ALVAREZ WINDEY, Ariel Justo (97893) ROBLES, Gabriel (95897)

TP4: Sistema de archivos e intérprete de comandos

static_assert

Se recomienda leer la función `diskaddr()` en el archivo `fs/bc.c`. ¿Qué es `super->s_nblocks`?

La variable `super` es un `struct` que representa el super bloque del filesystem. En particular el campo `s_nblocks` almacena la cantidad de bloques que tiene nuestro filesystem, en esta función validamos que el número de bloque pasado por parámetro no sea mayor a la cantidad de bloques que tiene nuestro disco.

```
diff --git a/fs/bc.c b/fs/bc.c
index ec59cfb..d131002 100644
--- a/fs/bc.c
+++ b/fs/bc.c
@@ -51,6 +51,22 @@ bc_pgfault(struct UTrapframe *utf)
     //
     // LAB 5: you code here:

+   // Redondeamos al tamaño de una página
+   addr = ROUNDDOWN(addr, PGSIZE);
+
+   // Alocamos la pagina en la región de mapeo al disco
+   r = sys_page_alloc(0, addr, PTE_W | PTE_U | PTE_P);
+   if (r < 0)
+       panic("[bc_pgfault] sys_page_alloc failed %e", r);
+
+   // Leemos desde el número de sector correspondiente
+   // y vamos mapeando desde addr (la funcion incrementa
+   // addr de SECTSIZE) indicandole la cantidad de sectores
+   // que le corresponden a un bloque (BLKSECTS)
+   r = ide_read(blockno * BLKSECTS, addr, BLKSECTS);
+   if (r < 0)
+       panic("[bc_pgfault] ide_read failed %e", r);
+
     // Clear the dirty bit for the disk block page since we just read the
     // block from disk
     if ((r = sys_page_map(0, addr, 0, addr, uvpt[PGNUM(addr)] & PTE_SYSCALL)) <
@@ -75,12 +91,34 @@ void
 flush_block(void *addr)
 {
     uint32_t blockno = ((uint32_t) addr - DISKMAP) / BLKSIZE;
+   int error;

     if (addr < (void *) DISKMAP || addr >= (void *) (DISKMAP + DISKSIZE))
         panic("flush_block of bad va %08x", addr);

     // LAB 5: Your code here.
```

```
-   panic("flush_block not implemented");
+   // Redondeamos al tamaño de una página
+   addr = ROUNDDOWN(addr, BLKSIZE);
+
+   // Checkeamos que la pagina este mapeada
+   if (!va_is_mapped(addr))
+       return;
+
+   // Checkeamos que la pagina tenga el bit de dirty
+   // caso contrario no es necesario volcar el contenido a disco
+   if (!va_is_dirty(addr))
+       return;
+
+   // Si llegamos aca, realizamos la escritura en disco
+   // los argumentos son los mismos que para la lectura
+   error = ide_write(blockno * BLKSECTS, addr, BLKSECTS);
+   if (error)
+       panic("[flush_block] ide_write failed %e", error);
+
+   // Limpiamos el bit de dirty de igual manera que en bc_pgfault
+   error = sys_page_map(0, addr, 0, addr, uvpt[PGNUM(addr)] & PTE_SYSCALL);
+   if (error)
+       panic("[flush_block] sys_page_map: %e", error);
 }

 // Test that the block cache works, by smashing the superblock and
diff --git a/fs/fs.c b/fs/fs.c
index 875f451..c021ea3 100644
--- a/fs/fs.c
+++ b/fs/fs.c
@@ -62,7 +62,20 @@ alloc_block(void)
    // super->s_nblocks blocks in the disk altogether.

    // LAB 5: Your code here.
-   panic("alloc_block not implemented");
+   // Recorremos todos los bloques y retornamos el primero libre
+   for (int blockno = 0; blockno < super->s_nblocks; blockno++) {
+       if (block_is_free((uint32_t) blockno)) {
+           // Ponemos en cero en el bitmap para
+           // marcar el bloque como ocupado
+           bitmap[blockno / 32] &= ~(1 << (blockno % 32));
+
+           // Hacemos flush del bitmap
+           flush_block(bitmap);
+
+           // retornamos el numero de bloque alocado
+           return blockno;
+       }
+   }
    return -E_NO_DISK;
 }

@@ -133,7 +146,54 @@ static int
 file_block_walk(struct File *f, uint32_t filebno, uint32_t **ppdiskbno, bool
alloc)
 {
    // LAB 5: Your code here.
-   panic("file_block_walk not implemented");
```

```
+   int error, indirect_block;
+
+   // Checkeamos filebno que este dentro del rango permitido
+   if (filebno >= (NDIRECT + NINDIRECT))
+       return -E_INVAL;
+
+   // Comprobamos el caso facil en el que no es un bloque indirecto
+   if (filebno < NDIRECT) {
+       // Lo guardamos en ppdiskbno si es distinto de NULL
+       if (ppdiskbno)
+           *ppdiskbno = &f->f_direct[filebno];
+       return 0;
+   }
+
+   // Comprobamos el caso de que no tengamos bloque de indireccion
+   if (f->f_indirect == 0) {
+       // En este caso si el flag de alloc es false tenemos un error
+       if (alloc == false)
+           return -E_NOT_FOUND;
+
+       // Caso contrario alocamos el bloque
+       // y comprobamos que no haya error
+       indirect_block = alloc_block();
+       if (indirect_block < 0)
+           return -E_NO_DISK;
+
+       f->f_indirect = indirect_block;
+
+       // Alocamos una pagina virtual para el bloque alocado
+       error = sys_page_alloc(0,
+                              diskaddr(indirect_block),
+                              PTE_U | PTE_W | PTE_P);
+       if (error)
+           panic("[file_block_walk] sys_page_alloc failed %e", error);
+
+       // Limpiamos el bloque alocado
+       memset(diskaddr(indirect_block), 0, BLKSIZE);
+
+       // Hacemos flush a disco
+       flush_block(diskaddr(indirect_block));
+   }
+
+   // Almacenamos el puntero correspondiente del bloque indirecto
+   if (ppdiskbno)
+       *ppdiskbno =
+               (uint32_t *) diskaddr(f->f_indirect) + filebno - NDIRECT;
+
+   return 0;
 }

 // Set *blk to the address in memory where the filebno'th
@@ -147,8 +207,28 @@ file_block_walk(struct File *f, uint32_t filebno, uint32_t
**ppdiskbno, bool all
 int
 file_get_block(struct File *f, uint32_t filebno, char **blk)
 {
-   // LAB 5: Your code here.
-   panic("file_get_block not implemented");
```

```
+   int errcode;
+   uint32_t *global_block_ref;
+
+   errcode = file_block_walk(f, filebno, &global_block_ref, true);
+
+   if (errcode)
+       return errcode;
+
+   // Puede ocurrir que el bloque que recuperamos no exista
+   // En ese caso lo alocamos
+
+   if (*global_block_ref == 0)
+       *global_block_ref = alloc_block();
+
+   //  -E_NO_DISK if a block needed to be allocated but the disk is full.
+
+   if (*global_block_ref < 0)
+       return -E_NO_DISK;
+
+   *blk = (char *) diskaddr(*global_block_ref);
+
+   return 0;
 }

 // Try to find a file named "name" in dir.  If so, set *file to it.
diff --git a/fs/serv.c b/fs/serv.c
index f38153b..295d2ba 100644
--- a/fs/serv.c
+++ b/fs/serv.c
@@ -222,7 +222,30 @@ serve_read(envid_t envid, union Fsipc *ipc)
                req->req_n);

    // Lab 5: Your code here:
-   return 0;
+   // Variable para validacion de errores
+   int r;
+   // Definimos el struct open file
+   struct OpenFile *open_file;
+
+   // Transformamos de file ID a OpenFile
+   r = openfile_lookup(envid, req->req_fileid, &open_file);
+   if (r < 0)
+       return r;
+
+   r = file_read(open_file->o_file,
+                 ret->ret_buf,
+                 MIN(req->req_n, PGSIZE),
+                 open_file->o_fd->fd_offset);
+
+   // Si hubo error retornamos
+   if (r < 0)
+       return r;
+
+   // Actualizamos el offset dentro del archivo
+   open_file->o_fd->fd_offset += r;
+
+
+   return r;
```

```
 }


@@ -240,7 +263,30 @@ serve_write(envid_t envid, struct Fsreq_write *req)
                req->req_n);

    // LAB 5: Your code here.
-   panic("serve_write not implemented");
+   // Definimos el struct open file y variable de error
+   int r;
+   struct OpenFile *open_file;
+
+   // Transformamos file ID a OpenFile
+   r = openfile_lookup(envid, req->req_fileid, &open_file);
+   if (r < 0)
+       return r;
+
+   // Escribimos en el archivo (esta funcion extiende
+   // el archivo en caso de ser necesario)
+   r = file_write(open_file->o_file,
+                  req->req_buf,
+                  req->req_n,
+                  open_file->o_fd->fd_offset);
+
+   // Si hubo error retornamos
+   if (r < 0)
+       return r;
+
+   // Actualizamos el offset dentro del archivo
+   open_file->o_fd->fd_offset += r;
+
+   return r;
 }

 // Stat ipc->stat.req_fileid.  Return the file's struct Stat to the
diff --git a/kern/env.c b/kern/env.c
index c6f9574..cdf841a 100644
--- a/kern/env.c
+++ b/kern/env.c
@@ -464,6 +464,10 @@ env_create(uint8_t *binary, enum EnvType type)
    // If this is the file server (type == ENV_TYPE_FS) give it I/O
    // privileges.
    // LAB 5: Your code here.
+   // Checkeamos con el type del enviroment si es el user File Server
+   // en ese caso le damos los privilegios de entrada-salida
+   if (new_env->env_type == ENV_TYPE_FS)
+       new_env->env_tf.tf_eflags |= FL_IOPL_3;
 }

 //
diff --git a/kern/pmap.c b/kern/pmap.c
index 70b26c0..03e944d 100644
--- a/kern/pmap.c
+++ b/kern/pmap.c
@@ -307,21 +307,19 @@ mem_init_mp(void)
    //
    // LAB 4: Your code here:
    uintptr_t kstacktop_i;
```

```
-   for (size_t i = 0 ; i < NCPU ; i++) {
+   for (size_t i = 0; i < NCPU; i++) {
        // kstacktop_i tendra la direccion de memoria virtual
        // del TOPE del stack del CPU i. Recordar que el stack
        // crece a direcciones de memoria virtual mas bajas,
-       // con lo cual el limite del stack_i estara en
+       // con lo cual el limite del stack_i estara en
        // kstacktop_i + KSTKSIZE. Ver memlayout.h
        // para mas informacion.
        kstacktop_i = KSTACKTOP - i * (KSTKSIZE + KSTKGAP);
-       boot_map_region(
-           kern_pgdir,
-           kstacktop_i - KSTKSIZE,
-           KSTKSIZE,
-           PADDR(percpu_kstacks[i]), // Usamos la PA indicada
-           PTE_W | PTE_P
-       );
+       boot_map_region(kern_pgdir,
+                       kstacktop_i - KSTKSIZE,
+                       KSTKSIZE,
+                       PADDR(percpu_kstacks[i]),  // Usamos la PA indicada
+                       PTE_W | PTE_P);
    }
 }

@@ -369,7 +367,7 @@ page_init(void)
    // Change your code to mark the physical page at MPENTRY_PADDR
    // as in use
    _Static_assert(MPENTRY_PADDR % PGSIZE == 0,
-                  "MPENTRY_PADDR is not page-aligned");
+                  "MPENTRY_PADDR is not page-aligned");

    // The example code here marks all physical pages as free.
    // However this is not truly the case.  What memory is free?
@@ -397,11 +395,11 @@ page_init(void)
        if (paddr == MPENTRY_PADDR) {
            continue;
        }
-       if (paddr >= PADDR(boot_alloc(0)) || paddr < IOPHYSMEM) {
-       // Si no es una dirección prohibida
-       // pages[i].pp_ref = 0; // Fue seteado con memset
-         pages[i].pp_link = page_free_list;
-         page_free_list = &pages[i];
+       if (paddr >= PADDR(boot_alloc(0)) || paddr < IOPHYSMEM) {
+           // Si no es una dirección prohibida
+           // pages[i].pp_ref = 0; // Fue seteado con memset
+           pages[i].pp_link = page_free_list;
+           page_free_list = &pages[i];
        }
    }
 }
@@ -811,7 +809,7 @@ mmio_map_region(physaddr_t pa, size_t size)
    // Actualizamos la variable base (estatica)
    uintptr_t ret = base;
    base += size;
-   return (void*)ret;
+   return (void *) ret;
 }
```

```
 static uintptr_t user_mem_check_addr;
diff --git a/kern/sched.c b/kern/sched.c
index 10bae84..4632ca1 100644
--- a/kern/sched.c
+++ b/kern/sched.c
@@ -38,7 +38,7 @@ sched_yield(void)
     }

     // Recorremos circularmente el arreglo envs
-    for (size_t i = 0 ; i < NENV ; i++) {
+    for (size_t i = 0; i < NENV; i++) {
         // Tomamos el modulo para hacer el recorrido circular
         idle = envs + ((curenv_idx + i) % NENV);
         // Checkeamos el primer proceso en RUNNABLE
diff --git a/kern/syscall.c b/kern/syscall.c
index acc4bed..ee91872 100644
--- a/kern/syscall.c
+++ b/kern/syscall.c
@@ -13,24 +13,28 @@
 #include <kern/sched.h>



-
-
 // Funcion propia para validar direcciones
 // de memoria virtual pasadas a la syscall
 static int
-check_va(const void * va) {
-    uintptr_t casted = (uintptr_t)va;
-    if (PGOFF(casted) || casted >= UTOP) return -E_INVAL;
-    else return 0;
+check_va(const void *va)
+{
+    uintptr_t casted = (uintptr_t) va;
+    if (PGOFF(casted) || casted >= UTOP)
+        return -E_INVAL;
+    else
+        return 0;
 }

 // Funcion propia para validar los permisos
 // Primero comprobamos que no tengan permisos invalidos
 // Segundo comprobamos que los mandatorios esten seteados
 static int
-check_permissions(int perm) {
-    if (perm & ~PTE_SYSCALL || ((perm & (PTE_U | PTE_P)) != (PTE_U | PTE_P)))
return -E_INVAL;
-    else return 0;
+check_permissions(int perm)
+{
+    if (perm & ~PTE_SYSCALL || ((perm & (PTE_U | PTE_P)) != (PTE_U | PTE_P)))
+        return -E_INVAL;
+    else
+        return 0;
 }

 // Print a string to the system console.
@@ -105,11 +109,12 @@ sys_exofork(void)
```

```
     // will appear to return 0.

     // LAB 4: Your code here.
-    struct Env * new_env;
+    struct Env *new_env;
     // Creamos un nuevo proceso y el padre sera el proceso actual
     int ret = env_alloc(&new_env, curenv->env_id);
     // Comprobamos errores de env_alloc
-    if (ret < 0) return (envid_t)ret;
+    if (ret < 0)
+        return (envid_t) ret;
     // Actualizamos el estado del nuevo proceso
     new_env->env_status = ENV_NOT_RUNNABLE;
     // Copiamos el trap frame del padre
@@ -139,21 +144,23 @@ sys_env_set_status(envid_t envid, int status)
     // envid's status.

     // LAB 4: Your code here.
-    struct Env * env;
+    struct Env *env;

     // Pasamos envid a struct Env
     // Ponemos el checkeo de permisos en true
     // que comprueba que el envid pasado
-    // corresponde a currenv o a un hijo
+    // corresponde a currenv o a un hijo
     // inmediato de currenv.
     int ret = envid2env(envid, &env, true);

     // Comprobamos errores

-    if (ret < 0) return ret;
+    if (ret < 0)
+        return ret;

     // Comprobamos status a setear validos
-    if (status != ENV_RUNNABLE && status != ENV_NOT_RUNNABLE) return -E_INVAL;
+    if (status != ENV_RUNNABLE && status != ENV_NOT_RUNNABLE)
+        return -E_INVAL;

     // Si pasamos las validaciones seteamos el status y retornamos
     env->env_status = status;
@@ -174,7 +181,34 @@ sys_env_set_trapframe(envid_t envid, struct Trapframe *tf)
     // LAB 5: Your code here.
     // Remember to check whether the user has supplied us with a good
     // address!
-    panic("sys_env_set_trapframe not implemented");
+    // Definimos el struct Env
+    struct Env *env;
+
+    // Comprobamos que la direccion del trapframe
+    if ((uintptr_t) tf >= UTOP)
+        return -E_INVAL;
+
+    // Pasamos de envid a struct Env
+    envid2env(envid, &env, true);
+
+    // Validamos que el proceso exista o el caller tenga permisos
```

```
+   if (env == NULL)
+       return -E_BAD_ENV;
+
+   // Copiamos el TrapFrame en el struct Env
+   memcpy(&env->env_tf, tf, sizeof(struct Trapframe));
+
+   // Nos aseguramos que el usuario siempre code en nivel
+   // de proteccion 3 (CPL 3)
+   env->env_tf.tf_cs = GD_UT | 3;
+
+   // Nos aseguramos que tendra las interrupciones habilitadas
+   env->env_tf.tf_eflags |= FL_IF;
+
+   // Y ademas IOPL a 0
+   env->env_tf.tf_eflags |= FL_IOPL_0;
+
+   return 0;
 }

 // Set the page fault upcall for 'envid' by modifying the corresponding struct
@@ -190,8 +224,8 @@ sys_env_set_pgfault_upcall(envid_t envid, void *func)
 {
    // LAB 4: Your code here.

-   //Chequeamos errores
-   struct Env * env;
+   // Chequeamos errores
+   struct Env *env;
    if (envid2env(envid, &env, true) < 0) {
        return -E_BAD_ENV;
    }
@@ -230,34 +264,38 @@ sys_page_alloc(envid_t envid, void *va, int perm)

    // LAB 4: Your code here.
    // Variables
-   struct Env * env;
-   struct PageInfo * page;
+   struct Env *env;
+   struct PageInfo *page;

    // Pasamos envid a struct Env
-   // Y comprobamos de igual manera
+   // Y comprobamos de igual manera
    // que en sys_env_set_status
    int error = envid2env(envid, &env, true);

    // Comprobamos errores
-   if (error) return error; // -E_BAD_ENV
+   if (error)
+       return error;  // -E_BAD_ENV

    // Validamos va
    error = check_va(va);
-   if (error) return error;
-
+   if (error)
+       return error;
+
```

```
     // Validamos permisos
     error = check_permissions(perm);
-    if (error) return error;
+    if (error)
+        return error;

     // Alocamos la pagina y validamos que haya memoria
     page = page_alloc(ALLOC_ZERO);
-    if (page == NULL) return -E_NO_MEM;
+    if (page == NULL)
+        return -E_NO_MEM;

     // Mapeamos la pagina y validamos errores
     error = page_insert(env->env_pgdir, page, va, perm);
     if (error) {
-        page_free(page); // Liberamos la pagina (nunca se incremento pp_ref)
-        return error;    // -E_NO_MEM
+        page_free(page);  // Liberamos la pagina (nunca se incremento pp_ref)
+        return error;  // -E_NO_MEM
     }

     // Success
@@ -293,42 +331,50 @@ sys_page_map(envid_t srcenvid, void *srcva, envid_t dstenvid,
void *dstva, int p
     // LAB 4: Your code here.
     // Variables
     struct Env *src_env, *dst_env;
-    struct PageInfo * page;
-    pte_t * pte;
+    struct PageInfo *page;
+    pte_t *pte;
     int error = 0;

     // Pasamos envid's a struct Env's
-    // Y comprobamos de igual manera
+    // Y comprobamos de igual manera
     // que en sys_env_set_status
     error = envid2env(srcenvid, &src_env, true);
-    if (error) return error;
+    if (error)
+        return error;
     error = envid2env(dstenvid, &dst_env, true);
-    if (error) return error;
+    if (error)
+        return error;

     // Validamos va's
     error = check_va(srcva);
-    if (error) return error;
+    if (error)
+        return error;
     error = check_va(dstva);
-    if (error) return error;
+    if (error)
+        return error;

     // Validamos que srcva este mapeado en src_env Address Space
     page = page_lookup(src_env->env_pgdir, srcva, &pte);
```

```
-   if (page == NULL) return -E_INVAL;
+   if (page == NULL)
+       return -E_INVAL;

    // Validamos permisos
    // Idem page alloc
    // Validamos permisos
    error = check_permissions(perm);
-   if (error) return error;
+   if (error)
+       return error;

    // Verificamos que no sea una pagina de solo lectura
    // y se la este intentando mapear con permisos de escritura
-   if (((*pte & PTE_W) == 0) && (perm & PTE_W)) return -E_INVAL;
+   if (((*pte & PTE_W) == 0) && (perm & PTE_W))
+       return -E_INVAL;

    // Mapeamos la pagina y validamos errores
    error = page_insert(dst_env->env_pgdir, page, dstva, perm);
-   if (error) return error;     // -E_NO_MEM
-
+   if (error)
+       return error;  // -E_NO_MEM
+
    return 0;
 }

@@ -346,20 +392,23 @@ sys_page_unmap(envid_t envid, void *va)

    // LAB 4: Your code here.
    // Variables
-   struct Env * env;
-   struct PageInfo * page;
+   struct Env *env;
+   struct PageInfo *page;
    int error = 0;
    error = envid2env(envid, &env, true);
-   if (error) return error;
+   if (error)
+       return error;

    // Validamos va
    error = check_va(va);
-   if (error) return error;
+   if (error)
+       return error;

    // Validamos que haya pagina mapeada sino
    // page_remove podria fallar en page_decref.
    page = page_lookup(env->env_pgdir, va, NULL);
-   if (page == NULL) return 0; // "silently succeeds"
+   if (page == NULL)
+       return 0;  // "silently succeeds"

    // Removemos la pagina
    page_remove(env->env_pgdir, va);
@@ -409,7 +458,7 @@ static int
```

```
 sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva, unsigned perm)
 {
    // LAB 4: Your code here.
-   struct Env * to_env;
+   struct Env *to_env;
    bool trans_page = false;
    int ret = envid2env(envid, &to_env, false);
    if (ret < 0) {
@@ -423,18 +472,20 @@ sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva,
unsigned perm)
        return -E_IPC_NOT_RECV;
    }

-   if ((uintptr_t)srcva < UTOP || (uintptr_t)to_env->env_ipc_dstva < UTOP) {
+   // Mapeamos si AMBOS procesos lo solicitaro
+   if ((uintptr_t) srcva < UTOP && (uintptr_t) to_env->env_ipc_dstva < UTOP) {
        //-E_INVAL if srcva < UTOP but srcva is not page-aligned.
-       if (((uintptr_t)srcva % PGSIZE) != 0) {
+       if (((uintptr_t) srcva % PGSIZE) != 0) {
            return -E_INVAL;
        }
        //-E_INVAL if srcva < UTOP and perm is inappropriate
        ret = check_permissions(perm);
-       if (ret) return -E_INVAL;
+       if (ret)
+           return -E_INVAL;

        //  -E_INVAL if srcva < UTOP but srcva is not mapped in the caller's
-       pte_t * pte;
-       struct PageInfo * page;
+       pte_t *pte;
+       struct PageInfo *page;
        if (!(page = page_lookup(curenv->env_pgdir, srcva, &pte))) {
            return -E_INVAL;
        }
@@ -443,13 +494,14 @@ sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva,
unsigned perm)
            return -E_INVAL;
        }

-       if (page_insert(to_env->env_pgdir, page, to_env->env_ipc_dstva, perm) < 0)
{
+       if (page_insert(to_env->env_pgdir, page, to_env->env_ipc_dstva, perm) <
+           0) {
            //-E_NO_MEM if there's not enough memory to map srcva in envid's
address space.
            return -E_NO_MEM;
        }

        trans_page = true;
-   }
+   }

    // Otherwise, the send succeeds, and the target's ipc fields are
    // updated as follows:
@@ -461,9 +513,11 @@ sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva,
unsigned perm)
    //    env_ipc_value is set to the 'value' parameter;
```

```
     to_env->env_ipc_value = value;
     //    env_ipc_perm is set to 'perm' if a page was transferred, 0 otherwise.
-    if (trans_page) to_env->env_ipc_perm = perm;
-    else to_env->env_ipc_perm = 0;
-
+    if (trans_page)
+        to_env->env_ipc_perm = perm;
+    else
+        to_env->env_ipc_perm = 0;
+
     // The target environment is marked runnable again
     to_env->env_status = ENV_RUNNABLE;

@@ -488,8 +542,9 @@ sys_ipc_recv(void *dstva)

     // Si dstva es < UTOP => Esperamos recibir una página,
     // En ese caso, validamos que la dirección esté alineada
-    if ((uintptr_t)dstva < UTOP) {
-        if (PGOFF(dstva)) return -E_INVAL;
+    if ((uintptr_t) dstva < UTOP) {
+        if (PGOFF(dstva))
+            return -E_INVAL;
     }

     // Indicamos donde queremos recibir la pagina de dato
@@ -540,26 +595,35 @@ syscall(uint32_t syscallno, uint32_t a1, uint32_t a2,
uint32_t a3, uint32_t a4,
         return (int32_t) sys_exofork();
     }
     case SYS_env_set_status: {
-        return (int32_t) sys_env_set_status((envid_t)a1, (int)a2);
+        return (int32_t) sys_env_set_status((envid_t) a1, (int) a2);
     }
     case SYS_page_alloc: {
-        return (int32_t) sys_page_alloc((envid_t)a1, (void *)a2, (int)a3);
+        return (int32_t) sys_page_alloc((envid_t) a1, (void *) a2, (int) a3);
     }
     case SYS_page_map: {
-        return (int32_t) sys_page_map((envid_t)a1, (void *)a2, (envid_t)a3, (void
*)a4, (int)a5);
+        return (int32_t) sys_page_map((envid_t) a1,
+                                      (void *) a2,
+                                      (envid_t) a3,
+                                      (void *) a4,
+                                      (int) a5);
     }
     case SYS_page_unmap: {
-        return (int32_t) sys_page_unmap((envid_t)a1, (void *)a2);
+        return (int32_t) sys_page_unmap((envid_t) a1, (void *) a2);
     }
     case SYS_ipc_recv: {
-        return (int) sys_ipc_recv((void *)a1);
+        return (int32_t) sys_ipc_recv((void *) a1);
     }
     case SYS_ipc_try_send: {
-        return (int) sys_ipc_try_send((envid_t)a1, (uint32_t)a2, (void *)a3,
(unsigned)a4);
+        return (int32_t) sys_ipc_try_send(
```

```
+               (envid_t) a1, (uint32_t) a2, (void *) a3, (unsigned) a4);
     }
-
     case SYS_env_set_pgfault_upcall: {
-        return (int) sys_env_set_pgfault_upcall((envid_t)a1, (void *)a2);
+        return (int32_t) sys_env_set_pgfault_upcall((envid_t) a1,
+                                                    (void *) a2);
+    }
+    case SYS_env_set_trapframe: {
+        return (int32_t) sys_env_set_trapframe((envid_t) a1,
+                                               (struct Trapframe *) a2);
     }

     default:
diff --git a/kern/trap.c b/kern/trap.c
index cabcbfd..fff82e0 100644
--- a/kern/trap.c
+++ b/kern/trap.c
@@ -53,6 +53,8 @@ extern void trap_19();
 extern void trap_20();

 extern void trap_32();
+extern void trap_33();
+extern void trap_36();

 extern void trap_48();

@@ -151,6 +153,10 @@ trap_init(void)

     // Timer interruption
     SETGATE(idt[IRQ_OFFSET + IRQ_TIMER], 0, GD_KT, trap_32, 0);
+    // Keyboard interruption
+    SETGATE(idt[IRQ_OFFSET + IRQ_KBD], 0, GD_KT, trap_33, 0);
+    // Serial port interruption
+    SETGATE(idt[IRQ_OFFSET + IRQ_SERIAL], 0, GD_KT, trap_36, 0);

     // SYSCALL interrupt
     SETGATE(idt[48], 0, GD_KT, trap_48, 3);
@@ -189,9 +195,9 @@ trap_init_percpu(void)
     // Obtenemos el cpunum (0 para cpu 1, 1 para cpu 2, etc...)
     int cpuid = cpunum();
     // Obtenenemos el struct cpuinfo del cpu en cuestión
-    struct CpuInfo * curcpu = &(cpus[cpuid]);
+    struct CpuInfo *curcpu = &(cpus[cpuid]);
     // De dicho cpu obtenemos la estructura Taskstate (que representa la TSS)
-    struct Taskstate * curts = &(curcpu->cpu_ts);
+    struct Taskstate *curts = &(curcpu->cpu_ts);

     // Calculamos el indice del task segment del core en cuestión
     uint16_t idx = (GD_TSS0 >> 3) + cpuid;
@@ -211,7 +217,8 @@ trap_init_percpu(void)

     curts->ts_iomb = sizeof(struct Taskstate);

-    gdt[idx] = SEG16(STS_T32A, (uint32_t)(curts), sizeof(struct Taskstate) - 1, 0);
+    gdt[idx] =
+            SEG16(STS_T32A, (uint32_t)(curts), sizeof(struct Taskstate) - 1, 0);
```

```
    gdt[idx].sd_s = 0;

@@ -233,10 +240,10 @@ trap_init_percpu(void)

    // Load the TSS selector (like other segment selectors, the
    // bottom three bits are special; we leave them 0)
-   //ltr(GD_TSS0);
+   // ltr(GD_TSS0);

    // Load the IDT
-   //lidt(&idt_pd);
+   // lidt(&idt_pd);
 }

 void
@@ -316,8 +323,16 @@ trap_dispatch(struct Trapframe *tf)
        return;
    }
    case IRQ_OFFSET + IRQ_TIMER: {
-       lapic_eoi();        // Avisamos al hardware que atrapamos la interrupcion
-       sched_yield();  // Actuamos en consecuencia de la interrupcion (round-
robin)
+       lapic_eoi();  // Avisamos al hardware que atrapamos la interrupcion
+       sched_yield();  // Actuamos en consecuencia de la interrupcion (round-
robin)
+       return;
+   }
+   case IRQ_OFFSET + IRQ_KBD: {
+       kbd_intr();
+       return;
+   }
+   case IRQ_OFFSET + IRQ_SERIAL: {
+       serial_intr();
        return;
    }
    case T_SYSCALL: {
@@ -478,24 +493,37 @@ page_fault_handler(struct Trapframe *tf)
        struct UTrapframe *u;

        // Si es una llamada recursiva entonces el esp estará dentro del rango de
[UXSTACKTOP; UXSTACKTOP-PGSIZE)
-       bool recursive = ((tf->tf_esp < UXSTACKTOP) && (tf->tf_esp >= UXSTACKTOP-
PGSIZE)) ? true : false;
-
+       bool recursive = ((tf->tf_esp < UXSTACKTOP) &&
+                         (tf->tf_esp >= UXSTACKTOP - PGSIZE))
+                                ? true
+                                : false;
+
        if (recursive) {
-           // Si es llamada recursiva, debemos dejar una palabra en blanco (4
bytes) entre el UTrapFrame
-           // Anterior y el nuevo
-           u = (struct UTrapframe *) (tf->tf_esp - 4 - sizeof(struct UTrapframe));
+           // Si es llamada recursiva, debemos dejar una palabra en
+           // blanco (4 bytes) entre el UTrapFrame Anterior y el
+           // nuevo
+           u = (struct UTrapframe *) (tf->tf_esp - 4 -
```

```
+                                       sizeof(struct UTrapframe));
            // Chequeamos que tengamos permisos para escribir el UTrapFrame en el
stack
            // Y el word (4 bytes adicionales) en blanco para distinguir llamada
recursiva
-           user_mem_assert(curenv, (void *) u, sizeof(struct UTrapframe) + 4,
PTE_W | PTE_P);
+           user_mem_assert(curenv,
+                           (void *) u,
+                           sizeof(struct UTrapframe) + 4,
+                           PTE_W | PTE_P);
        } else {
            // Si no es llamada recursiva, se debe escribir un UTrapFrame en
UXSTACKTOP
-           u = (struct UTrapframe *) (UXSTACKTOP - sizeof(struct UTrapframe));
-           user_mem_assert(curenv, (void *) u, sizeof(struct UTrapframe), PTE_W |
PTE_P);
+           u = (struct UTrapframe *) (UXSTACKTOP -
+                                      sizeof(struct UTrapframe));
+           user_mem_assert(curenv,
+                           (void *) u,
+                           sizeof(struct UTrapframe),
+                           PTE_W | PTE_P);
        }

        // Chequeamos que el handler de usuario sea accesible para el usuario
-       user_mem_assert(curenv, (void *) curenv->env_pgfault_upcall, 4, PTE_P);
-
+       user_mem_assert(
+               curenv, (void *) curenv->env_pgfault_upcall, 4, PTE_P);
+
        // Completamos el UTrapFrame copiando desde tf
        u->utf_fault_va = fault_va;
        u->utf_err = tf->tf_err;
@@ -506,7 +534,7 @@ page_fault_handler(struct Trapframe *tf)

        // Si es recursivo escribimos el ultimo byte en 0
        if (recursive) {
-           uintptr_t * aux = (uintptr_t *) u;
+           uintptr_t *aux = (uintptr_t *) u;
            aux += sizeof(struct UTrapframe);
            *aux = 0;
        }
diff --git a/kern/trapentry.S b/kern/trapentry.S
index fcf5080..43e876c 100644
--- a/kern/trapentry.S
+++ b/kern/trapentry.S
@@ -71,6 +71,8 @@ TRAPHANDLER_NOEC(trap_19, T_SIMDERR)
 TRAPHANDLER_NOEC(trap_20, 20)

 TRAPHANDLER_NOEC(trap_32, IRQ_OFFSET + IRQ_TIMER)
+TRAPHANDLER_NOEC(trap_33, IRQ_OFFSET + IRQ_KBD)
+TRAPHANDLER_NOEC(trap_36, IRQ_OFFSET + IRQ_SERIAL)
 TRAPHANDLER_NOEC(trap_48, T_SYSCALL)


diff --git a/lib/file.c b/lib/file.c
index f2e0b45..863aeb9 100644
```

```
--- a/lib/file.c
+++ b/lib/file.c
@@ -141,7 +141,28 @@ devfile_write(struct Fd *fd, const void *buf, size_t n)
    // remember that write is always allowed to write *fewer*
    // bytes than requested.
    // LAB 5: Your code here
-   panic("devfile_write not implemented");
+
+   // Variable de validación
+   int r;
+
+   // Guardamos el file ID en el struct Fsreq_write
+   fsipcbuf.write.req_fileid = fd->fd_file.id;
+
+   // Guardamos la canitdad de bytes a escribir en el struct Fsreq_write
+   // (checkeamos que los solicitados no sean mas que el largo del buffer)
+   fsipcbuf.write.req_n = MIN(n, sizeof(fsipcbuf.write.req_buf));
+
+   // Copiamos en Fsreq_write el buffer a escribir
+   memmove(fsipcbuf.write.req_buf,
+           buf,
+           MIN(n, sizeof(fsipcbuf.write.req_buf)));
+
+   // Realizamos el request al file system server
+   r = fsipc(FSREQ_WRITE, NULL);
+   if (r < 0)
+       return r;
+
+   return r;
 }

 static int
diff --git a/lib/fork.c b/lib/fork.c
index 8089079..6730030 100644
--- a/lib/fork.c
+++ b/lib/fork.c
@@ -25,6 +25,20 @@ pgfault(struct UTrapframe *utf)
    //   (see <inc/memlayout.h>).

    // LAB 4: Your code here.
+   // Recuperamos la PTE en cuestión
+   pte_t pte = uvpt[PGNUM(addr)];
+
+   // Verificamos que la página esté mapeada
+   if ((err & FEC_PR) == 0)
+       panic("[pgfault] pgfault por página no mapeada");
+
+   // Verificamos que el page fault no haya sido por una lectura
+   if ((err & FEC_WR) == 0)
+       panic("[pgfault] pgfault por lectura");
+
+   // Verificamos que la página tenga copy-on-write
+   if ((pte & PTE_COW) == 0)
+       panic("[pgfault] pgfault COW no configurado");

    // Allocate a new page, map it at a temporary location (PFTEMP),
    // copy the data from the old page to the new page, then move the new
@@ -34,7 +48,19 @@ pgfault(struct UTrapframe *utf)
```

```
     // LAB 4: Your code here.

-    panic("pgfault not implemented");
+    // Alocamos una nueva página en PFTEMP
+    r = sys_page_alloc(0, PFTEMP, PTE_W | PTE_U | PTE_P);
+    if (r)
+        panic("[pgfault] sys_page_alloc failed: %e", r);
+
+    // Alineamos y copiamos el contenido de la página
+    addr = (void *) ROUNDDOWN(addr, PGSIZE);
+    memmove(PFTEMP, addr, PGSIZE);
+
+    // Re-mapeamos correctamente
+    r = sys_page_map(0, PFTEMP, 0, addr, PTE_W | PTE_U | PTE_P);
+    if (r)
+        panic("[pgfault] sys_page_map failed: %e", r);
 }

 //
@@ -54,12 +80,45 @@ duppage(envid_t envid, unsigned pn)
     int r;

     // LAB 4: Your code here.
-    panic("duppage not implemented");
+
+    // Recuperamos la PTE asociada
+    pte_t pte = uvpt[pn];
+
+    // Reconstruimos la dirección de memoria virtual
+    void *va = (void *) (pn << PTXSHIFT);
+
+    if (pte & PTE_SHARE) {
+        // Si es una pagina que debe ser compartida con
+        // los mismos permisos que en el padre (por cuestiones
+        // del filesystem, asi persisten cambios de archivos abiertos)
+        r = sys_page_map(0, va, envid, va, pte & PTE_SYSCALL);
+        if (r)
+            panic("[duppage] sys_page_map: %e", r);
+    } else if ((pte & PTE_W) || (pte & PTE_COW)) {
+        // Si la página era de escritura o tenía copy on write
+
+        // Mapeamos la página en el hijo sin PTE_W
+        r = sys_page_map(0, va, envid, va, PTE_COW | PTE_U | PTE_P);
+        if (r)
+            panic("[duppage] sys_page_map: %e", r);
+
+        // Re-mapeamos la página en el padre sin PTE_W
+        r = sys_page_map(0, va, 0, va, PTE_COW | PTE_U | PTE_P);
+        if (r)
+            panic("[duppage] sys_page_map: %e", r);
+    } else {
+        // Si es una pagina de solo lectura simplemente la compartimos
+        r = sys_page_map(0, va, envid, va, PTE_U | PTE_P);
+        if (r)
+            panic("[duppage] sys_page_map: %e", r);
+    }
+
     return 0;
```

```
 }

 static void
-dup_or_share(envid_t dstenv, void *va, int perm) {
+dup_or_share(envid_t dstenv, void *va, int perm)
+{
     int r;

     // Si la pagina es de escritura
@@ -75,10 +134,10 @@ dup_or_share(envid_t dstenv, void *va, int perm) {
         // en el proceso padre (0 = currenv = proceso padre) en la direccion UTEMP
         if ((r = sys_page_map(dstenv, va, 0, UTEMP, perm)) < 0)
             panic("[dup_or_share] sys_page_map: %e", r);
-
+
         // Copia el contenido de la pagina addr (del padre)
         // en UTEMP (del padre) que esta mapeada con addr (del hijo)
-        // Es decir esta copiando el contenido padre de addr en
+        // Es decir esta copiando el contenido padre de addr en
         // la pagina del hijo (copia del A.S.)
         memmove(UTEMP, va, PGSIZE);

@@ -118,16 +177,16 @@ fork_v0(void)
     int r;

     // Creamos un proceso nuevo
-    // El kernel copia los registros y
-    // continua desde aqui tanto para padre
-    // (envid > 0 (envid del hijo))
+    // El kernel copia los registros y
+    // continua desde aqui tanto para padre
+    // (envid > 0 (envid del hijo))
     // como para el hijo (envid = 0).
     envid = sys_exofork();
     if (envid < 0)
         panic("[fork_v0] sys_exofork failed: %e", envid);
     if (envid == 0) {
         // Si envid es 0 entonces el proceso
-        // es el hijo, corregimos la variable
+        // es el hijo, corregimos la variable
         // thisenv y retornamos
         thisenv = &envs[ENVX(sys_getenvid())];
         return 0;
@@ -147,8 +206,10 @@ fork_v0(void)
             // Checkeamos que la page table entry este mapeada
             if (pte & PTE_P) {
                 // Como la pagina esta mapeada, llamamos a dup_or_share
-                dup_or_share(envid, (void*)addr, pte & PTE_SYSCALL);
-            }
+                dup_or_share(envid,
+                             (void *) addr,
+                             pte & PTE_SYSCALL);
+            }
         }
     }

@@ -179,7 +240,89 @@ envid_t
 fork(void)
```

```
 {
    // LAB 4: Your code here.
-   return fork_v0();
+   int error;
+
+   // Configuramos pgfault como el handler del padre
+   // Esto tambien reserva memoria para su pila de excepciones
+   set_pgfault_handler(pgfault);
+
+   // Creamos el proceso hijo y validamos correctamente
+   envid_t envid = sys_exofork();
+   if (envid < 0)
+       panic("[fork] sys_exofork failed: %e", envid);
+
+   if (envid == 0) {
+       // Si envid es 0 entonces el proceso
+       // es el hijo, corregimos la variable
+       // thisenv y retornamos
+       thisenv = &envs[ENVX(sys_getenvid())];
+       return 0;
+   } else {
+       // Es el proceso padre
+       // Usamos indices para poder iterar
+       // sobre la cantidad minima de paginas posibles
+       size_t pdx;
+       size_t ptx;
+
+       // Procesamos las paginas de memoria de 0 a UTOP
+       // Si la pagina esta mapeada invocamos a dup_or_share()
+       for (pdx = 0; pdx < PDX(UTOP); pdx++) {
+           // Recuperamos el page directory entry
+           pde_t pde = uvpd[pdx];
+
+           // Verificamos que la Page Table este alocada
+           // caso contrario la ignoramos y continuamos con la siguiente
+           if ((pde & PTE_P) == 0)
+               continue;
+
+           // Si está alocada recorremos las 1024 PTE,
+           // copiando las páginas alocadas.
+           for (ptx = 0; ptx < NPTENTRIES; ptx++) {
+               // Construimos la direccion virtual
+               // Usamos 0 para el offset
+               uintptr_t addr = (uintptr_t) PGADDR(pdx, ptx, 0);
+
+               // Usamos el PGNUM para acceder a uvpt con la VA construida
+               pte_t pte = uvpt[PGNUM(addr)];
+
+               // Si la dirección es el Stack de excepciones
+               // no lo duplicamos sino que alocamos una nueva
+               // página para el proceso hijo
+               if (addr == (UXSTACKTOP - PGSIZE)) {
+                   error = sys_page_alloc(envid,
+                                          (void *) addr,
+                                          PTE_W | PTE_U |
+                                                  PTE_P);
+                   if (error)
+                       panic("[fork] sys_page_alloc "
```

```
+                             "failed: %e",
+                             error);
+                 continue;
+             }
+
+             // Si la página no está alocada la salteamos
+             if ((pte & PTE_P) == 0)
+                 continue;
+
+             // Si la página está alocada llamamos a duppage()
+             duppage(envid, PGNUM(addr));
+         }
+     }
+
+     // Configuramos pgfault como el handler del hijo
+     error = sys_env_set_pgfault_upcall(envid,
+                                        thisenv->env_pgfault_upcall);
+     if (error)
+         panic("[fork] sys_env_set_pgfault_upcall failed: %e",
+               error);
+
+     // Seteamos al proceso hijo como RUNNABLE
+     error = sys_env_set_status(envid, ENV_RUNNABLE);
+     if (error)
+         panic("[fork] sys_env_set_status failed: %e", error);
+
+     return envid;
+ }
 }

 // Challenge!
diff --git a/lib/ipc.c b/lib/ipc.c
index 6765b0f..1846b38 100644
--- a/lib/ipc.c
+++ b/lib/ipc.c
@@ -23,27 +23,32 @@ int32_t
 ipc_recv(envid_t *from_env_store, void *pg, int *perm_store)
 {
     // LAB 4: Your code here.
-
+
     // No se espera recibir una página
     // Cualquier dirección por mayor o igual a UTOP
     // es interpretado por sys_ipc_recv como que
     // no se espera una página.
-    if (!pg) pg = (void *) UTOP;
+    if (!pg)
+        pg = (void *) UTOP;

     int ret = sys_ipc_recv(pg);
     if (ret < 0) {
         // Syscall con error
-        if (from_env_store) *from_env_store = 0;
-        if (perm_store) *perm_store = 0;
+        if (from_env_store)
+            *from_env_store = 0;
+        if (perm_store)
+            *perm_store = 0;
```

```
        return ret;
    }

    // Syscall exitosa

    // If 'from_env_store' is nonnull, then store the IPC sender's envid in
*from_env_store.
-   if (from_env_store) *from_env_store = thisenv->env_ipc_from;
+   if (from_env_store)
+       *from_env_store = thisenv->env_ipc_from;
    // If 'perm_store' is nonnull, then store the IPC sender's page permission in
*perm_store
-   if (perm_store) *perm_store = thisenv->env_ipc_perm;
+   if (perm_store)
+       *perm_store = thisenv->env_ipc_perm;
    // Return the value sent by the sender
    return thisenv->env_ipc_value;
 }
@@ -61,10 +66,10 @@ ipc_send(envid_t to_env, uint32_t val, void *pg, int perm)
 {
    // LAB 4: Your code here.

-   // Si pg es NULL, le pasaremos a sys_ipc_try_send un valor que entenderá que
-   // significa "no page"
+   // Si pg es NULL, le pasaremos a sys_ipc_try_send un valor que entenderá
+   // que significa "no page"
    if (!pg) {
-       pg = (void *)UTOP;
+       pg = (void *) UTOP;
    }

    int ret;
diff --git a/lib/spawn.c b/lib/spawn.c
index 0858caf..2e60fdc 100644
--- a/lib/spawn.c
+++ b/lib/spawn.c
@@ -323,5 +323,40 @@ static int
 copy_shared_pages(envid_t child)
 {
    // LAB 5: Your code here.
+   int r;
+
+   // Creamos los indices para iterar
+   size_t pdx;
+   size_t ptx;
+
+   // Iteramos hasta USTACKTOP para no afectar el stack de excepciones
+   for (pdx = 0; pdx < PDX(USTACKTOP); pdx++) {
+       pde_t pde = uvpd[pdx];
+
+       // Si la PDE no esta alocada continuamos
+       if ((pde & PTE_P) == 0)
+           continue;
+
+       // Si esta alocada recorremos las PTE's
+       for (ptx = 0; ptx < NPTENTRIES; ptx++) {
+           // Armamos la direccion virtual correspondiente
+           uintptr_t addr = (uintptr_t) PGADDR(pdx, ptx, 0);
```

```
+
+           pte_t pte = uvpt[PGNUM(addr)];
+
+           // Comprobamos que este alocada y tenga el bit PTE_SHARE
+           if ((pte & PTE_P) && (pte & PTE_SHARE)) {
+               // Si asi mapeamos la pagina
+               r = sys_page_map(0,
+                                (void *) addr,
+                                child,
+                                (void *) addr,
+                                pte & PTE_SYSCALL);
+               if (r < 0)
+                   return r;
+           }
+       }
+   }
+
    return 0;
 }
diff --git a/user/sh.c b/user/sh.c
index 8ec285e..def4425 100644
--- a/user/sh.c
+++ b/user/sh.c
@@ -55,7 +55,20 @@ again:
            // then close the original 'fd'.

            // LAB 5: Your code here.
-           panic("< redirection not implemented");
+           // Abrimos el archivo 't' como solo lectura
+           if ((fd = open(t, O_RDONLY)) < 0) {
+               cprintf("open %s for read: %e", t, fd);
+               exit();
+           }
+           // Si no es stdin llamamos a dup
+           if (fd != 0) {
+               r = dup(fd, 0);
+               if (r)
+                   cprintf("dup for %s failed: %e", t, r);
+
+               // Cerramos el file descriptor viejo
+               close(fd);
+           }
            break;

        case '>':  // Output redirection
```