Xilinx Zynq FPGA, TI DSP, MCU 기반의
프로그래밍 및 회로 설계 전문가 과정
#22

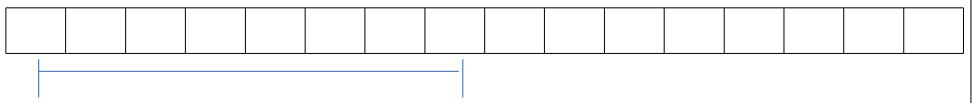
강사 : Innova Lee(이 상훈)
학생 : 김 시윤

배운내용 복

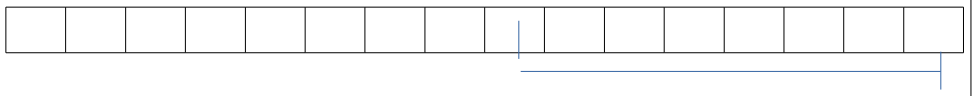
---wait---

```
#include <unistd.h>
#include <stdio.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))
        printf("(exit)status : 0x%x\n",WEXITSTATUS(status));//정상종료되었는지
        else if(WTERMSIG(status))//sig 를 맞아 종료되었는지
        printf("(signal)status : 0x%x, %s\n",status &
0x7f,WCOREDUMP(status) ?"core dumped" :"");
//signal 맞아 비정상종료 했는지.
//맨앞이 코어덤프 이 프로그램이 비정상적으로 종료되었을때 그 상태에 대한 정보 이 정보를 줄지 말지를 지정해주는게 코어덤프 1 이 셋팅 코어덤프 끈다. 0 이 나오면 코어덤프를 안끈다.
}

int main(void)
{
        pid_t pid;
        int status;
        if((pid = fork())>0)
        {
                wait(&status);//자식을 받은 어볼트에서 죽은 숫자값이 들어옴.
                term_status(status);
        }
        else if(pid == 0)
        abort();//signal 6 번
        else
        {
                perror("fork()");
                exit(-1);
        }
        return 0;
}
```

나름 COREDUMP 를 알아보았다.



정 상 종 료



비 정 상 종 료

비정상 종료했을때 coredump 를 쓸지 말지 결정하는 비트가 비정상종료 제일 앞 MSB 비트이다. 그래서 비정상종료의 리턴값을 알기 위해서는 &0x7f 를 해준다. 0x7f 를 하면 앞에 뺴고 다 1 이기 때문에 리턴해서 비정상종료 된 위치값을 알수있다.

--wait(signal ex)---

```
#include <unistd.h>
#include <stdio.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))
	printf("(exit)status : 0x%x\n",WEXITSTATUS(status));
	else if(WTERMSIG(status))
	printf("(signal)status : 0x%x,%s\n",status & 0x7f,WCOREDUMP(status)?"core
sumped":"");
}

void my_sig(int signo)
{
	int status;
	wait(&status);
	term_status(status);
}

int main(void)
{
	pid_t pid;
	int i;
	signal(SIGCHLD,my_sig);

//인자 int signum 와 함수에 주소가 들어간거 함수포인터는
//돌방상황이 발생하면 메뉴얼이 필요함 signal 이란것은 행동지침 등록해놓는거
//시그날 날라오면 마이시그 호출시켜  어떵상황에서 어떤 행동을 해야하는지 지침.등록
해놓고 자식만들어
	if((pid = fork())>0)
		for(i=0; i<10000; i++)
		{
			usleep(50000);
//자식이 만들어지자마자 5 초동안 자
		printf("%d\n",i+1);
		}
		else if(pid==0)
		sleep(5);
		else
		{
			perror("fork()");
			exit(-1);
		}

//sleep = 초단위 //usleep 은 마이크로 초
		//usleep 0.05 초 마다 숫자를 뿌려 슬립은 5 초 .
		// 100 에서 차일드 동작해
		//자식이 죽으면 부모로 가는데 부모가 돌고있어 자식이 죽으면 마이시그 호출
됨
		//그래서 위로 올라가.
		//위로 올라가니까 마이 시그가 보여
		//status 에 시그차일드 전달됨
		// 그 스테이터스를 비정상종료 했는지 정상종료했는지 본다.
		// 정상적으로 죽었으니 exit 에 걸림 정상종료
return 0;
}

//정상종료후 리턴 0 해서 정상종료임을 알림 메인에 리턴 0 이 있는이유 정상종료 했다는
뜻.
//어떤상황에 대한 대처를 하고싶을때 사용.
//wait = block
//wait 하는데 다른 신호 오면 블록 됨 자식이 죽었는데 수습이 안되 좀비 프로세스
//task_struct 가 포크할때마다 엄청나게 생겼어 좀비프로세스 , 이게 많다는건 메모리 많
이 차지한다는거 >이런일이 많이 일어나면 메모리 터져 nonblockin 이 필요함
```

---execlp(실행)---

```
//실행시키는거
// ps -ef 현재 구동중인 프로세스

#include <unistd.h>

int main(void)
{
	execlp("ps","ps","-e","-f",0);

//프로그램 이름 백터 0 1 2 0 은 끝.
	return 0;
}
```

---

execlp("실행시킬 프로그램 이름" ,argv0,argv1,argv2,0)
execlp 는 파일에 지정한 파일을 실행하며 argv 를 인자로 전달한다.
따라서 위 소스코드는 ps 를 실행하며 ps -e -f 를 인자로 전달한다.
따라서 ps -ef 와 동일한 결과를 얻는다.

---execve---

```
#include <unistd.h>
#include <stdio.h>
int main(void)
{

	execlp("ps","ps","-e","-f",0);
	printf("after\n");
	return 0;
}
//execve = 둔갑술
//fork 가 분신술이라면 얘는 둔갑술
//메모리레이아웃을 ps 로 바꾸면 ps 안에는 after 라는게 없음
//ps 로 메모리 레이아웃이 바뀌면서 밑에 프린트에프가 씹힘
//a.out 가 s-h-d-t , ps = s-h-d-t
//printf 나오고싶으면 포크를 해서 나오게 만들어야한다.
```

여기서 ps 라는 변수가 a.out 로 둔갑한다. 따라서 ps 에는 밑에 printf 가 존재하지 않는다.
만약 printf 를 출력하고 싶다면 fork 를 해 부모와 자녀를 분담 시켜줘야한다.

---execve&fork---

```
#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;
}
//자식이 ps 를 하고 죽고 wait 로 처리하고 프린트 해줌
```

---

fork 를 하여 프로세스가 부모랑 자식 두개의 프로세스가 된다. 부모는 자식이 죽을때까지 기다린다.
자식이 죽기전에는 동작을 하지 않는다. 그래서 wait 가 Blocking 이다.
자식은 execlp 를 하고 정상종료된다 그리고 부모가 prompt 를 프린트한다.

---execve&newpgm(execl)---

```
#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)
                //다른 여러 프로그램을 동시 다발적으로 돌린다.
                //자식을 뉴 피지엠으로 만들어서 뉴 피지엠을 구동시킴
                //뉴피지엠 원투
                execl("./newpgm","newpgm","one","two",(char *)0);
//캐릭터포인터 0 은 NULL 을 표현.
//newpgm 으로 둔갑 argv 에 newpgm,one,tow 입력
                return 0;
}
```

newpgm

```
#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;
}
```

---

execl 은 다른경로에 있는 파일을 실행시켜준다. 따라서 다른 포르세스인 newpgm 을 실행시켜
argv0 => newpgm , argv1 => one , argv2 => two 가 들어가게된다.

---execve&newpgm(execl)---

```
#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);
//함수라 생각하자 다름 프로세스의 메인함수로 접근하는 인>자들을 나열한거라 생각하면 쉽다.

        return 0;
}
```

newpgm

왼쪽의 뉴피지엠에서

int main(int argc , char **argv, char **env)
와 for(i=0;env[i];i++)를 추가시켜주면된다.
선생님께서 힌트를 많이주셔서 성공하였습니다. 감사합니다.

이유는 execl 이 newpgm 을 실행시키는데 거기 메인함수에는 env 라는 변수가 없기 때문에
함수에서 변수를 받을수 있게 설정해주었다.

---system(date)---

```
#include <stdio.h>

int main(void)
{
```

```
        while(1)
        system("date");//system 에 ls 입력하면 ls 동작
        //fork 를 한다음에  exect 를 한다
        printf("after\n");
        return 0;
}
```

system 에 날짜를 입력해서 날짜를 출력한다.
ls 를 입력하면 ls 를 출력한다.

이 소스코드는 fork 와 execve 를 같이 사용하면 구현 가능하다.

---Execve---

```
#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};//쉘에서 해단 커맨드 입력으로 들어온 데이트를 실행
        char *envp[] = {0};
        if((pid = fork())>0)
                wait(&status);
        else if(pid == 0)
                execve("/bin/sh",argv,envp);
}

int main(void)
{
        my_system("date");//인자로 문자열 넘김
        printf("after\n");
        return 0;
}
//여러개를 할라면 포문을 돌면서 포크 여러번 이그제트 여러번
```

==========

path 에 지정한 경로명의 파일을 실행하여 argv , envp 를 인자로 전달한다. 이 두개는 포인터 배열이고 포인터 배열의 마지막에는 NULL 을 지정해야 한다.
쉘이라는 경로명의 파일을 실행하여
/bin/sh -c date 를 실행하면 date 가 나온다 이 경로를 execve 를 이용하여 구현하였다.
여기서 envp 0 을 해준이유를 알아봤는데 execve 의 포맷이 절대경로 , argv,envp 이기때문에 우리는 envp 를 사용하지 않기 때문에 0 으로 NULL 값을 주고 집어넣은거 같다.

---daemon---

```
//daemon
#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);//부모를 먼저 죽였다 폐륜
        setsid();// setsid 일반프로세스 구동하면 pts 나오는데 가상터미널 세션아이디임 우리의 프로세스>가 터미널과 생명을 함께한다는 뜻 그래서 터미널 끄면 프로세스도 죽어 근데 이렇게 폐륜하고 셋에스아이디
 하면 소속이 없어짐. 셋에스아이디 하면 악마로 바뀌고 물음표 생김 여기까지 하면 데몬 된다는 소리.
        chdir("/");//체인지 디렉토리 체인지 디렉토리 하는이유는 어떤파일이든지 접근할 필요가 있을 수 >있기 때문에 위치를 루트로 바꿔준다.
        umask(0);//신경쓰지마 권한 다 설정한다는 뜻이야 루트에 있는 모든걸 다 사용하게 해줄게 유마스>크 0
        for(i=0;i<64;i++)//포문들면서 클로우즈 아이를 했어 클로우즈 하는 이유는 데몬은 자식이야 부모>의 것들을 상속받아 연을 끊으면서 다 클로우즈 하는거
                close(i);//0 표준입력 1 표준출력 기본적으로 리눅스는 64 개 열어놓는데 전부다 닫은거 입력
 출력 에러 없음
        signal(SIGCHLD,SIG_IGN);//데몬이 자식 프로세스를 만들수 있어 시그널은 어떠한 동작 지침을 설정
하는거 시그차일드가 오면 시그 이그노어 자식이 죽던말던 신경쓰지마
        return 0;
```

```
}

int main(void)
{
	daemon_init();
	for(;;);//데몬이닛이 끝난후 포문을 돌고있으니 안끝난다. 이 포문에서 계속 반복적으로 서비스
하>는 거 하면 서비스 되는거야
//ftp naver torrent google gameserver 이런거 같은거 다 데몬사용
//차량에서 영상처리 하는거 데몬으로 만들어야함!!!

		return 0;

}
//생명력이 뛰어나서 데몬
//일반 프로세스는 꺼버리면 죽는다.
```

---wait nonblock---

```
// wait -> waitpid
// 들어오는걸 예약시키고 순서대로 처리함
//게임서버에서 많이 일어남

wait 를 nonblock 으로 쓰고싶으면 wait pid 를 해주면 된다.
```

---signal GOD---

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

int main(void)
{
	signal(SIGINT,SIG_IGN);
	signal(SIGQUIT,SIG_IGN);
	signal(SIGKILL,SIG_IGN);
	pause();
	return 0;
}
//signal 을 싹다 막아놓으면 죽일수 있는 방법은?
```

```
//데몬이 동작할때 데몬을 없앨라는데 시그날을 없앨라면
//리눅스 세계에 신이 날리는 시그날이 있다.
//SIGKILL 은 막아놔도 막을수 없다. !! SIGKILL 은 신이다.
//이 킬을 어케 없애나
//킬 -9 하면된다.

KILL 은 신이라서 안없어진다 대단하다.
Kill -9 daemon-pid 를 입력하면 데몬이 꺼진다.
```