OS01 3주차 과제 - 201702081

- fork
  - 소스코드

```c
#include <stdio.h>
#include <stdlib.h> // exit
#include <unistd.h> // fork, getpid

int global_int = 1000;

int main(int argc, char *argv[])
{
    int local_int = 10;

    printf("global int : %d \n", global_int);
    printf("local int : %d \n\n", local_int);

    int ret_fork = fork();
    if (ret_fork < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    }


    // Child
    else if (ret_fork == 0) {
        printf("Child pid : %d \n", (int) getpid());

        printf("child global int before : %d \n", global_int);
        printf("child local int before: %d \n\n", local_int);


        // Child's Modification
        global_int = 99999;
        local_int = 100000;
        printf("Child changed global, local var to 99999, 100000 \n");
        sleep(2);

        // Print after Parent's Modification
        printf("[After Parent modified] \n");
        printf("child global : %d \n", global_int);
        printf("child local : %d \n", local_int);
        printf("\n");
```

```c
    // Parent
    } else {

        printf("parent global int before : %d \n", global_int);
        printf("parent local int before : %d \n\n", local_int);


        // Print after Child's Modification
        sleep(1);
        printf("[After Child modified] \n");
        printf("parent global : %d \n", global_int);
        printf("parent local : %d \n", local_int);
        printf("\n");

        // Parent's Modification
        global_int = 99999999;
        local_int = 100000000;
        printf("Parent changed global, local var to 99999999, 100000000 \n");
    }



    return 0;
}
```

- 실행결과 : 서로의 global, local 변수에 영향을 주지 않음

```
global int : 1000
local int : 10

parent global int before : 1000
parent local int before : 10

Child pid : 18670
child global int before : 1000
child local int before: 10

Child changed global, local var to 99999, 100000
[After Child modified]
parent global : 1000
parent local : 10

Parent changed global, local var to 99999999, 100000000
u201702081@os:~/Desktop/cnu_os_prac/03$ [After Parent modified]
child global : 99999
child local : 100000
```

- 결과 설명 : sleep로 시간차를 두어, 변경 순서가 지켜지도록 하였다. 부모와 자식의 변수는 서로의 영향을 받지 않았는데, 이는 프로세스가 새로 만들어진 것이기 때문에 독립적인 공간을 갖기 때문이다.

- wait

  - 소스코드 (순서대로 wait1 ~ wait3)

```c
#include <stdio.h>
#include <stdlib.h>     // exit
#include <unistd.h>     // getpid, fork, sleep
#include <sys/wait.h>

int main(int argc, char *argv[])
{
    int ret_fork = fork();
    if (ret_fork < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (ret_fork == 0) {
        // child (new process)
        printf("안녕 ? \n");
    } else {
        // parent goes down this path (original process)
        int ret_wait = wait(NULL);
        printf("잘가 \n");
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>     // exit
#include <unistd.h>     // getpid, fork, sleep
#include <sys/wait.h>

int main(int argc, char *argv[])
{
    int ret_fork = fork();
    if (ret_fork < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (ret_fork == 0) {
        // child (new process)
        printf("안녕 ? \n");
    } else {
        // parent goes down this path (original process)
        sleep(1);
        printf("잘가 \n");
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>     // exit
#include <unistd.h>     // getpid, fork, getppid
#include <sys/wait.h>

int main(int argc, char *argv[])
{
    int ret_fork = fork();
    if (ret_fork < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (ret_fork == 0) {
        // child (new process)
        printf("안녕 ? \n");
        system("touch /tmp/foofoofoo");
    } else {
        // parent goes down this path (original process)
        while(access("/tmp/foofoofoo", 00) == -1);
        system("rm -rf /tmp/foofoofoo");
        printf("잘가 \n");

    }
    return 0;
}
```

- 실행결과

```
u201702081@os:~/Desktop/cnu_os_prac/03$ ./wait1
안녕 ?
잘가
u201702081@os:~/Desktop/cnu_os_prac/03$ ./wait2
안녕 ?
잘가
u201702081@os:~/Desktop/cnu_os_prac/03$ ./wait3
안녕 ?
잘가
```

- 결과 설명

wait1 : wait() 를 통해 자식프로세스가 종료되면 "잘가" 를 출력

wait2 : 1초 sleep() 을 통해 자식이 먼저 종료되도록 함

wait3 : 자식에서 임시파일을 생성하여, 부모는 그 파일이 생성됨을 확인한 후에 "잘가" 출력

이 세가지는 부모/자식간의 동기화 방법이, 자식 종료 확인 / 잠시 기다림 / 임시파일 이용으로 차이가 있다.

- exec
  - 소스코드

```c
#include <stdio.h>
#include <stdlib.h>     // exit
#include <unistd.h>     // getpid, fork, execvp
#include <string.h>     // strdup
#include <sys/wait.h>

int main(int argc, char *argv[])
{
    printf("hello world (pid:%d)\n", (int) getpid());
    int ret_fork = fork();
    if (ret_fork < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (ret_fork == 0) {
        // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
        char* myargs[2];
        myargs[0] = strdup("/bin/ls");
        myargs[1] = NULL;
        execvp("/bin/ls", myargs);
        printf("this shouldn't print out");
    } else {
        // parent goes down this path (original process)
        int ret_wait = wait(NULL);
        printf("hello, I am parent of %d (ret_wait:%d) (pid:%d)\n",
               ret_fork, ret_wait, (int) getpid());
    }
    return 0;
}
```

  - 실행결과

```
u201702081@os:~/Desktop/cnu_os_prac/03$ ./exec
hello world (pid:18796)
hello, I am child (pid:18797)
exec  exec_201702081.c  fork  fork_201702081.c  open  open_201702081.c
hello, I am parent of 18797 (ret_wait:18797) (pid:18796)
```

  - 결과 설명

자식프로세스가 exec() 시스템콜을 사용하여 /bin/ls로 바뀐다. exec의 종류가 많은 이유는, 인자를 전달하는 방식 / 환경변수 전달 방식을 세부적으로 나누기 위함이다.

- open
  - 소스코드

```c
#include <stdio.h>
#include <stdlib.h>     // exit
#include <unistd.h>     // fork, close
#include <string.h>     // strdup
#include <fcntl.h>      // open(flags, modes)
#include <sys/wait.h>

int main(int argc, char *argv[])
{
    close(STDOUT_FILENO);
    open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);

    int ret_fork = fork();

    if (ret_fork < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);

    } else if (ret_fork == 0) {
        // child: redirect standard output to a file
        printf("자식 왔다감 \n");
        sleep(3);

    } else {
        // parent goes down this path (original process)

        printf("부모 왔다감 \n");


    }
    return 0;
}
```

  - 실행결과

```
u201702081@os:~/Desktop/cnu_os_prac/03$ ./open
u201702081@os:~/Desktop/cnu_os_prac/03$ cat ./p4.output
부모 왔다감
자식 왔다감
```

  - 결과 설명

    파일 open 이후 fork로 분기하여, 같은 파일에 부모, 자식이 같이 접근하니 두 내용이 동시에 있음을 확인할 수 있었다

- 어려웠던 점
  - 결과가 이렇게 나오는게 맞는건지 확실하게 모르겠다. 결과예시라도 있으면 좋겠다