

***Xilinx Zynq FPGA, TI DSP, MCU 기
반의 프로그래밍 및 회로 설계 전문가
과정***

**<리눅스 시스템 프로그래밍>
2018.03.26 - 23 일차**

강사 - 이상훈
gcccompil3r@gmail.com

학생 - 안상재
sangjae2015@naver.com

1. 소스코드 분석

1) 자식 프로세스가 종료되었을 때 blocking 방식으로 처리함. (wait() 은 자식 프로세스가 종료 될때까지 기다림.)

-WIFEXITED(status) : 정상종료일 때 참

-WEXITSTATUS(status) : 정상종료일 때 status 를 하위 8 비트값으로 바꿔줌.

-WTERMSIG(status) : 시그널에 의해 종료되었을 때 시그널 번호.

-WCOREDUMP(status) : 코어 파일 발생 여부

```
#include <stdio.h>
#include <unistd.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)) // 정상 종료
        printf("(exit)status : 0x%x\n", WEXITSTATUS(status));
    else if(WTERMSIG(status)) // 비정상 종료(signal)
        printf("(signal)status : 0x%x, %s\n", status & 0x7f, WCOREDUMP(status)?"core dumped:");
    /* signal 맞아 죽으면 기록을 할지 말지를 결정하는 것이 coredump bit
        WCOREDUMP()가 1 이되면 core dump */
}

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

    if((pid = fork()) > 0)
    {
        wait(&status); // 자식 프로세스의 ID 를 반환받음.(6)
        term_status(status);
    }
    else if(pid == 0)
        abort(); // abort()에 의해 자식이 비정상 종료됨.(signal 6 번)
    else
    {
        perror("fork()");
        exit(-1);
    }

    return 0;
}
```

2) 자식 프로세스의 종료를 nonblocking 방식으로 처리함.

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <stdlib.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) // SIGCHLD 전달
{
    int status;
    wait(&status);
    term_status(status);
}

/* signal(int signum,함수 포인터) */
// 어떤상황에서 어떻게 해야할지 지침을 정하는 메뉴얼
// 어떤상황이 발생하느냐에 따라 그에 맞는 상황을 정의할 수 있음(비동기 처리)

int main(void)
{
    pid_t pid;
    int i;
    signal(SIGCHLD, my_sig); // SIGCHLD 가 들어오면 my_sig 를 동작시켜!
                             // (context-switching 을 할때, 자식 프로세스가 SIGCHLD 를 보내줌.)
                             // 행동지침을 정의함.
                             // 자식 죽으면 my_sig 호출

    if((pid = fork())>0)
        for(i=0;i<10000;i++)
        {
            usleep(50000); // 0.05 초에 한번씩 i 가 증가하면서 출력됨.
            printf("%d\n",i+1);
        }
    else if(pid == 0)
        sleep(5); // 자식 프로세스는 5 초동안 대기함.
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```

3) execlp 함수는 첫번째 인자의 파일로 메모리 레이아웃을 바꾸고, 해당 파일경로의 프로세스로 갈아탄다.(프로세스를 새로 만드는 것이 아님.)

- execlp([파일명], [argv[0]], [argv[1]], 0)

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

int main(void)
{
    execlp("ps","ps","-e","-f", 0);// execlp 는 프로세스를 새로 갈아탐.(둔갑술) 메모리 레이아웃을 "ps"로 바꿈.
    printf("after\n"); // 프로세스를 새로 만드는 것이 아니라 갈아타는 것이기 때문에 after 는 출력안됨.
    return 0;
}
```

4) 자식 프로세스가 ps -ef 를 실행하고 종료되면 부모프로세스가 자식 프로세스의 pid 값을 wait()를 통해 받고, “prompt >\n”를 출력함.

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

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

    if((pid = fork()) > 0)
    {
        wait(&status);
        printf("prompt >\n");
    }
    else if(pid == 0)
        execlp("ps","ps","-e","-f",0); // "ps" 메모리로 둔갑

    return 0;
}
```

5) execl 은 첫번째인자(경로) 에 대한 프로세스로 갈아탄다.

```
#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); // newpgm 파일의 경로로 들어가서 argv 와 env 를 인자
    }
}
```

로 취함.

```
    }
    return 0;
}

////////////////////////////////////
<newpgm.c 파일>

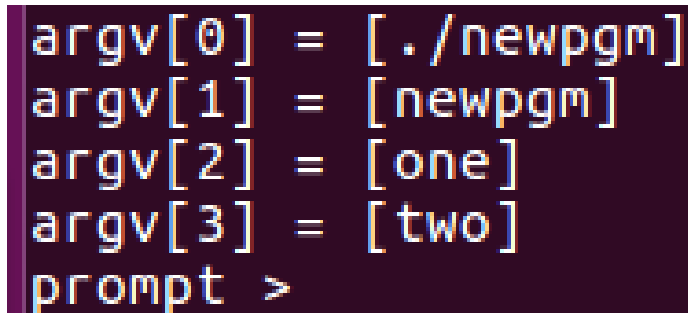
#include <stdio.h>

int main(int argc, char **argv, char **envp)    // envp 는 환경변수
{
    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;
}
```

5-1) 결과 사진



```
argv[0] = [ ./newpgm ]
argv[1] = [ newpgm ]
argv[2] = [ one ]
argv[3] = [ two ]
prompt >
```

6) system() 은 내부적으로 fork()를 한 다음 execve 를 수행함.

-system("date")는 날짜, 요일, 시간을 출력함.

-system() 시스템 콜을 구현함.

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

int my_system(char *cmd)    // system 안에서 fork 를 하고 execve 수행.
{
    pid_t pid;
    int status;
    char *argv[] = {"sh","-c",cmd,0};    // "sh"는 쉘, "-c" 해당 커맨드 실행 cmd 실행
    char *envp[] = {0};    //envp 는 환경변수

    if((pid = fork()) > 0)
        wait(&status);    // 자식이 종료될때까지 대기
    else if(pid == 0)
```

```

        execve("/bin/sh", argv, envp); “/bin/sh” 경로로 들어가서 argv, envp 를 인자로 전달한다.
    }

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

```

7) daemon process

* 조건

- 부모 프로세스가 종료되어야 함.
- 데몬 프로세스와 관련된 모든 파일이 닫혀있어야 함.
- 터미널을 종료해도 데몬 프로세스는 살아있음.

* 사용하는 이유

- 서버 같은 경우는 어떠한 경우에도 살아있어야 함.

```

#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) // 부모 프로세스는 바로 종료(폐륜)
        exit(0);

    setsid();      // 새로운 세션을 만드는 함수
    chdir("/");     // 디렉토리 루트로 이동.
    umask(0);      // 권한을 가짐.

    for(i=0;i<64;i++)
        close(i);    // 모든 파일과의 관계를 끊기 위해서 모두 닫음(데몬은 누구의 간섭도 안받음)

    signal(SIGCHLD,SIG_IGN); // 자식 프로세스가 종료되었을 때 SIGCHLD 가 오는 것을 무시
                             // 함.(자식이 죽던말던 신경 안씀.)

    return 0;
}

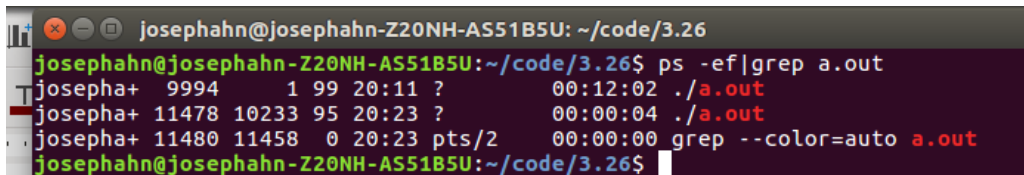
int main(void)
{
    daemon_init(); // daemon 프로세스를 초기화함.
    //sleep(20);
    while(1);      // daemon 프로세스가 종료되지 않게 함.

    return 0;
}

```

7-1) 결과 분석

- 결과 사진에서 '?' 표시가 있는 파일은 데몬파일을 뜻한다. 데몬 프로세스를 만드는 실행파일을 한번 실행할 때마다 데몬 프로세스가 1 개씩 만들어진다. 데몬 파일은 어떠한 명령어로도 제거되지 않으며, 오로지 kill -9 에 의해서만 제거가 된다. kill -9 [pid]를 실행하면 해당 pid 의 데몬 프로세스가 제거된다. Pkill -9 a.out 을 실행하면 모든 데몬프로세스가 제거된다.



```

josephahn@josephahn-Z20NH-AS51B5U: ~/code/3.26
josephahn@josephahn-Z20NH-AS51B5U:~/code/3.26$ ps -ef | grep a.out
josepha+ 9994      1 99 20:11 ?        00:12:02 ./a.out
josepha+ 11478 10233 95 20:23 ?        00:00:04 ./a.out
josepha+ 11480 11458  0 20:23 pts/2    00:00:00 grep --color=auto a.out
josephahn@josephahn-Z20NH-AS51B5U:~/code/3.26$
```

8) 데몬 프로세스는 kill -9 명령어로만 종료가 가능함.

```

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

int main(void)
{
    signal(SIGINT, SIG_IGN); // SIGINT = CCTL + 'C' (CCTL+'C' 을 씹는다.)
    signal(SIGQUIT, SIG_IGN); //
    signal(SIGKILL, SIG_IGN); // SIGKILL : 신의 철퇴(어떤 것으로도 못막음) KILL-9
    pause();

    return 0;
}

/*
SIGNAL 싹다 막아놓.
데몬보다 더 썸 것이 신이 날리는 SIGNAL 임. (kill -9 [pid])
*/
```