMINAL    PROBLEMS    OUTPUT    DEBUG CONSOLE
→ Desktop gcc os.c
→ Desktop ./a.out
Parent: value = 5

```

- 설명
 - ↳ 제대로 이해하지 못한 경우, 결과값이 20이 나올 것 이라고 착각할 수 있다.
 - ↳ parent와 child는 같은 공간을 공유하는 것이 아니라, 복사해서 사용
 - ↳ 즉, parent의 value와 child의 value는 다른 공간을 사용하는 다른 변수
 - ↳ child에서 바뀌는 것은 parent에 영향을 주지 않는다
- 결과 ⇒ child의 value는 20 이지만, parent의 value는 5 이다.

fork() fork() fork() 예제

```
int main(){
    fork();
    fork();
    fork();
}
```

- 총 몇개의 프로세스가 실행될까? ⇒ **8개**
- 그림 그리며 확인 해보기

for (fork()) 예제

```
int main(){
    pid_t pid;
    int i;
    for(i = 0; i < 4 ; i++)
        pid = fork();
    printf("Hello, fork() %d\n", pid);
    return 0;
}
```

- 총 16개의 프로세스 생성
- pid를 출력해보며 확인해 볼 수 있다.

```
TERMINAL    PROBLEMS    OUTPUT    DEBUG CONSOLE

→ Desktop gcc os.c
→ Desktop ./a.out
Hello, fork() 7837
Hello, fork() 0
Hello, fork() 7840
Hello, fork() 7841
Hello, fork() 0
Hello, fork() 0
Hello, fork() 7842
Hello, fork() 7844
Hello, fork() 0
Hello, fork() 0
Hello, fork() 7845
Hello, fork() 7847
Hello, fork() 0
Hello, fork() 7848
Hello, fork() 0
Hello, fork() 0
```

execvp() 예제

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

int main(){
    pid_t pid;
    pid = fork();

    if(pid == 0){ //child process
        execvp("/bin/ls", "ls", NULL); // ls 명령을 실행함
        printf("LINE J\n"); // *실행되지 않는다
    }
    else if(pid > 0){ //parent process
        wait(NULL);
        printf("Child Complete\n"); // 실행됨
    }

    return 0;
}
```

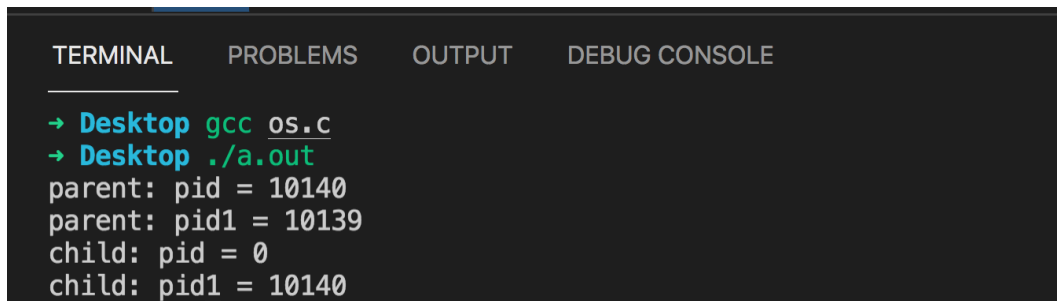
- 새로운 process는 fork()해서 생성하고, parent를 복제한 child를 생성한다.
- 하지만, fork()를 하더라도 parent와 child가 다른 기능을 수행하게 하고싶을 수 있다.
- execvp() 를 사용하여 child process가 기존 memory에 새로운 기능을 덮어씌운다.
 - ↳ 여기서 중요한점! execvp() 이후의 구문은 실행되지 않는다.
 - ↳ 즉, 위의 코드에서 "LINE J" 는 출력되지 않는다.
- 실행결과
 - ls명령어를 실행한 결과 + "Child Complete" 가 출력된다.

getpid() 예제

```
int main(){
    pid_t pid, pid1;
    pid = fork();

    if(pid == 0){ //child process
        pid1 = getpid();
        printf("child: pid = %d\n", pid);
        printf("child: pid1 = %d\n", pid1);
    }
    else if(pid > 0){ //parent process
        pid1 = getpid();
        printf("parent: pid = %d\n", pid);
        printf("parent: pid1 = %d\n", pid1);
    }
    return 0;
}
```

- 실행결과



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
→ Desktop gcc os.c
→ Desktop ./a.out
parent: pid = 10140
parent: pid1 = 10139
child: pid = 0
child: pid1 = 10140
```

- 설명
 - ↳ getpid()는 자신의 pid를 얻어온다.
 - ↳ parent.pid는 child.pid1과 같다.