TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 최대성
c3d4s19@naver.com

2018.03.27. – 023일차 수업

동기적 일처리 방식

-> 순차적으로 일을 스스로 끝내나가는 방식

비동기적 일처리 방식

-> 해야할 일을 위임하고 기다리는 방식

<unistd.h> 헤더파일 내부 함수

WIFEXITED(status)
-> 자식 프로세스가 정상적으로 종료되었다면 TRUE

WEXITSTATUS(status)
-> 자식 프로세스가 정상 종료되었을 때 반환한 값

WTERMSIG(status)
-> 자식 프로세스를 종료하도록 한 신호의 번호를 반환

WIFSIGNALED(status)
-> 자식 프로세스가 시그널에 의해 종료되었다면 TRUE

WIFSTOPPED(status)
-> 자식 프로세스가 중단되었다면 TRUE

WCOREDUMP(status)
-> 코어덤프 파일이 있으면 TRUE

[코어덤프: 프로그램이 Signal을 받아서 비정상 종료되면 어떻게 종료되었는지 알려주는 파일]

signal([신호], [함수 포인터])
-> [신호]가 들어오면 [함수 포인터]의 함수를 실행시킨다

wait(&status) <Blocking 함수>
-> 에러일 경우 -1, 정상종료일 경우 0, 비정상 종료일 경우 종료된 자식의 프로세스 ID 반환

waitpid([-1], [&status], [WNOHANG]) <NON Blocking 함수>
-> wait()과 같은 기능
[-1]옵션: 어떠한 자식 프로세스든지 기다림
[WNOHANG]옵션: 어떠한 자식도 종료되지 않았어도 즉시 리턴

NON Blocking
-> 처리할 일이 많으면 예약해놓고 순차적으로 처리함

signal 함수 이용 예제

```c
#include <stdio.h>
#include <unistd.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))     //정상종료
                printf("(exit)status :
                0x%x₩n",WEXITSTATUS(status));
        else if(WTERMSIG(status))//비정상 종료
                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);
}

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;
}
```

---

* "exec" 계열의 함수 <unistd.h>

-> 현재 프로세스가 첫번째 인자의 프로그램 파일로 갈아타고 나머지 인자를 가지고 실행된다.

(메모리 레이아웃이 첫번째 인자의 프로그램 파일로 바뀜)0

#include<unistd.h>

int execl(const char *path, const char *arg0, ..., const char *argn, (char *)0);

path에 지정한 경로명의 파일을 실행하며 arg0~argn을 인자로 전달한다. 관례적으로 arg0에는 실행 파일명을 지정한다. execl함수의 마지막 인자로는 인자의 끝을 의미하는 NULL 포인터((char*)0)를 지정해야 한다. path에 지정하는 경로명은 절대 경로나 상대 경로 모두 사용할 수 있다.

int execv(const char *path, char *const argv[]);

path에 지정한 경로명에 있는 파일을 실행하며 argv를 인자로 전달한다. argv는 포인터 배열이다. 이 배열의 마지막에는 NULL 문자열을 저장해야 한다.

int execle(const char *path, const char *arg0, ..., const char *argn, (char *)0, char *const envp[]);

path에 지정한 경로명의 파일을 실행하며 arg0~argn과 envp를 인자로 전달한다. envp에는 새로운 환경변수를 설정할 수 있다. arg0~argn을 포인터로 지정하므로, 마지막 값은 NULL 포인터로 지정해야 한다. Envp는 포인터 배열이다. 이 배열의 마지막에는 NULL 문자열을 저장해야 한다.

int execve(const char *path, char *const argv[], char *const envp[]);

path에 지정한 경로명의 파일을 실행하며 argv, envp를 인자로 전달한다. argv와 envp는 포인터 배열이다. 이 배열의 마지막에는 NULL 문자열을 저장해야 한다.

int execlp(const char *file, const char *arg0, ..., const char *argn, (char *)0);

file에 지정한 파일을 실행하며 arg0~argn만 인자로 전달한다. 파일은 이 함수를 호출한 프로세스의 검색 경로(환경 변수 PATH에 정의된 경로)에서 찾는다. arg0~argn은 포인터로 지정한다. execl 함수의 마지막 인자는 NULL 포인터로 지정해야 한다.

int execvp(const char *file, char *const argv[]);

file에 지정한 파일을 실행하며 argv를 인자로 전달한다. argv는 포인터 배열이다. 이 배열의 마지막에는 NULL 문자열을 저장해야 한다.

위의 exec함수들은 exec에 l, v, e, p등의 알파벳이 붙어있다

l, v :
argv 인자를 넘겨줄 때 사용
(l 일 경우는 char *로 하나씩 v 일 경우에는 char *[]로 배열로 한번에 넘겨줌)

e : 환경변수를 넘겨줄 때 사용
(e 는 위에서 v 와 같이 char *[]로 배열로 넘겨줌)

p : p가 있는 경우에는 환경변수 PATH를 참조하기 때문에 절대 경로를 입력하지 않아도 됨

* "exec" 계열 함수 사용 예제

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

int main(){
        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;
}
```

---

* DAEMON 프로세스

-> (쉽게 종료되지 않는 프로세스)

DAEMON 프로세스 예제

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

int daemon_init(){
        int i;
        if(fork() > 0)
                exit(0); //부모 프로세스 제거
        setsid();
      //새로운 세션을 생성하고 그 세션의 리더가 됨
        chdir("/");
        //디렉토리 루트로 변경
        umask(0);
        //루트 권한 부여
        for(i = 0; i < 64; i++)
                close(i);
        //자신과 연관된 주변 모두 닫기
        //(리눅스에서 보통 64까지 있음)
        signal(SIGCHLD, SIG_IGN);
        //signal(SIGINT, SIG_IGN);
        //signal(SIGQUIT, SIG_IGN);
        //signal(SIGKILL, SIG_IGN);
        //시그널 무시
        return 0;

        //pause(); //시그널 받을때까지 대기
}

int main(){
        daemon_init();
        while(1){
                sleep(1);
                printf("daemon₩n");
        }
        return 0;
}
```

리눅스 명령어 ps -ef 에서 TTY(세션 id)가 ? 이면

DAEMON 프로세스이다 (소속이 없음)