Pintos P3

- Benjamin Diaz
- Juan Galvez
- Rodrigo Cespedes

FRAME

Frame.h

```
#include <stdbool.h>
#include "threads/synch.h"

struct frame {
   struct lock frame_lock; //this is a lock for synch purposes
  struct page *frame_page; //self explanatory
  void *kernel_vaddr; //virtual address corresponding to the kernel
};

void frame_table_init ();
struct frame *frame_table_allocation (struct page *page_tmp);
```

Frame.c

Variables globales:

- static struct frame *all_frames : array de frames
- static int counter_f : número de frames

Funciones:

- alloc_all():
  - Hace parte de la lógica para inicializar frame table.
- alloc_frames_per_page_logic():
  - Trata de asignar un page a un frame y hacerle “lock”, tambien busca un page para hacer “evict”.

- alloc_frames_per_page():
  - Invoca a alloc_frames_per_page_logic() y retorna un frame con un page asignado si el proceso actual tiene “frame_tmp->frame_lock”asignado, caso contrario retorna null.
- frame_table_init():
  - Inicializa frame table.
- frame_table_allocation():
  - Invoca a alloc_frames_per_page() y devuelve el frame de retornado por esa función.

PAGE

Page.h

```c
/* Contiene un Page */
struct page {
    struct hash_elem hash_elem; // will be useful for saving the thread's pages elements
    struct frame *page_frame; //page frame
    struct thread *page_thread; // page thread (thread owns page)
    struct file *file; // file
    void *user_vaddr; //user virtual address
    bool is_readable; //to know if this can only be for reading
    off_t bytes; // number of bytes to read or write
    off_t offset; // file offset to precisely operate
    block_sector_t block_sector; // value used for validation

};

/* Funciones */
struct page *page_init_and_alloc(void *);
void unlock_page (const void *);
bool lock_page (const void *);
bool page_in (void *);
bool page_out (struct page *);
void free_page (struct hash_elem *);
```

Page.c

Funciones:

- free_page():
  - Elimina un page del frame table y se libera memoria.
- page_with_vaddr():
  - Retorna el page que tenga la VA, si no existe retorna null.
- lock_frame_by_page():
  - Hace lock de un frame dado un page, luego hace “swap_in”, retorna true si todo funcionó, caso contrario, retorna false.
- page_in():
  - Hace “page in” de un page dada un VA, para este proceso invoca a lock_frame_by_page() donde se hace el swap.

- page_out():
  - Hace “page_out” de un page invocando a “swap_out”, para este caso, page debe tener un frame lock, retorna true si se pudo hacer el page_out, caso contrario retorna false.
- page_init_and_alloc():
  - Añade un mapeo por el user virtual address al page table. Falla si el VADDR ya está mapeado o su la alocación de memoria falla.
- lock_page():
  - Intenta lockear un page en memoria física.
- unlock_page():
  - Desbloquea un page.

SWAP

Swap.h

```
#define SECTOR_PER_PAGE (PGSIZE / BLOCK_SECTOR_SIZE)
#define B_SECTOR_SIZE  ( (block_sector_t) - 1 )

struct page;

struct swap{

        struct block *swap_block; // checks if swap is enabled

        struct lock swap_lock; // ensures locking while swap operation is made

        struct bitmap *swap_bitmap; //keeps track of used swapped pages
};

void swap_init();
void swap_in (struct page *page_tmp);
bool swap_out (struct page *page_tmp);
```

Swap.c

Variables globales:

- static struct actual_swap:
  - Struct swap que será utilizado en las funciones.
- uint32_t variable:
  - Variable auxiliar, será utilizada en varias funciones.

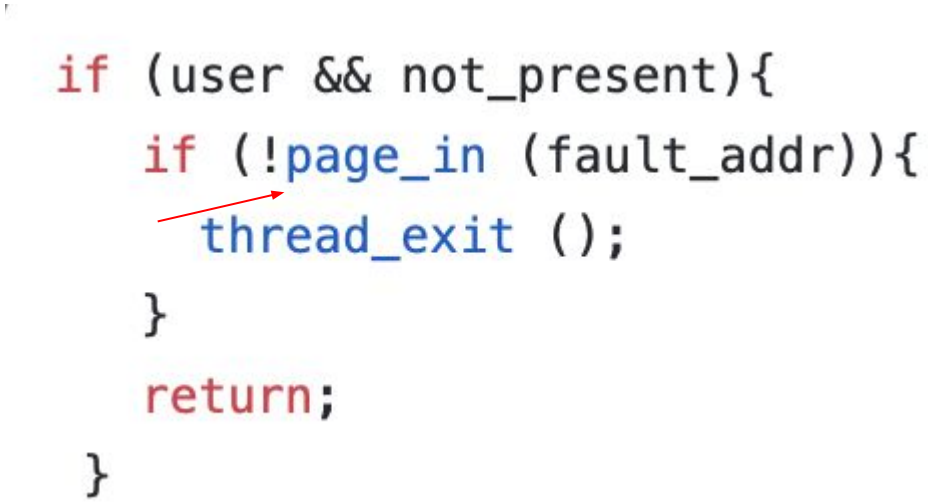
Funciones:

- swap_init():
  - Inicializa el swap.
- swap_in():
  - Trae un page dado a un frame.
- swap_out():
  - Saca un page dado de un frame.

Implementación

Exception.c

```
Page_fault(struct intr_frame *f)
```



Process.c

Page_exit(void) aca quitamos el page del page table

```
struct hash *hash_tmp = NULL;
hash_tmp = thread_current()->pages;
if (hash_tmp){
        hash_destroy (hash_tmp, free_page);
}
```

```
bool page_cmp (const struct hash_elem *hash_elem_a, const struct hash_elem *hash_elem_b, void *aux UNUSED){
  struct page *page_a = hash_entry (hash_elem_a, struct page, hash_elem);
  struct page *page_b = hash_entry (hash_elem_b, struct page, hash_elem);

  return page_a->user_vaddr < page_b->user_vaddr;
}

unsigned page_hash (const struct hash_elem *hash_elem_a, void *aux UNUSED){
  struct page *page = hash_entry (hash_elem_a, struct page, hash_elem);
  return ((uintptr_t) page->user_vaddr) >> PGBITS;
}
```

load(const char *file_name, void (**eip) (void), void **esp)

```
struct thread *t = thread_current ();
struct Elf32_Ehdr ehdr;
struct file *file = NULL;
off_t file_ofs;
bool success = false;
int i;


/* Allocate and activate page directory. */
t->pagedir = pagedir_create ();
if (t->pagedir == NULL)
  goto done;
process_activate ();

t->pages = malloc (sizeof (struct hash) );
if (t->pages == NULL){
  goto done;
}
hash_init (t->pages, page_hash, page_cmp, NULL);
```

---

load_segment(struct file *file, off_t, ofs, uint8_t *upage,

uint32_t read_bytes, uint32_t zero_bytes)

```
/* Nuestro */
 struct page *page_tmp = NULL;
 page_tmp = page_init_and_alloc(upage);
 if (!page_tmp){
   return false;
   }
```

setup_stack(void **esp, const char *file_name)

```
struct page *page = page_init_and_alloc(((uint8_t *) PHYS_BASE) - PGSIZE);
if (page) {
    page->page_frame = frame_table_allocation(page);
    if (page->page_frame){

      bool success;
      page->is_readable = false;
      char *filename_aux = NULL;
      size_t size_data_type;
      off_t offset = PGSIZE;
      size_t new_pgsize;
      void *dest = NULL;

      /*------------------------------------------------------------------------*/
      //Pushear argumentos
      size_data_type =  (strlen(file_name)+1);
      new_pgsize = ROUND_UP (size_data_type, sizeof (uint32_t));
      if (new_pgsize > offset){
              success =  false;
              goto Unlock_Frame_Table;
      }
      offset = offset - new_pgsize;
      dest = offset + page->page_frame->kernel_vaddr + (new_pgsize - size_data_type);
      memcpy (dest, file_name, size_data_type);
      filename_aux = dest;
      /*------------------------------------------------------------------------*/
```

Se repite varias veces

Syscall.c

Static char *get_new_file(const char *file)

```
if (kernel_file){

        int size;
        size = 0;
        char *user_page_aux;

        while (true){

                user_page_aux = pg_round_down(file);
                if (lock_page (user_page_aux)){
                        goto next;

                }else{
                        goto no_lock_worked;
                }
```

```
next:
while ( (user_page_aux + PGSIZE) > file ){

        kernel_file[size++] = *file;
        if (PGSIZE <= size){
                goto not_enough_space;
        }else if (*file == '\0'){
                unlock_page (user_page_aux);
                return kernel_file;
        }
        file++;
}

unlock_page (user_page_aux);
        }

    no_lock_worked:
    palloc_free_page(kernel_file);
    thread_exit();

    not_enough_space:
    unlock_page(user_page_aux);

}else{

    thread_exit ();
}
```

int SYSCALL_READ (int fd, void *buffer, unsigned length)

```
if (fd == STDIN_FILENO){
    int i;
    char input;
    for (i = 0; i < size_read; ++i){
        input = input_getc();
        ( lock_page (buf) ) ? buf[i] = input, unlock_page(buf) : thread_exit();
    }
    read_b = size_read;

}else{
    if (lock_page (buf)){
        lock_acquire(&lock_filesystem);
        result = file_read(fed->new_file_addr, buf, size_read);
        lock_release(&lock_filesystem);
        unlock_page(buf);
    }else{
        thread_exit ();
    }
}
```