TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

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

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

int main(void)
{
	pid_t pid;		//pid_t = int 같은거
	pid = fork();			//자식의 pid 값을 반환
	if(pid > 0)
		printf("paren\n");		//0 보다 크면 페런츠
	else if(pid ==0)
		printf("child\n");		//0 이면 차일드
	else
	{
	perror("fork())");				//에러가 나면 출력 ,어떤 문제가 생기는지 에러메시지
					를 출력
	exit(-1);
	}
	return 0;
}



int main(void)
{
	pid_t pid;
	pid = fork();		//프로세스를 생성(자식프로세스)		fork()의 리턴은 자식의 pid
값
	if(pid > 0)				//0 보다 크면 부모, 부모는 자식이있으니까 pid 가 있음
		printf("parent : pid = %d, cpid = %d\n", getpid(), pid);		//getpid()하면 자기
자신의 pid 를 보여줌
	else if(pid == 0)		//자식프로세스 = 자식이없다  그래서 0
		printf("child : pid = %d, cpid = %d\n", getpid(), pid);			//

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

```
int main(void)                  //대소문자 나오는이유는 프로세스 두개가 돌고있음
{
        pid_t pid;
        int i;
        pid = fork();
        if(pid >0)

                while(1)
                {
                        for(i = 0; i< 26; i++)
                        {
                                printf("%c", i+'A');            //대문자 출력
                                fflush(stdout);
                        }
                }

        else if(pid == 0)
        {
                while(1)
                {
                        for(i =0; i<26; i++)
                        {
                                printf("%c", i+'a');            //소문자
                                fflush(stdout);
                        }
                }
        }
        else
        {
                perror("fork() ");
                exit(-1);
        }
        printf("\n");
        return 0;


}



#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

int global = 100;                               //전역변수로 선언

int main(void)
{
        int local = 10;
        pid_t pid;
```

```
        int i;
        pid = fork();           //부모랑 자식으로

        if(pid > 0)                     //부모
        {
                printf("global : %d, local : %d\n", global, local);     //왜 글로벌값은 갱신이안되지 ?  프로세스가 달라서임!
        }                                                               //이걸 C.O.W 라고 함 copy on write ( 메모리의 쓰기 작업이발생할때 복사함!)
        else if(pid == 0)               //자식                          //ex 텍스트를 복사(명령어를알기위해)하고
                                                                        // 전역변수를 사용하니까 그때 전역변수를 복사
        {
                global++;
                local++;
                printf("global : %d, local : %d\n", global, local);             //자식부터 시작되었다고 했을때
        }
        else
        {
                perror("fork() ");
                exit(-1);
        }
        printf("\n");
        return 0;
}



void recursive_dir(char *dname)   //. 점이 전달됨
{
      struct dirent *p;  //
      struct stat buf;
      DIR *dp;
      chdir(dname);                     //경로 디렉토리 위치를 바꾸는 함수
      dp = opendir(".");                //디렉토리에 대한 포인터를 얻음
      printf("\ts: \n", dname);
      while(p =readdir(dp))                     //리스트가 다 순회할 때 까지 리스트를 출력함
            printf("%s\n", p->d_name);
      rewinddir(dp);                    //되감기, 포인터를 맨앞으로 가져다 놓음
      while(p = readdir(dp))
      {
            stat(p->d_name, &buf);              //파일종류를 buf 에 저장
            if(S_ISDIR(buf.st_mode))
                  if(strcmp(p->d_name,".") && strcmp(p->d_name, ".."))          // .이랑 ..이면 들어가지 말아라~
                        recursive_dir(p->d_name);           //다른 디렉토리에 들어가고 읽을게 없으면 넘긴다
      }
```

```
        chdir("..");                    //끝나면 이전 디렉토리로 돌아감
        closedir(dp);
}

int main(int argc, char *argv[])
{
        recursive_dir(".");
        return 0;
}
~
```

./a.out 하면

---

ls -l 이랑 거의 비슷하게 작동

---

paging Mechanism

왜 페이징을 사용할지에 대해 (메모리를 많이 필요로하기 때문에) 대용량의 데이터를 쓰기위해

페이징을 사용하면 어떤 부분에서 이점을 얻을수 있을까?

Fork

복사 = 분신술

복사는 하되 pid 는 다름

what is process ?

IS process ? Shared VM???

gcc -o text

-o <file> : 지정한 <file>로 출력 파일을 만든다.

이 옵션을 사용하지 않으면 기본으로 a.out 실행 파일이 생성된다

```
 #include <stdio.h>

int main(void)
{
        int a = 10;
        printf("&a = %#p\n", &a);
        sleep(1000);
        return 0;
}
```

```
#include <stdio.h>

int main(void)
{
	int *p = 0x7ffc25a5bef4;
	printf("&a : %#p\n", *p);
	return 0;
}
```

gcc -o text ps_text1.c. 을 하면 ./a.out 를 안하고 실행파일이 text 로 바뀐다

./test &위와같이 수행한후 주소값을 ps_test2.c 의 *p 값에 대입
0x7ffc25a5bef4
맨 앞에 숫자가 7 이라 유저영역

Segmentation fault (core dumped) 이게 나옴 (정상임)
나는 이유는 프로세스가 달라서 그럼 프로세스가 만들어지는 순간 task_struct 가 만들어짐
우리가 본권한은 파일자체의 권한

IPC (인터 프로세스 커뮤니케이션) 자기들끼리 정보를 전달하기 위해 사용

디멘드 온 페이징 (당장 필요한것만 복사)

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)          //./a.out   하면 1792 여기서 256 으로 나누면 7 이게  exit(7)    8
{                       //8 비트 라서 256 을나누거나 8 비트를 쉬프트함 -- 프로세스는 시그널을 맞으면 죽게됨!
        pid_t pid;
        int status;
        if((pid =fork()) >0 )
        {

        wait(&status);
        printf("status :0x%xd\n", WEXITSTATUS(status));
        }
        else if(pid == 0)
                abort();
        else
        {
                perror("fork()");
                exit(-1);
        }
        return 0;
}
-------------------
0x0
------------------


#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)          //./a.out   하면 1792 여기서 256 으로 나누면 7 이게  exit(7)    8
{                       //8 비트 라서 256 을나누거나 8 비트를 쉬프트함 -- 프로세스는 시그널을 맞으면 죽게됨!
        pid_t pid;
        int status;
        if((pid =fork()) >0 )
        {




        wait(&status);
```

```
        printf("status :0x%x\n", WEXITSTATUS(status));
        printf("status :0x%x\n", status -128);                  // - 128 안했을때는 0x86 이 나
        printf("status :0x%x\n", status &0x7f);                         //0x6 나옴 7f 는 127
}
        else if(pid == 0)
                abort();
        else
        {
                perror("fork()");
                exit(-1);
        }
        return 0;
}
```

```
 status :0x0
status :0x6
```

 ----------옴

kill -l 은

리눅스 상의 시그널들을 볼수있음