

TI DSP, MCU 및 Xilinx Zynq FPGA

프로그래밍 전문가 과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 하성용
accept0108@naver.com

22 일차
-리눅스 프로그래밍 5 일차

ls_moudle9.c

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

```
void recursive_dir(char *dname);
```

//먼저 동작을 위해 위에 써놓는방식

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

```
void recursive_dir(char *dname)
{
```

```
    struct dirent *p;
    struct stat buf;
    DIR *dp; //디렉토리안에있는 리스트들
    chdir(dname);
    dp=opendir(".");
    printf("\t%s:\n",dname);
    while(p=readdir(dp)) //이안에 리스트가 다순회를 할때까지
    {
        stat(p->d_name,&buf); // 밑에걸 디렉토리인지 아닌지 판별
        if(S_ISDIR(buf.st_mode))
            if(strcmp(p->d_name,".")&&strcmp(p->d_name,
".."))// . 이나 ..이면 들어가지말것. 같으면 //숫컷 . 은 넘어가는거
                recursive_dir(p->d_name); //p->d name 으로 들
어가 다돌고 .으로 돌아감
    }
    chdir(".."); //상위디렉토리 ..으로감
    closedir(dp);
}
```

fork
v1
페이징 매커니즘
왜 페이징을 사용하면 어떤부분에서 이점을 얻을수있을까
대용량의 데이터들 처리하려면 대용량의
밖에 메모리가필요

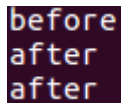
페이징을 사용하면
클럭스피드가 느
나가서 주소값을 보면 진짜주소가

fork=복사
복사는 가능. 자신을 복사한 복사라도 PID(고유값)는 다르다

제각각 흩어지기때문에 제어가안됨

```
fork1
#include<unistd.h>
#include<stdio.h>

int main(void)
{
    printf("before\n");
    fork();
    printf("after\n");
    return 0;
}
```

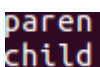


```
before
after
after
```

fork2.c //부모와 자식관의 관계 알아보기

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

int main(void)
{
    pid_t pid; //pid_t = int 랑 같은것,
    pid=fork(); //pid 는
    if(pid>0) //자식이 0 보다 크다면
        printf("paren\n");
    else if(pid==0)
        printf("child\n");
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```



```
paren
child
```

fork3.c

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

```

int main(void)
{
    pid_t pid;
    pid=fork(); //자식프로세스 생성
    if(pid>0) //pid 가 0 보다 큼 ( 부모)
        printf("parent : pid = %d, cpid=%d\n",getpid(),pid);
    자기자신의 프로세스아이디, cpid 는 포크의 리턴값받아옴,
    else if(pid==0) //자식프로세스
        printf("child : pid = %d, cpid=%d\n",getpid(),pid);
    else
    {
        perror("fork()");
        exit(-1);
    }
    return 0;
}

```

```

parent : pid = 3403, cpid=3404
child : pid = 3404, cpid=0

```

fork4.c
for 문을 while(1)로 감싸서 무한루프 만들기

```

#include<stdio.h>
#include<unistd.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); //시스템내부에있는 버퍼를 비울것
            }
        }
    }
    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;
}

```

Is process Shared VM? //프로세스가 Vm 을공유할수있는지

ps_test1.c

```

#include<stdio.h>

```

```

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

```

ps_test2.c

```

#include<stdio.h>

```

```

int main(void)
{
    int *p=0xbfc3e40;
    printf("&a:%#p\n",*p);
    return 0;
}

```

./test &
 위와 같이 수행한 후 나오는 주소값을 ps_test2.c 의 *p 값에 대입
 ./a.out 을 해서 확인

Segmentation fault (core dumped)

pgid = 어떻게 읽기속성과 쓰기속성을 가지고있는지확인

하나는 모터작동 하나는 레이더 신호해석(속도값,장애물)
 a 프로세스인 모터한테 정보를 전달하기위해서 할수있는게
 메시지큐
 파이프
 셰이드
 IPC(인터포스 커뮤니케이션 프로세스 내부 커뮤드니케이션이라는말)
 작업분담, 경우에 따라선 협업이 필요할때 하는게 IPC
 프로세스는 가상메모리를 공유할수없음
 작업을 효율적으로 분담하기위해(혼자하면 메모리부담이커져서)

ps_test1.c

```

#include<stdio.h>

```

```

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

```

ps_test2.c

```
#include<stdio.h>

int main(void)
{
    int *p=0x7ffdbb989454;
    printf("&a: %#p\n",*p);
    return 0;
}
```

ps_test3.c

```
#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;
}
```

```
global : 100, local:10
global : 101, local=11
```

C.O.W.
copy on write
쓰는동시에 복사가된다

돌아가야하니까 TEXT 가 필요
스택연혁으로 알고싶음
전역변수 접근 데이터 복사

메모리에 무언가 쓰기작업이되면 복사시작

ps_test4.c (논블로킹처럼 동작함)

```
#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));
            buf[ret]=0;
            printf("Keyboard:%s\n",buf);
        }
    }
    else if(pid ==0)
    {
        for(;;)
        {
            ret=read(fd,buf,sizeof(buf));
            buf[ret]=0;
            printf("myfifo:%s\n",buf);
        }
    }
    else
    {
        perror("fork() ");
        exit(-1);
    }
    close(fd);
    return 0;
}
```

생성후
mkfifo myfifo

우클릭 오픈터미널로 해당위치 터미널 열고
cat > myfifo

터미널 ./a.out 실행

fork9.c (실질적으로 이러한 process 를 Zombie Process 라 한다)

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

```

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

```

후 컴파일하고 실행하고

터미널열고

ps -ef|grep a.out

defunct 나오는게 좀비프로세스

parent & child

child 가 죽음(exit2)

보면 SIGCHLD (죽어서 처리해달라는 메시지)

wait(&status); // chile 가 어떻게 죽었는지를 봄
status 로 확인

특수한코드로 죽은걸 알게되면 그코드에 대한 조사를 해야되는데
즉 wait(&status)는 자식의 상태값을 받기위해 사용하는 시스템콜중 하나입니다

wait.c

```

#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:%d\n",status);
    }
    else if(pid==0)
        exit(7);
    else
    {

```



```

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

```

1792 나오는데
 이걸 296 으로 나누기
 7 이나옴 (exit7 이랑 일치)

How to Extract Status?

정상종료

비정상종료

256 으로 나누거나 8 을 시프트

프로세스는 시그널을 맞으면 기본적으로 죽음

모든 비정상종료를 총칭하는말이 시그널

wait3.c (오타있음)

```

#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>>8)&0xff);
    }
    else if(pid==0)
        exit(7);
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}

```

```

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

int main(void)
{
    pid_t pid;
    int status;
    if((pid=fork())>0)
    {
        wait(&status);
        printf("status:0x%x\n",WEXITSTATUS(status));
    }
    else if(pid==0)
        exit(7);
    else
    {
        perror("fork()");
        exit(-1);
    }
    return 0;
}

```

status:0x7

wait5.c

```

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

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

```

status:0x0

```
stty -a //사용되는 시그널값들을 보여줌
speed 38400 baud; rows 37; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rpnt = ^R;
werase = ^W; lnext = ^V; discard = ^O; min = 1; time = 0;
-parenb -parodd -cmspar cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc -ixany -imaxbel iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echopr
echoctl echoke -flusho -extproc
```

kill -l

SIG~ //리눅스상에 존재하는 모든 시그널 번호들

맨앞 1 비트는 용도가

포덤프가 어떻게죽었는지 확인하는거