TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/3/26
수업일수	23 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. wait( )

1.함수

wait () : 프로세스의 동기화를 위해 다른 프로세스가 끝날 때 까지 프로세스를 멈추로록 만드는 함수이다.
wait pid( ) : 특정 자식 프로세스를 기다리기 위해 사용한다.

-자식 프로세스의 상태를 확인하는 매크로들
WIFEXITED(status) : 자식이 정상적으로 종료되면 true 를 반환하는 매크로, 0이 아닌 값을 리턴.
WEXITSTATUS(status) : 자식의 종료 상태를 반환하는 매크로이다. exit()에서 ()안의 값(인자)로 주는 값을 말한다.
WTERMSIG(status) : 매크로가 참일 경우 자식 프로세스를 종료시킨 비정상 종료 이유를 구할 수 있음. 시그널 번호를 얻는다.
WIFSTOPPED(status) : 이 매크로가 참이면 자식 프로세스는 멈춰있는 상태로 자식 프로세스 상태를 알 수 있음.
WSTOPSIG(status) : 매크로가 참일 경우 자식 프로세스를 멈춤상태로 만든 시그널 번호를 얻는다.
WCOREDUMP(status) : 시스템에 따라서 WIFSIGNALED(status)가 참일 경우 자식 프로세스가 core 덤프 파일을 생성했는지를 확인하는 이 매크로를 제공해준다.

-waitpid( )에서 사용하는 pid 인자 값의 의미
pid < -1 pid 의 절대값과 동일한 프로세스 그룹 ID 의 모든 자식 프로세스의 종료를 기다린다.
Pid == -1 모든 자식프로세스의 종료를 기다린다.
Pid == 0 현재 프로세스의 프로세스 그룹 ID 를 가지는 모든 자식 프로세스의 종료를 기다린다.
Pid>0 pid 값에 해당하는 프로세스 ID 를 가진 자식 프로세스의 종료를 기다림

-waitpid()에서 사용하는 option 인자
WNOHANG : waitpid()를 실행했을 때, 자식 프로세스가 종료되어 있지 않으면 블록상태가 되지 않고 바로 리턴하게 해준다.
WUNTRACED : pid 에 해당하는 자식 프로세스가 멈출 상태일 경우 그 상태를 리턴한다. 프로세스 종료 뿐만 아니라 프로세스의 멈춤 상태도 찾아낸다.

2.wait( )

~소스코드

```
#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))//자식 프로세스가 정상적으로 종료되면
                printf("(exit)status : 0x%x₩n",WEXITSTATUS(status)); //16진수로 자식의 종료상태 출력
        else if(WTERMSIG(status)) //자식 프로세스가 비정상적으로 종료되면
                printf("(signal)status : 0x%x,%s₩n",status&0x7f,WCOREDUMP(status)?"core dumped":" ");
                        //16진수로 덤프파일을 출력한다.
}

int main(void)
{
        pid_t pid;
        int status;
        if((pid=fork()>0)) //parent
        {
                wait(&status);
                term_status(status);
        }
        else if(pid==0) //child
                abort();
```

```
        else
        {
                perror("fork()");
                exit(-1);
        }
        return 0;
}
```

~ 결과

(signal)status : 0x6,core dumped

2. signal( )

1.signal( )

→ system call

비동기 처리일 때 행동지침을 정해놓는다. 어떤 시그널이 발생하면 기존 방법으로 처리할지, 무시할지, 프로그램에서 직접 처리할지를 설정할 수 있다.

헤더 : #incldue<signal.h>

형태 : void (*signal (int signum,void(*handler)(int)))(int);

설정 옵션

SIG_DFL : 기존방법

SIG_IGN : 시그널 무시

함수이름 : 시그널이 발생하면 지정된 함수를 호출한다.

~소스코드

```
#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)) //자식 프로세스가 정상종료이면
                printf("(exit)status : 0x%x₩n",WEXITSTATUS(status)); // 자식의 종료상태 출력
        else if(WTERMSIG(status)) // 자식 프로세스가 비정상 종료이면
                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(void)
{
	pid_t pid;
	int i;
	signal(SIGCHLD,my_sig); //자식이 죽으면 my_sig 를 호출한다.
	if((pid=fork()>0)) //parent
	{
		for(i=0;i<100000;i++)
		{
		usleep(500000);//0.5초동안 잔다
		printf("%d₩n",i+1);
		}
	}
	else if(pid==0) //child
		sleep(5);//5초동안 잔다
			//자식 프로세스는 정상종료여서 리턴 0값 나오게 된다.
	else
	{
		perror("fork()");
		exit(-1);
	}
	return 0;
}
```

~결과
1
2
3
4
5
6
7
8
9
(exit)status : 0x0
10

2. ps -ef 구현

- execlp()
int execlp(const char *filename, const char *arg, ...);
filename : 실행파일명
arg :실행파일과 같이 수행할 인자

- execve()
형태 : execve(const char *path, char *const argv[], char envp[]);
path : 실행할 파일 경로명
argv[] : 실행파일과 같이 수행할 인자
envp[] : 실행파일에 넘기는 환경 변수

- execle()
형태 : execle(const char *path, char *const arg,..., char * const envp[]);
path : 실행할 파일 경로명
arg : 실행파일과 같이 수행할 인자들
envp[] : 실행파일이 수행되는 환경

~소스코드

```
#include<unistd.h>

int main(void)
{
execlp("ps","ps","-e","-f",0);//프로그램 이름 ps, 인자 ps ...  , 0
                                //0→ 여기가 끝이라는걸 알려줌
return 0;
}
```

3. ps 구현 응용

~소스코드

```
#include<unistd.h>

int main(void)
{
execlp("ps","ps","-e","-f",0);//프로그램 이름 ps,인자 ps ...  , 0->여기가 끝이라는걸 알려줌
printf("after₩n");
return 0;
}
```

~결과

after 가 출력되지 않는다.

fork() → 분신술

execlp → 둔갑술, 현재 프로세스를 대체한다. 메모리를 ps 로 바꾸니까. 메모리 레이아웃이 바뀜.

현재 프로세스를 ps 로 대체해서 execlp 아래의 printf 명령어가 실행되지 않는다.

4.

newpgm.c

```
#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]); //인자로 받은 것을 출력한다.
	return 0;
}
```

4.c

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

int main(void)
{
	int status;
	pid_t pid;
	if((pid=fork())>0)//parent
	{
		wait(&status);
		printf("prompt > ");
	}
	else if(pid==0) //child
		execl("./newpgm","newpgm","one","two",(char*)0);//내가만든 newpgm 을 실행한다.
	return 0;
}
```

~결과
argv[0]=[newpgm]
argv[1]=[one]
argv[2]=[two]
prompt >

5.
argc : 인자의 갯수
argv : 문자열의 배열
envp : 시스템 환경 변수

-newpgm.c

```
#include<stdio.h>

int main(int argc,char **argv,char **envp) // 인자로 배열 2개를 받음
{
        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.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("./newpgm",argv,env);
	}
	return 0;
}
```

~결과

```
argv[0]=[./newpgm]
argv[1]=[newpgm]
argv[2]=[one]
argv[3]=[two]
envp[0]=[name=Os_Hacker]
envp[1]=[age=20]
prompt >
```

3. system

system 은 내부적으로 fork 를 함

1.

~소스코드

```
#include<stdio.h>

int main(void)
{
system("date"); //날짜,시간 등을 출력한다.
                        //ls 를 입력하면 ls 실행하게 된다.
printf("after₩n");
return 0;
}
```

~결과

```
2018. 03. 26. (월) 16:14:16 KST
after
```

2.daemon process

일반 프로세스는 터미널을 끄면 죽는데 daemon 프로세스는 터미널을 꺼도 계속 실행된다.

데몬 프로세스는 ? 로 뜬다. (빨간색 부분)

~소스코드

```
#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)//parent
                exit(0);//종료시킴
        setsid(); //프로세스가 터미널과 생명을 같이한다. 터미널을 끄면 프로세스도 꺼짐
                  //소속이 없어짐. parent 을 죽이고 setsid 를 하면 '?' 가됨
        chdir("/"); //위치를 /로 바꿔줌 이유잘모르겟다...
        umask(0);//root 에 있는 모든것을 사용할 수 있게 해준다.
        for(i=0;i<64;i++)
                close(i);//하는 이유 : 기본적으로 리눅스는 64개를 열어놓는데 다 닫음
                        //입출력,에러를 닫아서 간섭못하게함
        signal(SIGCHLD,SIG_IGN);//자식이오면 시그널을  무시한다
        return 0;
}

int main(void)
{
        daemon_init();//데몬프로세스 만들기
```

```
        for(;;)//서비스 프로그램을 하기 위해 for 문 안에 소스코드 입력하면 됨
                sleep(20);
        return 0;
}
```

~결과

-일반 프로세스

터미널 끄니까 프로세스는 꺼지는 것을 확인

```
프로세스 끄기 전
xeno@xeno-NH:~/proj/0326$ ps -ef | grep a.out
xeno        1988   1927   0 16:51 pts/7      00:00:00 ./a.out
xeno        2186   2104   0 16:51 pts/4      00:00:00 grep --color=auto a.out
```

```
프로세스 끈 후
xeno@xeno-NH:~/proj/0326$ ps -ef | grep a.out
xeno        2209   2104   0 16:52 pts/4      00:00:00 grep --color=auto a.out
// pts/7이 사라짐 → 프로세스가 꺼졌다.
```

-daemon process

```
프로세스 끄기 전
xeno@xeno-NH:~$ ps -ef | grep a.out
xeno        2388   1365   0 16:54 ?          00:00:00 ./a.out
xeno        2421   2409   0 16:54 pts/4      00:00:00 grep --color=auto a.out
```

```
프로세스 끈 후
xeno@xeno-NH:~$ ps -ef | grep a.out
xeno        2388   1365   0 16:54 ?          00:00:00 ./a.out
xeno        2421   2409   0 16:54 pts/4      00:00:00 grep --color=auto a.out
```

3. 데몬 프로세스 응용

~소스코드

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

int main(void)
{
        signal(SIGINT,SIG_IGN);
        signal(SIGQUIT,SIG_IGN);
        signal(SIGKILL,SIG_IGN); //SIGKILL 은 못막는다.
        pause();
        return 0;
}
//ctrl+c 를 무시한다.
Ctrl + z 도 막으면 Ctrl+z 도 먹히지 않는다. 하지만 SiGKILL 은 막지 못한다.
```

~결과

```
xeno@xeno-NH:~/proj/0326$ vi 10.c
xeno@xeno-NH:~/proj/0326$ ps -ef | grep a.out
xeno        2388   1365   0 16:54 ?        00:00:00 ./a.out          //데몬 프로세스
xeno        2813   2784   0 17:25 pts/7    00:00:00 grep --color=auto a.out
xeno@xeno-NH:~/proj/0326$ kill -9 2388          //kill -9에 프로세스 아이디를 입력하면 프로세스 꺼짐

xeno@xeno-NH:~/proj/0326$ ps -ef | grep a.out
xeno        2818   2784   0 17:25 pts/7    00:00:00 grep --color=auto a.out
        //데몬 프로세스가 꺼진 것을 확인
```

파일 -디스크의 추상화

프로세스 -cpu 의 추상화

프로세스 pid 의 0,1과 파일 0,1이 다름