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

강사 – Innova Lee( 이상훈 )
gcccompil3r@gmail.com
학생 – 윤연성
whatmatters@naver.com

Signal

Waitpid

```
#include <unistd.h>

int main(void)
{
	execlp("ps", "ps", "-e", "-f", 0);	//ps -ef 랑 같음
	return 0;
}
```

```
#include <unistd.h>

int main(void)
{
	execlp("ps", "ps", "-e", "-f", 0);   // 형식상 ps 두개가 써저야댐
	printf("after\n");                   //after 은 왜 출력이 안될까~? 메모리 레이아웃(이 바뀜
                                             //ps 로 바뀌면서 에프터가 없어짐
                                             //포크를 쓰고 이그젝을 하면 둘다 할수있음
	return 0;
}
```

-o <file> : 지정한 <file>로 출력 파일을 만든다.
이 옵션을 사용하지 않으면 기본으로 a.out 실행 파일이 생성된다.

```
#include <stdio.h>

int main(int argc, char **argv)
{
	int i;
	for(i = 0; argv[i]; i++)
		printf("argv[%d] = [%s]\n", i, argv[i]);
	return 0;
}
```

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

```
int main(void)
{
        int status;
        pid_t pid;
        if((pid = fork()) >0)           //패런츠
        {
                wait(&status);
                printf("prpmpt >");
        }
        else if(pid == 0)               //자식
                execl("./newpgm", "newpgm", "one", "two", (char *)0);  //자식을 이그젝해서
                                        // newpgm 으로 만듬 인자로 newpgm  one two 들어옴
        return 0;               //끝나면 자식이 죽으니까 위의 wait 에서 처리
}
```

데몬 프로세스

부모가 먼저 죽는경우

서버는 터미널을 끄던 안끄던 살아있어야 된다 근데 터미널을 끄니까 꺼져버림

근데  데몬을쓰면 터미널을꺼도 살아있음~

```
#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)  //0 보다 크니까 패런츠
            exit(0);                    //패런츠를 죽임
      setsid();               // 터미널과 프로세스가  생명을 같이함
      chdir("/");                   //위치를 루트로 바꿔줌
      umask(0);                     //루트에 있는걸 모든걸 사용할수있게 허용해준다(설정하는거)
      for(i = 0; i<64; i++)
      close(i);                     //close 하는 이유는 데몬은 자식이기에 상속을받는데
                                    // 패런츠가 죽었으니까 닫아버림 누군가가 간섭을할수없음
      signal(SIGCHLD, SIG_IGN);           //signal 은 동작지침을 설정 (시그널이 오면 자식이 죽던말던
아무상관안함)
      return 0;
}
```

```
int main(void)
{
	daemon_init();
	for(;;)			//무한루프해야 확인가능
	sleep(20);

	return 0;
}
~
```

---

./a.out 하고
ps -ef |grep a.out  // 이걸로 확인할수있음

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

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

ys@ys-Z20NH-AS51B5U:~/my_proj/23day$ **ps -ef |grep a.out**

```
ys      27299  1648  0 16:56 ?        00:00:00 ./a.out
ys      27489 27329  0 17:21 pts/2    00:00:00 ./a.out
ys      27557  1648  0 17:25 ?        00:00:00 ./a.out
ys      27574 27561  0 17:25 pts/19   00:00:00 grep --color=auto a.out
```

(데몬이 살아있음)  ? 보면 알수있음

```
ys@ys-Z20NH-AS51B5U:~/my_proj/23day$ kill -9 27299
ys@ys-Z20NH-AS51B5U:~/my_proj/23day$ kill -9 27557
```

**(kill -9 xxx 하면 데몬도 죽임 (프로세스 자체를 죽임)**

```
ys@ys-Z20NH-AS51B5U:~/my_proj/23day$ ps -ef |grep a.out
ys      27489 27329  0 17:21 pts/2    00:00:00 ./a.out
ys      27585 27561  0 17:26 pts/19   00:00:00 grep --color=auto a.out
```

프로세스 꺼진거 확인

파일을 하드디스크의 추상화
프로세스가 cpu 의 추상화

pid = -1 : 부모의 모든 자식 프로세스를 나타낸다.
pid > 0 : pid 와 같은 자식 프로세스를 나타낸다.
pid < -1 : 부모의 그룹 id 와 같은 프로세스를 나타낸다.
pid == 0 : 해당 pid 의 그룹 id 과 같은 프로세스를 나타낸다.

**WNOHANG : 자식 프로세스가 없을 경우 바로 리턴**
**WUNTRACED : 자식 프로세스가 멈추 었다면 리턴**
**WCONTINUED : (커널 2.6.10 이상에서) SIGCONT 신호를 받고 멈추었던 자식 프로세스가 동작을 재개 할 때 리턴**

waitpid 의 리턴은 다음과 같다.
**1. 성공할 때 해당 자식 프로세스의 pid**
**2. WNOHANG 이 세팅되고 여러개의 자식 프로세스가 있을 경우, 자식이 멈추 었을 때 0 을 리턴**
**3. 에러 났을 경우 -1**

exec - 부모프로세스를 자식 프로세스로 대체

wait - 프로세스 동기화 (자식 프로세스가 끝날때 까지 기다리고 자식이 마치면 부모가 재개한다
wait(&status) -------- int status;
**pid = wait(&status); // 자식끝날때까지 대기**

exit - 프로세스 종료 (정상종료 exit(0);)

exec 의 관련 함수들

**execl – l** 의인자는 열거하는 방식이 **나열형**
excel(“bin/ls”, “ls”, “-l”, (char *)0);
**execv - v** 의 인자는 열거하는 방식이 **배열형**
char * argv[] = {“ls”, “-l”, (char*)0} execv(“/bin/ls”, argv);
**execlp -p** 의 인자는 실행파일의 이름만 지정 (경로를 알필요없음)
p 의 인자가없으면 경로를 알아야 한다
execlp(“**cal”, “cal**”, “12”, “2016”, (char *)0);
execlp**(“ls”, “ls**”, “-l”, (char *)0);

setsid() // 터미널과 프로세스가 생명을 같이함