Universidad de Buenos Aires
Facultad de Ingeniería

Sistemas operativos (75.08)

TP3: Multitarea con desalojo.

Alumno	**Padrón**
Martín Bosch	96749
Estanislao Ledesma	96622

Primer cuatrimestre 2017

Git diff

```
martin@martin:~/GitHub/sisop$ git diff --stat base_tp3
 jos/TP3.md                                  |  75
++++++++++++++++++++++++++++++++++++--
 jos/TP3.pdf                                 | Bin 0 -> 126390 bytes
 jos/__pycache__/gradelib.cpython-35.pyc | Bin 0 -> 18646 bytes
 jos/grade-lab4                              |   0
 jos/inc/memlayout.h                         |   3 ++
 jos/kern/Makefrag                           |   1 +
 jos/kern/env.c                              |  11 +++++-
 jos/kern/init.c                             |  33 +++++++++++++---
 jos/kern/pmap.c                             |  68
++++++++++++++++++++++++++----------
 jos/kern/sched.c                            |  29 +++++++++++++
 jos/kern/syscall.c                          | 189
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++----
 jos/kern/trap.c                             |  70
++++++++++++++++++++++++++++++----------
 jos/kern/trapentry.S                        |   2 ++
 jos/lib/fork.c                              |  58
++++++++++++++++++++++++++---
 jos/lib/ipc.c                               |  40 ++++++++++++++++++++--
 jos/user/contador.c                         |   2 ++
 jos/user/dumbfork.c                         |   2 ++
 19 files changed, 526 insertions(+), 58 deletions(-)
```

Make grade

```
martin@martin:~/GitHub/sisop/jos$ make grade
helloinit: OK (0.7s)
Part 0 score: 1/1

yield: OK (0.9s)
spin0: Timeout! FAIL (1.3s)
       AssertionError: ...
       SMP: CPU 0 found 1 CPU(s)
       enabled interrupts: 1 2
       GOOD [00000000] new env 00001000
       GOOD [00000000] new env 00001001
       GOOD I am 00001000 and my spin will go on #1
       I am 00001000 and my spin will go on #2
       I am 00001000 and my spin will go on #3
       ...
       I am 00001000 and my spin will go on #11
       I am 00001000 and my spin will go on #12
       GOOD I am 00001001 and I like my interrupt #1
       I am 00001000 and my spin will go on #13
       I am 00001000 and my spin will go on #14
       ...
       I am 00001000 and my spin will go on #97
```

```
        I am 00001000 and my spin will go on #98
        GOOD I am 00001000 and my spin will go on #99
        I am 00001000 and my spin will go on #100
        I am 00001000 and my spin will go on #101
        ...
        I am 00001000 and my spin will go on #176
        I am 00001000 and my spin will go on #177
        I am 00001000 and my spin will go on #178
        I am 00001000 and my spin will go on #179
        qemu-system-i386: terminating on signal 15 from pid 6316 (make)
        MISSING 'I am 00001001 and I like my interrupt #4'

        QEMU output saved to jos.out.spin0
Part 1 score: 1/2

dumbfork: FAIL (0.7s)
        AssertionError: ...
        SMP: CPU 0 found 1 CPU(s)
        enabled interrupts: 1 2
        GOOD [00000000] new env 00001000
        GOOD [00001000] new env 00001001
        GOOD 0: I am the child!
        [00001001] user fault va 00000000 ip 00000000
        TRAP frame at 0xf02c207c from CPU 0
        ...
        esp  0xeebfe000
        ss   0x----0023
        GOOD [00001001] free env 00001001
        1: I am the child!
        2: I am the child!
        ...
        7: I am the child!
        8: I am the child!
        GOOD 9: I am the child!
        10: I am the child!
        11: I am the child!
        ...
        17: I am the child!
        18: I am the child!
        GOOD 19: I am the child!
        GOOD [00001000] exiting gracefully
        GOOD [00001000] free env 00001000
        No runnable environments in the system!
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 6421 (make)
        MISSING '0: I am the parent.'
        MISSING '9: I am the parent.'
        MISSING '.00001001. exiting gracefully'

        QEMU output saved to jos.out.dumbfork
forktree: FAIL (2.0s)
        AssertionError: ...
        1000: I am ''
        [00001000] new env 00001001
        GOOD 1000: I am '0'
        [00001000] new env 00001002
        1000: I am '00'
        [00001000] new env 00001003
        GOOD 1000: I am '000'
        GOOD [00001000] exiting gracefully
        [00001000] free env 00001000
```

```
        [00001001] user fault va 00000000 ip 00000000
        ...
        [00001002] free env 00001002
        No runnable environments in the system!
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 6527 (make)
        MISSING '....: I am .1.'
        MISSING '....: I am .100.'
        MISSING '....: I am .110.'
        MISSING '....: I am .111.'
        MISSING '....: I am .011.'
        MISSING '....: I am .001.'
        MISSING '.00001001. exiting gracefully'
        MISSING '.0000200.. exiting gracefully'
        MISSING '.0000200.. free env 0000200.'

        QEMU output saved to jos.out.forktree
spin: Timeout! FAIL (31.7s)
        AssertionError: ...
        SMP: CPU 0 found 1 CPU(s)
        enabled interrupts: 1 2
        GOOD [00000000] new env 00001000
        GOOD I am the parent.  Forking the child...
        GOOD [00001000] new env 00001001
        GOOD I am the child.  Spinning...
        [00001001] user fault va 00000000 ip 00000000
        TRAP frame at 0xf02c207c from CPU 0
        ...
        flag 0x00000202
        esp  0xeebfe000
        ss   0x----0023
        [00001001] free env 00001001
        qemu-system-i386: terminating on signal 15 from pid 6635 (make)
        MISSING 'I am the parent.  Running the child...'
        MISSING 'I am the parent.  Killing the child...'
        MISSING '.00001000. destroying 00001001'
        MISSING '.00001000. free env 00001001'
        MISSING '.00001000. exiting gracefully'
        MISSING '.00001000. free env 00001000'

        QEMU output saved to jos.out.spin
Part 2 score: 0/3

yield2: FAIL (1.7s)
        AssertionError: ...
        enabled interrupts: 1 2
        SMP: CPU 1 starting
        GOOD [00000000] new env 00001000
        GOOD [00000000] new env 00001001
        Hello, I am environment 00001001.
        Hello, I am environment 00001000.
        ...
        All done in environment 00001001.
        All done in environment 00001000.
        GOOD [00001000] exiting gracefully
        [00001000] free env 00001000
        GOOD [00001001] exiting gracefully
        [00001001] free env 00001001
        No runnable environments in the system!
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
```

```
        qemu-system-i386: terminating on signal 15 from pid 6836 (make)
        MISSING 'Hello, I am environment 0000100., cpu 0'
        MISSING 'Hello, I am environment 0000100., cpu 1'
        MISSING 'Back in environment 0000100., iteration 4, cpu 0'
        MISSING 'Back in environment 0000100., iteration 4, cpu 1'

        QEMU output saved to jos.out.yield2
stresssched: Timeout! FAIL (31.4s)
        AssertionError: ...
        flag 0x00000202
        esp  0xeebfe000
        ss   0x----0023
        [00001001] free env 00001001
        qemu-system-i386: terminating on signal 15 from pid 6949 (make)
        MISSING '.000010... stresssched on CPU 0'
        MISSING '.000010... stresssched on CPU 1'
        MISSING '.000010... stresssched on CPU 2'
        MISSING '.000010... stresssched on CPU 3'

        QEMU output saved to jos.out.stresssched
Part 3 score: 0/2

sendpage: FAIL (0.7s)
        AssertionError: ...
        enabled interrupts: 1 2
        SMP: CPU 1 starting
        GOOD [00000000] new env 00001000
        GOOD [00001000] new env 00001001
        [00001000] user fault va 00000000 ip 00800edf
        TRAP frame at 0xf02c2000 from CPU 0
        ...
        esp  0xeebfdf98
        ss   0x----0023
        GOOD [00001000] free env 00001000
        [00001001] user fault va 00000000 ip 00000000
        TRAP frame at 0xf02c207c from CPU 0
        ...
        esp  0xeebfe000
        ss   0x----0023
        GOOD [00001001] free env 00001001
        No runnable environments in the system!
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 7131 (make)
        MISSING '1001 got message from 1000: hello child environment! how are you?'
        MISSING 'child received correct message'
        MISSING '1000 got message from 1001: hello parent environment! I'm good'
        MISSING 'parent received correct message'
        MISSING '.00001000. exiting gracefully'
        MISSING '.00001001. exiting gracefully'

        QEMU output saved to jos.out.sendpage
pingpong: FAIL (1.5s)
        AssertionError: ...
        SMP: CPU 2 starting
        SMP: CPU 3 starting
        GOOD [00000000] new env 00001000
        GOOD [00001000] new env 00001001
        [00001000] user fault va 00000000 ip 00800df8
        TRAP frame at 0xf02c2000 from CPU 0
        ...
        esp  0xeebfdf88
```

```
        ss   0x----0023
        GOOD [00001000] free env 00001000
        [00001001] user fault va 00000000 ip 00000000
        TRAP frame at 0xf02c207c from CPU 0
        ...
        esp  0xeebfe000
        ss   0x----0023
        GOOD [00001001] free env 00001001
        No runnable environments in the system!
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 7236 (make)
        MISSING 'send 0 from 1000 to 1001'
        MISSING '1001 got 0 from 1000'
        MISSING '1000 got 1 from 1001'
        MISSING '1001 got 8 from 1000'
        MISSING '1000 got 9 from 1001'
        MISSING '1001 got 10 from 1000'
        MISSING '.00001000. exiting gracefully'
        MISSING '.00001001. exiting gracefully'

        QEMU output saved to jos.out.pingpong
primes: FAIL (1.2s)
        AssertionError: ...
        SMP: CPU 2 starting
        SMP: CPU 3 starting
        GOOD [00000000] new env 00001000
        GOOD [00001000] new env 00001001
        [00001000] user fault va 00000000 ip 00800e78
        TRAP frame at 0xf02c2000 from CPU 0
        ...
        [00001001] free env 00001001
        No runnable environments in the system!
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 7344 (make)
        MISSING 'CPU .: 2 .00001001. new env 00001002'
        MISSING 'CPU .: 3 .00001002. new env 00001003'
        MISSING 'CPU .: 5 .00001003. new env 00001004'
        MISSING 'CPU .: 7 .00001004. new env 00001005'
        MISSING 'CPU .: 11 .00001005. new env 00001006'
        MISSING 'CPU .: 1877 .00001120. new env 00001121'

        QEMU output saved to jos.out.primes
Part 4 score: 0/3

faultread: OK (1.7s)
faultwrite: OK (2.2s)
faultdie: FAIL (1.0s)
        AssertionError: ...
        esp  0xeebfdf88
        ss   0x----0023
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 7676 (make)
        MISSING 'i faulted at va deadbeef, err 6'
        MISSING '.00001000. exiting gracefully'
        MISSING '.00001000. free env 00001000'

        QEMU output saved to jos.out.faultdie
faultregs: FAIL (1.8s)
        AssertionError: ...
```

```
        esp  0xeebfdf88
        ss   0x----0023
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 7781 (make)
        MISSING 'Registers in UTrapframe OK'
        MISSING 'Registers after page-fault OK'

        QEMU output saved to jos.out.faultregs
faultalloc: FAIL (2.0s)
        AssertionError: ...
        esp  0xeebfdf88
        ss   0x----0023
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 7888 (make)
        MISSING 'fault deadbeef'
        MISSING 'this string was faulted in at deadbeef'
        MISSING 'fault cafebffe'
        MISSING 'fault cafec000'
        MISSING 'this string was faulted in at cafebffe'
        MISSING '.00001000. exiting gracefully'
        MISSING '.00001000. free env 00001000'

        QEMU output saved to jos.out.faultalloc
faultallocbad: FAIL (1.2s)
        AssertionError: ...
        esp  0xeebfdf88
        ss   0x----0023
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 7996 (make)
        MISSING '.00001000. user_mem_check assertion failure for va deadbeef'
        MISSING '.00001000. free env 00001000'

        QEMU output saved to jos.out.faultallocbad
faultnostack: FAIL (1.8s)
        AssertionError: ...
        esp  0xeebfdfb8
        ss   0x----0023
        GOOD [00001000] free env 00001000
        No runnable environments in the system!
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 8101 (make)
        MISSING '.00001000. user_mem_check assertion failure for va eebff...'

        QEMU output saved to jos.out.faultnostack
faultbadhandler: FAIL (2.0s)
        AssertionError: ...

        >>>
        Welcome to the JOS kernel monitor!
        Type 'help' for a list of commands.
        qemu-system-i386: terminating on signal 15 from pid 8208 (make)
        MISSING '.00001000. user_mem_check assertion failure for va (deadb|eebfe)...'
        MISSING '.00001000. free env 00001000'

        QEMU output saved to jos.out.faultbadhandler
faultevilhandler: FAIL (1.3s)
        AssertionError: ...
```

```
       >>>
       Welcome to the JOS kernel monitor!
       Type 'help' for a list of commands.
       qemu-system-i386: terminating on signal 15 from pid 8315 (make)
       MISSING '.00001000. user_mem_check assertion failure for va (f0100|eebfe)...'
       MISSING '.00001000. free env 00001000'

       QEMU output saved to jos.out.faultevilhandler
Part 5 score: 2/9

Score: 4/20
```

Parte 0: Múltiples CPUs

Tarea: mem_init_mp

```
static void
mem_init_mp(void)
{
    for(uint32_t i = 0; i < NCPU; ++i) {
        boot_map_region(kern_pgdir, KSTACKTOP - KSTKSIZE - i * (KSTKSIZE +
KSTKGAP), KSTKSIZE, PADDR(percpu_kstacks[i]), PTE_W);
    }
}
```

Tarea: mpentry_addr

```
void
page_init(void)
{
    int phy_io_inicio = IOPHYSMEM;
    int phy_io_fin = EXTPHYSMEM;

    int phy_pages_inicio = (int)pages - KERNBASE;
    int phy_pages_fin = (int)pages + (sizeof(struct PageInfo)*npages) - KERNBASE;

    int phy_envs_inicio = (int)envs - KERNBASE;
    int phy_envs_fin = (int)envs + (sizeof(struct Env)*NENV) - KERNBASE;

    int phy_mpentry_inicio = MPENTRY_PADDR;
    int phy_mpentry_fin = MPENTRY_PADDR + PGSIZE;

    _Static_assert(MPENTRY_PADDR % PGSIZE == 0, "MPENTRY_PADDR is not
page-aligned");

    unsigned int i;
    for (i = 1; i < npages; i++) {
        int phy_page = i * PGSIZE;

        // Seccion de 384K para I/O
```

```
        if ( (phy_page >= phy_io_inicio && phy_page < phy_io_fin) ||

        // Region donde se encuentra el codigo del kernel.
        // En boot_aloc() aclara que la primer direc dada, que se asigna a pages,
        // es aquella que el linker no asigno al codigo del kernel.
               (phy_page >= phy_io_fin && phy_page < phy_pages_inicio) ||

        // La memoria ya asignada por boot_alloc()
               (phy_page >= phy_pages_inicio && phy_page < phy_pages_fin) ||

               (phy_page >= phy_envs_inicio && phy_page < phy_envs_fin) ||

        //Septima pagina
               (phy_page >= phy_mpentry_inicio && phy_page < phy_mpentry_fin) )
        {
               continue;
        }

        pages[i].pp_ref = 0;
        pages[i].pp_link = page_free_list;
        page_free_list = &pages[i];
    }
}
```

Tarea: mmio_map_region

```
void *
mmio_map_region(physaddr_t pa, size_t size)
{
    void* ret = (void*)base;
    size_t size_mult = ROUNDUP(size, PGSIZE);

    if (base + size_mult >= MMIOLIM) {
        panic("MMIO overflow");
    }

    boot_map_region(kern_pgdir, base, size_mult, pa, PTE_PCD | PTE_PWT | PTE_W);

    base += size_mult;

    return ret;
}
```

Parte 1: Planificador y múltiples procesos

Tarea: sched_yield

```
void
sched_yield(void)
{
    struct Env *idle;

    int i;
    int limit_loop;
    if (curenv!=NULL && ENVX(curenv->env_id) != NENV-1) {
        int envID = ENVX(curenv->env_id);

        i = envID+1;
        limit_loop = envID;
    } else {
        i = 0;
        limit_loop = NENV-1;
    }

    while (i != limit_loop) {
        if (envs[i].env_status == ENV_RUNNABLE) {
               env_run(&envs[i]);
        }
        if (i == NENV-1)
               i=0;
        else
               i++;
    }

    // If no envs are runnable, but the environment previously
    // running on this CPU is still ENV_RUNNING, it's okay to
    // choose that environment.
      if (curenv && curenv->env_status == ENV_RUNNING) {
        env_run(curenv);
    }

    // sched_halt never returns
    sched_halt();
}
```

Tarea: sys_yield

```
int32_t
syscall(uint32_t syscallno, uint32_t a1, uint32_t a2, uint32_t a3, uint32_t a4,
uint32_t a5)
{
```

```
    switch (syscallno) {


        case SYS_page_alloc:
               return sys_page_alloc(a1, (void *)a2, a3);

        case SYS_page_map:
               return sys_page_map(a1, (void *)a2, a3, (void *)a4, a5);
               break;

        case SYS_env_set_status:
               return sys_env_set_status(a1, a2);

        case SYS_ipc_try_send:
               return sys_ipc_try_send(a1, a2, (void *)a3, a4);

        case SYS_ipc_recv:
               return sys_ipc_recv((void *)a1);

        case SYS_cputs:
               sys_cputs((char*)a1, a2);
               return 0;

        case SYS_exofork:
               return sys_exofork();

        case SYS_cgetc:
               return sys_cgetc();

        case SYS_getenvid:
               return sys_getenvid();

        case SYS_env_destroy:
               return sys_env_destroy(a1);

        case SYS_yield:
               sys_yield();
               return 0;

        case SYS_page_unmap:
               return sys_page_unmap(a1, (void *)a2);

        default:
               return -E_INVAL;
    }
}
```

Tarea: contador_env

```
// Se utiliza la 'va' UTEMP ya que en su definicion dice: "Used for temporary page
mappings"
#define VGA_USER UTEMP
```

```
static int
env_setup_vm(struct Env *e)
{
    int i;
    struct PageInfo *p = NULL;

    // Allocate a page for the page directory
    if (!(p = page_alloc(ALLOC_ZERO)))
        return -E_NO_MEM;

    // You need to increment env_pgdir's pp_ref for env_free to work correctly.
    p->pp_ref += 1;
    e->env_pgdir = page2kva(p);
    // Copia la informacion que posee el pgdir del kernel (user pages, kernel
stack, etc)
    // en el pgdir del proceso.
    memcpy(e->env_pgdir, kern_pgdir, PGSIZE);

    // UVPT maps the env's own page table read-only.
    // Permissions: kernel R, user R
    e->env_pgdir[PDX(UVPT)] = PADDR(e->env_pgdir) | PTE_P | PTE_U;

    // Mapping del buffer VGA
    struct PageInfo *page_info = pa2page(0xb8000);
    int err = page_insert(e->env_pgdir, page_info, VGA_USER, PTE_P | PTE_U |
PTE_W);
    if (err < 0) {
        return -E_NO_MEM;
    }

    return 0;
}
```

Tarea: timer_irq

En trapentry.S:

```
/*
 * Lab 3: Your code here for generating entry points for the different traps.
 */

handler0:
    TRAPHANDLER_NOEC(divide, T_DIVIDE);
handler1:
    TRAPHANDLER_NOEC(debug, T_DEBUG);
handler2:
    TRAPHANDLER_NOEC(nmi, T_NMI);
handler3:
    TRAPHANDLER_NOEC(brkpt, T_BRKPT);
handler4:
    TRAPHANDLER_NOEC(oflow, T_OFLOW);
handler5:
    TRAPHANDLER_NOEC(bound_check, T_BOUND);
handler6:
```

```
    TRAPHANDLER_NOEC(illop, T_ILLOP);
handler7:
    TRAPHANDLER_NOEC(device, T_DEVICE);
handler8:
    TRAPHANDLER(dblflt, T_DBLFLT);
handler10:
    TRAPHANDLER(tss, T_TSS);
handler11:
    TRAPHANDLER(segnp, T_SEGNP);
handler12:
    TRAPHANDLER(stack, T_STACK);
handler13:
    TRAPHANDLER(gpflt, T_GPFLT);
handler14:
    TRAPHANDLER(pgflt, T_PGFLT);
handler16:
    TRAPHANDLER_NOEC(fperr, T_FPERR);
handler17:
    TRAPHANDLER(align, T_ALIGN);
handler18:
    TRAPHANDLER_NOEC(mchk, T_MCHK);
handler19:
    TRAPHANDLER_NOEC(simderr, T_SIMDERR);
handler32:
    TRAPHANDLER_NOEC(clock, IRQ_OFFSET+IRQ_TIMER);
handler48:
    TRAPHANDLER_NOEC(t_syscall, T_SYSCALL);
```

En trap.c:

```
...
void clock();
...

void
trap_init(void)
{
    extern struct Segdesc gdt[];

    // LAB 3: Your code here.
    SETGATE(idt[0], 0, GD_KT, divide, 0);
    SETGATE(idt[1], 0, GD_KT, debug, 0);
    SETGATE(idt[2], 0, GD_KT, nmi, 0);
    SETGATE(idt[3], 0, GD_KT, brkpt, 3);
    SETGATE(idt[4], 0, GD_KT, oflow, 0);
    SETGATE(idt[5], 0, GD_KT, bound_check, 0);
    SETGATE(idt[6], 0, GD_KT, illop, 0);
    SETGATE(idt[7], 0, GD_KT, device, 0);
    SETGATE(idt[8], 0, GD_KT, dblflt, 0);
    SETGATE(idt[10], 0, GD_KT, tss, 0);
    SETGATE(idt[11], 0, GD_KT, segnp, 0);
    SETGATE(idt[12], 0, GD_KT, stack, 0);
    SETGATE(idt[13], 0, GD_KT, gpflt, 0);
    SETGATE(idt[14], 0, GD_KT, pgflt, 0);
    SETGATE(idt[16], 0, GD_KT, fperr, 0);
    SETGATE(idt[17], 0, GD_KT, align, 0);
```

```
    SETGATE(idt[18], 0, GD_KT, mchk, 0);
    SETGATE(idt[19], 0, GD_KT, simderr, 0);
    SETGATE(idt[IRQ_OFFSET+IRQ_TIMER], 0, GD_KT, clock, 0);
    SETGATE(idt[48], 0, GD_KT, t_syscall, 3);

    // Per-CPU setup
    trap_init_percpu();
}

int
env_alloc(struct Env **newenv_store, envid_t parent_id)
{
...
    // Enable interrupts while in user mode.
    // LAB 4: Your code here.
    e->env_tf.tf_eflags = e->env_tf.tf_eflags | FL_IF;
…
}
```

Tarea: timer_preempt

```
static void
trap_dispatch(struct Trapframe *tf)
{
    // Handle processor exceptions.
    // LAB 3: Your code here.
    switch (tf->tf_trapno) {
...
        case IRQ_OFFSET+IRQ_TIMER:
               sched_yield();
...
               }
    }
}
```

Parte 2: Creación dinámica de procesos

Tarea: sys_exofork

```
static envid_t
sys_exofork(void)
{
    struct Env *child;

    int err = env_alloc(&child, thiscpu->cpu_env->env_id);

    if (!err) {
        return err;
    }
```

```
    child->env_status =ENV_NOT_RUNNABLE;
    child->env_tf = thiscpu->cpu_env->env_tf;

    return child->env_id;
}

static int
sys_env_set_status(envid_t envid, int status)
{
       if (status != ENV_RUNNABLE && status != ENV_NOT_RUNNABLE) {
        return -E_INVAL;
    }

    struct Env *child = NULL;
    int err = envid2env(envid, &child, 1);
    if (err < 0) {
        return -E_BAD_ENV;
    }

    child->env_status = status;
    return 0;
}

static int
sys_page_alloc(envid_t envid, void *va, int perm)
{
       struct Env *env_act = NULL;
    int err = envid2env(envid, &env_act, 1);
    if (err < 0) {
        return -E_BAD_ENV;
    }

    if ((uint32_t)va >= UTOP || ((uint32_t)va % PGSIZE)) {
        return -E_INVAL;
    }

    if ((perm & (PTE_U|PTE_P)) != (PTE_U|PTE_P)) {
        return -E_INVAL;
    }

    struct PageInfo *page_info = page_alloc(ALLOC_ZERO);
    if (!page_info) {
        return -E_NO_MEM;
    }

    int err2 = page_insert(env_act->env_pgdir, page_info, va, perm);
    if (err2 < 0) {
        page_free(page_info);
    }

    return err;
}
```

```
static int
sys_page_map(envid_t srcenvid, void *srcva, envid_t dstenvid, void *dstva, int
perm)
{
       struct Env *env_src = NULL;
    int err = envid2env(srcenvid, &env_src, 1);
    if (err < 0) {
        return -E_BAD_ENV;
    }

    struct Env *env_dst = NULL;
    int err2 = envid2env(dstenvid, &env_dst, 1);
    if (err2 < 0) {
        return -E_BAD_ENV;
    }

    if ((uint32_t)srcva >= UTOP || ((uint32_t)srcva % PGSIZE)) {
        return -E_INVAL;
    }

    if ((uint32_t)dstva >= UTOP || ((uint32_t)dstva % PGSIZE)) {
        return -E_INVAL;
    }

    if ((perm & (PTE_U|PTE_P)) != (PTE_U|PTE_P)) {
        return -E_INVAL;
    }

    pte_t *src_pte = pgdir_walk(env_src->env_pgdir, srcva, false);
    if ((perm & PTE_W) && ((*src_pte & PTE_W)==0)) {
        return -E_INVAL;
    }

    pte_t *dst_pte = pgdir_walk(env_dst->env_pgdir, dstva, true);
    if (!dst_pte) {
        return -E_NO_MEM;
    }

    *dst_pte = *src_pte;
    return 0;
}

static int
sys_page_unmap(envid_t envid, void *va)
{
    struct Env *env_act = NULL;
    int err = envid2env(envid, &env_act, 1);
    if (err < 0 || !env_act) {
        return -E_BAD_ENV;
    }

    if ((uint32_t)va >= UTOP || ((uint32_t)va % PGSIZE)) {
        return -E_INVAL;
```

```
    }

    page_remove(env_act->env_pgdir, va);

    return 0;
}
```

Tarea: fork_v0

```
static int
duppage(envid_t envid, unsigned pn)
{
    int r = 1;

    // LAB 4: Your code here.
    void *va = (void*) (pn * PGSIZE);
    if ((uvpt[pn] & PTE_W) || (uvpt[pn] & PTE_COW)) {
        r = sys_page_alloc(0, va, PTE_COW | PTE_U | PTE_P);
        if (r < 0) {
               return r;
        }
        r = sys_page_map(0, va, envid, va, PTE_COW | PTE_U | PTE_P);
    }
    return r;
}

static void
dup_or_share(envid_t dstenv, void *va, int perm) {

    if (!((uint32_t)va & PTE_W)) {
        sys_page_map(0, va, dstenv, va, perm);
    }

}

envid_t
fork_v0(void)
{
    envid_t envid;
    uint8_t *addr;
    int r;

    envid = sys_exofork();

    if (envid < 0)
        panic("sys_exofork: %e", envid);
    if (envid == 0) {
        thisenv = &envs[ENVX(sys_getenvid())];
        return 0;
    }

    for (addr = (uint8_t*) 0; (uint32_t)addr < UTOP; addr += PGSIZE) {
```

```
    //Si la pagina esta mapeada se llama a dup_or_share().
    if ( (uvpt[PGNUM(addr)] & PTE_P) ) {
              dup_or_share(envid, addr, PTE_SYSCALL);
        } else {
              duppage(envid, (uint32_t)addr / PGSIZE);
        }
    }

    // Start the child environment running
    if ((r = sys_env_set_status(envid, ENV_RUNNABLE)) < 0)
        panic("sys_env_set_status: %e", r);

    return envid;

}

envid_t
fork(void)
{
    // LAB 4: Your code here.
    return fork_v0();
}
```

Parte 3: Ejecución en paralelo (multi-core)

Ejercicio 4

```
void
trap_init_percpu(void)
{
    uint8_t cpuID = thiscpu->cpu_id;

    thiscpu->cpu_ts.ts_esp0 =  KSTACKTOP - cpuID * (KSTKSIZE + KSTKGAP);
    thiscpu->cpu_ts.ts_ss0 = GD_KD;
    ts.ts_iomb = sizeof(struct Taskstate);

    gdt[(GD_TSS0 >> 3)+cpuID] = SEG16(STS_T32A, (uint32_t)(&thiscpu->cpu_ts),
                                      sizeof(struct Taskstate)-1, 0);
    gdt[(GD_TSS0 >> 3)+cpuID].sd_s = 0;
    // A logical address consists of a 16-bit segment selector (supplying 13+1
address bits)
    // and a 16-bit offset. The segment selector must be located in one of the
segment registers.
    // That selector consists of a 2-bit Requested Privilege Level (RPL), a 1-bit
Table Indicator (TI),
    // and a 13-bit index. When attempting address translation of a given logical
address, the processor
    // reads the 64-bit segment descriptor structure from either the Global
Descriptor Table when TI=0 or
    // the Local Descriptor Table when TI=1.
    ltr( ((GD_TSS0 >> 3)+cpuID) << 3 );
```

```
    lidt(&idt_pd);
}
```

Ejercicio 5

```
void
i386_init(void)
{
...
    // Lab 4 multitasking initialization functions
    pic_init();

    // Acquire the big kernel lock before waking up APs
    // Your code here:
    lock_kernel();

    // Starting non-boot CPUs
    boot_aps();
…
}

void
mp_main(void)
{
    // We are in high EIP now, safe to switch to kern_pgdir
    lcr3(PADDR(kern_pgdir));
    cprintf("SMP: CPU %d starting\n", cpunum());

    lapic_init();
    env_init_percpu();
    trap_init_percpu();
    xchg(&thiscpu->cpu_status, CPU_STARTED); // tell boot_aps() we're up

    // Now that we have finished some basic setup, call sched_yield()
    // to start running processes on this CPU.  But make sure that
    // only one CPU can enter the scheduler at a time!
    //
    // Your code here:
    lock_kernel();
    sched_yield();

    // Remove this after you finish Exercise 4
    for (;;);
}

void
trap(struct Trapframe *tf)
{
...
    // The lower two-bits of the code segment descriptor will determine the
    // current privilege level that the code is executing at.
    if ((tf->tf_cs & 3) == 3) {
        // Trapped from user mode.
        // Acquire the big kernel lock before doing any
```

```
        // serious kernel work.
        // LAB 4: Your code here.
        assert(curenv);
        lock_kernel();

        // Garbage collect if current enviroment is a zombie
        if (curenv->env_status == ENV_DYING) {
               env_free(curenv);
               curenv = NULL;
               sched_yield();
        }
...
}

void
env_run(struct Env *e)
{
    if (curenv != NULL && curenv->env_status == ENV_RUNNING) {
        curenv->env_status = ENV_RUNNABLE;
    }
    curenv = e;
    curenv->env_status = ENV_RUNNING;
    curenv->env_runs += 1;
    lcr3(PADDR(curenv->env_pgdir));
    unlock_kernel();
    env_pop_tf(&curenv->env_tf);
}
```

Parte 4: IPC

Ejercicio 15

```
static int
sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva, unsigned perm)
{
    // LAB 4: Your code here.
    struct Env *env_send = NULL;
    int err = envid2env(envid, &env_send, 0);
    if (err < 0 || !env_send) {
        return -E_BAD_ENV;
    }

    if (!(uint32_t)env_send->env_ipc_recving) {
        return -E_IPC_NOT_RECV;
    }

    if ((uint32_t)srcva < UTOP && ((uint32_t)srcva % PGSIZE)) {
        return -E_INVAL;
    }

    if ((uint32_t)srcva < UTOP && !(perm & PTE_SYSCALL)) {
```

```
        return -E_INVAL;
    }

    pte_t *pte_srcva;
    struct Env *env_act = thiscpu->cpu_env;
    struct PageInfo *page_info = page_lookup(env_act->env_pgdir, srcva,
&pte_srcva);
    if ((uint32_t)srcva < UTOP && !page_info) {
        return -E_INVAL;
    }

    if ((perm & PTE_W) && (*pte_srcva & PTE_W)) {
        return -E_INVAL;
    }

    if (page_insert(env_send->env_pgdir, page_info, srcva, perm) < 0) {
        return -E_NO_MEM;
    }

    env_send->env_ipc_recving = 0;
    env_send->env_ipc_from = env_act->env_id;
    env_send->env_ipc_value = value;

    if ((uint32_t)srcva < UTOP) {
        env_send->env_ipc_perm |= perm;
    } else {
        env_send->env_ipc_perm = 0;
    }

    env_send->env_status = ENV_RUNNABLE;

    return 0;
}

static int
sys_ipc_recv(void *dstva)
{
    // LAB 4: Your code here.
    if ((uint32_t)dstva < UTOP && ((uint32_t)dstva % PGSIZE) ) {
        return -E_INVAL;
    }

    struct Env *env_act = thiscpu->cpu_env;

    env_act->env_ipc_recving = true;
    env_act->env_ipc_dstva = dstva;
    env_act->env_status = ENV_NOT_RUNNABLE;

    sched_yield();
    return 0;
}

void
ipc_send(envid_t to_env, uint32_t val, void *pg, int perm)
```

```
{
    // LAB 4: Your code here.
    void *aux = pg;
    if (!pg) {
        aux = (void *)UTOP;
    }

    int err;
    do {
        err = sys_ipc_try_send(to_env, val, pg, perm);
        if (err != 0) {
               sys_yield();
        }
    } while (err == -E_IPC_NOT_RECV);

    if (err != 0) {
        panic("Error al enviar!\n");
    }

}

int32_t
ipc_recv(envid_t *from_env_store, void *pg, int *perm_store)
{
    // LAB 4: Your code here.
    if (!from_env_store) {
        *from_env_store = thisenv->env_id;
    }

    if (!perm_store) {
        *perm_store = thisenv->env_ipc_perm;
    }

    void *aux = pg;
    if (thisenv->env_ipc_perm!=0 && !pg) {
        aux = (void *)UTOP;
    }

    int err = sys_ipc_recv(aux);
    if (err != 0) {
        if (!from_env_store) *from_env_store = 0;
        if (!perm_store) *perm_store = 0;
    }

    return err;
}
```