# Xilinx Zynq FPGA,TI DSP,
# MCU 기반의
# 프로그래밍 전문가 과정

날 짜 : 2018 . 3. 22

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 정한별
hanbulkr@gmail.com

# <wait 마지막>

**시그널은 메뉴얼이다. → (함수 포인터, 비동기식, 인터럽트, 시스템콜)**

### signal(SIGCHLD , my_sig)

→ 자식 프로세스(SIGCHLD)가 signal 을 맞았을 때 , 혹은 정상종료 될때 my_signal 에서 어떻게 처리를 하겠다.
→ my_sig 라는 함수 포인터가 들어온다. void( *x)(int)

### wait()

1. child 가 시그널 상태를 보낼 때 까지 기다리는 함수이다.
2. wait 은 blocking 함수이다.
3 . wait 의 nonblocking 은 waitpid 이다.

### waitpid(-1 , &status, 0)

signal 이 온 데로 순서대로 저장하고 처리할 수 있을 때 하나씩 처리함.
-1 은 누가 와도 다 처리 하겠다는 뜻,
status 는 저장할 상태 값,
0 은 결과를 return 할 때 까지 블록한다는 뜻.

**파라미터**

| | |
|---|---|
| pid | wait할 자식 프로세스의 유형<br><br>-1보다 작음: 그룹ID가 절대값과 같은 차일드 프로세스를 waiting합니다.<br>-1 : 아무 자식 프로세스 ID라도 waiting합니다.<br>0 : 자신과 같은 프로세스 그룹ID를 가진 차일드 프로세스를 waiting합니다.<br>0보다 큼: 넘겨진 pid인 자식 프로세스만 waiting합니다. |
| status | return된 자식 프로세스의 상태 변경값을 저장하는 변수입니다. (ouput only)<br>이 상태값의 의미를 해석하는 여러가지 macro함수를 제공합니다.<br><br>**WIFEXITED**(status) : 만약 정상종료했다면. 즉, _exit(2), exit(3) 또는 main(3)함수 return으로 종료했다면.<br>**WEXITSTATUS**(status) : WIFEXITED(status)가 0이 아닌 값일 경우<br>_exit(2), exit(3)의 파라미터로 넘긴 값, 또는 main(3)함수의 return 값을 얻습니다.<br>exit()나 main의 return 값이 왜 필요한 지 아시겠지요?<br><br>**WIFSIGNALED**(status) : 만약 시그널이 발생하여 비정상적으로 종료했다면.<br>**WTERMSIG**(status) : WIFSIGNALED(status)가 0이 아닌 값일 경우, signal 번호를 얻습니다.<br>**WCOREDUMP**(status) : WIFSIGNALED(status)가 0이 아닌 값일 경우,<br>core file이 생성되었는 지 여부를 return합니다.<br><br>**WIFSTOPPED**(status) : 만약 stop signal이 발생하였다면. ptrace(2) 등으로 child 프로세스를 tracing중일 때.<br>**WSTOPSIG**(status) : WIFSTOPPED(status)가 0이 아닌 값일 경우, child process를 중지 시킨 signal 번호.<br>**WIFCONTINUED**(status) : STOP되었던, 자식 프로세스가 SIGCONT로 재 실행되었다면. |
| option | 0 또는 아래의 상수의 조합으로 설정됩니다.<br><br>**0** : 결과를 return할 때까지 block합니다.<br>**WNOHANG** : 현재 종료된 자식 프로세스가 없으면 block하지 않고 바로 0을 반환함.<br>**WUNTRACED** : 자식 프로세스가 STOP하면 반환함<br>**WCONTINUED** : STOP되었던 자식 프로세스가 재 실행되면 반환함. |

**RETURN**

| | |
|---|---|
| 0 | option이 WNOHANG인 경우 종료된 자식 프로세스가 없으면 바로 0을 return 함 |
| 양수값 | 상태가 변경된 자식 프로세스 ID.<br><br>파라미터 status는 return된 자식 프로세스 ID의 상태값을 저장합니다. |
| -1 | wait중 오류가 발생하였으며, 상세 오류내용은 errno 전역 변수에 설정됩니다.<br><br>**ECHILD** : 자식 프로세스가 없거나, 설정된 pid가 자식 프로세스가 아님.<br>**EINTR** : WNOHANG으로 설정되지 않고, SIGCHLD signal이나 signal이 발생함.<br>**EINVAL** : option값이 유효하지 않음. |

```
#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)
{
	// 정상종료이다. 정상 종료 했으니가 0 이 나오는데 return 이 0 이기 때문이다.
	if(WIFEXITED(status))
		printf("(exit)status :0x%x\n",WEXITSTATUS(status));
	// 시그널에 맞아서 죽었다.
	// 시그널에 맞아 죽으면 core domp 의 setting 에 따라서 시그널 에러 번호가 뜬다.
	// 어떤 상황에 따라서 어떻게 동작 시킬지 비동기 처리 할 수 있다.
	// 어떤 상황에 따른 처리를 하고 싶을 때는 시그널로 처리해야한다.
	else if(WTERMSIG(status))
		printf("(siganl)status : 0x%x,%s\n",
			status & 0x7f, WCOREDUMP(status) ? "core dumped" :"");
}

void my_sig(int signo)
{
	int status;
	// 차일드가 시그널 상태를 보낼 때 까지 기다리는 함수이다.
	// wait 는 blocking 이다.
	wait(&status);
	// term_status 에 의해서 어떻게 죽었는지 확인한다.
	term_status(status);
}

int main(void)
{

	pid_t pid;
	int i;
	// 자식 프로세스가 시그널을 맞으면(인터럽트) my_sig 함수를 호출하겠다.
	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;
}
```

## <execve> 실행 → 둔갑술

| #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. |

**1. l : argv 인자를 (char *)하나씩 넘겨줄 때 사용합니다. (마지막 NULL 문자를 인자로 가짐)**

**2. v : argv 인자를 배열 단위로 넘겨줄 때 사용합니다. (char *[]로  한번에 넘겨줌, 포인터 배열 끝에 NULL 포함 해야함)**

**3. e : 환경변수를 넘겨줄 때 사용합니다. (e 는 위에서 v 와 같이 char *[]로 배열로 넘겨줍니다.)**

**4. p : p 가 있는 경우에는 환경변수 PATH 를 참조하기 때문에 절대경로를 입력하지 않아도 됩니다.**

# execve1.c

```
include <stdio.h>

int main(void)
{
	int i;
	// ps 라는 파일 경로로 가서 argv[0] = “ps”, argv[1] = “-e”, argv[2] = “-f”, 끝은 l 이니 null 로 끝.
	execlp(“ps”, “ps”, “-e”, “-f”, 0);
	// 이 밑으로는 실행이 되지 않음.
	// 메모리 레이아웃이 ps 로 바뀐다. exec 명려어를 실행 했으니.. 다른 프로세스로  새로 바뀜.
	printf(“after\n”);
	scanf(“%d”, &i);
	printf(“%d\n”,i);
	return 0;
}
```

# execve2.c

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

// 위에 문제를 이렇게 해야 해결할 수 있음...
int main(int argc, char **argv)
{
	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;
}
```

# execve4.c

```
#include<unistd.h>
#include<stdio.h>
int main(void)
{
	int status;
	pid_t pid;
	if((pid= fork())>0)
	{
		wait(&status);
		printf("prompt >");
	}
```

```
        else if(pid == 0)
                // 명령어에 p 가 안들어 있으니 절대경로 들어가야 한다.
                execl("./newpgm", "newpgm","one","two", (char*)0);
        return 0;
}
```

## execve5.c

```
#include<stdio.h>
#include<unistd.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 에서 v 랑 e 가 들어갔으니.. 인자를 포인터 배열로 받고 환경변수도 받아야함.
                execve("./newpgm", argv , env);
        return 0;
}
```

## execve6.c

```
#include<stdio.h>

int main(void)
{
        for(;;){
                //시스템콜 date 명령어를 무한루프로 돌림. *내부적으로 fork()가 있음.
                system("date");
                sleep(1);
        }
        printf("after\n");
        return 0;
}
```

## execve7.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;
	// sh = 배쉬 쉘, -c 해당 커맨드를 실행해라, 커맨드 하나, 0 널문자로 종료)
	char *argv[]= {"sh", "-c", cmd,0};
	char *envp[]= {0};
	if((pid = fork())>0)
		wait(&status);
	else if(pid == 0)
		execve("/bin/sh",argv,envp);
}

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

## execve9.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)
		// 부모를 죽임
		exit(0);
	// 새로운 세션을 생성-> 인간에서 악마로 됨. damon 이 된다. TTY 가 ? 로 됨.
	setsid();
	// root 라는 디렉토리로 감.
	chdir("/");
	// root(현재 디렉토리)로 갔으니까 그 디렉토리의 권한을 다 주겠다. chmod 0777
	umask(0);
	// 기본적으로 리눅스에서 64 개의 기본 입력, 출력, 에러.... 해서 64 개를 켜줌. 그걸 다 꺼줌.
	for(i = 0; i<64;i++)
		close(i);
	// 자식이 죽어도 무시해라.
	signal(SIGCHLD, SIG_IGN);
	return 0;
}

int main(void)
{
```

```
        // 게임 서버, 포털 ,
        daemon_init();
        for(;;)
                ;
        return 0;
}
```

## execve10.c

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

int main(void)
{
        signal(SIGINT, SIG_IGN);
        signal(SIGQUIT, SIG_IGN);
        signal(SIGKILL, SIG_IGN);
        //시그널을 받을 때 까지 기다리는 함수
        pause();
        return 0;
}
```