

TI DSP, MCU 및 Xilinx Zynq FPGA 프로그램 전문가 과정

강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 - 은태영

zero_bird@naver.com

wait()

```
tewill@tewill-B85M-D3H: ~/Downloads
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

void term_status(int status)
{
    if(WIFEXITED(status))
        printf("(exit)status : 0x%x\n", WEXITSTATUS(status));
    else if(WTERMSIG(status))
        printf("(signal)status : 0x%x, %s\n", status & 0x7f,
            WCOREDUMP(status) ? "core dumped" : "");
}

int main(void)
{
    pid_t pid;
    int status;
    if((pid = fork()) > 0)
    {
        wait(&status);
        term_status(status);
    }
    else if(pid == 0)
        abort();
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```

13,29

All

- ❖ fork 를 통해 자식을 생성한다.
- ❖ 부모는 wait 를 통해 블럭 상태로 대기한다.
- ❖ 자식은 abort 를 통해 비정상 종료를 한다.
- ❖ 부모는 자식의 시그널을 받아 term_status 를 실행한다.
- ❖ term_status 는 정상 종료인지 비정상 종료인지 if 문을 통하여 구분하고, 조건에 맞게 출력한다.

wait()

```
tewill@tewill-B85M-D3H: ~/Downloads
int main()
{
    pid_t pid;
    int i;
    signal(SIGCHLD, my_sig);
    if((pid = fork()) > 0)
        for(i = 0; i < 10000; i++)
        {
            usleep(50000);
            printf("%d\n", i + 1);
        }
    else if(pid == 0)
        sleep(5);
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
-- INSERT --
```

45,14 Bot

- ❖ signal(int signum, void 핸들러(int))
- ❖ 어떠한 상황일 때, 핸들러 를 실행하도록 한다.
- ❖ 상황에 따라 제어가 가능하다.(비동기 처리)
- ❖ signal 을 만들고, SIGCHLD 가 들어올 경우, my_sig 를 실행한다.
- ❖ fork 로 자식을 생성한 후, 자식은 5초 후 죽는다.
- ❖ 부모는 0.05초 단위로 숫자를 출력한다.

wait()

```
tewill@tewill-B85M-D3H: ~/Downloads
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

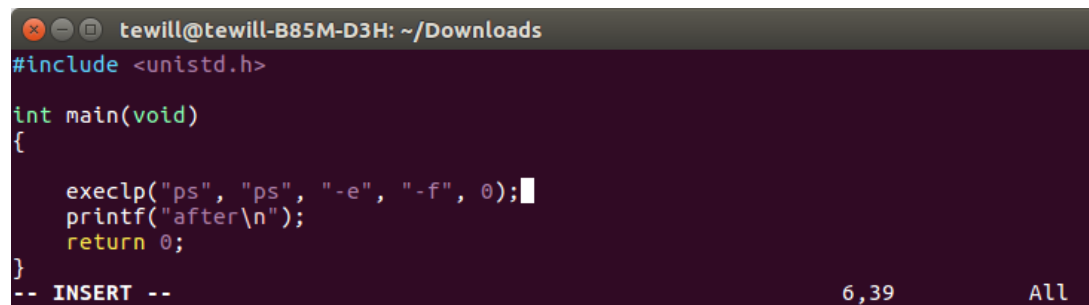
void term_status(int status)
{
    if(WIFEXITED(status))
        printf("(exit)status : 0x%x\n", WEXITSTATUS(status));
    else if(WTERMSIG(status))
        printf("(signal)status : 0x%x, %s\n",
            status & 0x7f, WCOREDUMP(status) ? "core dumped" : "");
}

void my_sig(int signo)
{
    int status;
    wait(&status);
    term_status(status);
}

-- INSERT --
```

- ❖ 자식이 죽는 순간 sigchild 가 my_sig 에 전달되며 실행된다.
- ❖ wait 을 통해 status에 시그널을 받는다.
- ❖ term_status 를 실행하여, 정상 종료인지 비정상 종료인지 확인하고, 상황에 맞게 출력한다.
- ❖ 5초 후에 죽기 때문에, 부모에서 0.05초 단위로 출력되는 숫자가 100 에 근접했을 때 시그널 출력을 확인 가능하다.
- ❖ waitpid() : wait 의 논 블러킹 형태이다. sigchild 로 들어온 것들을 기록하고, 순서대로 처리한다.(예약 후 실행)

wait()



```
tewill@tewill-B85M-D3H: ~/Downloads
#include <unistd.h>

int main(void)
{
    execlp("ps", "ps", "-e", "-f", 0);
    printf("after\n");
    return 0;
}
-- INSERT --
```

- ❖ `execlp()` : 프로세스 실행한다. 첫번째 인자로 프로그램 이름을 받고, 마지막 인자 0 을 통해 끝을 나타낸다.
- ❖ 프로세스의 형태를 변경시킨다.
- ❖ 메모리 레이아웃을 실행시키는 프로그램으로 변경한다.
- ❖ 즉, `ps` 의 기능만 실행되고, `after` 는 출력되지 않는다.

wait()

```
tewill@tewill-B85M-D3H: ~/Downloads
#include <unistd.h>
#include <stdio.h>

int main(void)
{
    int status;
    pid_t pid;
    if((pid = fork()) > 0)
    {
        wait(&status);
        printf("prompt >");
    } else if(pid == 0)
    {
        execlp("ps", "ps", "-e", "-f", 0);
    }
    return 0;
}
```

1,1 All

- ❖ 이를 해결하는 방법으로 fork 로 자식을 생성 후, execlp 를 실행한다.
- ❖ ps 가 실행 된 후, 'prompt >' 가 출력되는 걸 확인 할 수 있다.

wait()

```
tewill@tewill-B85M-D3H: ~/Downloads
#include <unistd.h>
#include <stdio.h>

int main(void)
{
    int status;
    pid_t pid;
    if(pid = fork() > 0)
    {
        wait(&status);
        printf("prompt >");
    }
    else if(pid == 0)
        execl("./newpgm", "newpgm", "one", "two", (char *) 0);
    return 0;
}
```

1,1 All

- ❖ 해당 방법을 이용하여 ps 뿐만이 아니라, 유저가 생성한 실행 파일도 사용 가능하다.
- ❖ 현재 폴더에 위치한 'newpgm' 을 실행시킨다.
- ❖ 인자로 newpgm 과 one, two 를 보내준다.

wait()

```
tewill@tewill-B85M-D3H: ~/Downloads
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
    int i;
    for(i = 0; argv[i]; i++)
        printf("argv[%d] = [%s]\n", i, argv[i]);

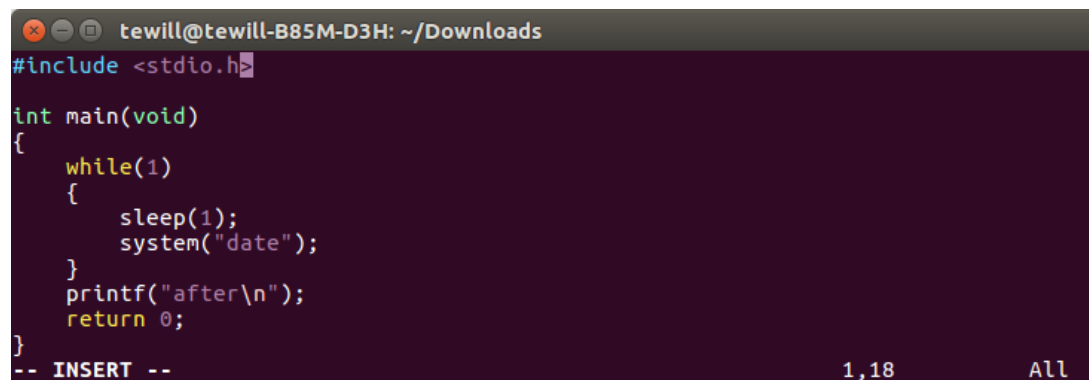
    for(i = 0; envp[i]; i++)
        printf("envp[%d] = [%s]\n", i, envp[i]);
    return 0;
}
-- INSERT --
```

```
tewill@tewill-B85M-D3H: ~/Downloads
#include <unistd.h>
#include <stdio.h>

int main(void)
{
    int status;
    pid_t pid;
    char *argv[] = {"/newpgm", "newpgm", "one", "two", 0};
    char *env[] = {"name = OS_Hacker", "age = 20", 0};
    if((pid = fork()) > 0)
    {
        wait(&status);
        printf("prompt >\n");
    } else if(pid == 0){
        execve("./newpgm", argv, env);
    }
    return 0;
}
```

- ❖ 두가지 인자 종류를 보내는 형태이다.
- ❖ argv 만 사용 시 env 를 인식 못한다.
- ❖ 따로 받는 인자 envp 를 만들어서 env 를 받는다.

wait()



```
tewill@tewill-B85M-D3H: ~/Downloads
#include <stdio.h>

int main(void)
{
    while(1)
    {
        sleep(1);
        system("date");
    }
    printf("after\n");
    return 0;
}
-- INSERT --
```

- ❖ system() : 내부적으로 fork 를 한 후 exe 를 실행한다.
- ❖ date 가 실행되는 모습을 볼 수 있다.
- ❖ 반복문을 통해 프로세스가 계속 가동하게 한다.
- ❖ 하지만 터미널이 종료될 경우, 해당 프로세스도 종료가 된다.

wait()

```
tewill@tewill-B85M-D3H: ~/Downloads
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int my_system(char *cmd)
{
    pid_t pid;
    int status;
    char *argv[] = {"sh", "-c", cmd, 0};
    char *envp[] = {0};
    if((pid = fork()) > 0)
        wait(&status);
    else if(pid == 0)
        execve("/bin/sh", argv, envp);
}

int main(void)
{
    my_system("date");
    printf("after\n");
    return 0;
}
```

10,39 All

- ❖ system() 을 fork 와 wait 을 이용하여 구현했다.
- ❖ “sh” 셸
- ❖ “-c” 실행시킨다.
- ❖ cmd = date.
- ❖ execve 는 execlp와 다르게 환경 변수를 갖는다.

wait()

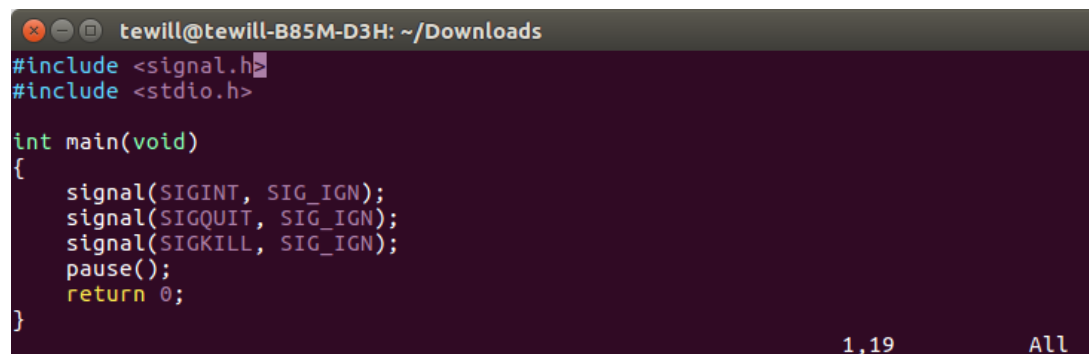
```
tewill@tewill-B85M-D3H: ~/Downloads
#include <sys/types.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

int daemon_init(void)
{
    int i;
    if(fork() > 0)
        exit(0);
    setsid();
    chdir("/");
    umask(0);
    for(i = 0; i < 64; i++)
        close(i);
    signal(SIGCHLD, SIG_IGN);
    return 0;
}

int main(void)
{
    daemon_init();
    for(;;)
    {
        sleep(1);
        printf("hi\n");
    }
    return 0;
}
-- INSERT --
```

- ❖ 터미널이 죽어도 살아남는 데몬 프로세스이다.
- ❖ 데몬 함수를 실행 후, 반복문을 통해 유지시킨다.
- ❖ 데몬 함수 내에서 자식을 생성 후 부모를 죽인다.
- ❖ TTI 가상 터미널 (세션 ID) : 터미널과 생명을 같이 한다.
- ❖ setsid() : TTI 소속이 없어진다.(데몬 상태)
- ❖ chdir("/") : / 위치로 이동시킨다.
- ❖ umask(0) : 권한 설정. root 에 있는 모든 것을 사용 가능하다.
- ❖ 상속 받는 것을 달는다. 리눅스는 기본적으로 64개의 파일을 열어놓기 때문에 64번 반복한다.
- ❖ SIG_IGN : 상관하지 않는다. 즉, 자식이 죽어도 상관하지 않는다는 뜻이다.

wait()



```
tewill@tewill-B85M-D3H: ~/Downloads
#include <signal.h>
#include <stdio.h>

int main(void)
{
    signal(SIGINT, SIG_IGN);
    signal(SIGQUIT, SIG_IGN);
    signal(SIGKILL, SIG_IGN);
    pause();
    return 0;
}
```

1,19 All

- ❖ 이를 응용해서 시그널 들을 막을 수 있다.
- ❖ 하지만, SIGKILL 은 막을 수 없다.
- ❖ 데몬 프로세스를 통해 서비스 프로그램 등이 이용된다. 예를 들어 fpt, 토렌트, 웹서버, 구글, 게임 서버 등 데몬 프로세스를 통해 유지시킨다.
- ❖ 데몬 프로세스를 죽이기 위해서는 이니시, 즉 SIGKILL 을 사용하거나 디바이스를 종료해야 한다.

