

TI DSP, MCU 및 Xilinx Zynq FPGA

프로그래밍 전문가 과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 하성용
accept0108@naver.com

23 일차
리눅스 프로그래밍 6 일차

정상종료와 비정상종료 파악

wait() // 부모 프로세스가 자식 프로세스가 종료했음을 확인하는 함수
waitpid() // 인자로 프로세스 ID 를 받음으로써 특정 자식 프로세스의 종료를 기다릴 수 있다.

status // 프로세스의 상태를 가져오기 위해서 사용한다.

WIFEXITED // 자식 프로세스가 정상 종료

WEXITSTATUS // 자식 프로세스가 정상 종료일 때 하위 8 비트 값

exit()를 호출하기 위한 인자나 return 값이 설정되고
종료된 자식의 반환 코드의 최하위 8 비트를 평가한다.
이 매크로는 정상종료 일때만 평가된다.

WIFSIGNALED // 자식 프로세스가 비정상 종료

WTERMSIG // 시그널에 의해 종료했을 때 시그널 번호

자식프로세스를 종료하도록한 신호의 번호를 반환한다.
당연히 WIFSIGNALED 가 non_zero 일 경우에만 사용할수 있다.

WCOREDUMP // 코어 파일 발생 여부

term_status1.c

```
#include<unistd.h>
#include<stdio.h>
#include<errno.h> // 오류번호와 일치하는 오류 메시지 문자열
#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" : ""); //맨앞이 코어덤프, 코어덤프는 프로그램이 시그널에 맞아 죽게되서 비정상적으로 종
료되었을때 이 프로그램에 대한 덤프정보를 0 이면 안주고 1 이면 줌
}

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;
    }
}

```

(signal)status : 0x6, core dumped

```

wait10.c
#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); //자식프로세스가 죽어서 신호보내는걸 기다려주는것
    term_status(status);
}

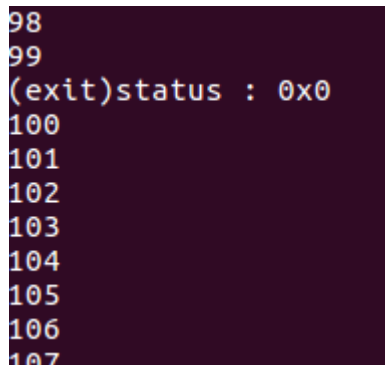
int main(void)
{
    pid_t pid;
    int i;
    signal(SIGCHLD, my_sig); // 죽으면 시그차일로 my_sig 를받아 실행
    if((pid=fork())>0)
        for(i=0; i<10000; i++)
        {
            usleep(50000); //0.05 초단위
            printf("%d\n",i+1);
        }
    else if(pid==0)
        sleep(5); //1 초단위의 메시지 출력하기까지의 지연시간
    else
    {

```

```

        perror("fork() ");
        exit(-1);
    }
    return 0;
}

```



```

98
99
(exit)status : 0x0
100
101
102
103
104
105
106
107

```

wait11.c

execve.c //무언가를 실행시키는것

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

```

int main(void)
{
    execlp("ps","ps","-e","-f",0);
    return 0;
}

```

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

```

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

```

메인프로세스가 돌아다 execlp 을 만나자마자 ps 프로세스로 갈아타고
printf 패스하고 쪽돔

fork 를 두고 동작하면 에프터 출력됨

fork 와 execve 의 합성

execve2.c

```

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

```

```

int main(void)
{
    int status;
    pid_t pid;
    if((pid=fork())>0)

```

```

    {
        wait(&status);
        printf("prompt >");
    }
    else if(pid==0)
        execlp("ps",argv[1],argv[2],argv[3],NULL);
    return 0;
}

```

execve3.c

```

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

int main(void)
{
    int status;
    pid_t pid;
    if((pid=fork()) > 0 ) //fork 가 자식프로세스를 만들어서 프로세스아이드를 반환,
    부모는 자식이 있어서 0 이 아님
    {
        wait(&status);
        printf("prompt > ");
    }
    else if(pid==0) //자식이 죽으면
        execlp("ps", "ps", "-e", "-f", 0);
    return 0;
}

```

// 만든 프로그램들을 동시다발적으로 동작시키겠다

newpgm.c

```

#include<stdio.h>

int main(int argc, char **argv) // argc:명령행옵션의 개수저장되는곳, argv:명령행 옵션의 문자
열들이 실제로 저장되는 배열
{
    int i;
    for(i=0; argv[i]; i++)
        printf("argv[%d]=[%s]\n",i,argv[i]);
    return 0;
}

```

new_a.c

```

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

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

```

```

        return 0;
    }

```

```

gcc -o newpgm newpgm.c //이름변경
gcc new_a.c //컴파일
./a.out //실행

```

```

argv[0]=[newpgm]
argv[1]=[one]
argv[2]=[two]
prompt>

```

실습 (힌트 : 환경변수, 이 파일이 아니라 newpgm 을 수정하라.)

execve5.c //실행이 가능하도록 만들어보라

```

#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;
}

```

답 :

```

#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;
}

```

execve6.c

```

#include<stdio.h>

```

```

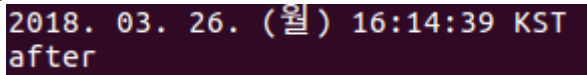
int main(void)
{
    system("date"); //시스템은 내부적으로 fork 를 하고 execve 을 한것
    printf("after\n");
}

```

```

        return 0;
    }

```



execve7.c //시스템 안에서 fork 를 하고 execve 을 한다는것

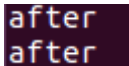
```

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

int my_system(char *cmd) //문자로넘겨서 포인터로 받았고 cmd
{
    pid_t pid;
    int status;
    char *argv[]={ "sh", "-c", cmd, 0}; //sh 는 쉘, 커널을까면 나오는게 쉘-c 는 해당커맨
드 실행하라는소리, 널문자로 종료
    char *envp[]={ 0};
    if((pid=fork())>0)
        wait(&status);
    else if(pid==0) //자식태어났으니 0
        execve("bin/sh",argv,envp); //자식프로세스가 이프로그램으로바뀜
}

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

```



Daemon Process

Zombie Process 와는 반대로 부모가 먼저 죽는 경우

execve9.c //데몬 만드는방법 아는게 중요

```

#include<sys/types.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); //부모 프로세스를 종료한다
    setsid(); //세션(tty)→즉, 데몬을 만든다
//setsid()를 사용하는 이유: fork 로 생성된 자식프로세스를 현재세션과 무관하게 동작시키기 위함이다
    chdir("/"); //root 라는 현재 작업 디렉토리를 변경
    umask(0); //현재있는디렉토리의 권한을 받음
}

```

즉, 루트에있는 모든권한을 사용할수있게해주는것

```
for(i=0; i<64; i++)
    close(i); //부모와 연이끊기면서 다 close
signal(SIGCHLD,SIG_IGN); /*
SIGCHLD: 자식프로세스들 중의 하나라도 종료되거나 멈출때마다 부모 프로세스에게 보내어진다,
IGN(이그노어):시그널이 무시되도록 정한다 */
return 0;
}
```

```
int main(void)
{
    daemon_init();
    for(;;)
    ;
    return 0;
}
```

실습 - 무한루프

execve6.c

#include<stdio.h>

```
int main(void)
{
    system("date"); //시스템은 내부적으로 fork 를 하고 execve 을 한것
    printf("after\n");
    return 0;
}
```

#include<stdio.h>

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

ctrl + z 멈춤

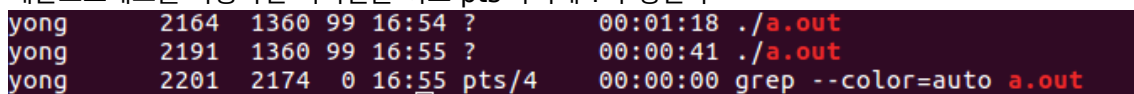
man waitpid

ps -ef

ps -ef |grep a.out //터미널 따로 켜져있는지 pts 죽은거 확인, pts ? 나오면 재부팅

서버를 만든다면 안정적으로 서비스를 제공하려면 이렇게 만들어서는 안됨
견고하게 만들기위해서 터미널을 꺼도 죽지않게하기위해 데몬프로세스사용

데몬프로세스를 사용하면 터미널을 꺼도 pts 자리에 ?가 생긴다



A terminal screenshot showing the output of the 'ps -ef' command. It lists three processes owned by 'yong'. The first two have a '?' in the 'TTY' column, while the third has 'pts/4'. The command 'grep --color=auto a.out' is shown at the bottom, highlighting the 'a.out' files in red.

USER	PID	PPID	C	ST	TTY	TIME	COMMAND
yong	2164	1360	99	16:54	?	00:01:18	./a.out
yong	2191	1360	99	16:55	?	00:00:41	./a.out
yong	2201	2174	0	16:55	pts/4	00:00:00	grep --color=auto a.out

두개 실행시키면 두개가 안죽는다

kill -2164 // 데몬프로세스 죽이는 명령어

#include<signal.h>

#include<stdio.h>


```
int main(void)
{
    signal(SIGINT,SIG_IGN); //SIGINT:프로세스에 인터럽트, 즉차단하라
                           //SIG_IGN: 시그널무시하라
    signal(SIGQUIT,SIG_IGN); //SIGQUIT: 코어 덤프를 남기고 프로세스 종료, 시그널 무시
    signal(SIGKILL,SIG_IGN); //SIGKILL: 프로세스 죽이기, 시그널 무시
    pause();
    return 0;
}
```

kill -9 2164 // 막을수없음

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

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

파일은 하드디스크의 추상화
프로세스는 cpu 의 추상화

tty //세션 ID