10. Homework
1) Run the program below. What happens? Explain the result.
ex1.c:

```
void main(){
   int x;
   x=fork();
   printf("x:%d\n", x);
}
```

강의노트대로 코드를 작성한 모습이다.

x: 0, x: 4506이 순서대로 출력된 모습이다. 0이 출력된 것이 child process이거 4506이 출력된 것이 parent process인데 이 는 scheduling이 어떻게 되느냐에 따라 순서가 정해진다.

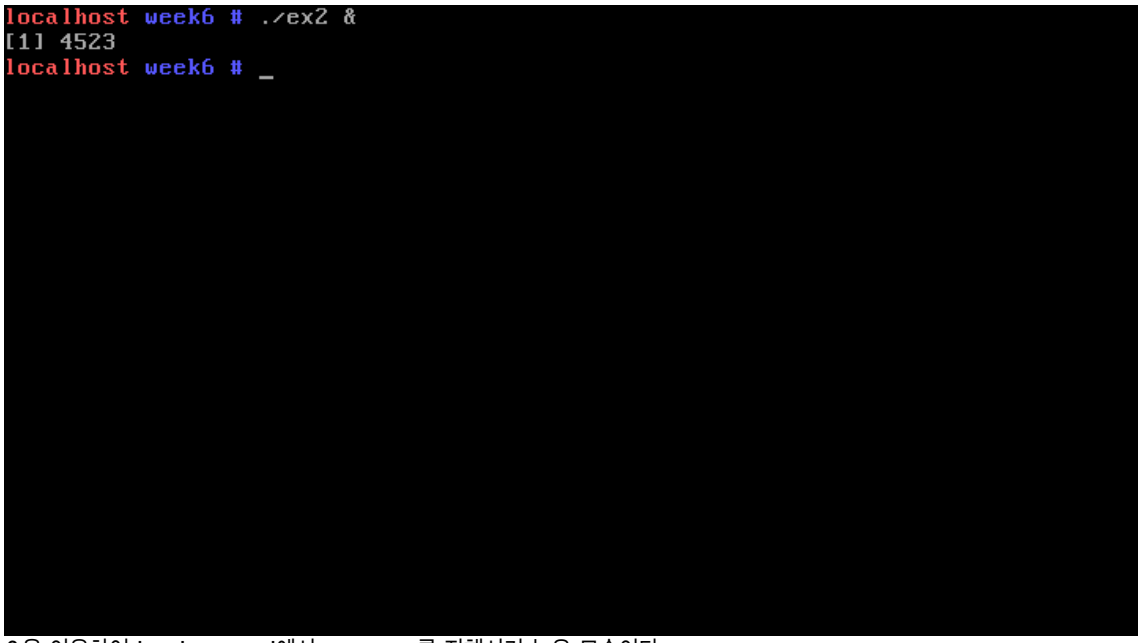
2) Try below and explain the result.

ex1.c:

```
void main(){
   fork();
   fork();
   for(;;);
}
```

```
# gcc –o ex1 ex1.c
# ./ex1 &
# ps –ef
```

강의노트대로 코드를 작성한 모습이다.



&을 이용하여 background에서 process를 진행시켜 놓은 모습이다.

```
root       182     2  0 01:40 ?        00:00:00 [kswapd0]
root       225     2  0 01:40 ?        00:00:00 [aio/0]
root       909     2  0 01:40 ?        00:00:00 [scsi_eh_0]
root       926     2  0 01:40 ?        00:00:00 [khpsbpkt]
root       952     2  0 01:40 ?        00:00:00 [kpsmoused]
root       956     2  0 01:40 ?        00:00:00 [kstriped]
root       959     2  0 01:40 ?        00:00:00 [kondemand/0]
root       968     2  0 01:40 ?        00:00:00 [rpciod/0]
root      1057     2  0 01:40 ?        00:00:00 [kjournald]
root      1155     1  0 01:40 ?        00:00:00 /sbin/udevd --daemon
root      4290     1  0 01:41 ?        00:00:00 /usr/sbin/syslog-ng
root      4405     1  0 01:41 ?        00:00:00 /usr/sbin/cron
root      4469     1  0 01:41 tty1     00:00:00 /bin/login --
root      4471     1  0 01:41 tty2     00:00:00 /sbin/agetty 38400 tty2 linux
root      4473     1  0 01:41 tty3     00:00:00 /sbin/agetty 38400 tty3 linux
root      4475     1  0 01:41 tty4     00:00:00 /sbin/agetty 38400 tty4 linux
root      4477     1  0 01:41 tty5     00:00:00 /sbin/agetty 38400 tty5 linux
root      4479     1  0 01:41 tty6     00:00:00 /sbin/agetty 38400 tty6 linux
root      4488  4469  0 01:41 tty1     00:00:00 -bash
root      4523  4488 17 01:44 tty1     00:00:01 ./ex2
root      4524  4523 17 01:44 tty1     00:00:01 ./ex2
root      4525  4524 17 01:44 tty1     00:00:01 ./ex2
root      4526  4523 17 01:44 tty1     00:00:01 ./ex2
root      4527  4488  0 01:45 tty1     00:00:00 ps -ef
localhost week6 # _
```

original process (4523)의 첫 번째 fork에서 process(4524)가 복제되었다.
그리고 original process (4523)의 두 번째 fork에서 process(4526)이 복제되었다.
original process의 첫 번째 fork에서 복제된 process (4524)의 fork에서 process(4525)가 복제되었다.
총 4개의 process존재하는 상태이다.

3) Run following code. What happens? Explain the result.
ex1.c:

```
void main(){
  int i; float y=3.14;
  fork();
  fork();
  for(;;){
    for(i=0;i<10000000;i++) y=y*0.4;
    printf("%d\n", getpid());
  }
}
```

```
#include <stdio.h>
void main(){
        int i;
        float y = 3.14;
        fork();
        fork();
        for(;;){
                for(i = 0; i < 10000000; i++)
                        y = y * 0.4;
                printf("%d\n", getpid());
        }
}
~
~
~
~
~
~
~
~
~
~
~
~
"ex3.c" [converted] 12L, 166C                                 12,1          All
```

강의노트의 코드를 작성한 모습이다.

```
4614
4617
4616
4614
4615
4617
4616
4615
4614
4616
4617
4615
4614
4616
4617
4614
4615
4616
4617
4616
4614
4615
4617
4614
```

fork가 연속으로 2번 호출되어 4개의 process가 된다. 4514, 4515, 4516, 4517이라는 4개의 process가 존재하는데 이 들이 각각 scheduling에 따라 순서가 정해져서 process가 진행 되는 모습이다.

4) Try below and explain the result.

ex1.c:

```
void main(){
    char *argv[10];
  argv[0]=”./ex2”;
  argv[1]=0;
  execve(argv[0], argv, 0);
}
```

ex2.c

```
void main(){
```

```
   printf("korea\n");
}
#gcc –o ex1 ex1.c
#gcc –o ex2 ex2.c
#./ex1
```

```
#include <stdio.h>
void main(){
        char* argv[10];
        argv[0]="./ex4_2";
        argv[1]=0;
        execve(argv[0], argv, 0);
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"ex4_1.c" [converted] 7L, 110C                          4,2-9         All
#include <stdio.h>
void main(){
        printf("korea\n");
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"ex4_2.c" [converted] 4L, 54C                           4,1           All
```

강의노트의 코드를 작성한 모습이다.

```
localhost week6 # ./ex4_1
korea
localhost week6 # _
```

첫 번째 코드를 실행시키면 exec으로 두 번째 코드를 호출하기 때문에 두 번째 코드가 실행되고 korea가 출력된 것을 볼 수 있다.

5) Run following code and explain the result.

```
void main(){
    char *argv[10];
  argv[0]=”/bin/ls”;
  argv[1]=0;
  execve(argv[0], argv, 0);
}
```

```
#include <stdio.h>
void main(){
        char* argv[10];
        argv[0]="/bin/ls";
        argv[1]=0;
        execve(argv[0],argv,0);
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                    6,24-31       All
```

강의노트대로 코드를 작성한 모습이다.
/bin/ls인 인자를 1개 넘겨주어 exec을 하는 코드이다.

```
localhost week6 # ./ex5
ex1  ex1.c  ex2  ex2.c  ex3  ex3.c  ex4_1  ex4_1.c  ex4_2  ex4_2.c  ex5  ex5.c
localhost week6 # _
```

ls를 실행했을 때와 동일한 결과를 볼 수 있다.

6) Run following code and explain the result.

```
void main(){
    char *argv[10];
  argv[0]=”/bin/ls”;
  argv[1]="-l";
  argv[2]=0;
  execve(argv[0], argv, 0);
}
```

```
#include <stdio.h>
void main(){
        char* argv[10];
        argv[0]="/bin/ls";
        argv[1]="-l";
        argv[2]=0;
        execve(argv[0], argv, 0);
}
```

강의노트대로 코드를 작성한 모습이다.
/bin/ls와 -l의 인자 2개를 넘겨주어 exec를 하는 모습이다.

```
localhost week6 # ./ex6
total 84
-rwxr-xr-x 1 root root 6934 Oct  4 01:42 ex1
-rw-r--r-- 1 root root   78 Oct  4 01:42 ex1.c
-rwxr-xr-x 1 root root 6896 Oct  4 01:43 ex2
-rw-r--r-- 1 root root   62 Oct  4 01:43 ex2.c
-rwxr-xr-x 1 root root 6972 Oct  4 01:47 ex3
-rw-r--r-- 1 root root  166 Oct  4 01:47 ex3.c
-rwxr-xr-x 1 root root 6900 Oct 11 02:19 ex4_1
-rw-r--r-- 1 root root  110 Oct 11 02:19 ex4_1.c
-rwxr-xr-x 1 root root 6898 Oct 11 02:18 ex4_2
-rw-r--r-- 1 root root   54 Oct 11 02:19 ex4_2.c
-rwxr-xr-x 1 root root 6898 Oct 11 02:20 ex5
-rw-r--r-- 1 root root  108 Oct 11 02:20 ex5.c
-rwxr-xr-x 1 root root 6898 Oct 11 02:21 ex6
-rw-r--r-- 1 root root  125 Oct 11 02:21 ex6.c
localhost week6 #
```

ls -l를 실행했을 때와 동일한 결과를 볼 수 있다.

11. Homework

1) Try the shell code in section 7. Try Linux command such as "/bin/ls", "/bin/date", etc.

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
void main(){
        int x,y;
        char buf[50];
        char* argv[2];
        for(;;){
                printf("$");
                scanf("%s", buf);
                argv[0]=buf;
                argv[1]=NULL;

                x=fork();
                if(x==0){
                        printf("I am child to execute %s\n", buf);
                        y = execve(buf, argv, 0);
                        if(y<0){
                                printf("exec failed. errno is %d\n", errno);
                                exit(1);
                        }
                } else wait();
        }
}
"ex7.c" [converted] 24L, 389C                                 24,1          All
```

강의노트대로 코드를 작성한 모습이다.

fork를 하여 child process로 복제 해주고 parent process는 child process가 죽을 때 까지 wait하며, child process는 exec으로 입력받은 것을 실행하는 코드이다.

```
localhost week6 # ./ex7
$/bin/ls
I am child to execute /bin/ls
ex1    ex2    ex3    ex4_1    ex4_2    ex5    ex6    ex7
ex1.c  ex2.c  ex3.c  ex4_1.c  ex4_2.c  ex5.c  ex6.c  ex7.c
$/bin/date
I am child to execute /bin/date
Thu Oct 11 02:28:40 KST 2018
$
```

/bin/ls를 입력했을 때 ls 와 같은 결과를, /bin/date를 입력했을 때 date와 같은 결과가 보여지는 것을 알 수 있다.

2) Print the pid of the current process (current->pid) inside rest_init() and kernel_init(). The pid printed inside rest_init() will be 0, but the pid inside kernel_init() is 1. 0 is the pid of the kernel itself. Why do we have pid=1 inside kernel_init()?

```
        run_init_process("/bin/init");
        run_init_process("/bin/sh");

        panic("No init found.  Try passing init= option to kernel.");
}

static int __init kernel_init(void * unused)
{
        printk("kernel_init pid : %d\n", current->pid);

        lock_kernel();
        /*
         * init can run on any cpu.
         */
        set_cpus_allowed(current, CPU_MASK_ALL);
        /*
         * Tell the world that we're going to be the grim
         * reaper of innocent orphaned children.
         *
         * We don't want people to have to make incorrect
         * assumptions about where in the task array this
         * can be found.
         */
        init_pid_ns.child_reaper = current;
                                                        830,1         95%
```

kernel_init() 함수 안에 pid를 출력하는 코드를 추가한 모습이다.

```
 * We need to finalize in a non-__init function or else race conditions
 * between the root thread and the init thread may cause start_kernel to
 * be reaped by free_initmem before the root thread has proceeded to
 * cpu_idle.
 *
 * gcc-3.4 accidentally inlines this function, so use noinline.
 */

static void noinline __init_refok rest_init(void)
        __releases(kernel_lock)
{
        printk("rest_init pid : %d\n", current->pid);

        int pid;

        kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);
        numa_default_policy();
        pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
        kthreadd_task = find_task_by_pid(pid);
        unlock_kernel();

        /*
         * The boot idle thread must execute schedule()
         * at least once to get things moving:
                                                      435,1         49%
```

rest_init() 함수 안에 pid를 출력하는 코드를 추가한 모습이다.

```
Checking 'hlt' instruction... OK.
SMP alternatives: switching to UP code
Freeing SMP alternatives: 17k freed
ACPI: Core revision 20070126
Parsing all Control Methods:
Table [DSDT](id 0001) - 253 Objects with 28 Devices 80 Methods 4 Regions
Parsing all Control Methods:
Table [SSDT](id 0002) - 0 Objects with 0 Devices 0 Methods 0 Regions
 tbxface-0598 [00] tb_load_namespace     : ACPI Tables successfully acquired
ACPI: setting ELCR to 0200 (from 0e00)
evxfevnt-0091 [00] enable               : Transition to ACPI mode successful
rest_init pid : 0
kernel_init pid : 1
CPU0: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz stepping 01
SMP motherboard not detected.
Brought up 1 CPUs
net_namespace: 244 bytes
NET: Registered protocol family 16
No dock devices found.
ACPI: bus type pci registered
PCI: PCI BIOS revision 2.10 entry at 0xfda26, last bus=0
PCI: Using configuration type 1
Setting up standard PCI resources
evgpeblk-0956 [00] ev_create_gpe_block    : GPE 00 to 07 [_GPE] 1 regs on int 0x9
                                                      110,11        39%
```

res_init의 pid는 0, kernel_init의 pid는 1로 출력된 모습이다.
이는 res_init에서 kernel_init을 호출 할 때 kernel_thread(kernel_init) 형식으로 호출하는데, 이 때 process를 복제하여 호출하기 때문에 process가 하나 더 생긴 것이다.

3) What happens if the kernel calls "kernel_init" directly instead of calling kernel_thread(kernel_ init, ...) in rest_init()? Trace the kernel code and show where the kernel falls into panic.

```
 *
 * gcc-3.4 accidentally inlines this function, so use noinline.
 */

static void noinline __init_refok rest_init(void)
        __releases(kernel_lock)
{
        printk("rest_init pid : %d\n", current->pid);

        int pid;

        kernel_init(NULL);
//      kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);

        numa_default_policy();
        pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
        kthreadd_task = find_task_by_pid(pid);
        unlock_kernel();

        /*
         * The boot idle thread must execute schedule()
         * at least once to get things moving:
         */
        init_idle_bootup_task(current);
```

원래 kernel_thread(kernel_init) 부분을 주석처리하고 바로 kernel_init()을 호출한 모습이다.

```
EAX: f000ff53 EBX: c0605380 ECX: 00000000 EDX: c05b9f00
ESI: c0605380 EDI: c05b9f00 EBP: 00000000 ESP: c05b9ee8
 DS: 007b ES: 007b FS: 00d8 GS: 0000 SS: 0068
Process swapper (pid: 0, ti=c05b8000 task=c05553a0 task.ti=c05b8000)
Stack: 00000000 c0605818 c1208818 00000000 c0118084 0000000f 00000296 00000000
       c0605818 c1208818 00000000 c012c7e8 c011a480 00000000 00000000 00000000
       c05b9f28 c05b9f28 00000000 00000000 00000000 c05b9f3c c05b9f3c c0559a34
Call Trace:
 [<c0118084>] try_to_wake_up+0x17/0xd0
 [<c012c7e8>] kthread_create+0x65/0xa7
 [<c011a480>] migration_thread+0x0/0x18a
 [<c0410a5c>] migration_call+0x2d/0x3a5
 [<c011a480>] migration_thread+0x0/0x18a
 [<c05ccaa9>] migration_init+0x19/0x40
 [<c03f983c>] rest_init+0x6c/0x2c7
 [<c0236b3a>] acpi_enable_subsystem+0x3a/0x110
 [<c05bf5de>] start_kernel+0x2cb/0x2d2
 =======================
Code: 00 00 e8 f2 e6 ff ff 89 da 89 e8 5b 5e 5f 5d e9 45 bf 2f 00 55 89 c5 57 89
 d7 56 53 9c 58 fa 89 07 bb 80 53 60 c0 8b 45 04 89 de <8b> 40 10 03 34 85 00 40
 5b c0 89 f0 e8 11 be 2f 00 8b 45 04 8b
EIP: [<c0117e94>] task_rq_lock+0x17/0x4b SS:ESP 0068:c05b9ee8
---[ end trace ca143223eefdc828 ]---
Kernel panic - not syncing: Attempted to kill the idle task!
_
```

정상적으로 부팅이 되지 않은 모습이다. 이는 kernel_init을 호출할 때 process를 증가시키지 않고 그냥 진행하였는데, 나중에 process를 하나 삭제할 때 1개 밖에 없는 데 삭제하려고 시도했기 때문에 오류가 발생한 모습이다.

4) The last function call in start_kernel() is rest_init(). If you insert printk() after rest_init(), it is not displayed during the system booting. Explain the reason.

```
void start_kernel(){
    .............
    printk("before rest_init\n");  // this will be printed out
    rest_init();
    printk("after rest_init\n");   // but this will not.
}
```

```
        cgroup_init();
        cpuset_init();
        taskstats_init_early();
        delayacct_init();

        check_bugs();

        acpi_early_init(); /* before LAPIC and SMP init */


        /* Do the rest non-__init'ed, we're now alive */
        printk("before rest_init\n");
        rest_init();
        printk("agter rest_init\n");
}

static int __initdata initcall_debug;

static int __init initcall_debug_setup(char *str)
{
        initcall_debug = 1;
        return 1;
}
__setup("initcall_debug", initcall_debug_setup);
                                                        673,29-36     76%
```

rest_init()의 앞 뒤로 출력문을 추가한 모습이다.

```
Checking 'hlt' instruction... OK.
SMP alternatives: switching to UP code
Freeing SMP alternatives: 17k freed
ACPI: Core revision 20070126
Parsing all Control Methods:
Table [DSDT](id 0001) - 253 Objects with 28 Devices 80 Methods 4 Regions
Parsing all Control Methods:
Table [SSDT](id 0002) - 0 Objects with 0 Devices 0 Methods 0 Regions
 tbxface-0598 [00] tb_load_namespace     : ACPI Tables successfully acquired
ACPI: setting ELCR to 0200 (from 0e00)
evxfevnt-0091 [00] enable               : Transition to ACPI mode successful
before rest init
kernel_init pid : 1
CPU0: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz stepping 01
SMP motherboard not detected.
Brought up 1 CPUs
net_namespace: 244 bytes
NET: Registered protocol family 16
No dock devices found.
ACPI: bus type pci registered
PCI: PCI BIOS revision 2.10 entry at 0xfda26, last bus=0
PCI: Using configuration type 1
Setting up standard PCI resources
evgpeblk-0956 [00] ev_create_gpe_block   : GPE 00 to 07 [_GPE] 1 regs on int 0x9
E486: Pattern not found: after                                  110,1         38%
```

부팅 후 dmesg > x | vi x 를 이용하여 부팅메시지를 본 모습이다.
before 출력문을 출력되었지만 after 출력문을 not found 된 모습이다. 이는 cpu_idle에서 무한 loop를 돌며 진행되기 때문에 그 이후로는 진행되지 않아 출력이 되지 않은 모습이다.

5) The CPU is either in some application program or in Linux kernel. You always should be able to say where is the CPU currently. Suppose we have a following program (ex1.c).

```
void main(){
  printf("korea\n");
}
```

When the shell runs this, CPU could be in shell program or in ex1 or in kernel. Explain where is CPU for each step of this program. Start the tracing from the moment when the shell prints a prompt until it prints next prompt.

```
shell: printf("$");                 // CPU is in shell
    => write(1, "$", 1)             // CPU is in c library
    => INT 128                      // CPU is in c library
kernel:
     sys_write()                    // CPU is in kernel and display '$' in screen
                                    // after sys_write() kernel schedules shell again
```

                                // and CPU goes back to shell
shell: scanf("%s", buf);   // CPU is in shell
.....
.....

printf(“$”) (CPU in shell)
→ write (CPU in C library))
→ sys_write (CPU in kernel)

scanf(“%s”, buf) (CPU in shell)
→ read (CPU in C library)
→ sys_read (CPU in kernel)

6) Trace fork, exec, exit, wait system call to find the corresponding code for the major steps of each system call.

fork

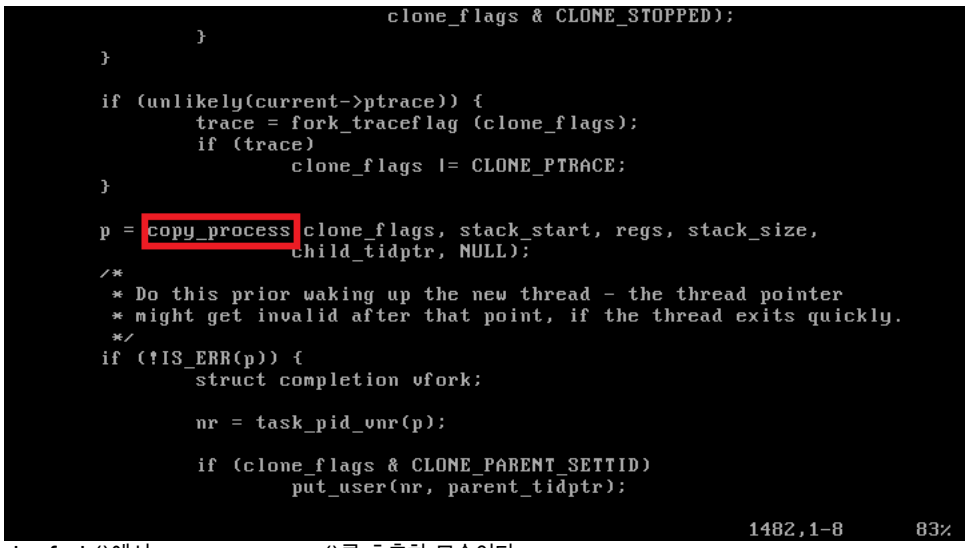
```
        /*
         * Restore %gs if needed (which is common)
         */
        if (prev->gs | next->gs)
                loadsegment(gs, next->gs);

        x86_write_percpu(current_task, next_p);

        return prev_p;
}

asmlinkage int sys_fork(struct pt_regs regs)
{
        return do_fork(SIGCHLD, regs.sp, &regs, 0, NULL, NULL);
}

asmlinkage int sys_clone(struct pt_regs regs)
{
        unsigned long clone_flags;
        unsigned long newsp;
        int __user *parent_tidptr, *child_tidptr;

        clone_flags = regs.bx;
        newsp = regs.cx;
                                                        750,16        89%
```

sys_fork() 함수에서 do_fork()를 호출한 모습이다.

```
/*
 *  Ok, this is the main fork-routine.
 *
 * It copies the process, and if successful kick-starts
 * it and waits for it to finish using the VM if required.
 */
long do_fork(unsigned long clone_flags,
              unsigned long stack_start,
              struct pt_regs *regs,
              unsigned long stack_size,
              int __user *parent_tidptr,
              int __user *child_tidptr)
{
        struct task_struct *p;
        int trace = 0;
        long nr;

        /*
         * We hope to recycle these flags after 2.6.26
         */
        if (unlikely(clone_flags & CLONE_STOPPED)) {
                static int __read_mostly count = 100;

                if (count > 0 && printk_ratelimit()) {
                                                        1447,3        81%
```

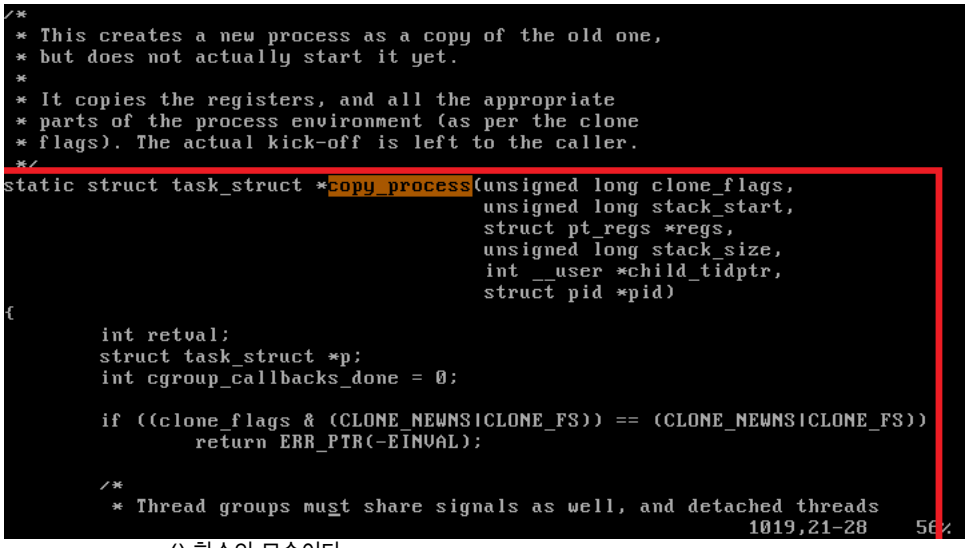
do_fork() 함수의 모습이다.

전체적인 fork의 진행을 포함한다.



```
                                clone_flags & CLONE_STOPPED);
                }
        }

        if (unlikely(current->ptrace)) {
                trace = fork_traceflag (clone_flags);
                if (trace)
                        clone_flags |= CLONE_PTRACE;
        }

        p = copy_process(clone_flags, stack_start, regs, stack_size,
                         child_tidptr, NULL);
        /*
         * Do this prior waking up the new thread - the thread pointer
         * might get invalid after that point, if the thread exits quickly.
         */
        if (!IS_ERR(p)) {
                struct completion vfork;

                nr = task_pid_vnr(p);

                if (clone_flags & CLONE_PARENT_SETTID)
                        put_user(nr, parent_tidptr);

                                                        1482,1-8      83%
```

do_fork()에서 copy_process()를 호출한 모습이다.



```
/*
 * This creates a new process as a copy of the old one,
 * but does not actually start it yet.
 *
 * It copies the registers, and all the appropriate
 * parts of the process environment (as per the clone
 * flags). The actual kick-off is left to the caller.
 */
static struct task_struct *copy_process(unsigned long clone_flags,
                                        unsigned long stack_start,
                                        struct pt_regs *regs,
                                        unsigned long stack_size,
                                        int __user *child_tidptr,
                                        struct pid *pid)
{
        int retval;
        struct task_struct *p;
        int cgroup_callbacks_done = 0;

        if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
                return ERR_PTR(-EINVAL);

        /*
         * Thread groups must share signals as well, and detached threads
                                                        1019,21-28    56%
```

copy_process() 함수의 모습이다.
process를 복사해서 빈 메모리로 넣는 기능이다.

```
         * for various simplifications in other code.
         */
        if ((clone_flags & CLONE_SIGHAND) && !(clone_flags & CLONE_VM))
                return ERR_PTR(-EINVAL);

        retval = security_task_create(clone_flags);
        if (retval)
                goto fork_out;

        retval = -ENOMEM;
        p = dup_task_struct(current);
        if (!p)
                goto fork_out;

        rt_mutex_init_task(p);

#ifdef CONFIG_TRACE_IRQFLAGS
        DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
        DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
#endif
        retval = -EAGAIN;
        if (atomic_read(&p->user->processes) >=
                        p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
                if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
                                                        1038,30-37    58%
```

dup_task_struct()를 호출한 모습이다.

```
static struct task_struct *dup_task_struct(struct task_struct *orig)
{
        struct task_struct *tsk;
        struct thread_info *ti;
        int err;

        prepare_to_copy(orig);

        tsk = alloc_task_struct();
        if (!tsk)
                return NULL;

        ti = alloc_thread_info(tsk);
        if (!ti) {
                free_task_struct(tsk);
                return NULL;
        }

        *tsk = *orig;
        tsk->stack = ti;

        err = prop_local_init_single(&tsk->dirties);
        if (err) {
                                                        188,11-18      9%
```

dup_task_struct() 함수의 모습이다.
현재 process를 복사하는 기능이다.

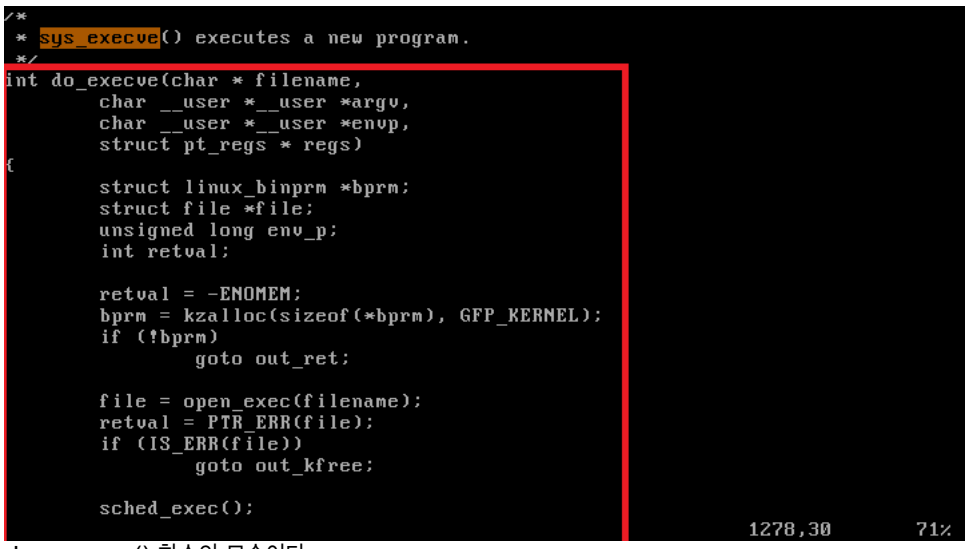
```
                if ((p->ptrace & PT_PTRACED) || (clone_flags & CLONE_STOPPED)) {
                        /*
                         * We'll start up with an immediate SIGSTOP.
                         */
                        sigaddset(&p->pending.signal, SIGSTOP);
                        set_tsk_thread_flag(p, TIF_SIGPENDING);
                }

                if (!(clone_flags & CLONE_STOPPED))
                        wake_up_new_task(p, clone_flags);
                else
                        __set_task_state(p, TASK_STOPPED);

                if (unlikely (trace)) {
                        current->ptrace_message = nr;
                        ptrace_notify ((trace << 8) | SIGTRAP);
                }

                if (clone_flags & CLONE_VFORK) {
                        freezer_do_not_count();
                        wait_for_completion(&vfork);
                        freezer_count();
                        if (unlikely (current->ptrace & PT_TRACE_VFORK_DONE)) {
                                                                1512,1-8      84%
```

남은 과정을 진행하는 부분이다.

execve

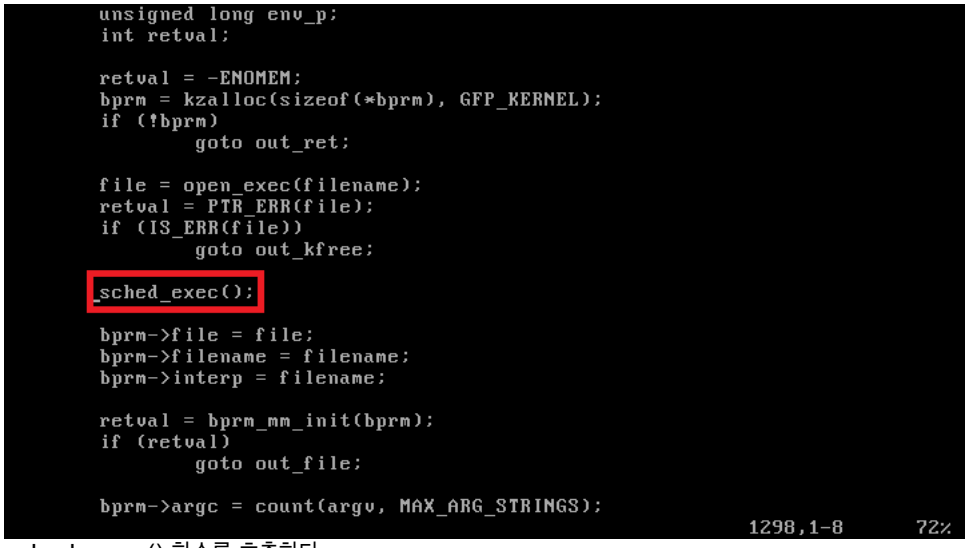
```
/*
 * sys_execve() executes a new program.
 */
asmlinkage int sys_execve(struct pt_regs regs)
{
        int error;
        char * filename;

        filename = getname((char __user *) regs.bx);
        error = PTR_ERR(filename);
        if (IS_ERR(filename))
                goto out;
        error = do_execve(filename,
                        (char __user * __user *) regs.cx,
                        (char __user * __user *) regs.dx,
                        &regs);
        if (error == 0) {
                /* Make sure we don't return using sysenter.. */
                set_thread_flag(TIF_IRET);
        }
        putname(filename);
out:
        return error;
}
                                                                788,1         95%
```

sys_execve() 에서 do_execve() 함수를 호출하는 모습이다.



```
/*
 * sys_execve() executes a new program.
 */
int do_execve(char * filename,
        char __user *__user *argv,
        char __user *__user *envp,
        struct pt_regs * regs)
{
        struct linux_binprm *bprm;
        struct file *file;
        unsigned long env_p;
        int retval;

        retval = -ENOMEM;
        bprm = kzalloc(sizeof(*bprm), GFP_KERNEL);
        if (!bprm)
                goto out_ret;

        file = open_exec(filename);
        retval = PTR_ERR(file);
        if (IS_ERR(file))
                goto out_kfree;

        sched_exec();
                                                    1278,30       71%
```

do_execve() 함수의 모습이다.
전반 적인 execve 진행과정을 가지고 있다.
open_exec() 함수를 호출하여 해당 파일을 열어 내용을 읽는다.



```
        unsigned long env_p;
        int retval;

        retval = -ENOMEM;
        bprm = kzalloc(sizeof(*bprm), GFP_KERNEL);
        if (!bprm)
                goto out_ret;

        file = open_exec(filename);
        retval = PTR_ERR(file);
        if (IS_ERR(file))
                goto out_kfree;

        sched_exec();

        bprm->file = file;
        bprm->filename = filename;
        bprm->interp = filename;

        retval = bprm_mm_init(bprm);
        if (retval)
                goto out_file;

        bprm->argc = count(argv, MAX_ARG_STRINGS);
                                                    1298,1-8      72%
```

sched_exec() 함수를 호출한다.

```
                wake_up_process(mt);
                put_task_struct(mt);
                wait_for_completion(&req.done);

                return;
        }
out:
        task_rq_unlock(rq, &flags);
}

/*
 * sched_exec - execve() is a valuable balancing opportunity, because at
 * this point the task has the smallest effective memory and cache footprint.
 */
void sched_exec(void)
{
        int new_cpu, this_cpu = get_cpu();
        new_cpu = sched_balance_self(this_cpu, SD_BALANCE_EXEC);
        put_cpu();
        if (new_cpu != this_cpu)
                sched_migrate_task(current, new_cpu);
}

/*
                                                        2532,2        30%
```

sched_exec() 함수에서는 scheduling을 해주는 부분이다.

```
        retval = copy_strings_kernel(1, &bprm->filename, bprm);
        if (retval < 0)
                goto out;

        bprm->exec = bprm->p;
        retval = copy_strings(bprm->envc, envp, bprm);
        if (retval < 0)
                goto out;

        env_p = bprm->p;
        retval = copy_strings(bprm->argc, argv, bprm);
        if (retval < 0)
                goto out;
        bprm->argv_len = env_p - bprm->p;

        retval = search_binary_handler(bprm,regs);
        if (retval >= 0) {
                /* execve success */
                free_arg_pages(bprm);
                security_bprm_free(bprm);
                acct_update_integrals(current);
                kfree(bprm);
                return retval;
        }
                                                        1324,2-9      74%
```

do_execve() 함수의 나머지 부분이다.

exit

```
EXPORT_SYMBOL_GPL(do_exit);

NORET_TYPE void complete_and_exit(struct completion *comp, long code)
{
        if (comp)
                complete(comp);

        do_exit(code);
}

EXPORT_SYMBOL(complete_and_exit);

asmlinkage long sys_exit(int error_code)
{
        do_exit((error_code&0xff)<<8);
}

/*
 * Take down every thread in the group.  This is called by fatal signals
 * as well as by sys_exit_group (below).
 */
NORET_TYPE void
do_group_exit(int exit_code)
{
                                                        1039,0-1      62%
```

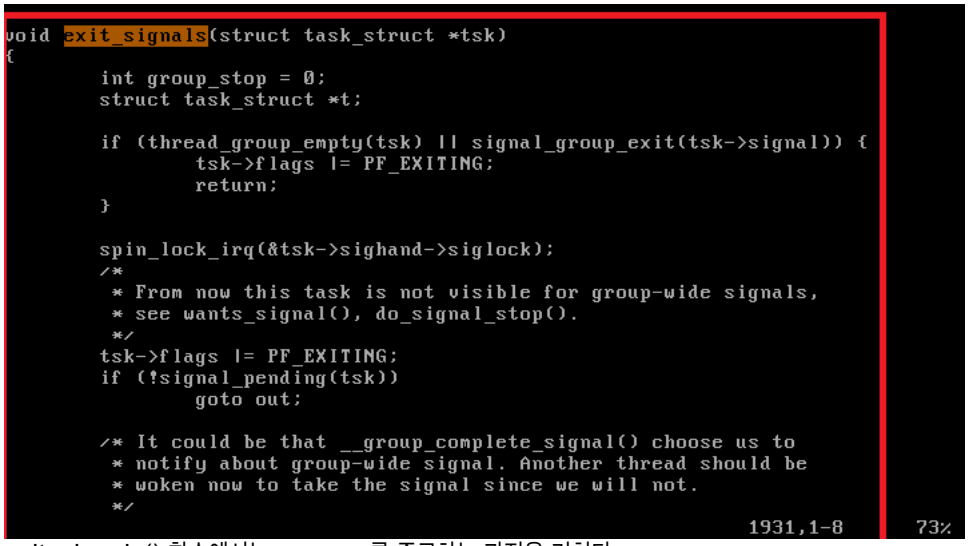
sys_exit()에서 do_exit()를 호출하는 모습이다.

```
NORET_TYPE void do_exit(long code)
{
        struct task_struct *tsk = current;
        int group_dead;

        profile_task_exit(tsk);

        WARN_ON(atomic_read(&tsk->fs_excl));

        if (unlikely(in_interrupt()))
                panic("Aiee, killing interrupt handler!");
        if (unlikely(!tsk->pid))
                panic("Attempted to kill the idle task!");

        if (unlikely(current->ptrace & PT_TRACE_EXIT)) {
                current->ptrace_message = code;
                ptrace_notify((PTRACE_EVENT_EXIT << 8) | SIGTRAP);
        }

        /*
         * We're taking recursive faults here in do_exit. Safest is to just
         * leave this task alone and wait for reboot.
         */
        if (unlikely(tsk->flags & PF_EXITING)) {
                                                        905,2-9       53%
```

do_exit() 함수의 모습이다.
전체적인 exit 진행과정을 가지고 있다.

```
                tsk->flags |= PF_EXITPIDONE;
                if (tsk->io_context)
                        exit_io_context();
                set_current_state(TASK_UNINTERRUPTIBLE);
                schedule();
        }

        exit_signals(tsk);  /* sets PF_EXITING */
        /*
         * tsk->flags are checked in the futex code to protect against
         * an exiting task cleaning up the robust pi futexes.
         */
        smp_mb();
        spin_unlock_wait(&tsk->pi_lock);

        if (unlikely(in_atomic()))
                printk(KERN_INFO "note: %s[%d] exited with preempt_count %d\n",
                                current->comm, task_pid_nr(current),
                                preempt_count());

        acct_update_integrals(tsk);
        if (tsk->mm) {
                update_hiwater_rss(tsk->mm);
                update_hiwater_vm(tsk->mm);
                                                        928,2-9       56%
```

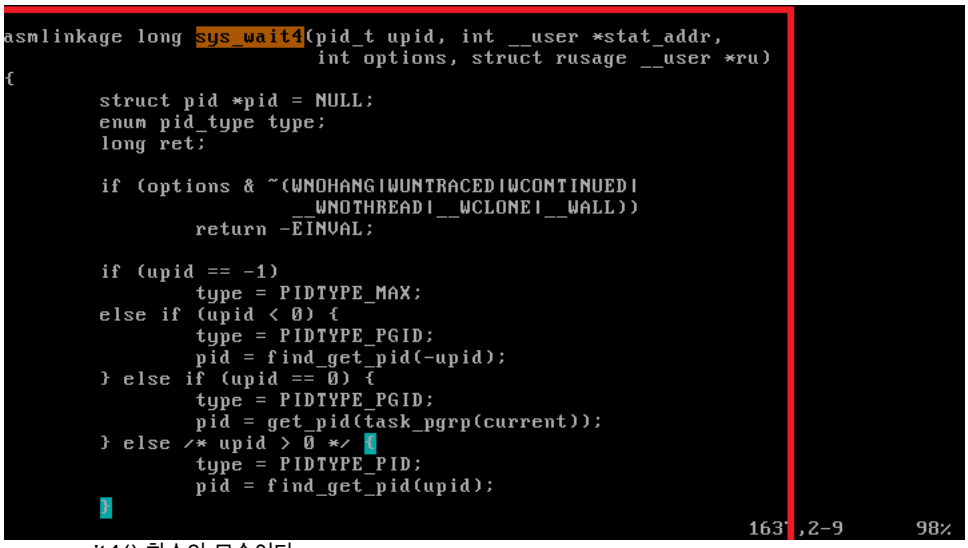
exit_signals() 함수를 호출하는 모습이다.



```
void exit_signals(struct task_struct *tsk)
{
        int group_stop = 0;
        struct task_struct *t;

        if (thread_group_empty(tsk) || signal_group_exit(tsk->signal)) {
                tsk->flags |= PF_EXITING;
                return;
        }

        spin_lock_irq(&tsk->sighand->siglock);
        /*
         * From now this task is not visible for group-wide signals,
         * see wants_signal(), do_signal_stop().
         */
        tsk->flags |= PF_EXITING;
        if (!signal_pending(tsk))
                goto out;

        /* It could be that __group_complete_signal() choose us to
         * notify about group-wide signal. Another thread should be
         * woken now to take the signal since we will not.
         */
                                                        1931,1-8      73%
```

exit_signals() 함수에서는 process를 종료하는 과정을 거친다.

wait



```
asmlinkage long sys_wait4(pid_t upid, int __user *stat_addr,
                          int options, struct rusage __user *ru)
{
        struct pid *pid = NULL;
        enum pid_type type;
        long ret;

        if (options & ~(WNOHANG|WUNTRACED|WCONTINUED|
                        __WNOTHREAD|__WCLONE|__WALL))
                return -EINVAL;

        if (upid == -1)
                type = PIDTYPE_MAX;
        else if (upid < 0) {
                type = PIDTYPE_PGID;
                pid = find_get_pid(-upid);
        } else if (upid == 0) {
                type = PIDTYPE_PGID;
                pid = get_pid(task_pgrp(current));
        } else /* upid > 0 */ {
                type = PIDTYPE_PID;
                pid = find_get_pid(upid);
        }
                                                        1637,2-9      98%
```

sys_wait4() 함수의 모습이다.

```
        if (upid == -1)
                type = PIDTYPE_MAX;
        else if (upid < 0) {
                type = PIDTYPE_PGID;
                pid = find_get_pid(-upid);
        } else if (upid == 0) {
                type = PIDTYPE_PGID;
                pid = get_pid(task_pgrp(current));
        } else /* upid > 0 */ {
                type = PIDTYPE_PID;
                pid = find_get_pid(upid);
        }

        ret = do_wait(type, pid, options | WEXITED, NULL, stat_addr, ru);
        put_pid(pid);

        /* avoid REGPARM breakage on x86: */
        asmlinkage_protect(4, ret, upid, stat_addr, options, ru);
        return ret;
}

#ifdef __ARCH_WANT_SYS_WAITPID
                                                          1638,0-1      99%
```

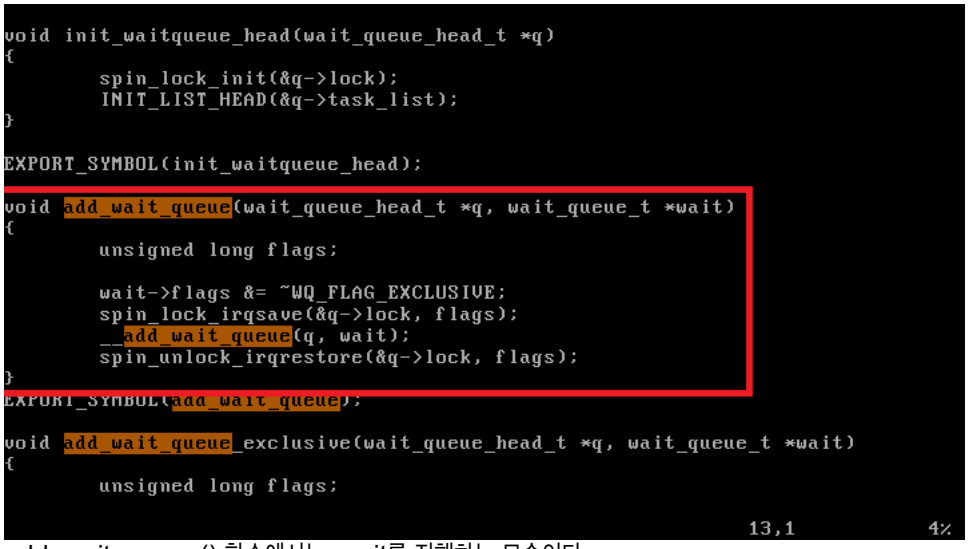
do_wait() 함수를 호출하는 모습이다.

```
static long do_wait(enum pid_type type, struct pid *pid, int options,
                    struct siginfo __user *infop, int __user *stat_addr,
                    struct rusage __user *ru)
{
        DECLARE_WAITQUEUE(wait, current);
        struct task_struct *tsk;
        int flag, retval;

        add_wait_queue(&current->signal->wait_chldexit,&wait);
repeat:
        /* If there is nothing that can match our critier just get out */
        retval = -ECHILD;
        if ((type < PIDTYPE_MAX) && (!pid || hlist_empty(&pid->tasks[type])))
                goto end;

        /*
         * We will set this flag if we see any child that might later
         * match our criteria, even if we are not able to reap it yet.
         */
        flag = retval = 0;
        current->state = TASK_INTERRUPTIBLE;
        read_lock(&tasklist_lock);
        tsk = current;
                                                          1467,6-13     83%
```

do_wait() 함수의 모습이다.
전체적인 wait 의 진행과정을 가지고 있다.

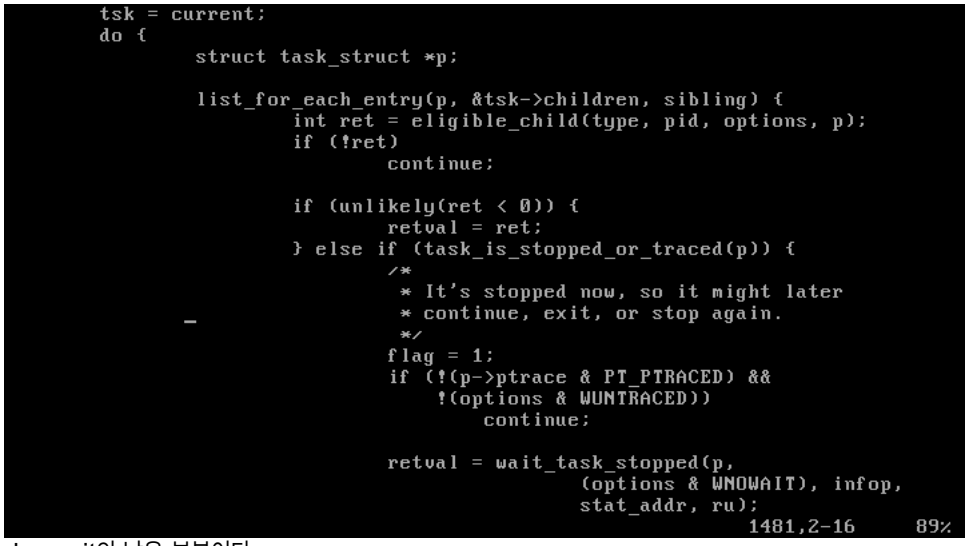
```
                    struct rusage __user *ru)
{
        DECLARE_WAITQUEUE(wait, current);
        struct task_struct *tsk;
        int flag, retval;

        add_wait_queue(&current->signal->wait_chldexit,&wait);
repeat:
        /* If there is nothing that can match our critier just get out */
        retval = -ECHILD;
        if ((type < PIDTYPE_MAX) && (!pid || hlist_empty(&pid->tasks[type])))
                goto end;

        /*
         * We will set this flag if we see any child that might later
         * match our criteria, even if we are not able to reap it yet.
         */
        flag = retval = 0;
        current->state = TASK_INTERRUPTIBLE;
        read_lock(&tasklist_lock);
        tsk = current;
        do {
                struct task_struct *p;

                                                          1453,2-9      88%
```

add_wait_queue() 함수를 호출하는 모습이다.



```
void init_waitqueue_head(wait_queue_head_t *q)
{
        spin_lock_init(&q->lock);
        INIT_LIST_HEAD(&q->task_list);
}

EXPORT_SYMBOL(init_waitqueue_head);

void add_wait_queue(wait_queue_head_t *q, wait_queue_t *wait)
{
        unsigned long flags;

        wait->flags &= ~WQ_FLAG_EXCLUSIVE;
        spin_lock_irqsave(&q->lock, flags);
        __add_wait_queue(q, wait);
        spin_unlock_irqrestore(&q->lock, flags);
}
EXPORT_SYMBOL(add_wait_queue);

void add_wait_queue_exclusive(wait_queue_head_t *q, wait_queue_t *wait)
{
        unsigned long flags;
                                                              13,1          4%
```

add_wait_queue() 함수에서는 wait를 진행하는 모습이다.



```
        tsk = current;
        do {
                struct task_struct *p;

                list_for_each_entry(p, &tsk->children, sibling) {
                        int ret = eligible_child(type, pid, options, p);
                        if (!ret)
                                continue;

                        if (unlikely(ret < 0)) {
                                retval = ret;
                        } else if (task_is_stopped_or_traced(p)) {
                                /*
                                 * It's stopped now, so it might later
                                 * continue, exit, or stop again.
                                 */
                                flag = 1;
                                if (!(p->ptrace & PT_PTRACED) &&
                                    !(options & WUNTRACED))
                                        continue;

                                retval = wait_task_stopped(p,
                                                (options & WNOWAIT), infop,
                                                stat_addr, ru);
                                                              1481,2-16     89%
```

do_wait의 남은 부분이다.