

1. Fork

1.1) 소스코드

```
u201600253@sejung-VirtualBox: ~/Desktop/test05
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int global = 1;
6 int main()
7 {
8     int local;
9     local = 10;
10    printf("first : global(%d), local(%d)\n", global, local);
11
12    pid_t childpid = fork();
13
14    if(childpid < 0){
15        printf("fork failed\n");
16        return 1;
17    }
18
19    if(childpid > 0){
20        global++, local++;
21        printf("Parent : global(%d), local(%d)\n", global, local);
22        printf("I'm parents. My PID is %d\n", (int)getpid());
23        printf("My child's PID : %d\n", childpid);
24        printf("\n");
25        sleep(1);
26    }
27    else {
28        global++, local++;
29        printf("Child : global(%d), local(%d)\n", global, local);
30        printf("I'm child. My PID is %d\n", (int)getpid());
31        printf("My parent's PID : %d\n", (int)getppid());
32        printf("\n");
33    }
34    return 0;
35 }
```

1.2) 결과

```
u201600253@sejung-VirtualBox:~/Desktop/test05$ ./a.out
first : global(1), local(10)
Parent : global(2), local(11)
I'm parents. My PID is 8536
My child's PID : 8537

Child : global(2), local(11)
I'm child. My PID is 8537
My parent's PID : 8536
```

1.3) 설명

Fork에 의해 자식 프로세스를 생성할 시점에 부모 프로세스의 전역변수와 지역 변수 값을 복제한다. 즉, 두 프로세스는 독립된 메모리를 사용한다.

2. Wait1

2.1) 소스코드

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5
6 int main()
7 {
8     int pid;
9     int status;
10
11     pid = fork();
12
13     if(pid < 0) {
14         printf("fork failed\n");
15         exit(1);
16     }
17
18     if(pid == 0) {
19         printf("child's process(PID = %d) : 안녕\n", (int)getpid());
20         // 자식프로세스가 생성되면 PID값과 "안녕"을 출력한다
21         sleep(2);
22     } else {
23         wait(&status); // wait함수를 써서 자식이 종료할때까지 대기하게 한다
24         printf("parent's process(PID = %d) : 잘가\n", (int)getpid());
25         // 부모프로세스이면 PID값과 "잘가"를 출력한다.
26     }
27 }
```

2.2) 결과

```
waiting child
child's process(PID = 8986) : 안녕
parent's process(PID = 8985) : 잘가
```

2.3) 설명

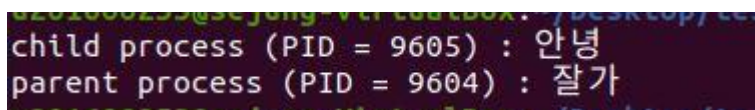
Wait을 통해 부모는 자식이 끝날 때까지 기다린다. 그리고 자식이 끝났다는 상태를 얻은 뒤 부모도 종료한다.

3. Wait2

3.1) 소스코드

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/wait.h>
4 #include <unistd.h>
5
6 int main()
7 {
8     pid_t cpid;
9     int status;
10
11     cpid = fork();
12
13     if(cpid == 0) { // 자식프로세스가 생성되면
14         printf("child process (PID = %d) : ", (int)getpid()); // PID값과
15         sleep(5);
16         printf("안녕\n"); // 안녕을 출력한다
17         exit(0);
18     }
19     else if(cpid > 0){ // 부모프로세스이면
20         int ret;
21         printf("parent process (PID = %d) : ", (int)getpid()); // PID값과
22         ret = waitpid(cpid, &status, 0);
23         // waitpid함수를 사용하여 특정 PID의 자식프로세스가 종료하기를 기다린다
24         printf("잘가\n");
25         exit(0);
26     }
27     else { // 프로세스 생성이 안되면
28         printf("fork failed\n");
29         return 1;
30     }
31     return 0;
32 }
```

3.2) 결과



```
child process (PID = 9605) : 안녕
parent process (PID = 9604) : 잘가
```

3.3) 설명

Waitpid함수는 자식이 종료될때까지 기다린다. 옵션을 사용해서 차단을 방지할 수 있고, 기다릴 자식 프로세스를 좀 더 상세하게 지정할 수 있다. 성공했을 때 PID를 반환하지만 오류가 생기면 -1을 반환한다. 함수의 옵션 중 첫번째 인자인 fork값은 자식인지 판단한다. 두번째 인자는 자식 프로세스가 정상적으로 종료했는지를 알려주고, 세번째 인자인 0은 wait함수와 동일한 동작을 한다.

4. Wait3

4.1) 소스코드

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <semaphore.h>
5 #include <errno.h>
6 #include <string.h>
7 #include <fcntl.h>
8
9 int main() {
10     sem_t *mutex = sem_open("sem_name", O_CREAT, 0644, 0);
11     pid_t pid = fork();
12
13     if(pid < 0) {
14         printf("fork failed!\n");
15         exit(1);
16     }
17     else if(pid == 0) {
18         printf("child process (PID = %d) : ", (int)getpid());
19         printf("안녕\n");
20         sem_post(mutex);
21     }
22     else {
23         sem_wait(mutex);
24         printf("parent process (PID = %d) : 잘가\n", (int)getpid());
25     }
26     return 0;
27 }
```

4.2) 결과

```
child process (PID = 9875) : 안녕
parent process (PID = 9874) : 잘가
```

4.3) 설명

뮤텍스를 써서 세마포어의 네번째 인자값이 0으로 설정하였다. post되는 부분을 자식프로세스가 생기는 곳에 놓고, wait하는 부분을 부모코드에 놓았다. 즉, 뮤텍스가 post되기전까지 부모가 실행 대기상태를 유지하는 코드를 짰다.

5. Exec

5.1) 소스코드

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6
7 int main()
8 {
9     printf("<start> PID : %d\n", (int)getpid());
10    int fork_val = fork();
11
12    if(fork_val < 0) { // fork에 실패하면
13        printf("fork failed!\n");
14        exit(1); // 종료
15    }
16    else if(fork_val == 0) { // 자식프로세스를 생성하면
17        printf("<child process> (PID : %d)\n", (int)getpid()); // 자식프로세
18        스 PID를 출력하고
19        execl("/bin/ls", "/bin/ls", "-al", "/tmp", NULL); // execl함수를 통>
20        해 결과적으로 ls -al /tmp를 터미널에서 수행한 값이 나온다.
21    }
22    else { // 부모프로세스이면
23        printf("<parent process> (PID : %d)\n", (int)getpid());
24    }
25    return 0;
26 }
```

5.2) 결과

```
u201600253@sejung-VirtualBox:~/Desktop/test05$ ./exec_201600253
<start> PID : 10496
<parent process> (PID : 10496)
u201600253@sejung-VirtualBox:~/Desktop/test05$ <child process> (PID : 10497)
total 68
drwxrwxrwt 17 root      root      4096 4월  4 02:12 .
drwxr-xr-x 20 root      root      4096 3월 18 20:16 ..
-rw----- 1 u201600253 u201600253  0 3월 29 00:10 config-err-JD64Xr
drwxrwxrwt 2 root      root      4096 3월 29 00:09 .font-unix
drwxrwxrwt 2 root      root      4096 3월 29 00:10 .ICE-unix
drwx----- 2 u201600253 u201600253 4096 3월 29 00:10 ssh-y5fRcxoI7mS5
drwx----- 3 root      root      4096 3월 29 00:10 systemd-private-494d33664
6ea4950bdd2e9202434defa-color.service-74Uvhg
drwx----- 3 root      root      4096 4월  4 00:55 systemd-private-494d33664
6ea4950bdd2e9202434defa-fwupd.service-yULnlj
drwx----- 3 root      root      4096 3월 29 00:09 systemd-private-494d33664
6ea4950bdd2e9202434defa-ModemManager.service-g366Ef
drwx----- 3 root      root      4096 3월 29 00:09 systemd-private-494d33664
6ea4950bdd2e9202434defa-switcheroo-control.service-RyQfoI
drwx----- 3 root      root      4096 3월 29 00:09 systemd-private-494d33664
6ea4950bdd2e9202434defa-systemd-logind.service-MQtWeg
drwx----- 3 root      root      4096 3월 29 00:09 systemd-private-494d33664
6ea4950bdd2e9202434defa-systemd-resolved.service-VxC1Fg
drwx----- 3 root      root      4096 3월 29 00:09 systemd-private-494d33664
6ea4950bdd2e9202434defa-systemd-timesyncd.service-IUE8th
drwx----- 3 root      root      4096 3월 29 00:10 systemd-private-494d33664
6ea4950bdd2e9202434defa-upower.service-BsjHRe
drwxrwxrwt 2 root      root      4096 3월 29 00:09 .Test-unix
drwx----- 2 u201600253 u201600253 4096 4월  4 02:12 tracker-extract-files.100
0
drwxrwxrwt 2 root      root      4096 3월 29 00:10 .X11-unix
drwxrwxrwt 2 root      root      4096 3월 29 00:09 .XIM-unix
```

5.3) 설명

시간이 지남에 따라 새로운 기능이 추가된 함수들이 필요했기 때문이다. 그리고 같은 라이브러리에 묶여 있는 오래된 프로그램을 변형하지 않고 서는 오래된 함수의 기능을 바꿀 수 없기 때문이다.

6. Open

6.1) 소스코드

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <fcntl.h>
5 #include <sys/wait.h>
6 #include <unistd.h>
7 #include <sys/types.h>
8
9 int main()
10 {
11     int file;
12     pid_t pid;
13
14     file = open("data.txt", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
15     // data.txt파일이 없으면 생성하고, 이는 쓰기전용, 있다면 덮어쓰고, user>
    권한을 갖는다.
16     if(file == -1) { // 만약 해당 파일을 여는데 문제가 생기면
17         printf("failed to open file\n");
18         exit(1);
19     }
20     else { // 해당 파일을 열 수 있다면
21         printf("successfully opened file\n");
22     }
23
24     int val = fork(); // 시스템콜을 통해서 fork한다
25
26     if(val < 0) { // fork를 실패하면
27         printf("failed fork\n");
28         exit(1);
29     }
30     else if(val == 0) { // 자식프로세스를 생성하면
31         printf("child process\n");
32         char msg[] = "I'm child. I write this letter in file\n";
33         write(file, msg, strlen(msg)); // 해당 글귀를 file에 적어준다
34     }
35     else { // 부모프로세스이면
36         printf("parent process\n");
37         char msg[] = "I'm parent. Closed file won't be called\n";
38         write(file, msg, strlen(msg)); // 해당 글귀를 file에 적어준다
39     }
40     close(file); // open했으니 file을 닫는다
41     return 0;
42 }
```

6.2) 결과

```
successfully opened file
parent process
u201600253@sejung-VirtualBox:~/Desktop/test05$ child process
cat data.txt
I'm parent. Closed file won't be called
I'm child. I write this letter in file
```

6.3) 설명

실행결과로 만든 파일에 부모에서 쓴 문자열과 자식에서 쓴 문자열이 들어있다. 즉, 2 개의 프로세스가 close 호출 없이 동시에 접근해서 쓰기 호출을 할 수 있는 것 같다.

7. 느낀점

이론 수업때 듣고 다 이해해도, 실습을 하려고 하면 제가 배웠던 개념이 어디서 적용되는건지 잘 모르겠습니다. 그래서 이론과 실습의 연관성을 잘 못찾겠습니다. 실습 설명하실 때 이론 수업때 배웠던 개념이 이런 방법으로 쓰인다고 말씀해주실 수 있나요? 그리고 실습때 리눅스가 운영체제와 어떻게 연관되는지도 설명해주실 수 있나요? 항상 잘 듣고 있습니다. 감사합니다.