

**TI DSP, MCU 및 Xilinx Zynq
FPGA
프로그래밍 전문가 과정**

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 문한나

mhn97@naver.com

임베디드 어플리케이션 구현

1.

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

int main(void){

    int fd,ret;
    char buf[1024];
    fd = open("myfifo",O_RDWR);
    fcntl(0,F_SETFL,O_NONBLOCK);
    fcntl(fd,F_SETFL,O_NONBLOCK);

    for(;;){

        if((ret = read(0,buf,sizeof(buf)))>0){

            buf[ret-1]=0;
            printf("Keyboard input : [%s]\n",buf);
        }
        if((ret = read(fd,buf,sizeof(buf)))>0){
            buf[ret-1]=0;
            printf("pipe input : [%s]\n",buf);
            if(strcmp(buf,"type.c\n")){
                open("type.c",O_WRONLY | O_CREAT , 0644);
            }
        }
    }

    close(fd);

    return 0;

}
```

```
mhn@mhn-Z20NH-AS51B5U: ~/linux/t/tt
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$ gcc test5.c
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$ mkfifo myfifo
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$ ls
a.out myfifo test5.c
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$ ./a.out test5.c
pipe input : [hi~]
hello~
Keyboard input : [hello~]
pipe input : [a]
pipe input : [a]
pipe input : [a]
pipe input : [c]
^C
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$ ls
a.out myfifo test5.c type.c
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$
```

```
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$ cat > myfifo
hi~
a
a
a
c
^C
mhn@mhn-Z20NH-AS51B5U:~/linux/t/tt$
```

2.

```
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#define BUF_SIZE          128

//jmp_buf env1;
int i = -1;

void my_sig(int signo){
    //void my_sig(){

        printf("You lose\n");
        exit(0);

    }

void my_sig2(int signo){
    //void my_sig(){

        printf("clap!\n");

    }

int main(void){

    int randd;
    int input;
    int a;
    int count=0;
    char buf[1024] = {0};
    char msg[32];
```

```

    signal(SIGALRM,my_sig);
    printf("369 게임을 시작합니다. 입력은 2 초 이내입니다\n");

re;;

    i++;
    printf("%d\n",i);

    for(i ; i<1000; i++){

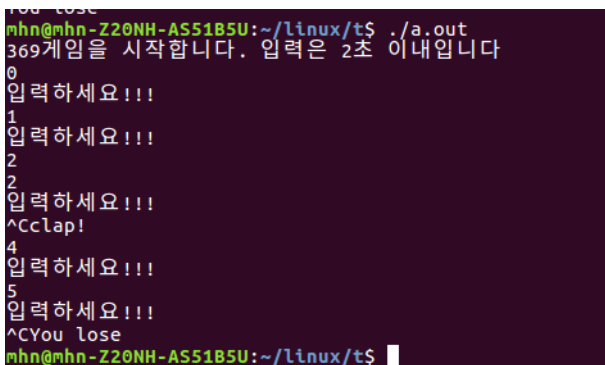
        //          printf("%d\n",i);
        printf("입력하세요!!!\n");
        read(0,buf,sizeof(buf));
        input = atoi(buf);
        signal(SIGALRM,my_sig);
        alarm(2);

        if((i%3) == 1){
            signal(SIGINT,my_sig2);
            goto re;
        }else
            signal(SIGINT,my_sig);

    }

    return 0;
}

```



```

mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ./a.out
369 게임을 시작합니다. 입력은 2 초 이내입니다
0
입력하세요!!!
1
입력하세요!!!
2
2
입력하세요!!!
^Cclap!
4
입력하세요!!!
5
입력하세요!!!
^CYou lose
mhn@mhn-Z20NH-AS51B5U:~/linux/t$

```

3. file system manager

- memory manager
- task manager
- device manager
- network manager

4. file

5. * 사용자 임의대로 재구성이 가능하다.

오픈소스이다.

* 열악한 환경에서도 HW 자원을 적절히 활용하여 동작한다.

* 커널의 크기가 작다.

사실 작은 사이즈는 아니지만 윈도우보다는 작다

* 완벽한 멀티유저, 멀티태스킹 시스템

* 뛰어난 안정성

네트워크 장비에 들어간다

* 빠른 업그레이드

예로 nvidia 사건을 들 수 있다.

* 강력한 네트워크 지원

* 풍부한 소프트웨어

GNU 재단

6. 2^{32} (4GB) 크기의 가상 주소 공간을 표현할 수 있으며 0~3GB 까지가 User Space 구간이고 3GB~4GB 까지가 Kernel 의 Space 구간이다.

7. pagefault handler 를 보고 유저인지 커널인지 확인한다.

만약 유저이면 권한이 없으니 segmentaion fault 를 출력하고

커널이면 페이지를 재할당 받은 후 접근한다.

8. elf

9. TGID 값이 같으면서 PID 값이 다르면 같은 프로세스 안에 있는 스레드이며, TGID 와 PID 값이 같으면 프로세스이며, 리더이다.

10. task_struct

11. 현재 태스크의 task_struct 구조체를 가리킬 수 있게 해 준다. (thread_info 가 가리키는 task)

```
mhn@mhn-Z20NH-AS51B5U: ~/kernel/linux-4.4
1 #ifndef _ALPHA_CURRENT_H
2 #define _ALPHA_CURRENT_H
3
4 #include <linux/thread_info.h>
5
6 #define get_current()    (current_thread_info()->task)
7 #define current         get_current()
8
9 #endif /* _ALPHA_CURRENT_H */
```

```

44
45 /*
46  * low level task data that entry.S needs immediate access to.
47  * __switch_to() assumes cpu_context follows immediately after cpu_domain.
48  */
49 struct thread_info {
50     unsigned long    flags;           /* low level flags */
51     int              preempt_count;   /* 0 => preemptable, <0 => bug */
52     mm_segment_t     addr_limit;     /* address limit */
53     struct task_struct *task;        /* main task structure */
54     __u32             cpu;           /* cpu */
55     __u32             cpu_domain;    /* cpu domain */
56     struct cpu_context_save cpu_context; /* cpu context */
57     __u32             syscall;       /* syscall number */
58     __u8              used_cp[16];   /* thread used copro */
59     unsigned long     tp_value[2];   /* TLS registers */
60 #ifdef CONFIG_CRUNCH
61     struct crunch_state crunchstate;
62 #endif
63     union fp_state     fpstate __attribute__((aligned(8)));
64     union vfp_state     vfpstate;
65 #ifdef CONFIG_ARM_THUMBEE
66     unsigned long      thumbee_state; /* ThumbEE Handler Base register */
67 #endif
68 };

```

12. Process 는 메모리를 공유하고 Thread 는 종속적이다.

13. System Context, Memory Context, HW Context 가 있다.

HW Context 는 Context-Switching 을 하기 위한 구조이다.

14. deadline 은 실시간 태스크 스케줄링 방식으로, deadline 이 가장 가까운(즉, 가장 급한)태스크를 스케줄링 대상으로 선정한다. 태스크들은 RBtree 에 정렬되어있다.

15. TASK_INTERRUPTIBLE 은 현재 대기중인 TASK 가 INTERRUPT 를 수신해도 지장이 없는 것이고, TASK_UNINTERRUPTIBLE 은 현재 대기중인 TASK 가 자신이 기다리는 사건 외에는 방해 받아서는 안되는 경우여서 중요한 연산시 사용한다.

16. $O(N)$ 은 태스크의 개수가 늘어나면 그만큼 스케줄링에 걸리는 시간도 선형적으로 증가하는 알고리즘이고, $O(1)$ 은 태스크의 개수가 많은 적든 스케줄링에 걸리는 시간이 일정한 알고리즘이다. 따라서 처리해야할 태스크가 적으면 $O(N)$ 방식이, 많으면 $O(1)$ 방식이 좋다.

17. fork() 시 효율성을 위해 기존의 캐시를 참조하여 2 번 cpu 에 만들어진다

18. fork() 시 캐시를 참조하는게 효율적이긴 하지만 많은 프로세스가 존재하는 경우에는 경쟁이 심해지기 때문에 캐시를 참조하지 않고 2 번 cpu 에 새로 생성된다

19. 리눅스에선 접근속도가 같은 메모리의 집합을 뱅크라고 부르며, 이것을 표현하는 구조가 node 이다.

uma 는 node 가 한개여서 메모리에 접근하는 속도가 모두 같고, 모든 cpu 가 메모리를 공유한다.

이 노드는 전역변수 `contig_page_data` 를 통해 관리한다.

numa 는 node 가 복수개여서 메모리에 접근하는 속도가 다르며, 이기종 아키텍처에 사용된다.
이 노드들은 pglist_data 배열에서 리스트로 관리된다.

```
25
26 #ifndef CONFIG_NEED_MULTIPLE_NODES
27 struct pglist_data __refdata contig_page_data = {
28     .bdata = &bootmem_node_data[0]
29 };
30 EXPORT_SYMBOL(contig_page_data);
31 #endif
32
```

20. Static Priority – 0~99 (총 100 단계)

Dynamic Priority – 100~139 (총 40 단계)

21. 물리메모리가 1GB 이상일 때 896MB 이상의 메모리 영역을 ZONE_HIGHMEM 이라 한다.

이 영역은 커널의 가상 주소공간과 동적으로(간접참조) 연결하여 사용한다.

22. 페이지 프레임이라 하며, 크기는 4kb 이다.

구조체 이름은 page 이다.

23. 버디 할당자는 외부단편화를 줄일 수 있는 구조이다.

버디 할당자는 zone 구조체에 존재하는 free_area[] 배열을 통해 구축된다.

free_area 10 개의 배열을 가지며 0~9 까지 해당 free_area 가 관리하는 할당의 크기를 나타낸다.

버디는 2 의 정수 승 개수의 페이지 프레임들을 할당해주며 리눅스 구현상 최대 4MB($2^9 * 4KB$)이다.

(2 의 승수로 구동되므로 비트연산으로 처리하면 빠르게 처리가 가능하다.)

만약 2 개의 페이지가 요청되었다면

order(0)은 4kb x 2^0 즉, 페이지 단위로 보겠다는 뜻이며 xor 연산으로 할당하여 비트맵에 표시한다.

oeder(1)은 4kb x 2^1 로 8kb 단위로 보며 양쪽을 비교하여 할당한다.

oeder(2)은 4kb x 2^2 로 16kb 단위로 보고, oeder(3)은 4kb x 2^3 로 32kb 단위로 본다.

24. 슬랩할당자는 내부단편화를 줄일 수 있는 구조이다.

슬랩할당자는 먼저 4kb 페이지 프레임을 할당받은 후 이것은 32byte 로 쪼개놓고 할당하는 방식이다.

추후 사용자가 할당받았던 공간을 해제한다면 버디로 반납하지 않고 미리 받아놨던 공간에 다시 가지고 있다.

(일종의 캐시로 사용하는 것)

슬랩할당자는 kmem_cache_alloc()으로 할당받을 수 있고, 이것 외에도 kmalloc() / kfree()를 제공한다.

25. task_struct 밑에 mm_struct 가 관리

26. task_struct 밑에 mm_struct 밑에 vm_area_struct 가 관리

27. 페이지 테이블(pgd)

28. 결국 커널도 프로그램이기 때문에 수행되기 위해서는 메모리가 필요하다. 따라서 Stack, Heap, Data, Text 모두 필요하다.

29. 디스크에 저장되어 있는 실행 파일의 어느 부분을 읽어서 물리메모리에 올려놓을 것인가는 elf 포맷 파일의 헤더를 읽음으로써 알 수 있다. 메모리에 올리는 작업은 sys_execve()가 하며, depend on paging 을 수행한다. 적재된 실행 파일을 수행하려면 주소변환을 위한 페이지 테이블이 필요하다(pgd) 몫을 페이지 테이블의 엔트리를 탐색하는 인덱스로 사용하고, 나머지는 페이지 프레임 내에서 offset 으로 사용한다.

30. 가상주소를 사용하는 운영체제가 원활히 수행될 수 있도록 하기 위해 가상 주소로부터 물리 주소로의 변환을 담당하는 별도의 하드웨어(cpu 가 장착)

31. 디스크 블록이라 부르며 크기는 4KB 이다.

32. 디바이스 드라이버

33.

do_fork()함수가 호출되고 수행되기 위해 필요한 자원을 할당한다.

```
1764 #ifndef CONFIG_HAVE_COPY_THREAD_TLS
1765 /* For compatibility with architectures that call do_fork directly rather than
1766  * using the syscall entry points below. */
1767 long do_fork(unsigned long clone_flags,
1768             unsigned long stack_start,
1769             unsigned long stack_size,
1770             int __user *parent_tidptr,
1771             int __user *child_tidptr)
1772 {
1773     return _do_fork(clone_flags, stack_start, stack_size,
1774                   parent_tidptr, child_tidptr, 0);
1775 }
1776 #endif
1777
1778 /*
1779  * Create a kernel thread.
1780  */
1781 pid_t kernel_thread(int (*fn)(void *), void *arg, unsigned long flags)
1782 {
1783     return _do_fork(flags|CLONE_VM|CLONE_UNTRACED, (unsigned long)fn,
1784                   (unsigned long)arg, NULL, NULL, 0);
1785 }
1786
```

34. 최상위 디렉토리(루트)가 어디에 위치해있는지 정보를 가지고 있다.

35. 사용자 태스크들은 시스템 호출을 통해 파일시스템에 접근한다.

하지만 리눅스의 파일 시스템은 한두가지가 아니다. 이를 고려하여 파일시스템과 사용자 태스크 사이에 여러 포맷들을 다 커버할 수 있는 가상적인 층을 만들었다. 이것이 VFS 이다.

VFS 는 사용자가 시스템 콜을 호출하면 파일의 인자를 확인하여 구조체를 만든다. 이 구조체를 인자로 하여 파일 시스템에 요청을 하면 파일시스템은 inode 를 활용하여 넘겨받은 구조체에 값을 넣어준 후 리턴하는 방식으로 사용된다.

36. 외부 인터럽트와 내부 인터럽트가 있다.

외부 인터럽트는 pin 을 통해 외부에서 들어오며 현재 수행중인 태스크와 관련없는 비동기적인 하드웨어적 사건을 의미한다.

내부인터럽트는 cpu 자체가 감지하며 동기적인 사건으로 소프트웨어적 사건을 의미한다.

37. fault

trap
abort

38. fault 는 대표적으로 페이지 폴트가있다. 페이지 폴트는 폴트를 일으킨 명령어 주소를 eip 에 넣어 두었다가 해당 핸들러가 종료되고 나면 eip 에 저장되어있는 주소부터 다시 수행을 시작한다.

trap 은 시스템 콜이 대표적이다. trap 을 일으킨 다음 주소를 eip 에 넣어두고 다음 부터 다시 수행한다.

abort 는 종료시킨다.

39. i_node 객체에 i_rdev 필드에 주 번호와 부번호를 저장하는데, 부 번호를 통해 같은 디바이스 안에서 갯수를 구별하여 사용한다.

40. sys_call_table

41.

intel - eax

arm - r7 레지스터

42. pgd

43.

44. 커널도 프로그램이기 때문이다(스택 필요)

45. 슬랩 할당자

46. 버디 할당자

47. 세마포어(Semaphore)는 공유된 자원의 데이터를 여러 프로세스가 접근하는 것을 막는 것이고, 뮤텍스(Mutex)는 공유된 자원의 데이터를 여러 스레드가 접근하는 것을 막는 것이다.

48. insmod

49. rmmod

50. kernel stack 을 할당받으며, thread_info 구조체를 포함하고 있다.

51.

52.

53.

버디할당자는 외부단편화를 줄일 수 있다. 하지만 작은 용량이 필요해도 최소단위가 4kb 이기 때문에 메모리가 낭비될 수 있다. 이같은 내부단편화를 해결하기 위해 슬랩할당자를 쓴다

슬랩할당자는 먼저 4kb 페이지 프레임을 버디로부터 할당받은 후 이것은 32byte 로 쪼개놓고 할당하는 방식이다. 추후 사용자가 할당받았던 공간을 해제한다면 버디로 반납하지 않고 미리 받아놔던 공간에 다시 가지고 있다. (일종의 캐시로 사용하는 것)

54.

55.

56.

57. 대표적으로 CISC Architecture 는 x86, RISC Architecture 는 arm 으로 예를 들 수 있다.

CISC Architecture 는 명령어의 길이가 가변적으로 구성된 것으로 한 명령어의 길이를 줄여 디코딩 속도를 높이고 최소크기의 메모리 구조를 가진다.(하드웨어의 비중이 크다)

RISC Architecture 는 고정된 길이의 명령어를 사용한다.

적은수의 명령어로 명령어 집합을 구성하며 기존의 복잡한 명령은 보유한 명령어를 조합해서 사용한다. (소프트웨어의 비중이 크다)

58.

59. 분기 명령어를 사용할 때 깨진다(call, jmp 등)

60.

61.

62.

63.

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

```
int main(int argc, char **argv){

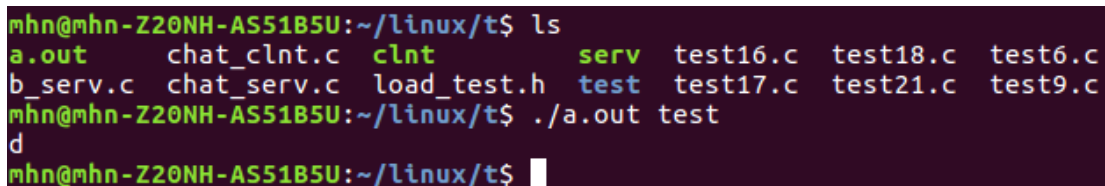
    struct stat buf;
    char ch;
    stat(argv[1], &buf); // 상태 보기

    if(S_ISDIR(buf.st_mode))
        ch = 'd';
    if(S_ISREG(buf.st_mode))
        ch = '-';
    if(S_ISFIFO(buf.st_mode))
        ch = 'p';
    if(S_ISLNK(buf.st_mode))
        ch = 'l';
    if(S_ISSOCK(buf.st_mode))
        ch = 's';
    if(S_ISCHR(buf.st_mode))
        ch = 'c';
    if(S_ISBLK(buf.st_mode))
        ch = 'b';
    printf("%c\n", ch);

    return 0;
```

// 두번째 인자로 받은 것이 무엇인지 판별

```
}
```



```
mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ls
a.out  chat_clnt.c  clnt  serv  test16.c  test18.c  test6.c
b_serv.c  chat_serv.c  load_test.h  test  test17.c  test21.c  test9.c
mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ./a.out test
d
mhn@mhn-Z20NH-AS51B5U:~/linux/t$
```

64.

65.

66. 다운 : wget <https://micros.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.4.tar.gz>

압축 풀기 : tar zxvf linux-4.4.tar.gz

찾기 : 압축을 푼 폴더에서 vi -t task_struct

67. 사용자가 작성한 프로그램을 저장하면 운영체제는 이 내용을 파일이라는 객체로 관리한다. 파일은 inode 와 함께 디스크에 4kb 크기의 디스크 블록에 저장된다. 디스크에 있는 소스파일을 컴파일하면 실행파일이 생성되며, 실행파일은 inode 를 통해 4kb 단위로 메모리에 올려진다. 메모리에 올려지면 태스크라는 객체가 생성된것이며 (task_struct 가 만들어진다), 세그먼트 테이블(가상 메모리)과 페이지 테이블(물리 메모리)을 이용하여 관리한다. 세그먼트에서 페이지로 가는 과정을 페이지징이라 하며, 페이지징된 프레임(태스크)은 우선순위에 따라 cpu 의 제어권을 가진다. 현재 동작중인 프로세스는 run queue 에, 우선순위에서 밀리거나 작업 중 제어권을 뺏긴 프로세스는 wait queue 에 들어가며, 프로세스는 값이 꼬이는 것을 방지하기 위해 task_struct 에 정보를 저장한 후 제어권을 넘겨준다. 이렇게 아주 빠른 속도로 여러 프로세스들이 서로 제어권을 넘겨주며 cpu 를 사용하는 것이 Context-Switching 이며, 덕분에 Multi-Tasking 이 가능하다.

68. lsmod

69. 유저가 system call 을 요청하면 라이브러리 함수 glibc 를 호출하여 요청한 번호를 ax 레지스터에 저장한다. cpu 가 int 0x80 을 인자로 하여 system call 함수를 호출한다.

컨텍스트 스위칭 후 제어권을 커널로 넘긴다.

Idt_table 의 128 번으로 sys_call_table 을 본다.

ax 레지스터에 저장해놨던 번호를 보고 함수포인터를 구동한다.

70.

71.

72.

```
#include <sys/types.h>
```

```
#include <dirent.h>
```

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

```
int main(void){
```

```
    DIR *dp; /
```

```
    int i=0;
```

```
    struct dirent *p;
```

```
    dp = opendir(".");
```

```
    while(p = readdir(dp)){
```

```
        if(p->d_name[0] == '.')
```

```
            continue;
```

```
        printf("%-16s",p->d_name);
```

```
        if((i+1) % 5 == 0)
```

```
            printf("\n");
```

```
        i++;
```

```
    }printf("\n");
```

```
    closedir(dp);
```

```
    return 0;
```

```
}
```

```

mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ls
a.out      chat_clnt.c  clnt      serv      test16.c  test18.c  test6.c
b_serv.c   chat_serv.c  load_test.h test      test17.c  test21.c  test9.c
mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ./a.out
serv        test9.c      chat_clnt.c  test21.c    a.out
test6.c     chat_serv.c  test17.c    test18.c    load_test.h
test        test16.c     b_serv.c    clnt
mhn@mhn-Z20NH-AS51B5U:~/linux/t$

```

73. set follow-fork-mode parent

74. cow 는 copy on write 의 약자로 미리 모든 메모리를 할당받는 것이 아니라 실제로 쓸 때 할당을 받는 것을 말한다. 만약 fork 후 자식 프로세스가 exec 를 하게 되면 부모의 모든 메모리 레이아웃을 복제하게 되는데, 중요한 점은 모든 부분이 필요하지 않을 수도 있다. 이렇게 되면 기껏 복제해 놓은 메모리를 사용하지 않게 될 수 있는데, 이런 경우를 방지하기 위해 필요할 경우에만 할당을 받는 구조이다.

75. Blocking 은 동기 방식으로 순서가 중요할 때 쓰며, Non-Blocking 은 비동기 방식으로 실시간 처리가 중요할 때 쓴다.

76.

```

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

int main(void){

    pid_t pid;
    int status;
    if((pid = fork())>0){
        wait(&status);
        term_status(status);
    }
    else if(pid == 0)
        abort();
    else{

```

```

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

```

```

mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ./a.out
(signal)status : 0x6, core dumped
mhn@mhn-Z20NH-AS51B5U:~/linux/t$

```

77.

```

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

int daemon_init(void){

    int i;
    if(fork() > 0)
        exit(0);
    setsid();
    chdir("/");
    umask(0);
    for(i=0;i<64;i++)
        close(i);
    signal(SIGCHLD,SIG_IGN);
    return 0;

}

int main(void){

    daemon_init();
    for(;;){

    }

    return 0;

}

```

```

mhn@mhn-Z20NH-AS51B5U: ~/linux/t
mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ps -ef | grep a.out
mhn      7535   2411  99  13:20  ?        00:00:18 ./a.out
mhn      7538   7515   0  13:20 pts/18   00:00:00 grep --color=auto a.out
mhn@mhn-Z20NH-AS51B5U:~/linux/t$

```

78.

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

void my_sig(int signo){

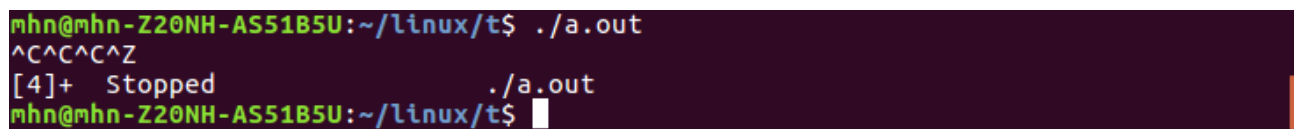
    exit(0);

}

int main(void){

for(;;){
    signal(SIGINT,SIG_IGN);
    signal(SIGQUIT,my_sig);
}

    return 0;
}
```



```
mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ./a.out
^C^C^C^Z
[4]+  Stopped                  ./a.out
mhn@mhn-Z20NH-AS51B5U:~/linux/t$
```

79. goto 는 스택을 해제할 수 없기 때문에 함수에서 함수로 이동할 수 없다. 이런 경우에는 setjmp 를 사용한다.

80. 리눅스의 자원인 파일을 관리하기 위한 기능이다.

81.

82.

83.

84.

85.

86.

87.

88.

server

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/epoll.h>

#define BUF_SIZE          128
#define MAX_CLNT          256

typedef struct sockaddr_in      si;
typedef struct sockaddr *      sp;

int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
pthread_mutex_t mtx;

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void send_msg(char *msg, int len)
{
    int i;

    pthread_mutex_lock(&mtx);

    for(i = 0; i<clnt_cnt; i++)
        write(clnt_socks[i], msg, len);

    pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void *arg)
{
    int clnt_sock = *((int *)arg);
    int str_len = 0, i;
    char msg[BUF_SIZE];

    while((str_len = read(clnt_sock , msg, sizeof(msg))) != 0)
        send_msg(msg, str_len);

    pthread_mutex_lock(&mtx);

    for(i = 0; i<clnt_cnt; i++){
        if(clnt_sock == clnt_socks[i])
        {
            while(i++ < clnt_cnt-1)
                clnt_socks[i] = clnt_socks[i+1];
            break;
        }
    }
}
```

```

        }
    }

    clnt_cnt--;
    pthread_mutex_unlock(&mtx);
    close(clnt_sock);

    return NULL;
}

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock;
    si serv_addr, clnt_addr;
    socklen_t addr_size;
    pthread_t t_id;

    if(argc != 2)
    {
        printf("Usage: %s <port>\n", argv[0]);
        exit(1);
    }

    pthread_mutex_init(&mtx, NULL);

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

    if(serv_sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("bind() error");
    if(listen(serv_sock, 22) == -1)
        err_handler("listen() error!");

    for(;;)
    {
        addr_size = sizeof(clnt_addr);
        clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);

        pthread_mutex_lock(&mtx);
        clnt_socks[clnt_cnt++] = clnt_sock;
        pthread_mutex_unlock(&mtx);

        pthread_create(&t_id, NULL, clnt_handler, (void *) &clnt_sock);
        pthread_detach(t_id);
        printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
    }
    close(serv_sock);
    return 0;
}

```


clnt

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/epoll.h>

#define BUF_SIZE          128
#define NAME_SIZE         32

typedef struct sockaddr_in  si;
typedef struct sockaddr *sp;

char name[NAME_SIZE] = "DEFAULT";
char msg[BUF_SIZE];

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n',stderr);
    exit(1);
}

void *send_msg(void *arg)
{
    int sock = *((int *)arg);
    char name_msg[NAME_SIZE + BUF_SIZE];

    for(;;)
    {
        fgets(msg, BUF_SIZE, stdin);

        if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
        {
            close(sock);
            exit(0);
        }

        sprintf(name_msg, "%s %s", name, msg);
        write(sock , name_msg, strlen(name_msg));
    }
    return NULL;
}

void *recv_msg(void *arg)
{
    int sock = *((int *)arg);
    char name_msg[NAME_SIZE +BUF_SIZE];
    int str_len;
```

```

for(;;)
{
    str_len = read(sock , name_msg, NAME_SIZE +BUF_SIZE-1);

    if(str_len ==-1)
        return (void*)-1;

    name_msg[str_len]=0;
    fputs(name_msg, stdout);
}
return NULL;
}

int main(int argc, char **argv)
{
    int sock;
    si serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

    if(argc !=4)
    {
        printf("Usage: %s <IP> <port> <name> \n", argv[0]);
        exit(1);
    }
    sprintf(name, "[%s]", argv[3]);
    sock = socket(PF_INET, SOCK_STREAM, 0);

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock , (sp)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error!");

    pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
    pthread_create(&rcv_thread, NULL, rcv_msg, (void*)&sock);
    pthread_join(snd_thread, &thread_ret);
    pthread_join(rcv_thread, &thread_ret);

    close(sock);
    return 0;
}

```

```

mhn@mhn-Z20NH-A551B5U:~/linux/t$ gcc -o chat_serv chat_serv.c -lpthread
mhn@mhn-Z20NH-A551B5U:~/linux/t$ ./chat_serv 7777
Connected Client IP: 127.0.0.1
Connected Client IP: 127.0.0.1

```

```

mhn@mhn-Z20NH-A551B5U:~/linux/t$ gcc -o chat_clnt chat_clnt.c -lpthread
mhn@mhn-Z20NH-A551B5U:~/linux/t$ ./chat_clnt 127.0.0.1 7777 a
hi
[a] hi
hello
[a] hello
~~~
[a] ~~~

```

89.

server

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/epoll.h>
#include "load_test.h"

#define BUF_SIZE          128
#define MAX_CLNT          256

typedef struct sockaddr_in      si;
typedef struct sockaddr *      sp;

//jmp_buf env1;

int black_cnt = 0,cnt = 0;
double run_time = 0;
double ratio = 0;
int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
pthread_mutex_t mtx;

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void get_runtime(tv start, tv end)
{
    end.tv_usec = end.tv_usec - start.tv_usec;
    end.tv_sec = end.tv_sec - start.tv_sec;
    end.tv_usec += end.tv_sec * 1000000;
    run_time = end.tv_usec;
    printf("runtime = %lf sec\n", end.tv_usec / 1000000.0); //확인용
}

//double send_msg(char *msg, int len)
void send_msg(char *msg, int len)
{
    int i,ret;
    tv start,end;

    pthread_mutex_lock(&mtx);

    for(i = 0; i<clnt_cnt; i++)
        write(clnt_socks[i], msg, len);
}
```

```

        pthread_mutex_unlock(&mtx);

    }
    void *clnt_handler(void *arg)
    {
        int clnt_sock = *((int *)arg);
        int str_len = 0, i;
        char msg[BUF_SIZE];
        double r_t=0;
        tv start,end;

        while((str_len = read(clnt_sock , msg, sizeof(msg))) != 0){
            gettimeofday(&start,NULL);

                send_msg(msg, str_len);
                cnt++;
            gettimeofday(&end,NULL);

            get_runtime(start,end);

            ratio = run_time / cnt;
            printf("ratio = %lf\n",ratio);

            if(ratio < 2){
                goto q;
            }
        }

        pthread_mutex_lock(&mtx);

    q:

        close(clnt_sock);
        shutdown(clnt_sock, SHUT_RD);

        for(i = 0; i<clnt_cnt; i++){
            if(clnt_sock == clnt_socks[i])
            {
                while(i++ < clnt_cnt-1)
                    clnt_socks[i] = clnt_socks[i+1];
                break;
            }
        }

        clnt_cnt--;
        pthread_mutex_unlock(&mtx);
        close(clnt_sock);

        return NULL;
    }

    int main(int argc, char **argv)
    {
        int serv_sock, clnt_sock ;
        si serv_addr, clnt_addr;
        socklen_t addr_size;

```

```

pthread_t t_id;

if(argc != 2)
{
    printf("Usage: %s <port>\n", argv[0]);
    exit(1);
}

pthread_mutex_init(&mtx, NULL);

serv_sock = socket(PF_INET, SOCK_STREAM, 0);

if(serv_sock == -1)
    err_handler("socket() error");

memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(atoi(argv[1]));

if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("bind() error");
if(listen(serv_sock, 22) == -1)
    err_handler("listen() error!");

for(;;)
{
    addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);

    pthread_mutex_lock(&mtx);
    clnt_socks[clnt_cnt++] = clnt_sock;
    pthread_mutex_unlock(&mtx);

    pthread_create(&t_id, NULL, clnt_handler, (void *) &clnt_sock);
    pthread_detach(t_id);
    printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
}
close(serv_sock);
return 0;
}

```

clnt

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/epoll.h>

```

```

#define BUF_SIZE      128
#define NAME_SIZE     32

```

```

typedef struct sockaddr_in  si;
typedef struct sockaddr *sp;

```

```

char name[NAME_SIZE] = "DEFAULT";
char msg[BUF_SIZE];

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void *send_msg(void *arg)
{
    int sock = *((int *)arg);
    char name_msg[NAME_SIZE + BUF_SIZE];

    for(;;)
    {
        fgets(msg, BUF_SIZE, stdin);

        if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
        {
            close(sock);
            exit(0);
        }

        sprintf(name_msg, "%s %s", name, msg);
        write(sock, name_msg, strlen(name_msg));
    }
    return NULL;
}

void *recv_msg(void *arg)
{
    int sock = *((int *)arg);
    char name_msg[NAME_SIZE + BUF_SIZE];
    int str_len;

    for(;;)
    {
        str_len = read(sock, name_msg, NAME_SIZE + BUF_SIZE - 1);

        if(str_len == -1)
            return (void*)-1;

        name_msg[str_len] = 0;
        fputs(name_msg, stdout);
    }
    return NULL;
}

int main(int argc, char **argv)
{
    int sock;
    struct serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

```

```

if(argc !=4)
{
    printf("Usage: %s <IP> <port> <name> \n", argv[0]);
    exit(1);
}
sprintf(name, "[%s]", argv[3]);
sock = socket(PF_INET, SOCK_STREAM, 0);

memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
serv_addr.sin_port = htons(atoi(argv[2]));

if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("connect() error!");

pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
pthread_create(&rcv_thread, NULL, rcv_msg, (void*)&sock);
pthread_join(snd_thread, &thread_ret);
pthread_join(rcv_thread, &thread_ret);

close(sock);
return 0;
}

```

```

mhn@mhn-Z20NH-A551B5U:~/linux/t$ ./b_serv 7777
Connected Client IP: 127.0.0.1
runtime = 0.000014 sec
ratio = 14.000000
runtime = 0.000014 sec
ratio = 7.000000
runtime = 0.000011 sec
ratio = 3.666667
runtime = 0.000014 sec
ratio = 3.500000
runtime = 0.000013 sec
ratio = 2.600000
runtime = 0.000013 sec
ratio = 2.166667
runtime = 0.000014 sec
ratio = 2.000000
runtime = 0.000014 sec
ratio = 1.750000

```

```
mhn@mhn-Z20NH-AS51B5U:~/linux/t$ ./chat_clnt 127.0.0.1 7777 a
a
[a] a
a
[a] a
a
[a] a
aa
[a] aa
a
[a] a
a
[a] a
a
[a] a
a
[a] a
a
mhn@mhn-Z20NH-AS51B5U:~/linux/t$ a
```

90

91

92

93 프로세스 안에서 여러 스레드가 있다고 가정할 때 동시에 접근해서는 안되는 공유자원구간이 있다.(이 구간에 둘 이상이 접근하면 값이 꼬일수 있음) 이 영역을 Critical Section 이라 한다.

94 open()을 하면 라이브러리 함수 glibc 가 호출되어 요청한 번호를 ax 레지스터에 저장한다.

cpu 가 int 0x80 을 인자로 하여 system call 함수를 호출한다. 컨텍스트 스위칭 후 제어권을 커널로 넘긴다.

Idt_table 의 128 번으로 sys_call_table 을 본다.

ax 레지스터에 저장해뒀던 번호를 보고 sys_call_table 에서 함수포인터를 구동한다.

```
regs->ax = i386_sys_call_table[nr](
    (unsigned int)regs->bx, (unsigned int)regs->cx,
    (unsigned int)regs->dx, (unsigned int)regs->si,
    (unsigned int)regs->di, (unsigned int)regs->bp);
```

```
1 #
2 # 32-bit system call numbers and entry vectors
3 #
4 # The format is:
5 # <number> <abi> <name> <entry point> <compat entry point>
6 #
7 # The abi is always "i386" for this file.
8 #
9 0      i386      restart_syscall      sys_restart_syscall
10 1      i386      exit                  sys_exit
11 2      i386      fork                  sys_fork          sys_fork
12 3      i386      read                  sys_read
13 4      i386      write                 sys_write
14 5      i386      open                  sys_open          compat_sys_open
15 6      i386      close                 sys_close
16 7      i386      waitpid               sys_waitpid       sys32_waitpid
17 8      i386      creat                 sys_creat
18 9      i386      link                  sys_link
19 10     i386      unlink                 sys_unlink
20 11     i386      execve                sys_execve        compat_sys_execve
21 12     i386      chdir                 sys_chdir
22 13     i386      time                  sys_time          compat_sys_time
```

95 리눅스 커널 기능 중 하드웨어 종속적인 부분들이 구현된 디렉토리이다.

이 디렉토리는 cpu 타입에 따라 하위 디렉토리로 다시 구분된다.

96

exynos – 삼성

omap – ti

zynq – 자일링스

s3c24xx, s3c64xx - 삼성

keystone – dsp

ipc – nxp

bmc – 라즈베리파이

davinci – ti(블랙박스용)

stm32 - cortex R 이 사용

tegra - nvicia 가 사용하는 리눅스 플랫폼

97 ti dsp

98

```
107 */
108 #define switch_to(prev, next, last) \
109     asm volatile(SAVE_CONTEXT
110         "movq %%rsp,%P[threadrsp](%[prev])\n\t" /* save RSP */
111         "movq %P[threadrsp](%[next]),%%rsp\n\t" /* restore RSP */
112         "call __switch_to\n\t"
113         "movq \"__percpu_arg([current_task])\",%%rsi\n\t"
114         __switch_canary
115         "movq %P[thread_info](%rsi),%r8\n\t"
116         "movq %%rax,%rdi\n\t"
117         "testl %[_tif_fork],%P[ti_flags](%r8)\n\t"
118         "jnz ret_from_fork\n\t"
119         RESTORE_CONTEXT
120         : "=a" (last)
121         : [next] "S" (next), [prev] "D" (prev),
122           [threadrsp] "i" (offsetof(struct task_struct, thread.sp)),
123           [ti_flags] "i" (offsetof(struct thread_info, flags)),
124           [_tif_fork] "i" (_TIF_FORK),
125           [thread_info] "i" (offsetof(struct task_struct, stack)),
126           [current_task] "m" (current_task)
127           __switch_canary_iparam
128           : "memory", "cc" __EXTRA_CLOBBER)
129
130
131 #endif /* CONFIG_X86_32 */
132
133 #endif /* _ASM_X86_SWITCH_TO_H */
134
```

```
371
372 struct thread_struct {
373     /* Cached TLS descriptors: */
374     struct desc_struct    tls_array[GDT_ENTRY_TLS_ENTRIES];
375     unsigned long         sp0;
376     unsigned long         sp;
377 #ifdef CONFIG_X86_32
378     unsigned long         sysenter_cs;
379 #else
380     unsigned short        es;
381     unsigned short        ds;
382     unsigned short        fsindex;
383     unsigned short        gsindex;
384 #endif
385 #ifdef CONFIG_X86_32
386     unsigned long         ip;
387 #endif
388 #ifdef CONFIG_X86_64
389     unsigned long         fs;
390 #endif
391     unsigned long         gs;
392
393     /* Save middle states of ptrace breakpoints */
394     struct perf_event      *ptrace_bps[HBP_NUM];
395     /* Debug status used for traps, single steps, etc... */
396     unsigned long          debugreg6;
397     /* Keep track of the exact dr7 value set by the user */
398     unsigned long          ptrace_dr7;
399     /* Fault info: */
400     unsigned long          cr2;
401     unsigned long          trap_nr;
402     unsigned long          error_code;
403 #ifdef CONFIG_VM86
404     /* Virtual 86 mode info */
405     struct vm86             *vm86;
406 #endif
407
```

thread_struct 에서 트랩 넘버를 가지고 있다

99 pc, __vectors_start + 0x1000

```
1209 _vectors_start:
1210     W(b)    vector_rst
1211     W(b)    vector_und
1212     W(ldr)  pc, __vectors_start + 0x1000
1213     W(b)    vector_pabt
1214     W(b)    vector_dabt
1215     W(b)    vector_addrxcptn
1216     W(b)    vector_irq
1217     W(b)    vector_fiq
1218
1219     .data
1220
1221     .globl  cr_alignment
1222 cr_alignment:
1223     .space  4
1224
1225 #ifdef CONFIG_MULTI_IRQ_HANDLER
```

1209,1 99%

100

태어나서 공부를 제일 많이 하는 것 같다. 고 3 때도 6 시간이상씩은 자면서 공부했으니 말이다.

그래도 둘러보면 나보다 훨씬 열심히 하는 사람들이 있어서 자극이 많이 된다.

하루하루 그 날 분량을 흡수하기 위해 노력하지만 그러다 보면 예전에 했던 것들을 많이 잊어버리게 된다.

그래서 주말을 잘 활용해야겠다는 생각을 매번 하지만, 생각에만 그쳤던 적이 훨씬 많은 것 같다.

이제 프로젝트도 있고 공부할 것도 점점 많아지고 하니 다시 복습을 해야 할 필요성을 절실히 느껴진다(당장 오늘 시험에서도..) 다음 주 있을 쉬는 기간을 잘 활용해서 복습을 끝내야겠다