



**UASLP**

Universidad Autónoma  
de San Luis Potosí



**FACULTAD DE  
INGENIERÍA**

## **Práctica 1. Archivo de Intercambio.**

Sistemas operativos B.

Profesora: M.I. Ortiz Hernández Marcela.

grupo: 2402-03.

Semestre: 2019-2020/II.

Cantu Olivares Pedro de Jesus.

Castro Silva David Alejandro.

Puente Villanueva Juan Arturo.

**13/Marzo/2020.**

## 1.- Resumen:

El objetivo de esta práctica es la creación de archivos de intercambio, para que contenga la información específica de los archivos de prueba ejecutados en la línea de comandos en NachOS.

Los conocimientos necesarios para llevarla a cabo son:

- ☐ Funciones con las que cuenta el sistema de archivos de nachos para la manipulación de archivos.
- ☐ Comprender y conocer los parámetros de las funciones para crear, abrir, leer, escribir y cerrar un archivo en nachos.
- ☐ Identificar las partes que forma un archivo ejecutable en nachos(es decir, la estructura interna del archivo ejecutable).
- ☐ Identificar donde se hace la verificación de que el proceso cabe en memoria.
- ☐ Ubicar y comprender cómo se carga el contenido del archivo ejecutable en memoria.
- ☐ Identificar de donde se puede obtener el nombre del archivo ejecutable a partir de la línea de comandos del administrador de memoria.

## 2.- Instrucciones:

En esta práctica se deberá generar el archivo de intercambio para cada archivo de prueba que sea ejecutado a través de la línea de comandos del administrador de memoria.

Un archivo ejecutable de nachos está formado por un encabezado y de los segmentos que contienen los datos y el código del proceso.

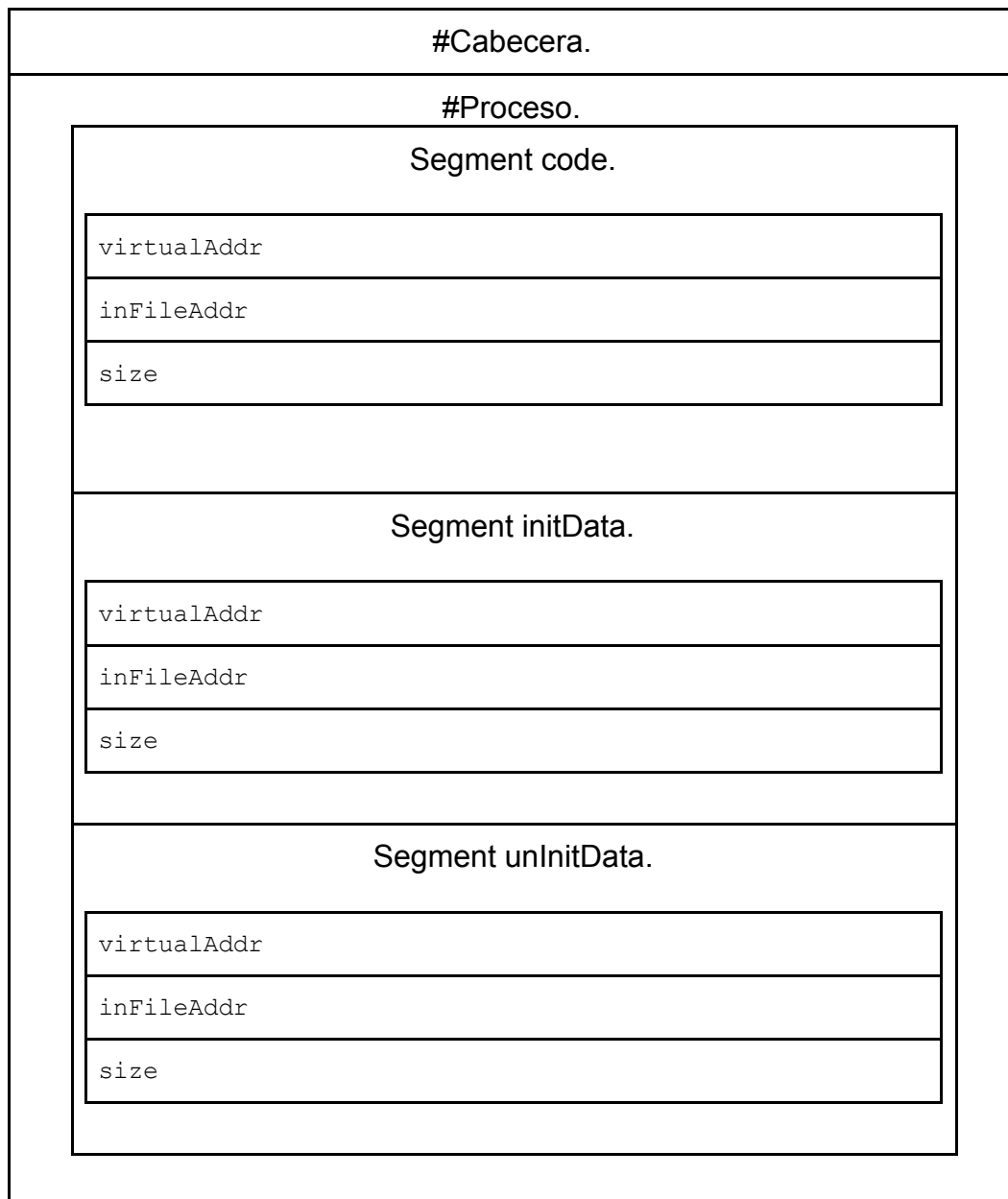
Se debe modificar Nachos de tal manera que antes de que se cargue el proceso a memoria se genere el archivo de intercambio, es decir, se debe crear un archivo y escribir los segmentos de datos y código del archivo ejecutable en el archivo recién creado. El archivo de intercambio se identificará por la extensión swp. Posteriormente, se deberá comparar el contenido del archivo ejecutable (despreciando el encabezado) y el contenido del archivo de intercambio y deberán ser iguales byte a byte. Esta verificación se realiza con el comando hexdump y se explica su funcionamiento en el archivo adjunto.

Los archivos de intercambio deberán generarse en el directorio donde se encuentra programa ejecutable de prueba.

Cada equipo deberá poner entre comentarios los nombres completos de los integrantes y la fecha de entrega de la práctica en el archivo addrspace.h y addrspace.cc

### 3. Explique y realice un diagrama de la estructura de un archivo ejecutable en nachos.

La estructura de un archivo ejecutable en NachOS se divide en tres segmentos. El código, los datos inicializados y los datos no inicializados. Estos segmentos tienen como información una dirección virtual, otra dirección dentro del archivo al que pertenecen y por último un tamaño. La información sobre estos tres segmentos se guarda en una cabecera.



#### 4. Indique el archivo y anexe el código donde encontró la definición de los segmentos del archivo ejecutable en nachos.

Recuperado de /nachos/bin/nof.h :

```
/* noff.h
 *   Data structures defining the Nachos Object Code Format
 *
 *   Basically, we only know about three types of segments:
 *   code (read-only), initialized data, and uninitialized data
 */

#define NOFFMAGIC    0xbadfad    /* magic number denoting Nachos
                                * object code file
                                */

typedef struct segment {
    int virtualAddr;    /* location of segment in virt addr space */
    int inFileAddr;    /* location of segment in this file */
    int size;    /* size of segment */
} Segment;

typedef struct noffHeader {
    int noffMagic;    /* should be NOFFMAGIC */
    Segment code;    /* executable code segment */
    Segment initData;    /* initialized data segment */
    Segment uninitData;    /* uninitialized data segment --
                            * should be zero'ed before use
                            */
} NoffHeader;
```

#### 5. Liste todos los archivos que modificó para esta práctica e indique una breve descripción de la modificación.

/nachos/userprog/progtest.cc:

##### Descripción:

En la función StartProcess se sabe el nombre de el archivo que se ejecutara. Por lo que en esta función se crea el nuevo archivo de intercambio llamando a la instancia global fileSystem, y pasandole el nombre (nombre de archivo + .swp) y el tamaño sacado del OpenFile del ejecutable.

```
void
StartProcess(char *filename)
{
    //Añadido practical:-----
    //crea el archivo y comprueba que efectivamente se haya creado.
```

```

if(!fileSystem->Create(strcat(filename, ".swp"), executable->Length()-40))
{
    printf("No se pudo crear el archivo\n");
}
else
{
    printf("archivo %s creado. \n", filename);
    swapOpenFile = fileSystem->Open(filename);
}
}

```

### **/nachos/userprog/addrspace.cc:**

#### **Descripción:**

En el constructor de AddrSpace que se llama desde progestest, se verifica si el archivo OpenFile para el intercambio se creó (es diferente de NULL). Si es así se procede a leer byte por byte, con un ciclo for la información del archivo ejecutable a partir del byte 40 (saltar el header del archivo).

Para este propósito se requiere un buffer(arreglo de caracteres) llamado temp. Dentro del ciclo. Llamar al método ReadAt del ejecutable hace el trabajo, de leer la información del ejecutable y ponerlo en el buffer. Posteriormente se procede a pasar esa información a el archivo de intercambio.

```

AddrSpace::AddrSpace(OpenFile *executable)
{
    //añadido Practical:-----
    if(swapOpenFile != NULL)
    {
        char *aux;
        for(int i = 0 ; i < executable->Length()-40; i++)
        {
            aux = new *char;
            executable->ReadAt(aux, 1, 40+i);
            swapOpenFile->Write(aux, 1);
        }
    }
    else
    {
        printf("El archivo de intercambio no existe\n");
    }
}

```

### **/nachos/threads/system.h:**

#### **Descripción:**

En system.h se tienen declaraciones de variables globales. Y puesto que el archivo de intercambio se manipula tanto en progestest.cc y addrspace.cc se decidió hacerla global. De otra manera se tendría que hacer otro constructor de AddrSpace,

pasando el nombre del archivo de intercambio. Se siguió la forma de declarar globalmente `fileSystem`, para declarar el `OpenFile` de intercambio.

```
#ifndef FILESYS_NEEDED          // FILESYS or FILESYS_STUB
#include "filesys.h"
extern FileSystem *fileSystem;
//añadido en Practical:-----
extern OpenFile *swapOpenFile;
//-----
#endif
```

### **/nachos/threads/system.cc:**

#### **Descripción:**

En `system.cc` se tienen definiciones de variables globales.

```
#ifndef FILESYS_NEEDED
FileSystem *fileSystem;
//añadido en la practical:-----
OpenFile *swapOpenFile;
//-----
#endif
```

### **6. Escriba el pseudocódigo para la generación del archivo de intercambio.**

1. Declarar archivo de intercambio.
2. Crear archivo de intercambio con el nombre del ejecutable(más la extensión) , y el tamaño calculado.
3. verificar que el archivo de intercambio se haya generado correctamente.
4. Leer la información del archivo ejecutable exceptuando la cabecera(40 bytes), guardandola en un buffer.
5. Escribir la información del buffer al archivo de intercambio creado en el paso 2.

### **7. Escriba el nombre del directorio de nachos donde se crean los archivos de intercambio de su práctica:**

**`./nachos/test/`**

### **8. Indique el nombre del método o la función utilizada en su práctica además de explicar los parámetros requeridos para:**

#### **a) Crear el archivo de intercambio**

**se declara en `system.h` y se crea en `progtest.cc`.**

`fileSystem->Create(strcat(filename, ".swp"), executable->Length()-40)`

Este método de `fileSystem(global)`, crea un archivo, para ello recibe por parámetros el nombre del archivo y el tamaño del mismo. El nombre se crea concatenando el

nombre del ejecutable original con la extensión “.swp”. Y el tamaño llamando al método Length del ejecutable(OpenFile), restando 40 bytes de la cabecera.

### **b) Leer archivo ejecutable**

El archivo ejecutable como OpenFile se puede leer con el método ReadAt. executable->ReadAt(aux,1,40+i). Recibe como parámetro el buffer donde se guardará la información a la que se dará lectura. el tamaño de la lectura, y por último la dirección desde la que hará la lectura.

### **c) Escribir en el archivo de intercambio**

swapOpenFile->Write(aux,1);. Este método de swapOpenFile(OpenFile). Recibe como parámetro el buffer que contiene la información que se escribiera y el tamaño.

## **9. Escriba el nombre del archivo y anexe el código donde se declaró el apuntador del archivo de intercambio.**

**En /nachos/threads/system.h:**

```
#ifndef FILESYS_NEEDED          // FILESYS or FILESYS_STUB
#include "fileys.h"
extern FileSystem *fileSystem;
//añadido en Practical:-----
extern OpenFile *swapOpenFile;
//-----
#endif
```

## **10. Indique el archivo donde obtiene el nombre del archivo ejecutable para posteriormente agregarle la extensión swp y anexe el código.**

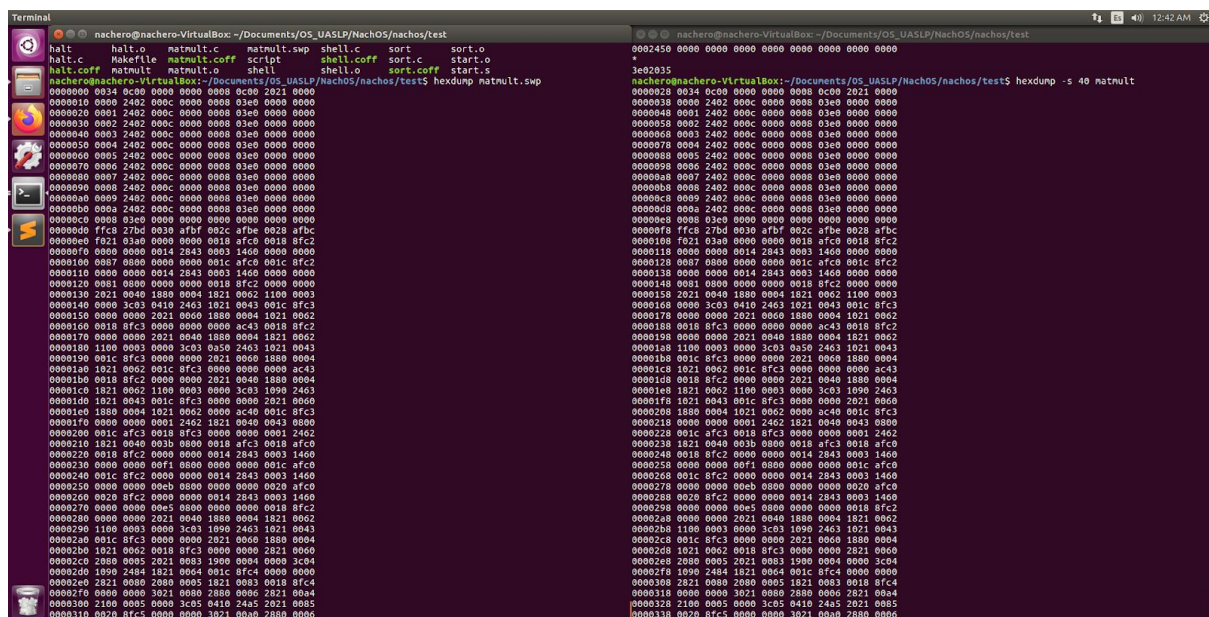
**/nachos/userprog/progtest.cc:**

```
void
StartProcess(char *filename)
{
//Añadido practical:-----
//crea el archivo y comprueba que efectivamente se haya creado.
if(!fileSystem->Create(strcat(filename, ".swp"), executable->Length()-40))
{
    printf("No se pudo crear el archivo\n");
}
else
{
    printf("archivo %s creado. \n", filename);
    swapOpenFile = fileSystem->Open(filename);
}
}
```

11. Escriba el comando que utilizó para comparar el contenido del programa ejecutable y el del archivo de intercambio para matmult.

- hexdump -s 40 matmult.
- hexdump matmult.swp.

12. Abra dos terminales, en la primera despliegue el contenido del archivo ejecutable de matmult omitiendo el encabezado, en la segunda terminal despliegue el contenido del archivo de intercambio generado para este programa de prueba (completo). Capture la pantalla y agregue la imagen.



The image shows two terminal windows side-by-side. The left window displays the output of the command `hexdump -s 40 matmult`, showing a hex dump of the matmult executable starting at address 00000000. The right window displays the output of the command `hexdump matmult.swp`, showing a hex dump of the swap file matmult.swp starting at address 0002450. Both windows show a large amount of hexadecimal data in columns.

13. Mencione problemas que se hayan presentado en la implementación del archivo de intercambio.

- La implementación de la generación de el archivo de intercambio se intentó hacer al principio de una manera muy diferente. Leyendo la información desde un buffer de tipo NoffHeader. Lo cual nunca sucedió.
- También se pensó en crear otro constructor para la clase AdrSpace para pasar el nombre del archivo ejecutable. Lo cual se descarto.
- Por último. La correcta ubicación de la declaración del apuntador para el archivo de intercambio, para su uso en diferentes partes de NachOS.

14. Mencione como realizó la depuración y prueba de su práctica.

- Se fue verificando paso a paso los cambios realizados con la ayuda de impresiones en pantalla(`printf`). Tales como el nombre del archivo que se generó. El tamaño. Se dejaron condicionales que, imprimen mensajes de



error si el archivo no se genera correctamente, y otro en la escritura del mismo si es NULL en ese momento.

- Después con la ayuda de los comandos descritos en el punto 11 se compararon los contenidos de los archivos.

**15. Describa a detalle lo que realizó cada integrante del equipo en esta práctica (sea objetivo y realista).**

**Pedro:**

Intente resolver el problema de varias maneras que no funcionaron, pero me ayudaron a entender el funcionamiento de los procesos involucrados(estuve cerca).

**David:**

Propuse una nueva estrategia para la solución que funcionó.

**Arturo:**

Fui el depurador. Mi código funcionó primero y fue el que se enseñó en la revisión.

Al final todos aportamos a la práctica para que la implementación funcionara correctamente.

**16. Escriba la cantidad total de horas que trabajaron para esta entrega(efectivas de trabajo).**

4-5 Horas.

**17. Conclusiones**

Nachos maneja archivos de manera relativamente sencilla. Esto nos permite generar archivos a partir de otros y así implementar diferentes estrategias de sistemas operativos más complejos, tal es la memoria virtual.

Los archivos de intercambio almacenan segmentos de un proceso en ejecución. Para poder simular la memoria virtual son necesarios dichos archivos.