

Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – **Innova Lee(이상훈)**
gcccompil3r@gmail.com

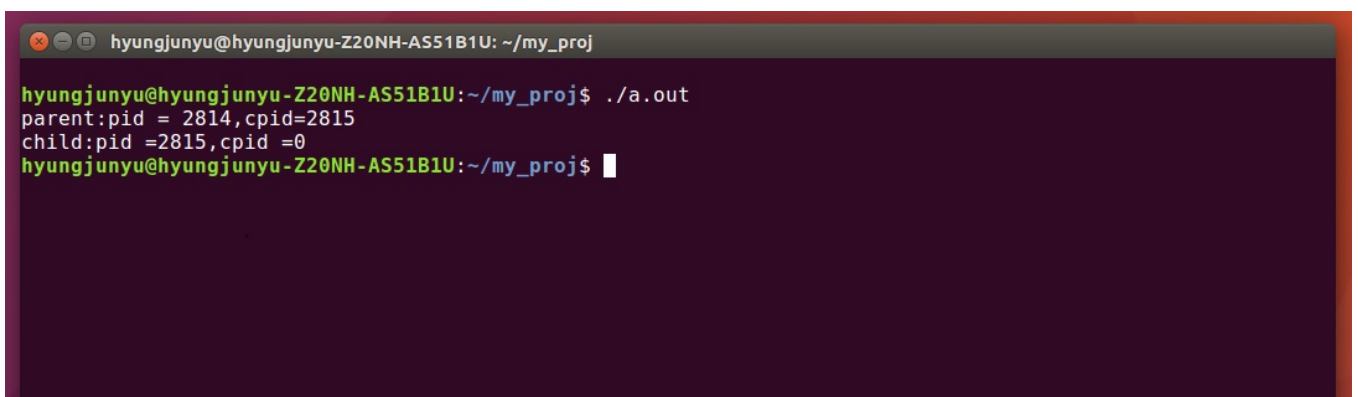
학생 hyungjun Yu(유형준)
love592946@naver.com

일단 먼저 fork에 대한 개념이 없으므로
3월23일(금요일)에 했던 내용을 복습하자.

```
#include<unistd.h>
#include<errno.h>
#include<stdlib.h>

int main()
{
    pid_t pid;
    pid = fork();
    if(pid>0)
        printf("parent:pid = %d,cpid=%d\n",getpid(),pid);
    else if(pid ==0)
        printf("child:pid =%d,cpid =%d\n",getpid(),pid);
    else
    {
        perror("fork()");
        exit(-1);
    }
    return 0;
}
```

---->> 이 소스코드를 보면 parent의 pid값과 child의 getpid()값(PID)
이 같은 걸 알 수 있다.



```
hyungjunyu@hyungjunyu-Z20NH-AS51B1U: ~/my_proj
hyungjunyu@hyungjunyu-Z20NH-AS51B1U:~/my_proj$ ./a.out
parent:pid = 2814,cpid=2815
child:pid =2815,cpid =0
hyungjunyu@hyungjunyu-Z20NH-AS51B1U:~/my_proj$
```

```

int global = 100;

int main()
{
    int local =10;
    pid_t pid;
    int i;
    pid= fork();
    if(pid>0)
    {
        global ++;
        printf("global:%d,local:%d\n",global,local);
    }
    else if(pid ==0)
    {
        global++;
        local++;
        printf("global: %d, local: %d\n", global,local);
    }
    else
    {
        perror("fork()");
        exit(-1);
    }
    printf("\n");
    return 0;
}

```

----->> 이 예제를 보면 전역변수 global에 대해 부모프로세스와 자식프로세스는 별개의 것을 알 수 있다.

```

hyungjunyu@hyungjunyu-Z20NH-AS51B1U: ~/my_proj
hyungjunyu@hyungjunyu-Z20NH-AS51B1U:~/my_proj$ vi test2.c
hyungjunyu@hyungjunyu-Z20NH-AS51B1U:~/my_proj$ gcc test2.c
hyungjunyu@hyungjunyu-Z20NH-AS51B1U:~/my_proj$ ./a.out
global:101,local:10
global: 101, local: 11
hyungjunyu@hyungjunyu-Z20NH-AS51B1U:~/my_proj$

```

오늘 했던 내용을 정리해 보자. 일단 잘 모르지만 대충 이해해보자. ㅏㅏ

```

void term_status(int status)
{
    if(WIFEXITED(status))
        printf("(exit)status : 0x%xWn", WEXITSTATUS(status));
    else if(WTERMSIG(status))
        printf("(signal)status : 0x%x,%sWn",
                status & 0x7f,WCOREDUMP(status) ?
"core dumped":""); //코어덤프 이프로그램이 비정상종료됐을때 어느 메모
리상태에서 어떻게 죽었는지에대해 기록해야되는데 그
                걸할지말지를 지정해주는비트 1이 나오면 끝나는거고 0이
나오면 안끝나.
}

int main(void)
{
    pid_t pid;
    int status;

```

0326

```

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

```

*/

/* *****signal 예제 --> 어떤상황이 발생하냐에따라 그상황을 제어할수있다. SIGCHLD 걸리니까 my sig가 동작함. -> 비동기처리.

```

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

    wait(&status); //SIGCHLD 전달됨. SIGCHLD가 많으면 ->
while(wait(&status))로 해준다? (wait는 블로킹이다.) 자식이 많이생긴만큼 task_struct마니 생김. 메모리를 많이 차지한다. 결국 >시스템이 뻗는다. 자동재부팅됨. 논블로킹방식이 있으면 되겠죠? 그게바로
while(waitpid(    이게 wait의 논블로킹. 논블로킹방식이란 들어오는애들을 순번을줘서 예약제로 처리함.
// waitpid => -1이되면 어떤자식이든 처리하겠다.
//wnohang --> 즉시리턴. 종료된 자일드가 없으면.SIGCHL 없으면.
    term_status(status); //비정상종료인지 정상종료인지 보겠다.
--> 정상적으로 sleep했으므로 signal맞아 죽은게 아닌(자식은) exit에 걸린다.
}

```

```

int main(void)
{
    pid_t pid;
    int i;
    signal(SIGCHLD, my_sig); // 시그널의 2번째 인자는 함수(주소). 함수
    수포인터는 비동기. 인터럽트?. SIGCHLD가 오면 my_sig를 등록시켜! 메뉴얼
    같은거임. (어떤상황에서 어떤동작을 시킬지)
    if((pid = fork()) > 0) // 0보다 크게 부모 0은 자식.
    {
        for(i = 0; i < 10000; i++)
        {
            usleep(50000); // sleep은 초단위 usleep은 ms(마
            이크세컨)단위.  $10^{-6}$ . ms =  $10^{-3}$  --> 0.05초마다 숫자값을 뿌림
            그래서 100에서 시그널이 호출됨.
            printf("%d\n", i+1);
        }
    }
    // 자식 죽자마자 my_sig 호출.
    else if(pid == 0)
        sleep(5);
    else
    {
        perror("fork()");
        exit(-1);
    }
    return 0; // 해당 프로세스는 정상종료되었다.
}

*/
-----
-----
-----

/* *****execve -> 명령을 실행시키는거. ps -ef 하는거랑 결과가
같은거임. 둔갑술!

```

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

```

int main(void)
{
    execlp("ps", "ps", "-e", "-f", 0); // 첫번째는 프로그램이름. 다음부
    터 인자. 형식상 그렇게써야함. 0은 끝이라는거임
    printf("after\n"); // 왜 이걸 실행이 안될까? -> 메모리 레이아웃이
    ps로 바뀌기 때문에. a.out의 메모리 레이아웃이랑 ps의 메모리 레이
    아웃이랑 다르다. 둔갑술!
    실행시키려면 어케? fork를 쓰자. 밑에 예
    제.
    return 0;
}

```

```

}

int main(void)
{
    int status;
    pid_t pid;
    if((pid = fork()) > 0)// 1.fork를 하니까 자식과 부모과 생성
    됨.
    {
        wait(&status);//3.자식은 ps실행하자마자 죽으니까 wait
    로 처리하고 프롬트가 출력이된다.
        printf("prompt > Wn");
    }
    else if(pid == 0)
        execlp("ps", "ps", "-e", "-f", 0);//2.자식은 ps를 수행하
    고
    return 0;
}
*/

```

/* **newpgm.c

```

int main(int argc, char **argv)
{
    int i;
    for(i = 0; argv[i]; i++)
        printf("argv[%d] = [%s]Wn", i, argv[i]);
    return 0;
}

```

gcc -g -o newpgm 0326.c 하는 이유가 -o가 이름바꿔주는거임. 0326.c에있
는걸 실행파일 newpgm으로 생성.
그래서 밑에서 newpgm을 exelc 으로 실행시키면 돌아가는거임.

```

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;
}

```

```

*/

```

```

/*
int main(int argc , char **argv, char **envp)
{
    int i,j;
    for(i = 0; argv[i]; i++)

        printf("argv[%d] = [%s]\n",argv[i]);

    for (i= 0; envp[i]; i++)
        printf("envp[%d] = [%s]\n", envp[i]);

return 0;
}

```

```

-----
-----
-----

```

```

// *****좀더 어려운.

```

```

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 >");
    }
    else if(pid == 0)
        execl("./newpgm", *argv, *env);
    return 0;
}
*/

```

```

// **system
// 이건 어떻게 동작할까? -> system은 내부적으로 fork를 한다음에
exec을 하는거임.

```

```

/*
int main(void)

```



```

{
    sleep(1); //터미널을 끄면 프로세스 종료. 근데 이렇게 게임만들면 안
    되겠지? 저장이안되니.
    for(;;)
        system("date");
    printf("afterWn");

    return 0;
}
*/
/*
int my_system(char *cmd)//인자 문자열이니깐 포인터로 받는다
{
    pid_t pid;
    int status;
    char *argv[] = {"sh","-c",cmd,0};// sh는 셸을 만든다. -c 해당
    커맨드를 실행하라. date를 실행하라.
    char *envp[] = {0};
    if((pid = fork()) > 0)
        wait(&status); //종료코드를 받고 끝내냄.
    else if(pid == 0)
        execve("/bin/sh",argv,envp);

}

int main(void)
{
    my_system("date");
    printf("afterWn");
    return 0;
}
*/
-----
-----
-----
// *****Daemon process -> zombie process와 반대로 부모가
먼저 죽는다.
/*
int daemon_init(void)
{
    int i;
    if(fork() > 0)
        exit(0);
    setsid();//내 터미널이 프로세스와 생명을 같이한다. exit(0)으
    로 parent죽이고 이거실행하면 데몬이 된다.
    chdir("/");//??
    umask(0);//권한설정해준다는 뜻. 루트에있는 모든걸 사용하게 해
    주겠다 라는 뜻.

```

0326

```
for(i = 0; i < 64 ; i++)//0번이 입력 1번이 출력 3번ㅇ 이래 +
그이후 64개있음.
    close(i);// 데몬은 자식. 부모의 것들을 상속. 패륜저질
렀으니 연을 끊어함. 그래서 다 클로즈하는것.
    signal(SIGCHLD,SIG_IGN);//_IGN 자식이 죽던말던 신경안쓴다
(ignore). 그리고 parent까지 죽인다.
    return 0;
}
//게임서버 포털사이트 다 데몬으로 되어있다. 뭔가 차량에서 영상처리하
는 프로세스를 만들때 데몬으로 만듬.
// 프로세스로 만드는데순간 닫는순간 절벽돌진.
int main(void)
{
    daemon_init();
    for(;;)// ? 없어지느거 확인하려고 한거임
    ;
    return 0;
}
//ps -ef | grep a.out 하면 뭐가뜨는데 이게 무슨 명령어일까.
//데몬은 pts가 다 ?로 뜬다. 그리고 터미널 종료해도 죽지않고 계속 떠있
는다.

*/

int main(void)
{
    signal(SIGINT,SIG_IGN); // SIGINT는 컨트롤c.-> 컨트롤c를 무시
하겠다.
    signal(SIGQUIT,SIG_IGN);
    signal(SIGKILL,SIG_IGN);//SIGKILL은 신의철퇴. IGN로 못막음.
데몬 죽일수있는. kill -9. pid(프로세스 고유식별번호)
    pause();
    return 0;
}
//참고하자면 file은 하드디스크의 추상화고 process는 CPU의 추상화다.
```