TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

1. 시스템 프로그래밍 – 5 파일입출력

```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<errno.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": "");// 0x7f core dump
}


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;
}
```

- Fork() – 자식 프로세스 생성
- Abort() signal로 자식 프로세스를 죽임.
- Signal 맞아 죽었기 때문에 term_status에서 status와 0x7f 비트로 core dump 확인하고 6을 출력

- Core는 프로그램의 비정상적인 종료가 발생하는 경우 커널에서 해당 프로세스와 관련된 메모리를 덤프시킨 파일을 말함.

- <core dump bit>
- 어느 메모리 상태에서 어떻게 죽었는지를
- 확인 할지 말지 지정해주는 비트가 코어 덤프
- 1 코어덤프 끈다.
- 0 코어덤프 안 끈다.

1. 시스템 프로그래밍 – 5 파일입출력

```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<errno.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": "");// 0x7f core dump
}

void my_sig(int signo){

    int status;  // sigchld 넘어옴.
    wait(&status);
    term_status(status);  // child 상태확인

}

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

    signal(SIGCHLD, my_sig);  // sigchld 날라오면 my_sig 동작시켜 ( signal - manual )
    //비동기 처리  -- 장확에 대한 대처를 하고 싶으면 다 signal로 등록!!

    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;
}
```

- Signal() 함수

Signal함수는 비동기 처리 방식으로

특정한 상황에 대처를 하고 싶을 때 사용된다.

그렇기 때문에 비상 대피가 필요할 때의 manual과

같은 역할을 수행한다.

1. 시스템 프로그래밍 – 5 파일입출력

```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<errno.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": "");// 0x7f core dump
}

void my_sig(int signo){

    int status;  // sigchld 넘어옴.
    //  wait(&status);  // wait blocking 함수
//  term_status(status);  // child 상태확인

    while(waitpid( -1, &status, WNOHANG) > 0)  // waitpid non-blocking
        term_status(status);                  // 대기를 주고 나중에 다 처리해 줌. call center

}

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

    signal(SIGCHLD, my_sig);  // sigchld 날라오면 my_sig 동작시켜 ( signal - manual )
    //비동기 처리  -- 상황에 대한 대처를 하고 싶으면 다 signal로 등록!!

    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;
}
```

- wait과 waitpid

자식 프로세스의 상태를 확인하는 wait함수는 blocking함수이기 때문에 자식 프로세스를 보고 있으면 다른 자식 프로세스는 볼 수가 없다.

반면, waitpid는 non-blocking함수이기 때문에 여러 자식의 상태도 모두 확인할 수 있다.

Waitpid는 콜센터가 굉장히 바쁠 때 연락처를 적어두고 다음에 연락을 주듯, 여러 프로세스들이 대기를 하고 순차적으로 처리를 하는 방식을 사용하여 모든 자식 프로세스의 상태를 확인해준다.

1. 시스템 프로그래밍 – 5 파일입출력

```c
#include<unistd.h>
#include<stdio.h>
int main(void){
        //name argument 1 2 3 , 0 = NULL 여기가 끝이다.
    execlp("ps", "ps", "-e", "-f", 0);//ps -ef와 같은 결과, 현재 사용중인 프로세스를 보여줌.
                    // 만든 프로그램 실행
    // execlp는 메모리 레이아웃을 ps로 바꾸기 때문에 after는 출력이 안됨.
    // 둔감술~!!!
    printf("after \n");
    return 0;
}
```

- Execlp 함수

Execlp는 내가 만든 실행파일을 실행시켜 준다.

하지만, 메모리 레이아웃을 내가 만든 실행 파일로 바꾸기 때문에 main에 아무리 다른 코드를 적어도 실행이 되지 않는다.

```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){
        execlp("ps", "ps", "-e", "-f", 0);
    }

    return 0;
}
```

- Fork & execlp

Fork로 자식 프로세스를 만들고 자식 프로세스에서는 내가 만든 실행 파일을 실행시키면

부모 프로세스에서는 내가 원하는 동작을 시킬 수 있다.

1. 시스템 프로그래밍 – 5 파일입출력

```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);
    }
    return 0;
}
```

- Execl()

Newpgm 실행 시키고 argv로 newpgm, one과 two를 인자로 취함. 0은 여기가 끝이라는 걸 표현.

```c
#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;
}
```

- Execve()

argv와 envp 두 인자를 모두 사용!

1. 시스템 프로그래밍 – 5 파일입출력

```c
#include <stdio.h>

int main(void){

    system("date");// fork후 exec
    printf("after\n");
    return 0;
}
```

- system()

Fork후 exec를 하는 함수.

```c
#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}; //shell, command, cmd
    char *envp[] = {0};

    if( (pid = fork()) > 0  ){
        wait(&status);
    }
    else if(pid == 0){
            execve("/bin/sh", argv, envp);  // shell 만들고 argv 실행

    }
}
int main(void){
    for(;;){
        sleep(1);
        my_system("date");
    }
    printf("after\n");
    return 0;
}
```

- system()함수를 fork로 구현하기

1. 시스템 프로그래밍 – 5 파일입출력

```c
#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);
    setsid(); // 프로세스 상태 ?로 만듬
    chdir("/"); // 실행 시켜야 할 파일이 있으니 root로 보냄
    umask(0); // 권한 설정
    for(i=0;i<64;i++)
        close(i); //  리눅스 기본 64개 파일 열으니 전부 닫음.
    signal(SIGCHLD,SIG_IGN);  // 자식이 생기면 무시하라 - signal 매뉴얼
    return 0;
}

int main(void){

    daemon_init();
    for(;;);
    return 0;
}
```

- Daemon()

데몬 함수는 서비스 프로그램을 만들려면 필수적으로 만들어야 하는 함수이다.

데몬 함수를 끄려면 kill -9를 사용하면 된다.