Xilinx Zynq FPGA, TI DSP MCU 기반의

프로그래밍 및 회로 설계 전문가

강사 이상훈

(Innova Lee)

Gcccompil3r@gmail.com

학생 김민호

minking12@naver.com

```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<dirent.h>
#include<sys/stat.h>
#include<string.h>
#include<stdlib.h>
#include<errno.h>
#include<fcntl.h>
#include<sys/wait.h>
/*

void recursive_dir(char *name);// 이 함수를 쓰겠다고 미리 선언?해주는 개념, 프로토 타입 기술
int main(int argc,char *argv[])
{
    recursive_dir(".");
    return 0;
}

void recursive_dir(char *dname)
{
    struct dirent *p;//각종 파일들
    struct stat buf;
    DIR *dp;
    chdir(dname);//. 이 저장(포인터를 현재위치로)
    dp=opendir(".");
    printf("\t%s:\n",dname);
    while(p=readdir(dp))//모두 확인
        printf("%s\n",p->d_name);
    rewinddir(dp);//되감기 pos 초기 위치로 돌아감
    while(p=readdir(dp))
    {
        stat(p->d_name,&buf);//파일 종류를 알수있음.
        if(S_ISDIR(buf.st_mode))
            if(strcmp(p->d_name,".")&&strcmp(p->d_name,".."))// . 과 .. 은 생
                recursive_dir(p->d_name);
    }
    chdir("..");//상위 폴더로 이동
    closedir(dp);
}
*/
/*
int main()
{
    printf("before\n");
    fork();//자기 밑에줄부터 복사.
    printf("after\n");
    return 0;
}
*/
```

```c
/*
int main()
{
    pid_t pid;//int 와 같다 (pid_t)
    pid=fork();//프로세스 생성 (자식의 pid 값 리턴-> 0), fork를 하니까 밑 라인들을 복사
    if(pid>0)
        printf("parent\n");//처음에는 0보다 크니까 if문 구동
    else if(pid==0)//자식 프로세스는 0을 리턴하니까 엘스 이프문 구동
        printf("child\n");
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```

```c

int main()
{
    pid_t pid;
    pid=fork();//프로세스 생성(자식의 pid 값=0)
    if(pid>0)
        printf("parent : pid = %d,cpid=%d\n",getpid(),pid);
    else if(pid==0)
        printf("child : pid =%d,cpid=%d\n",getpid(),pid);
    else
    {
        perror("fork() ");//perror 는 에러종류를 알아보는것.
        exit(-1);
    }
    return 0;
}
```

```c
int main()
{

    pid_t pid;
    int i;
    pid=fork();
    if(pid>0)//처음에는 부모 프로세스 실행
    {
        while(1)
        {
            for(i=0;i<26;i++)
            {
                //sleep(1);
                printf("%c ",i+'A');//부모가 대문자 출력 0보다 크니까
                fflush(stdout);
            }
        }
    }
    else if(pid==0)//자식 리턴값이 0이니까 출력
    {
        while(1)
        {
            for(i=0;i<26;i++)
            {
                //sleep(1);
                printf("%c ",i+'a');//위에 fork 에서 자식 값을 0을 반환하니까 여기서 소문자 출력
                fflush(stdout);
            }
        }                       //프로세스가 2개가 돌아가니까 대문자 소문자 번갈아 가면서 출력됨 (fork는 프로세스를 여러개 하는 . 뭐 그런)
    }
    else
    {
        perror("fork() ");
        exit(-1);
    }

    printf("\n");
    return 0;
}
*/
```

```c
int global=100;

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

    if(pid>0)//부모 프로세스
    {
        printf("global : %d,local : %d\n",global,local);//100,10 출력
    }
    else if(pid==0)//자식 프로세스
    {
        global++;
        local++;
        printf("global : %d,local : %d\n",global,local);//101, 11 출력
    }
    else
    {

        perror("fork() ");
        exit(-1);
    }
    printf("\n");
    return 0;
}
```

```c

/*//실행해보면 <defunct> 좀비 파일을 볼수있다.
int main()
{
    pid_t pid;
    if((pid=fork())>0)//실행되고 1000초간 실행 정지됨
        sleep(1000);
    else if(pid==0)//실행되면 바로 종료
        ;
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}

*/
/*
int main()
{
    pid_t pid;
    int status;
    if((pid=fork())>0)
    {
        wait(&status);//어떻게 종료되었는지 확인하는 함수. status를 받는다.
        printf("status : %d\n",status);
    }
    else if(pid==0)//자식 프로세스 종료
        exit(7);//7을 넘긴다.
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}//결과값이 1792 가 출력 ->
//1792/256=7 exit 7
*/
```

```
219 /*
220 int main()//잘모르겠음 ...
221 {
222     pid_t pid;
223     int status;
224     if((pid=fork())>0)
225     {
226         wait(&status);//어떻게 종료되었는지 확인하는 함수. status를 받는다.
227         printf("status : %d\n",status);
228     }
229     else if(pid==0)//자식 프로세스 종료
230         exit(7);//7을 넘긴다.
231     else
232     {
233         perror("fork() ");
234         exit(-1);
235     }
236     return 0;
237 }//결과값이 1792 가 출력 ->
238 //1792/256=7 exit 7
239 */
240
241 /*
242 int main()
243 {
244     pid_t pid;
245     int status;
246     if((pid=fork())>0)
247     {
248         wait(&status);
249         printf("status : 0x%x\n",(status>>8)&0xff);
250     }
251     else if(pid==0)
252         exit(7);
253     else
254     {
255         perror("fork() ");
256         exit(-1);
257     }
258     return 0;
259 }
```

```
285
286 int main()//잘모르겠음 아는게 없음 ...
287 {
288     pid_t pid;
289     int status;
290     if((pid=fork())>0)
291     {
292         wait(&status);
293         printf("status : 0x%d\n",status&0x7f);//WEXITSTATUS(status));
294     }
295     else if(pid==0)//abort는 프로세스를 비정상적종료 시키고 코어 덤프파일을 생성한다.
296         abort();//0x0이 나오는 이유는 비정상종료 앞의 8자리
297     else        //abort 는 시그널을 생성함 시그널 종류 보는법은 kill -l
298     {
299         perror("fork() ");
300         exit(-1);
301     }
302     return 0;
303 }
304
305
306
-- INSERT --
```

프로세스 계통도(남꺼를 참고해도 모르겠습니다….

프로세스의 포인터에는
pptr(parent point)
cptr(child point) 자식이 생기면 거기로 cptr 을 옮긴다.
yptr(같은 부모로부터 자신보다 더 늦게 생성된 프로세스의 포인터)
optr(같은 부모로부터 자신보다 더 일직 생성된 프로세스에 대한 포인터)

실행이 되고 있는 프로세스는 런쿠에 존재. 실행을 위해 대기하고 있는 프로세스는 웨이트 큐에 존재.