Tp1

Integrantes: - Etchegaray Campisi, Rodrigo; 96856
- Soldo, Federico; 98288

```
diff --git a/TP1.md b/TP1.md
index a03729f..9eb39db 100644
--- a/TP1.md
+++ b/TP1.md
@@ -1,92 +1,21 @@
-TP1: Memoria virtual en JOS
+#TP1: Memoria virtual en JOS
 ==============================

-page2pa
--------------
-
+##page2pa
 Recibe un puntero a un struct PageInfo *pp y devuelve la direccion de
memoria fisica donde comienza la pagina. A este puntero le resta la
direccion donde se encuentra el primer struct PageInfo del arreglo de
paginas para asi quitarle el offset y obtener el virtual page number, y le
hace un shift PGSHIFT = 12, debido a que las physical page son de 4kb
para alinearlo.

+...


 boot_alloc_pos
 --------------

-a. Al hacer readelf kernel -a, pudimos ver que en el codigo objeto lo
ultimo que direcciona son las variables globales. La ultima es pages
direccionada virtualmente en 0xf011794c.
-Entonces al entrar a la primer llamada de boot_alloc, lo que ocurre es
que end apunta a esta direccion, la proxima libre luego de direccionar
todo el codigo objeto en memoria.
-Luego nextfree es igual a esta direccion redondeada para "arriba" a
4096 (pagesize), y esto es lo que devolvera la funcion en su primer
llamada.
-
-
-b. make gdb
-gdb -q -s obj/kern/kernel -ex 'target remote 127.0.0.1:26000' -n -
x .gdbinit
-Reading symbols from obj/kern/kernel...done.
-Remote debugging using 127.0.0.1:26000
-warning: No executable has been specified and target does not support
-determining executable automatically.  Try using the "file" command.
```

```
-0x0000fff0 in ?? ()
-(gdb) b boot_alloc
-Breakpoint 1 at 0xf0100a59: file kern/pmap.c, line 89.
-(gdb) c
-Continuing.
-The target architecture is assumed to be i386
-=> 0xf0100a59 <boot_alloc>:        mov    %eax,%edx
-
-Breakpoint 1, boot_alloc (n=981) at kern/pmap.c:89
-89     {
-(gdb) si
-=> 0xf0100a5b <boot_alloc+2>:  cmpl   $0x0,0xf0117538
-98             if (!nextfree)
-(gdb) p nextfree
-$1 = 0x0
-(gdb) p (char*) end
-$2 = 0xf0100702 <cons_init>
"U\211\345\203\354\b\350\266\373\377\377\350\277\372\377\377\200=4
u", <incomplete sequence \360>
-(gdb) si
-=> 0xf0100a62 <boot_alloc+9>:  je     0xf0100a9f <boot_alloc+70>
-0xf0100a62       98             if (!nextfree)
-(gdb) si
-=> 0xf0100a9f <boot_alloc+70>: mov    $0xf011894f,%eax
-101                     nextfree = ROUNDUP((char *) end, PGSIZE);
-(gdb)
-=> 0xf0100aa4 <boot_alloc+75>: and    $0xfffff000,%eax
-0xf0100aa4       101                    nextfree = ROUNDUP((char *) end,
PGSIZE);
-(gdb)
-=> 0xf0100aa9 <boot_alloc+80>: mov    %eax,0xf0117538
-0xf0100aa9       101                    nextfree = ROUNDUP((char *) end,
PGSIZE);
-(gdb)
-=> 0xf0100aae <boot_alloc+85>: jmp    0xf0100a64 <boot_alloc+11>
-0xf0100aae       101                    nextfree = ROUNDUP((char *) end,
PGSIZE);
-(gdb)
-=> 0xf0100a64 <boot_alloc+11>: test   %edx,%edx
-110            if (n != 0)
-(gdb) p nextfree
-$3 = 0xf0118000 ""
-(gdb) p (char*) end
-$4 = 0xf0100702 <cons_init>
"U\211\345\203\354\b\350\266\373\377\377\350\277\372\377\377\200=4
u", <incomplete sequence \360>
-(gdb) si
-=> 0xf0100a66 <boot_alloc+13>:je     0xf0100ab0 <boot_alloc+87>
-0xf0100a66       110            if (n != 0)
```

```
-(gdb)
-=> 0xf0100a68 <boot_alloc+15>: mov    0xf0117538,%eax
-112                    result = nextfree;
-(gdb)
-=> 0xf0100a6d <boot_alloc+20>: lea    0xfff(%eax,%edx,1),%edx
-113                    nextfree = ROUNDUP((char *) (nextfree+n), PGSIZE);
-(gdb)
-=> 0xf0100a74 <boot_alloc+27>: and    $0xfffff000,%edx
-0xf0100a74       113                    nextfree = ROUNDUP((char *)
(nextfree+n), PGSIZE);
-(gdb)
-=> 0xf0100a7a <boot_alloc+33>: mov    %edx,0xf0117538
-0xf0100a7a       113                    nextfree = ROUNDUP((char *)
(nextfree+n), PGSIZE);
-(gdb)
-=> 0xf0100a80 <boot_alloc+39>: cmp    $0x400000,%edx
-114                    if ((int)nextfree > PGSIZE*1024) panic("boot_alloc: out
of memory\n");
-(gdb) p result
-$5 = 0xf0118000 ""
-(gdb) p (char*) end
-$6 = 0xf0100702 <cons_init>
"U\211\345\203\354\b\350\266\373\377\377\350\277\372\377\377\200=4
u", <incomplete sequence \360>
-
+...


 page_alloc
 ----------

-Como se explico anteriormente, page2pa recibe un puntero a un struct
PageInfo *pp y devuelve la direccion de memoria fisica donde comienza
la pagina. Por otro lado, page2kva, utiliza la macro KADDR que toma una
direccion fisica y devuelve la kernel virtual address correspondiente.
Entonces, page2kva recibe tambien un struct PageInfo *pp que convierte
a direccion fisica usando page2pa, para luego usar la macro KADDR y asi
devolver una kernel virtual address.
+...
+
+
diff --git a/__pycache__/gradelib.cpython-36.pyc b/__pycache__/
gradelib.cpython-36.pyc
index 7f5c571..cfe05fd 100644
Binary files a/__pycache__/gradelib.cpython-36.pyc and b/__pycache__/
gradelib.cpython-36.pyc differ
diff --git a/kern/pmap.c b/kern/pmap.c
index be16065..88608e7 100644
--- a/kern/pmap.c
```

```
+++ b/kern/pmap.c
@@ -95,8 +95,7 @@ boot_alloc(uint32_t n)
         // which points to the end of the kernel's bss segment:
         // the first virtual address that the linker did *not* assign
         // to any kernel code or global variables.
-        if (!nextfree)
-        {
+        if (!nextfree) {
                 extern char end[];
                 nextfree = ROUNDUP((char *) end, PGSIZE);
         }
@@ -107,16 +106,7 @@ boot_alloc(uint32_t n)
         //
         // LAB 2: Your code here.

-        if (n != 0)
-        {
-                result = nextfree;
-                nextfree = ROUNDUP((char *) (nextfree+n), PGSIZE);
-                if ((int)nextfree > PGSIZE*1024) panic("boot_alloc: out of
memory\n");
-                return result;
-        }
-        else return nextfree;
-
-
+        return NULL;
 }

 // Set up a two-level page table:
@@ -138,7 +128,7 @@ mem_init(void)
         i386_detect_memory();

         // Remove this line when you're ready to test this function.
-        // panic("mem_init: This function is not finished\n");
+        panic("mem_init: This function is not finished\n");

         //////////////////////////////////////////////////////////////////////
         // create initial page directory.
@@ -164,12 +154,6 @@ mem_init(void)
         // to initialize all fields of each struct PageInfo to 0.
         // Your code goes here:

-        pages = (struct PageInfo*) boot_alloc(npages*sizeof(struct
PageInfo));
-        memset(pages, 0, npages*sizeof(struct PageInfo));
-
-
-
```

```
-

        //////////////////////////////////////////////////////////////////////
        // Now that we've allocated the initial kernel data structures, we set
@@ -193,8 +177,6 @@ mem_init(void)
        //      (ie. perm = PTE_U | PTE_P)
        //    - pages itself -- kernel RW, user NONE
        // Your code goes here:
-       size_t size = ROUNDUP(npages*sizeof(struct PageInfo), PGSIZE);
-       boot_map_region(kern_pgdir, UPAGES, PTSIZE, PADDR(pages),
PTE_U | PTE_P);

        //////////////////////////////////////////////////////////////////////
        // Use the physical memory that 'bootstack' refers to as the kernel
@@ -208,9 +190,6 @@ mem_init(void)
        //     Permissions: kernel RW, user NONE
        // Your code goes here:

-       boot_map_region(kern_pgdir, KSTACKTOP-KSTKSIZE, KSTKSIZE,
PADDR(bootstack), PTE_W | PTE_P);
-
-
        //////////////////////////////////////////////////////////////////////
        // Map all of physical memory at KERNBASE.
        // Ie.  the VA range [KERNBASE, 2^32) should map to
@@ -220,9 +199,6 @@ mem_init(void)
        // Permissions: kernel RW, user NONE
        // Your code goes here:

-       size = ROUNDDOWN((2^32)-KERNBASE, PGSIZE);
-       boot_map_region(kern_pgdir, KERNBASE, size, (physaddr_t) 0,
PTE_W | PTE_P);
-
        // Check that the initial page directory has been set up correctly.
        check_kern_pgdir();

@@ -280,27 +256,11 @@ page_init(void)
        // Change the code to reflect this.
        // NB: DO NOT actually touch the physical memory corresponding
to
        // free pages!
-
-
        size_t i;
-       physaddr_t pp_address;
-       void* ka;
-
-       for (i = 0; i < npages; i++)
-       {
```

```
-               pp_address = page2pa(&pages[i]);
-
-               if (! ((i == 0) | ( (pp_address >= IOPHYSMEM) &
(pp_address < PADDR(boot_alloc(0)))) ) )
-               {
-                       pages[i].pp_ref = 0;
-                       pages[i].pp_link = page_free_list;
-                       page_free_list = &pages[i];
-               }
-               else
-               {
-                       pages[i].pp_link = NULL;
-               }
-
+       for (i = 0; i < npages; i++) {
+               pages[i].pp_ref = 0;
+               pages[i].pp_link = page_free_list;
+               page_free_list = &pages[i];
        }
 }

@@ -319,16 +279,8 @@ page_init(void)
 struct PageInfo *
 page_alloc(int alloc_flags)
 {
-       if (page_free_list) {
-       struct PageInfo *ret = page_free_list;
-       page_free_list = page_free_list->pp_link;
-       ret->pp_link = NULL;
-
-       if (alloc_flags & ALLOC_ZERO) memset( (void*) page2kva(ret), 0,
PGSIZE);
-
-       return ret;
-   }
-   return NULL;
+       // Fill this function in
+       return 0;
 }

 //
@@ -341,10 +293,6 @@ page_free(struct PageInfo *pp)
        // Fill this function in
        // Hint: You may want to panic if pp->pp_ref is nonzero or
        // pp->pp_link is not NULL.
-       if ((pp->pp_ref != 0) | (pp->pp_link != NULL) ) panic("page_free:
Try to free a page that cannot be freed\n");
-       pp->pp_link = page_free_list;
-       pp->pp_ref = 0;
-       page_free_list = pp;
 }

 //
@@ -380,25 +328,13 @@ page_decref(struct PageInfo *pp)
 // Hint 3: look at inc/mmu.h for useful macros that mainipulate page
 // table and page directory entries.
 //
-
 pte_t *
 pgdir_walk(pde_t *pgdir, const void *va, int create)
 {
-       pde_t * pgdir_entry = &pgdir[PDX(va)]; /*Esta bien ?*/
-       if (!(*pgdir_entry & PTE_P))
-       {
-               if (!create) return NULL;
-               struct PageInfo* page = page_alloc(1);
-               if (!page) return NULL;
-               page->pp_ref = 1;
-               *pgdir_entry = page2pa(page) | PTE_P | PTE_U | PTE_W;
-       }
-       pte_t * page_table_entry = KADDR(PTE_ADDR(*pgdir_entry));
-       return page_table_entry + PTX(va);
+       // Fill this function in
+       return NULL;
 }

-
-
 //
 // Map [va, va+size) of virtual address space to physical [pa, pa+size)
 // in the page table rooted at pgdir.  Size is a multiple of PGSIZE, and
@@ -413,13 +349,7 @@ pgdir_walk(pde_t *pgdir, const void *va, int
create)
 static void
 boot_map_region(pde_t *pgdir, uintptr_t va, size_t size, physaddr_t pa,
int perm)
 {
-       pte_t * pte;
-       for (size_t i = 0; i < size/PGSIZE; i++)
-       {
-               pte = pgdir_walk(pgdir, (void*) (va + (uintptr_t) (i*PGSIZE)),
1);
-               *pte = (pa + i*PGSIZE) | perm | PTE_P;
-       }
-
+       // Fill this function in
 }
 //
@@ -450,15 +380,7 @@ boot_map_region(pde_t *pgdir, uintptr_t va,
size_t size, physaddr_t pa, int perm
 int
 page_insert(pde_t *pgdir, struct PageInfo *pp, void *va, int perm)
 {
-       /*VER Requirements*/
-       pte_t * pte = pgdir_walk(pgdir, va, 1);
-       if (!pte) return -E_NO_MEM;
-       pp->pp_ref++;
-       if ((*pte & PTE_P))
-       {
-               page_remove(pgdir, va);
-       }
-       *pte = page2pa(pp) | perm | PTE_P;
+       // Fill this function in
        return 0;
 }

@@ -476,10 +398,8 @@ page_insert(pde_t *pgdir, struct PageInfo *pp,
void *va, int perm)
 struct PageInfo *
 page_lookup(pde_t *pgdir, void *va, pte_t **pte_store)
 {
-       pte_t* pte = pgdir_walk(pgdir, va, 0);
-       if (!pte) return NULL;
-       physaddr_t pa = PTE_ADDR(*pte);
-       return pa2page(pa);
+       // Fill this function in
+       return NULL;
 }

 //
@@ -500,14 +420,7 @@ page_lookup(pde_t *pgdir, void *va, pte_t
**pte_store)
 void
 page_remove(pde_t *pgdir, void *va)
 {
-       struct PageInfo * page = page_lookup(pgdir, va, 0);
-       if (page)
-       {
-               page_decref(page);
-               pte_t * pte = pgdir_walk(pgdir, va, 0);
-               if (pte) *pte = 0;
-               tlb_invalidate(pgdir, va);
-       }
+       // Fill this function in
 }
 //
make clean
make[1]: Entering directory '/home/eche/Documents/JOS'
rm -rf obj jos.in qemu.log
make[1]: Leaving directory '/home/eche/Documents/JOS'
./grade-lab2
make[1]: Entering directory '/home/eche/Documents/JOS'
+ as kern/entry.S
+ cc kern/entrypgdir.c
+ cc kern/init.c
+ cc kern/console.c
+ cc kern/monitor.c
+ cc kern/pmap.c
+ cc kern/kclock.c
+ cc kern/printf.c
+ cc kern/kdebug.c
+ cc lib/printfmt.c
+ cc lib/readline.c
+ cc lib/string.c
+ ld obj/kern/kernel
+ as boot/boot.S
+ cc -Os boot/main.c
+ ld boot/boot
+ mk obj/kern/kernel.img
make[1]: Leaving directory '/home/eche/Documents/JOS'
running JOS: OK (0.6s)
  Physical page allocator: OK
  Page management: OK
  Kernel page directory: OK
  Page management 2: OK
  Large pages: SKIPPED
Score: 4/4
```