Xilinx Zynq FPGA, TI DSP,MCU기반의 프로그래밍 및 회로 설계 전문가 과정

강사 - Innov (이상훈)

gcccompil3r@gmail.com

학생 - 이유성

dbtjd1102@naver.com

재귀함수를 이용한 ls-R구현 :현재 디렉토리부터 최하위 디렉토리까지 순회하면서 파일 출력

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<dirent.h>
#include<sys/stat.h>
#include<string.h>


void recursive_dir(char *dname);   //함수선언을 미리 하고  main함수 아래에 함수 내용을 서술하는 방식
int main(int argc, char **argv)
{
        recursive_dir(".");
        return 0;
}
void recursive_dir(char *dname)
{
        struct dirent *p; //디렉토리 포인터에 있는 리스트들(각종 파일들)
        struct stat buf; //파일 상태
        DIR *dp;
        chdir(dname); //디렉토리 포인터를 현재위치로 바꾸어주는 시스템 콜
        dp = opendir(dp))//리스트가 다 순회할 때까지 리스트를 출력함
         printf("%s\n",p->d_name);
        rewinddir(dp);// 되감기 포인터를 맨 앞으로 가져다 놓음

        while(p = readdir(dp))
        {
                stat(p->a_name,&buf); //st.mode를 얻어옴

                        if(S_ISDIR(buf.st_mode))
                                if(strcmp(p->d_name,".")&& strcmp(p->d_name,"..")) //".",".."재낌
                                        recursive_dir(p->d_name);
        }
        chdir("..");   //상위로 이동(처음에 들어갈때 보다 한단계 더 상위로 끝남
        closedir(dp);
}
```

1.ork()함수 // 프로세스 하나 더 만듬.

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

int main(void)
{

        printf("before\n");
        fork();



        printf("after\n");
        return 0;
}
```

//fork()실행하면 fork()함수 줄포함 이후 내용이 복사

2.

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

int main(void)
{
        pid_t pid;  // pid_t는  int같이 씀(pid_t프로세스 번호를 저장하는 타입)
        pid = fork();   //자식이 있을 경우 자식의 pid값 반환 자식 없을 경우 0 반환.
                        //복사되면 자식이 먼저 실행(일반적)

        if(pid >0)
                printf("parent\n");
        else if(pid==0)
                printf("child\n");
        else
        {
                perror("fork() "); //무슨일이 잘못 되었는지 출력하라고 하는것.
                exit(-1);
        }
        return 0;
}
```

// 자식 프로세스는 생성 즉시 곧바로 프로세스 생성 x

errno.h
error number 정적 메모리의 위치에 저장된 오류 코드를 통해 오류 상태를 보고 및 검색하기
위한 매크로를 정의한다
한 값(오류번호)은 애러 감지 했을 때 특정 라이브러리 함수에 의해 errno에 저장된다
프로그램 시작시 이값엔 0이 들어있다.

3.

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

int main(void)
{
        pid_t pid;
        pid = fork() ;              //parent는 0보다 클때  ,, child는 0일때,
        if(pid >0)
                printf("parent : pid = %d,cpid = %d\n",getpid(),pid);//(현재 자신의 프로세스
id값,자식의 프로세스 id값)
        else if(pid ==0)
                printf("child : pid = %d, cpid = %d\n",getpid(),pid); //pid =0인 이유는 자식이
없다.  반환값 성공할 경우 자식 프로세스의 PID가 부모에게 리턴되며 자식에게는 0이
리턴된다 실패할 경우 -1리턴, 적절한 errno값이 설정된다.
        else
        {
                perror("fork() ");
                exit(-1);
        }
        return 0;
}


// 부모의 cpid값과 자식 pid값이 같은 것을 확인 하면 됨.

//프로세스가 순차적으로 돌수도 있고 동시에 돌수도 있다.

//프로세스 만들어지면 task_struct가 만들어짐.

//프로세스 A B 가상메모리 공유할 수 없다.
//공유되게 -> 파이프 ,queue message
//A B나눈 이유는 작업을 효율적으로 분담하기 위해서 (특화된 작업을 위해)
```

4.운영체제가 멀티 테스킹 한다는 것처럼 보임 (프로세스 2개가 순차적으로 빠르게 실행 context switching이 이루어진다.)

```
#include<unistd.h>
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>
int main(void)
{
    pid_t pid;
        int i;
        pid = fork();
         if(pid > 0){
                while(1){
                        for(i = 0; i <26 ; i++){
                                printf("%c ",i+'A');
                                fflush(stdout);
                         }
                }
         }
         else if(pid ==0){
                while(1){
                          for(i=0 ; i<26 ; i++){
                                printf("%c ",i+'a');
                                fflush(stdout);
                         }
                }
         }
         else{
                perror("fork() ");
                 exit(-1);
         }
        printf("\n");
        return 0;
}
```

5. 프로세스들의 데이터 공유

file1.c

```
#include<stdio.h>

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

// gcc -o test 파일명

ps_test2.c

```
#include<stdio.h>
int main(void)
{
	int *p= (file1.c의 a의 주소) ;
	printf("&a : %#p\n", *p); //a의 주소
	return 0;
}
```

gcc ps_test2.c

./test &
여기서 나오는 주소값을 ps_test2.c 의 *p에 대입
./a.out 을 해서 확인.

프로세스들은 virtual memory(가상 메로리)를 공유하지 않는다.(각각의 프로세스는 다른 프로세스의 페이징에 대한 권한이 없다)
만약 접근하게 되면 segmentation fault가 뜸.

다른 프로세스에게 데이터를 주려면 파이프,message queue,shared momory(물리메모리)를
이용한다.

PIC :우선순위 인터럽트 제어기(작업을 분담할 때 정보를 전달하는 매커니즘)
프로세스를 왜 분할할까? . .메모리에 데이터를 전부 올릴 수 없기 때문에.

6.

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

int global = 100;

int main(void)

{
        int local = 10;
        pid_t pid;
        int i;
        pid =fork();//

        if(pid >0)
        {
                printf("global : %d,local : %d\n",global,local);
        }
        else if( pid ==0)       {
                global++;
                local++;
                printf("global : %d , local : %d\n",global,local);
        }
        else
        {
                perror("fork() ");
                exit(-1);
        }
        printf("\n");
        return 0;
}
```

글로벌 값이 갱신 되지 않는다.(프로세스가 다르기 때문)
C.O.W: 메모리에 쓸때 복사를 한다. fork()에 의해 자식 프로세스가 생성이 되면
부모 프로세스를 그대로 복사 하지만 데이터는 필요할때 복사한다(COW할때)
TEXT - STACT - DATA 순으로 복사

7. 파이프 파일을 만들고 멀티테스킹 확인

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

int main(void)
{
	int fd,ret;
	char buf[1024];
	pid_t pid;
	fd = open("myfifo",O_RDWR);
	if(pid = fork() >0)
	{
		for(;;)
		{
		ret = read(0,buf,sizeof(buf)); //blocking (키보드의 입력을 기다림)
		buf[ret] = 0;
		printf("Keyboard : %s\n",buf);
		}
	}
	else if(pid ==0)
	{
		for(;;)
		{
		ret = read(fd ,buf,sizeof(buf));//파이프의 입력을 기다림
		buf[ret] = 0;
		printf("myfifo : %s\n",buf);
		}
	}
	else
	{
		perror("fork() ");
		exit(-1);
	}
	close(fd);
	return 0;

}
 //블로킹함수지만 myfifo에서의 입력이 들어오면 자식 프로세스에서
//fd 만 보고 있기 때문에 논 블로킹 함수처럼 동작함
```

8.좀비 프로세스

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

//실질적으로 이러한 process를 zombie process라 한다.

int main(void)
{
        pid_t pid;
        if((pid = fork()) >0)
                sleep(1000);  //1000초 중지
        else if(pid ==0)
        ;          //자식 프로세스 종료
        else
        {
                perror("fork() ");
                exit(-1);
        }
        return 0;
}

//결과가 <defunct>라고 나오는 것이 좀비 프로세스다.
// 1000초 후 깨어나면 defunct가 없어짐
//자식 프로세스가 죽어 부모 프로세스가 처리해줘야하는데 자고 있어서 좀비 프로세스가 생성되었다.
```

9.wait()

```
#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)
{                               //자식 프로세스를 생성하면 자식프로세스의 pid값 자식없으면 0반환
        pid_t pid ;         //자식이 죽으면 시그널값을 반환 부모에게
        int status;
        if((pid = fork())>0)
        {
                wait(&status);  //자식의 시그널 값을 받을때까지 대기 .자식이 안죽으면 계속 대기 자식이 죽으면 시그널값을 status에 >저장
                printf("status : %d\n",status);
        }
        else if(pid ==0)
                exit(7);        // 정상적으로 죽음   /비정상종료는 시그먼테이션폴트 ,,무한루프 데이터를 너무 많이 먹으면 메모리 깨져 Kill 됨
        else
        {
                perror("fork() ");
                exit(-1);
        }
        return 0;
}


// 1792가 나온 이유?  7 256  exit 7 2^8 status(16비트 왼쪽 8비트 0 죽으면 비정상 종료 오른쪽 8비트 0  죽으면 정상종료)
// 프로세스는 시그널 맞으면 죽게 됨 (기본적으로)


wait()함수 자식프로세스가 종료될 때까지 기다린다
자식 프로세스가 종료되면 전체 16bit 레지스터 상위 8bit에 값을 종료된 값을 반환.
(sginal_struct에)
프로세스 앞 8비트에는 프로세스가 종료되게 한 exit
함수의 인수가 기록 된다. 뒤 8비트는 0이 저장
 정상종료 비정상종료
```

10. abort

```
#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)
{
	pid_t pid ;
	int status;
	if((pid = fork())>0)
	{
		wait(&status);
		printf("status : 0x%x\n",(status&0x7f));
	}
	else if(pid ==0)
		abort();//강제 종료 ->비정상 종료 0을 반환
	else
	{
		perror("fork() ");
		exit(-1);
	}
	return 0;

}
```

결과 status :0x0

프로세스 계통도

pptr부모 포인터
cptr자식 포인터
yptr 같은 부모로 부터 자기보다 먼저 생성된 프로세스에 대한 포인터
optr 같은 부모로 부터 자기보다 늦게 생성된 프로세스에 대한 포인터
부모 포인터는 자신의 자식중 제일 늦게 생성된 프로세스를 가르킨다.

Run Queue에는 실행 중인 프로세스
Wait Queue에는 실행을 위해 대기 중인 프로세스