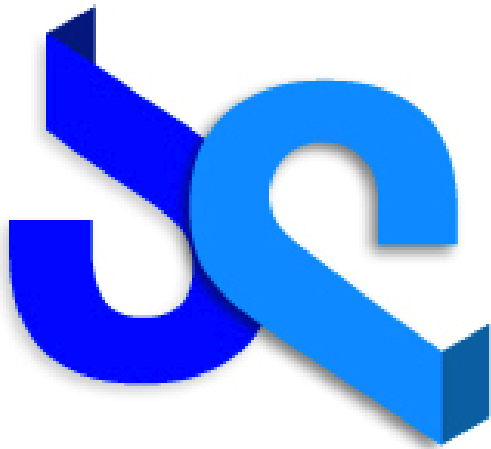




Universidad Tecnológica Nacional – Facultad Regional Buenos Aires  
Ingeniería en Sistemas de Información  
Sistemas Operativos (082027)



# sistemas operativos

**Trabajo Práctico– 1er cuatrimestre 2011**

**Plataforma de Almacenamiento Distribuida en  
Capa de Aplicación**

**Revisión 5**



# Indice

---

<b>Indice</b> .....	2
<b>1.-Introducción</b> .....	4
<b>2.- Objetivos del Trabajo Práctico</b> .....	4
<b>3.- Características del sistema</b> .....	5
<b>4.- Aspectos de diseño de los procesos</b> .....	7
Servidor FTP (FTPS).....	7
Definición .....	7
Desarrollo .....	7
KSS –Kernel Sumamente Simple.....	8
Definición .....	8
Desarrollo .....	8
System Calls del KSS.....	8
Tabla de Descriptores (TDD) .....	9
Archivo de configuración .....	9
Nombres de archivo.....	10
Acceso a los dispositivos de almacenamiento .....	10
Proceso de formateo: .....	11
FSS – File System Simplificado.....	11
Definición .....	11
Desarrollo .....	13
Comportamientos definidos.....	14
Dispositivo de almacenamiento Virtual (VDA).....	14
Definición: .....	14
Desarrollo: .....	15
Comportamientos definidos.....	17
<b>5.-Descripción de las entregas</b> .....	19
5.1.-Primer entrega .....	19
5.2.-Segunda entrega .....	20
5.3.-Tercer entrega.....	21
5.4.-Cuarta entrega .....	21
4.5.-Quinta entrega – Entrega Final .....	22



<b>6. Requerimientos técnicos y limitaciones</b> .....	23
<b>7. Anexos</b> .....	26
Anexo A :: Archivo Log y Debugging .....	26
Formato del archivo Log .....	26
Eventos significativos a ser logueados obligatoriamente .....	27
Anexo B :: Systems Calls del KSS:.....	28
Anexo C :: Protocolos de comunicación .....	31
Protocolo MPS – (ex IPC/IRC) .....	31
Handshake .....	32
Anexo D :: Persistencia del FSS.....	33
Anexo E :: Documentación.....	33
Links recomendados: .....	34



## 1.-Introducción

---

El trabajo práctico de este cuatrimestre pretende simular mediante el uso de procesos de tipo usuario interconectados por sockets TCP/IP una estructura de almacenamiento distribuida para dar soporte a un servidor de FTP estándar.

El objetivo del mismo es que mediante el desarrollo de esta simplificación el alumno se interiorice y comprenda los diversos los mecanismos que utiliza el sistema operativo para gestionar los medios de almacenamiento y su relación e interacción con los sistemas de archivos. No se pretende enfocar en el funcionamiento específico de un determinado sistema operativo sino por el contrario mostrar conceptos y aspectos de diseño generales.

Dada la complejidad del mismo, este trabajo fue dividido en entregas parciales las cuales permiten diseñar la arquitectura definitiva, de forma gradual, simplificando su desarrollo.

Se enumeran a continuación los conceptos teóricos y prácticos más significativos sobre sistemas operativos que cubre el trabajo práctico y que el alumno aprenderá:

- ✚ Creación y manipulación de procesos y threads mediante la API del sistema.
- ✚ Sincronización de Procesos, IPC y manejo de señales.
- ✚ Administración de Memoria
- ✚ Manejo del file system a través de la interfaz de sistema.
- ✚ Introducción a los Sistemas Distribuidos
- ✚ Arquitecturas basadas en capas físicas, protocolos y mensajería.
- ✚ Diferencias entre las plataformas más populares del mercado.

## 2.- Objetivos del Trabajo Práctico

---

Desde el punto de vista académico el trabajo está diseñado para:

- ✚ Que los alumnos adquieran los conocimientos prácticos del uso de un conjunto de herramientas que ofrecen los sistemas operativos modernos.
- ✚ Que entiendan la importancia de una norma o protocolo estándar en la comunicación entre procesos y diferentes plataformas.
- ✚ Que dominen los problemas específicos de este tipo de implementaciones.

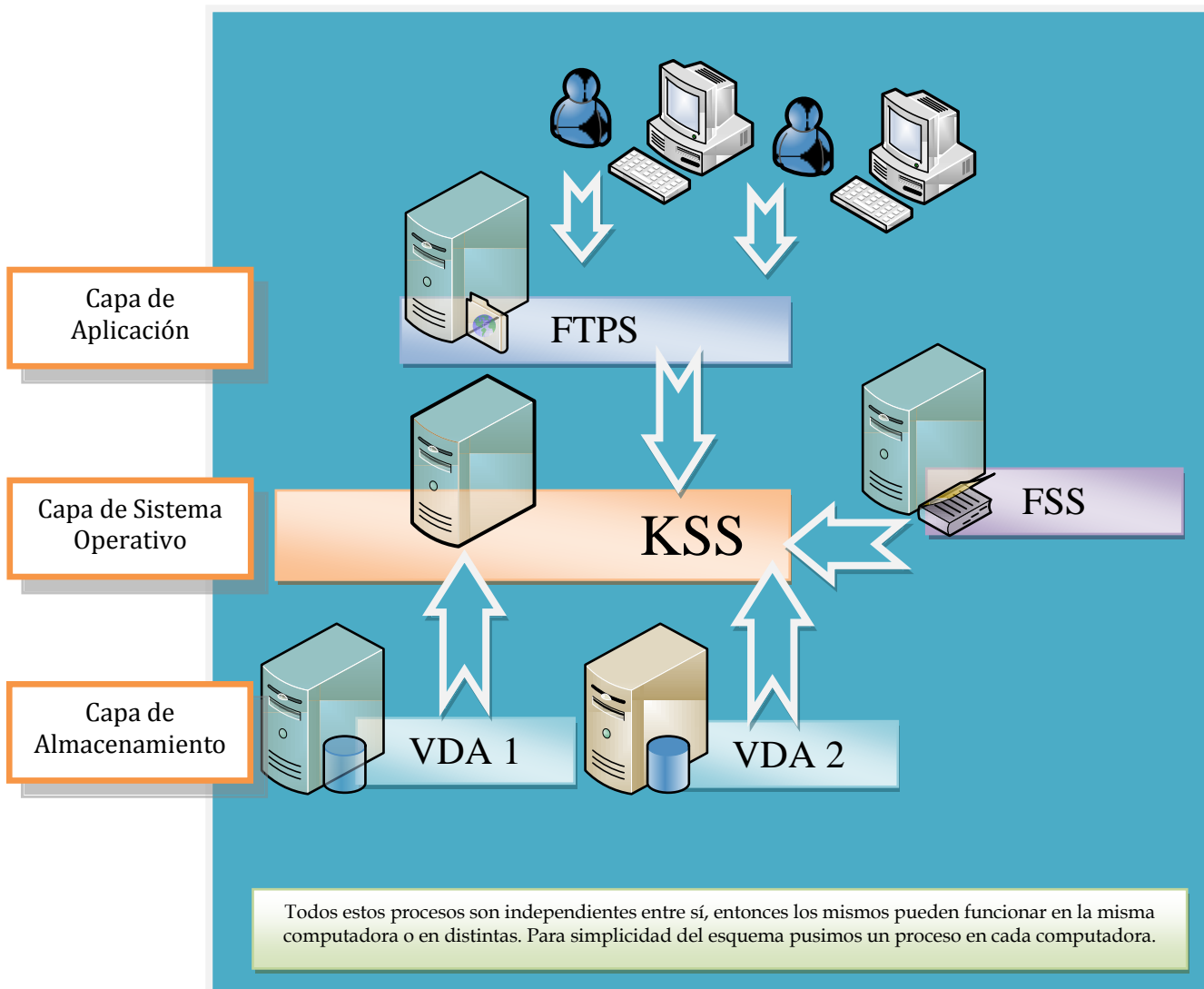


- ✚ Que apliquen en forma práctica el uso de lenguaje C en implementaciones de bajo nivel.
- ✚ Que el grupo de alumnos aprendan el trabajo en equipo, las problemáticas y las responsabilidades que eso implica.

### 3.- Características del sistema

#### Diagrama de componentes

A continuación se hace una breve introducción a cada uno de los componentes que intervienen en el sistema. Con el objetivo de tener una clara visión de las responsabilidades de cara proceso y para simplificar su comprensión y el mismo ha sido dividido en tres capas: *Aplicación, Sistema Operativo y Almacenamiento*





## *Capa de Aplicación*

Compuesta por los componentes que consumen servicios que facilita la capa de Sistema Operativo mediante la API provista por el mismo

- ✚ **Servidor FTP - FTPS (Windows):** Servidor FTP que recibe conexiones de clientes FTP estándar y utiliza los servicios del KSS para el almacenamiento y gestión de los archivos.

## *Capa de Sistema Operativo*

Compuesta por los componentes que proveen servicios a la capa de Aplicación mediante una API. Consumen servicios de la capa de almacenamiento para tareas de persistencia de datos.

- ✚ **Kernel Sumamente Simple - KSS (Linux):** Proceso encargado de gestionar el acceso a los dispositivos de almacenamiento y brindarles una coherencia gracias a la información extraída del FSS
- ✚ **File System Simple - FSS (Linux):** Proceso que contiene los índices que ordenan la información de los bloques de dispositivos de almacenamiento virtuales en archivos.

## *Capa de Almacenamiento*

Compuesta por los dispositivos que facilitan la persistencia de información. En la vida real los discos pueden comunicarse con la capa de Sistema Operativo mediante diversos métodos de conexión, sea de forma directa (DAS) o mediante una red de transporte de datos como puede ser IP o FiberChannel. El protocolo más utilizado para estos casos es el SCSI.

- ✚ **Dispositivos de Almacenamiento Virtuales - VDA (Windows):** Proceso que almacena el contenido de los archivos en bloques numerados de 512 bytes. Además funciona como un simulador de un disco convencional para propósitos educativos.



## ***4.- Aspectos de diseño de los procesos***

---

### ***Servidor FTP (FTPS)***

Los servidores FTP convencionales, o de transferencia de archivos por su sigla en inglés, permiten a un usuario que utilice un cliente FTP estándar subir y descargar archivos que estén almacenados en el mismo.

**Ejemplo:** <ftp://ftp.freshrpms.net/>

### ***Definición***

Este trabajo práctico tiene como objetivo desarrollar un servidor FTP que pueda ser consultado con cualquier cliente que, lógicamente, cumpla con el estándar, el cual está detallado en: <http://www.rfc-es.org/rfc/rfc0959-es.txt>

Es importante recalcar que dada la extensión del protocolo, solo serán soportadas algunas funciones.

Un servidor FTP convencional, al recibir un mensaje solicitando que se le envíe un archivo, deberá corroborar que el mismo exista en su disco local, abrirlo y enviar el contenido del mismo utilizando el protocolo de mensajería correspondiente.

Sin embargo, el servidor de este trabajo práctico tiene una característica particular. El mismo en vez de utilizar archivos guardados localmente, haciendo uso de la API provista por el sistema operativo, se comunicará mediante el paso de mensajes, con el proceso KSS y hará uso del sistema de almacenamiento distribuido desarrollado por el alumno.

### ***Desarrollo***

Esta aplicación se desarrollará para la plataforma Windows y será multithreaded, ya que cada hilo atenderá las peticiones de un cliente, existiendo la posibilidad de que más de un cliente esté haciendo uso del servidor en un instante dado. De esta manera se aprovecha al máximo las capacidades de los sistemas multiprocesadores actuales.



## *KSS - Kernel Sumamente Simple*

### *Definición*

Este es el proceso principal de la denominada capa de sistema operativo, ya que es el encargado de coordinar los demás elementos de la plataforma para proveer el servicio de almacenamiento.

En un sistema operativo real, esta tarea la lleva a cabo el Kernel. Como en este trabajo se simulan muy pocas características del mismo, de ahí proviene su nombre.

### *Desarrollo*







El KSS será un proceso funcionando a nivel de aplicación en una computadora con sistema operativo Linux. El mismo estará a la espera de conexiones TCP/IP de Servidores FTP, y luego de realizar la correspondiente autenticación, (ver anexo handshake), quedará a la espera de mensajes solicitando servicios. Al recibir uno, llamará a la rutina de código que atiende esa solicitud.

Dado que la simple presencia de este proceso Kernel no es suficiente para hacer uso de la plataforma, este denegará solicitudes hasta que el mismo se encuentre en estado operacional. Este estado se alcanza al contar con la conexión de un proceso FSS y al menos un dispositivo de almacenamiento formateado.

Por último, el KSS contará con un Shell. Una consola en modo texto que permitirá ejecutar funciones propias del Kernel para la administración del mismo y de los diversos dispositivos (ver anexo Shell)

### *System Calls del KSS*

El kernel pondrá a disposición de las aplicaciones las siguientes system calls las cuales podrán ser accedidas mediante el paso de mensajes utilizando el protocolo MPS. Ver Anexo B para la especificación de cada rutina.

-  `sys_open()`: Abre un archivo existente.
-  `sys_read()` Lee un flujo de 1 KByte de un archivo.
-  `sys_write()` Escribe un flujo de 1 KByte en un archivo.
-  `sys_close()` Cierra un archivo abierto.
-  `sys_flush()` Vuelca el contenido del buffer de escritura al disco.
-  `sys_flist()` lista un directorio del FSS.





## Tabla de Descriptores (TDD)

El proceso Kernel KSS manejará una tabla de archivos llamada Tabla de Descriptores (TDD) la cual contiene información de los archivos que en un instante de tiempo estén abiertos y sean utilizados.

Esta tabla será propia del KSS y deberá contener los siguientes campos y tipos de datos:

Descriptor único (unsigned int)	Nombre del VDA (char*)	Nombre de archivo (char*)	Tipo de apertura (int)	Tamaño total bytes (long)	Buffer (1KB)	Lista de sectores que componen el archivo (struct)
5	discoA	passwd.txt	0	1450	{contenido del archivo}	{lista de número de sectores que componen el archivo}
6	discoA	documento.doc	1	7750	...	...

### Descriptor único:

Al igual que los sistemas operativos convencionales este número hará de referencia o “handle” para un archivo abierto. Luego de ser solicitada la apertura de un archivo, el KSS brindará el número de descriptor con el que se manejarán hasta que el mismo sea cerrado.

### Nombre del VDA y Nombre del Archivo:

Estos datos refieren exactamente la ubicación y el nombre del archivo que se está utilizando.

### Tamaño total bytes:

La TDD debe contar todo el tiempo con el tamaño actual del archivo. Si este se está leyendo dicho valor no se modificará, pero si se estuviera escribiendo dicho valor deberá ser actualizado.

### Buffer:

En este trabajo práctico todas las lecturas y escrituras que se realicen a los dispositivos de almacenamiento se harán en bloques de tamaño fijo de 1KB y pasarán previamente por este buffer. De esta manera, al igual que cualquier sistema que tenga un productor y un consumidor, se amortiguan las diferencias de velocidades entre dispositivos.

### Lista de sectores que componen el archivo:

Será una lista anidada que contendrá el valor numérico de los sectores que componen un archivo. Al igual que el buffer será detallada más adelante en este documento.

## Archivo de configuración

Al iniciar el proceso, este obtendrá las siguientes características de un archivo de configuración:



- ✚ Puerto TCP en el que escuchará conexiones de dispositivos de almacenamiento y FSS
- ✚ Puerto TCP en el que escuchará conexiones de FTPS

## *Nombres de archivo*

Los nombres de archivo que manejará la plataforma corresponden a la siguiente sintaxis:

`/[VDA]/[nombre de archivo]`

Donde [VDA] refiere al nombre del dispositivo de almacenamiento virtual y [nombre de archivo] al nombre de dicho archivo el cual puede ser una secuencia de **hasta 30 caracteres combinando alfanuméricos y el carácter punto**.

Ejemplo:

`/discoA/PassWord99.txt`

*Nota: Tanto los nombres de los archivos y como los nombres de los dispositivos de almacenamiento deberán ser CASE-SENSITIVE (sensibles a mayúsculas) y no se soportan espacios.*

## *Acceso a los dispositivos de almacenamiento*

Como vimos anteriormente, al KSS se le conectarán diversos dispositivos de almacenamiento. Luego del correspondiente handshake, el KSS conocerá el nombre identificador correspondiente a dicho dispositivo.

Con dicho identificador, mediante el Shell, el administrador podrá montar dicho dispositivo de almacenamiento.

Al recibir la solicitud de un montaje, el KSS consultará al FSS, mediante el correspondiente paso de mensajes, si posee un formato existente para dicho dispositivo de almacenamiento virtual (VDA), en otras palabras si posee una tabla de sectores libres y una tabla de archivos.

En caso afirmativo se concluirá el montaje y los datos del VDA estarán a disposición de las aplicaciones.

En caso contrario se solicitará por pantalla que formatee dicho dispositivo (ver Proceso de formateo)

El montaje se realizará siguiendo una estructura de directorios donde el directorio raíz es “/”, cada VDA es montado como un directorio separado, el separador de directorios es “/”.



## *Proceso de formateo:*

Se le solicitará al VDA su información física CHS (Cylinders, Heads and Sectors) utilizando el protocolo correspondiente.

Con dicha información el KSS calculará la cantidad de sectores totales y la capacidad máxima de almacenamiento. Luego le solicitará al FSS que cree la tabla de sectores libres para ese disco.

Recuerde que este procedimiento se puede utilizar también como una forma rápida y efectiva de vaciar el contenido de un VDA.

### REFLEXIONE:

¿Qué ocurren con los datos almacenados en un VDA concluido el proceso de formateo? ¿Acaso estos datos son eliminados? ¿Ocurre lo mismo en un sistema operativo comercial?

## *FSS – File System Simplificado*

### *Definición*

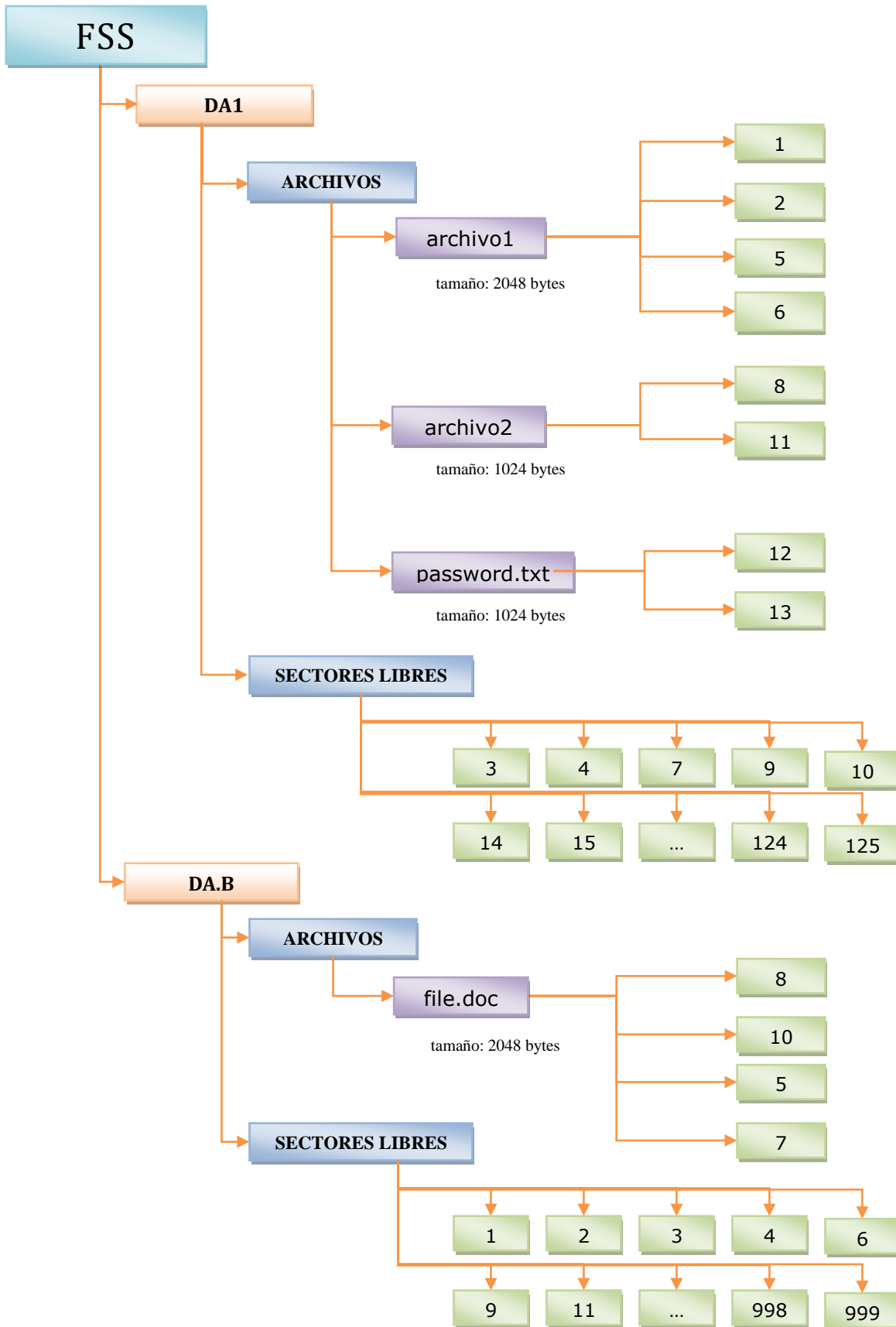
Este proceso será un simple índice que le permitirá al KSS asociar un nombre de archivo con los sectores que lo componen en el correspondiente dispositivo de almacenamiento virtual.

A diferencia de los sistemas de archivos convencionales que se almacenan dentro del propio dispositivo de almacenamiento en sectores predefinidos del disco, este será externo y en una aplicación remota. El KSS consultará la información requerida y le indicará que ejecute ciertos procedimientos mediante el correspondiente paso de mensajes.

Por simplicidad el FSS no soportará estructuras de directorios.

Para la administración del espacio libre, se deberá desarrollar y utilizar una estructura de tipo bitmap (también llamada bit vector ó bit-array). La misma deberá ser operada con operadores de manejo de bits (“bitwise operators”).

En la siguiente figura podemos observar la estructura general que deberá tener el FSS para almacenar los archivos.





Note que cada dispositivo de almacenamiento (DA1 y DA.B) tiene una lista de los archivos grabados en el mismo y una lista de aquellos sectores que aún no han sido grabados.

Podemos decir que el dispositivo de almacenamiento DA1 contiene a los archivos: “**archivo1**”, “**archivo2**” y “**password.txt**” y que el “archivo1” está compuesto por 4 sectores (1, 2, 5 y 6) de este dispositivo de almacenamiento, por lo cual su tamaño final es de 2048 bytes. Recuerde que cada sector del disco es de tamaño fijo de 512 bytes.

Note que los sectores usados por los archivos no figuran en la lista de “sectores libres”

### REFLEXIONE:

¿Qué ocurriría con la consistencia del archivo si cambiáramos el orden de sus sectores. ¿Sería el mismo? ¿Qué pasaría si dos archivos compartieran el mismo sector?

¿Qué ocurre si el archivo no ocupa exactamente los 2048 bytes. ¿Qué ocurre si ocupara menos? ¿Si ocupara más?

*Es muy importante que el sistema mantenga la consistencia de esta información ya que un dato grabado en un sector que luego figure como libre puede desencadenar en que dicho sector sea sobre escrito y una posterior corrupción de los datos.*

Adicionalmente se recomienda leer los capítulos 10 y 11 del Libro Sistemas Operativos de Silberschatz-Galvin

## **Desarrollo**

Estará desarrollado para la plataforma Linux. Como soporte de datos se usará el propio sistema de archivos del sistema operativo, usando directorios y archivos para indicar los archivos existentes y sus sectores asignados. Al final del documento se provee un anexo que explica cómo se persistirá su estructura de datos.

Como todos los procesos, el FSS, proveerá, una API con la cual se podrá comunicar con el KSS mediante el paso de mensajes utilizando el protocolo MSP.

**existeArchivo():** Dado el nombre de un VDA y un nombre de archivo, verificará que dicho archivo exista en el dispositivo.

**infoArchivo():** Dado el nombre de un VDA y un nombre de un archivo, devolverá la tabla de sectores que componen al archivo y su tamaño total.

**eliminarArchivo():** Dado el nombre de un VDA y un nombre de un archivo, eliminará dicho registro de la tabla de archivos y agregará los



sectores que lo componían a la tabla de sectores libres usando la primitiva: *liberarSectores()*.

**crearArchivo()**: Dado el nombre de un VDA y un nombre de archivo, creará el registro en la tabla de archivos asignando espacio cero y con una lista de sectores vacía (sin sectores).

**actualizarTamaño()**: Dado el nombre de un VDA, un nombre de archivo y un valor *long*, actualizará campo correspondiente al tamaño del archivo en el registro de la tabla de archivos.

**crearTablaSectoresLibres()**: Dado el nombre de un VDA y una cantidad de sectores int, creará un bitmap de sectores libres desde el 0 a la cantidad N-1 para el correspondiente VDA.

**dosSectoresLibres()**: Dado el nombre de un VDA, quitará dos sectores libres de la lista y devolverá sus identificadores.

**asignarSectores()**: Dado el nombre de un VDA, un nombre de archivo y dos número de sector, asignará esos sectores al archivo.

**liberarSectores()**: Dado el nombre de un VDA y una lista de sectores, agrega dichos sectores a la tabla de sectores libres de dicho VDA.

**listarDirectorio()**: Dado el nombre de un VDA, devolverá los nombres y tamaños de todos los archivos existentes dentro de dicho VDA.

**formatear()**: Dado el nombre de un VDA y una cantidad de sectores int, vaciará la tabla de archivos y creará una nueva tabla de sectores libres invocando las primitivas correspondientes (*eliminarArchivo*, *crearTablaSectoresLibres*).

## **Comportamientos definidos**

Al iniciar el proceso este obtendrá la dirección IP y puerto TCP del KSS de un archivo de configuración e intentará conectarse. Si la conexión es satisfactoria, quedará a la espera de requerimientos del mismo.

## **Dispositivo de almacenamiento Virtual (VDA)**

### **Definición:**

Este proceso simulará un disco rígido real. Tendrá como características sus valores de CHS, retardo rotacional y posición actual del cabezal de lectura, los cuales serán provistos inicialmente por archivo de configuración. **Por simplicidad este disco contará con un solo cabezal.**

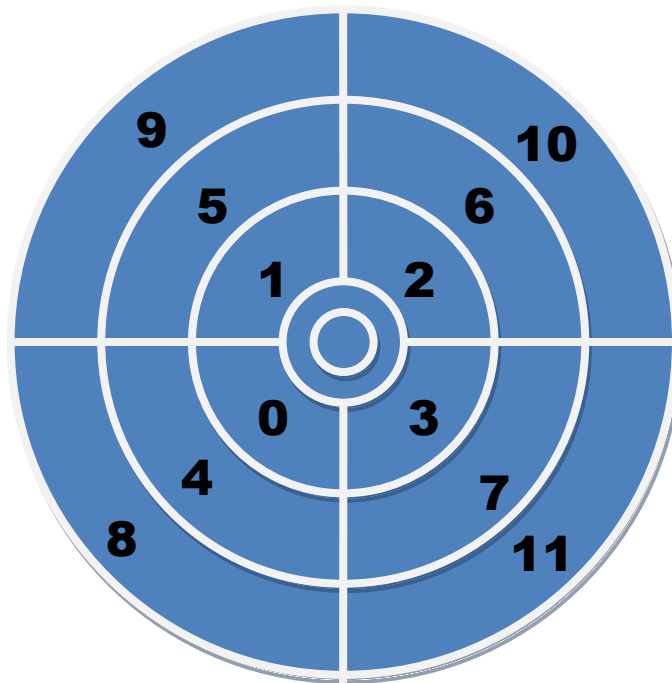


Al recibir una solicitud de sector, deberá calcular **el tiempo que le insumiría ubicar dicho sector si se tratara de un disco duro convencional** utilizando un algoritmo que será asignado a cada grupo individualmente e imprimirlo por pantalla. Luego responder la solicitud.

Utilizará una base BerkeleyDB para almacenar los sectores de datos de 512 Bytes de forma que los mismos puedan persistir una vez dado de baja el proceso y los organizará identificándolos mediante el número de sector iniciando por el cero.

El alumno deberá desarrollar también por código una caché de disco mediante una estructura que permita almacenar en memoria 10 sectores. El algoritmo de reemplazo de la caché será LRU (Last Recently Used).

Este proceso contará también con una simple consola que permitirá simular el pedido de sectores.



La figura muestra un ejemplo de la disposición de los sectores en caso de tratarse de un disco de 1 cabezal, 3 cilindros y 4 sectores por pista: CHS (1, 3, 4)

### *Desarrollo:*



Este proceso será desarrollado para funcionar en Windows. Al iniciar, verificará en su archivo de configuración si tiene la cache activada y creará las estructuras correspondientes.

Como todos los procesos, el disco, proveerá, una API definida por el alumno con la cual se podrá comunicar con el KSS mediante el paso de mensajes utilizando el protocolo de mensajería simple MSP.

**getSectores():** Recibe como parámetro dos números enteros de sectores.

Antes de devolver la información contenida en dichos sectores al solicitante, calculará e imprimirá en el log:

- el tiempo que tardaría en servir dicho requerimiento un disco rígido real
- los sectores que tendría que atravesar antes de servirlos
- la posición actual del cabezal
- promedio de cache hit/miss

Devuelve 1024 bytes donde los primeros 512 Bytes corresponden al primer sector solicitado y los siguientes al segundo. Si la lectura En caso de que el número de sector sea “null” se deberá responder con 512 bytes en ‘\0’

**putSectores():** Recibe una estructura de 1032 bytes contiguos

<b>Id sector 1</b> <b>(4 Bytes)</b>	<b>Buffer sector 1</b> <b>(512 Bytes)</b>	<b>Id sector 2</b> <b>(4 Bytes)</b>	<b>Buffer sector 2</b> <b>(512 Bytes)</b>
----------------------------------------	----------------------------------------------	----------------------------------------	----------------------------------------------

Antes de escribir los sectores y notificar el éxito de la operación al solicitante, el proceso calculará y escribirá en el archivo log del proceso:

- el tiempo que tardaría en escribir dicho sector
- los sectores que tuvo que atravesar antes de escribirlo
- la posición actual del cabezal

Devuelve un número entero con la cantidad de bytes escritos en el disco o cero en caso que la operación falle.

**getCHS():** No recibe parámetros y devuelve tres enteros correspondientes a la información física obtenida de archivo de configuración.

## Consola

El VDA cuenta con una consola la cual podrá ser accedida por teclado en todo momento de operación del disco.

La consola deberá soportar al menos los siguientes comandos:

**posicionCabezal():** Imprimirá en pantalla la posición actual del cabezal





**obtenerSectores([lista sectores]):** recibirá como parámetro una lista de no más de 5 sectores y deberá mostrar en pantalla por cada sector la información descrita en los comportamientos definidos junto con los primero 10 y los últimos 10 caracteres de dicho sector.

**Ejemplo:** obtenerSectores(100, 101, 102)

```

Posición actual: 0
Sectores Leídos: 21, 32, 43, 54, 65, 76, 87, 98, 99, 100
Tiempo consumido: 16ms
Sector 100: "Información..."del final."
Sectores Leídos: 101
Tiempo consumido: 2ms
Sector 101: "primeros "...ultimos 10 "

Sectores Leídos: 102 - En cache
Tiempo consumido: 0.1ms
Sector 102: "aabbccdde"... "1234567890"

```

Es importante recalcar que si bien el proceso simula un disco real dados sus parámetros físicos, esta información será únicamente mostrada en pantalla. **De ninguna manera se pretende que el tiempo de respuesta de los mensajes del proceso coincida con los tiempos calculados.**

## Comportamientos definidos

Al iniciar el proceso este obtendrá las siguientes características de un archivo de configuración:

- Nombre (no deberá repetirse en el sistema)
- Dirección IP y puerto TCP del proceso KSS
- CHS. La cantidad de sectores debe ser siempre par.
- Revoluciones por minuto
- Cache

Luego calculará y mostrará en el archivo log del proceso y en pantalla:

- Tiempo medio y peor tiempo de búsqueda de un sector
- Tasa de transferencia
- Capacidad total del disco
- Cantidad de sectores
- Tiempo medio de acceso de la cache

Si el proceso no encontrara un archivo de configuración o en el mismo estuviera explícitamente determinado que no debe conectarse a un KSS, el mismo deberá funcionar solo en modo consola y atender los pedidos que por allí sean solicitados.

### REFLEXIONE:



Basándose en la figura, qué ocurriría si con el cabezal posicionado en el sector 4 quisiera leer el sector 10. Investigue como funciona un disco rígido convencional y cuál es el tiempo que demoraría.

Libro de Sistemas Operativos - **Silberschatz-Galvin:**

- Capítulo 12: Disco
- Capítulo 13: I/O



## 5.-Descripción de las entregas

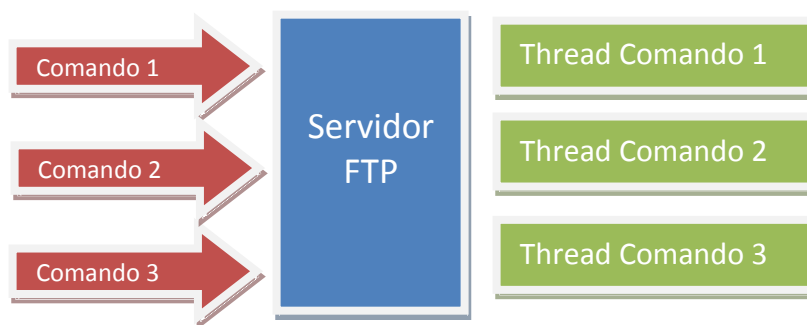
*Es importante recalcar que si bien las entregas tienen sus fechas de entrega y están numeradas y ordenadas existe mucho trabajo que correctamente, dividido entre los integrantes del grupo, se puede realizar de forma paralela.*

### 5.1.-Primer entrega

En esta entrega se deberá desarrollar la primera versión del Servidor FTP. Se iniciará por línea de comando y tomará los parámetros de un archivo de configuración (que se encontrará en mismo directorio donde corre la aplicación).

Una vez iniciado, aceptará las conexiones de clientes sin importar su usuario o password (modo anónimo) y negociará con el cliente el **modo pasivo**.

Una vez establecida la conexión desde el cliente creará un thread para cada cliente para manejar la conexión de comandos y otro thread para la conexión de datos (dos threads por cliente) según el protocolo FTP.



Para simplificar la familiarización del alumno con el trabajo práctico y la correcta modularización de las funciones, se recomienda que en esta entrega el servidor comparta la información de un directorio local de la computadora configurado por archivo.

Consultar el documento “Simulando una sesión FTP” (no incluido en esta especificación) para mayor información sobre la implementación del protocolo FTP

Deberá soportar al menos los siguientes comandos

Comando	Descripción
cd directorio	Cambia el directorio de trabajo en el servidor



ls	Muestra el contenido del directorio en el que estamos en el servidor
delete archivo	Borra un archivo en el servidor.
get archivo	Obtiene un archivo.
put archivo	Envía el archivo indicado al directorio activo del servidor.
? o help	Muestra una lista de los comandos disponibles
noop	(No Operation) Se le comunica al servidor que el cliente está en modo de no operación, el servidor usualmente responde con un "ZZZ" y refresca el contador de tiempo inactivo del usuario.
pasv / port	Comandos necesarios para iniciar la comunicación de modo pasivo.
pwd	Muestra el directorio activo en el servidor

#### Modo de testeo de la entrega

- Se recomienda verificar la integridad de los archivos transferidos en ambos extremos con el comando md5sum. <http://www.etree.org/md5com.html#download>
- Controlar que el uso de CPU corresponda con el procesador correcto en caso de correr sobre un sistema multiprocesador.

**Tiempo estimado** dos semanas

**Tipo de entrega** No Obligatoria

**Fecha** 07/05/2011

## 5.2.-Segunda entrega

En esta entrega se desarrollará el proceso de dispositivos de almacenamiento virtuales (VDA).

Es de suma importancia dejar la consola funcionando y testear las planificaciones del cabezal.

Se recomienda que el alumno tenga un método para probar la correcta escritura y lectura de sectores mediante una ampliación de la consola.

#### Modo de testeo de la entrega

- Leer y escribir bloques en los dispositivos de almacenamiento
- Mediante la consola probar el funcionamiento del simulador de disco.



<b>Tiempo estimado</b>	Tres semanas
<b>Tipo de entrega</b>	No Obligatoria
<b>Fecha</b>	28/05/2011

### 5.3.-Tercer entrega

En esta entrega se creará el proceso KSS. Aceptará conexiones del FTPS y normales de los dispositivos de almacenamiento. Desde el Shell, el KSS deberá poder ejecutar comandos e imprimirá los resultados en pantalla.

#### KSS Shell

El KSS permitirá la conexión de un Shell mediante el uso de sockets Unix. Este Shell que podrá ejecutar al menos los siguientes comandos:

**mount** : Montar un VDA

**umount**: Desmontar un VDA

**format**: Formatear un VDA\*

**ls**: Listar los archivos dentro un VDA o de la raíz.\*

**tdd\_dump**: Imprimir el contenido de la TDD

**md5sum**: Bajar el contenido de un archivo al disco local y ejecutar un md5sum sobre el mismo.\*

\*Estos comandos no estarán disponibles en esta entrega.

#### Modo de testeo de la entrega

- Conectar y desconectar VDAs.
- Utilizar rutinas para leer y escribir sectores en los discos utilizando el buffer intermedio de la OFT.

<b>Tiempo estimado</b>	Tres semanas
<b>Tipo de entrega</b>	Obligatoria
<b>Fecha</b>	18/06/2011

### 5.4.-Cuarta entrega

En esta entrega, a la estructura actual se debe agregar el proceso FSS.



El mismo se conectará al KSS y pondrá a su disposición sus rutinas para ordenar lógicamente los sectores de los VDA (*Ver Anexo: Primitivas del FSS*)

En esta entrega también se completará el funcionamiento del Shell del KSS, lo que le permitirá entre otras cosas formatear discos que al montarse no tengan formato.

El KSS publicará las correspondientes system calls para acceder a los archivos. (*Ver Anexo: System Calls del KSS*)

#### Modo de testeo de la entrega

- Montar una unidad de disco. Darle formato. Crear, borrar y leer archivos enviando mensajes al KSS.

**Tiempo estimado** dos semanas

**Tipo de entrega** No Obligatoria

**Fecha** 02/07/2011

### 4.5.-Quinta entrega - Entrega Final

En esta entrega se modificará las funciones dentro del servidor FTP para que en vez de utilizar la Windows API y el file system local, interactúe con la API del KSS y por ende pueda utilizar el sistema de archivos desarrollado por el alumno.

Al recibir una solicitud de parte de un cliente FTP, utilizará el vínculo que mantiene con el KSS para solicitar la información necesaria. Una vez recibida dicha información de parte del KSS, le responderá al cliente FTP su solicitud.

#### Modo de testeo de la entrega

- Analizar posibles problemas de concurrencia en el acceso a la API. Recordar que el servicio de transferencia corre múltiples hilos de ejecución.
- Lo recomendable en esta etapa de integración es realizar el testing de stress (*Prueba diseñada para determinar la respuesta de un sistema bajo condiciones de carga*). Es importante tratar de desarrollar toda la funcionalidad con una semana (como mínimo) de anticipación a la entrega final a fin de realizar el testing de la aplicación por completo y reducir al máximo posible la cantidad de bugs. Recordar que es indispensable, para la aprobación del trabajo práctico, la entrega completa de la funcionalidad y su correcto funcionamiento.

**Tiempo estimado** dos semanas



<b>Tipo de entrega</b>	Obligatoria
<b>Fecha</b>	16/07/2011

## ***6. Requerimientos técnicos y limitaciones***

---

### **COMUNICACIONES – SOCKETS**

Para la comunicación cliente/servidor se utilizarán los protocolos especificados en el trabajo práctico y se implementarán utilizando sockets (modelo Berkeley) orientados a la conexión del tipo *AF\_INET* para IPv4 (en las dos plataformas).

Para la programación de sockets en Windows, se va a utilizar la biblioteca Winsock declarada en *winsock2.h*. Se requiere linkear con la dependencia *Ws2\_32.lib*.

### **INICIALIZACIÓN DE LA BIBLIOTECA WINSOCK**

Antes de llamar a cualquier función de la biblioteca Winsocks, se deberá inicializar dicha librería indicando la versión de Window Sockets y obteniendo los detalles de la implementación.

Finalizada la utilización de la biblioteca ó ante algún error en la inicialización de la misma, se deberá liberar los recursos y finalizar el uso de la biblioteca Winsock.

Esta inicialización se realiza solamente una vez por ejecución de la aplicación y no debe ser inicializada desde la Dll.

Se utilizará la versión 2.2.

### **PROCESS MANAGEMENT Y MULTITHREADING**

#### **WINDOWS**

Para la creación de threads en Windows se utilizará la función `_beginthreadex` perteneciente a la C Runtime library. De esta manera se evitarán posibles memory leaks. (Para investigar: ¿cuales son las causas de este comportamiento al usar la API `CreateThread`?).

Cada thread tendrá por defecto un stack space de 1 megabyte.

#### **LINUX**

En caso de ser requerido, solo se podrá utilizar la API de la biblioteca `pthread` que forma parte del estándar POSIX (NPTL – Native POSIX Thread Library).



Cada Thread que la aplicación genere deberá tener por defecto el stack space en 1024 kilobytes por arriba del PTHREAD\_STACK\_MIN.

Para la creación de procesos, en caso de ser requerido, se utilizará el modelo Fork-one Model.

## **THREAD SYNCHRONIZATION**

En Windows se utilizarán Kernel Objects para la sincronización. Estos objetos serán los utilizados por las llamadas wait functions para coordinar la ejecución de múltiples threads al pasar de un estado señalizado a uno no señalizado.

A continuación se dará una descripción de los mecanismos que ofrece la Windows API y que estarán permitidos utilizar para la sincronización de threads en user-mode.

Los siguientes kernel objects podrán ser utilizados para la sincronización y queda a criterio del alumno cual utilizar según crea conveniente:

- Events Kernel Objects
- Semaphore Kernel Objects
- Mutex
- Threads
- Waitable timers

En Linux también se podrá utilizar cualquier tipo de synchronization:

- Mutex Locks
- Semaphores
- Condition variables
- Read-Write Locks

## **MEMORY MANAGEMENT**

### **WINDOWS**

Cada thread que la aplicación genere, creará su propio bloque de páginas adicional (su propio heap) en el espacio de dirección del proceso y al que solo él tendrá acceso. Esto lo hará para tener un manejo más eficiente de memoria y evitar el overhead generado por la sincronización entre threads. La cantidad de memoria que deberán ocupar las páginas al ser inicializadas es de 1024 Kb. No habrá restricciones en cuanto a tamaño. El tamaño máximo estará limitado por la cantidad de memoria disponible en el sistema.

Dado que el tiempo de vida de cada thread será corto, este no se ocupará de la fragmentación que se genere en el heap. Cuando el thread ya no necesite el heap este lo deberá eliminar para liberar los recursos.

En ningún momento un thread que no sea el principal utilizará la memoria del heap creado por defecto en el proceso ó heap global. Para esto se utilizarán las funciones de manejo de memoria proporcionadas por la API de Windows. No está





permitido el uso de la biblioteca estándar de C para manejo de memoria (Stdlib.h: malloc, free, etc).

### **LINUX**

No existen restricciones en cuanto al uso de memoria.

### **FILE SYSTEM**

Toda operación de I/O que involucre manejo de archivos en Windows deberá ser usando la Windows API correspondiente en su versión ANSI (en caso de existir UNICODE). No está permitido el uso de la biblioteca estándar de C para manejo archivos (Stdlib.h: fread, fwrite, fopen, etc).

Para Linux no existe tal restricción.



## 7. Anexos

---

### *Anexo A :: Archivo Log y Debugging*

#### **MANEJO DE ERRORES Y EXCEPCIONES**

La aplicación mostrará por consola y generará en el correspondiente archivo Log el código de error y la descripción de dicho código obtenido del sistema tanto para las llamadas a la API de Windows como para las System Calls de Linux. Para los errores pertinentes a la aplicación de deberá respetar las normas de logeo del trabajo práctico.

#### **ALGUNAS ACLARACIONES PARA LA PLATAFORMA WINDOWS**

Dichos códigos se encuentran definidos en el header WinError.h. Para la obtención del message string a partir cierto código generado, se utilizará la función más apropiada de la API de Windows en su correspondiente versión ANSI (en caso de existir UNICODE).

#### **WINDOWS Y LINUX**

Queda prohibido el uso de cualquier mecanismo de exception handling ya sea para sentencias del tipo exception handler (`__try __except`) ó termination handlers (`__try __finally`) en Windows.

Si existiese una API de Windows que permita el control de errores mediante este mecanismo, generando una excepción ante un error, se deberá forzar a que retorne un código de error para ser logeado tal como se explica en la sección de Formato del Archivo Log.

### *Formato del archivo Log*

El archivo Log deberá respetar el siguiente formato de presentación en todos los procesos que se ejecuten en cualquiera de los tres sistemas operativos.

En caso de utilizar distintos niveles detalle, el cambio entre uno u otro debe ser configurable por el usuario.

Se le recomienda al alumno registrar en este archivo **los eventos más importantes** de la ejecución de la aplicación, así **como los valores necesarios para conocer el estado del sistema** en un determinado momento. Esto es muy importante ya que en instancias finales de evaluación es probable que se haga uso de este archivo en situaciones donde la aplicación falla.

[FECHA] [NOMBREPROCESO] [PIDPROCESO][THREADID]: TIPOLOG: DATO

#### **DESCRIPCIÓN**

**Fecha**

Fecha del sistema. Deberá respetar el siguiente formato [HH:mm:ss.SSS].

**Nombre Proceso**

Nombre del proceso que está escribiendo en el Log.

**PID Proceso**

Process ID del proceso que está escribiendo en el Log.

**Thread ID**

ID del thread que escribe en el archivo. Opcional para el thread principal del proceso.

**TipoLog**

INFO, WARN, ERROR ó DEBUG nivel de detalle según lo que consideren apropiado.

**Data**

Descripción del evento ó cualquier información que se considere apropiada.

## Eventos de logeo obligatorio

Algunos eventos deberán ser logeados obligatoriamente con cierta información de contexto, a saber:

**FTP server**

- Configuración inicial recibida por archivo
- Conexión y desconexión de cliente ftp (info: ip cliente, puerto del cliente, etc)
- Inicio y finalización de transferencia de archivo (info: nombre archivo, envío/recepción, tamaño archivo, etc)

**KSS**

- Configuración inicial recibida por archivo
- Comienzo y finalización de cada una de las llamadas a su API (info: parámetros de las llamadas, valor de retorno)
- Conexión y desconexión de dispositivos de almacenamiento (info: identificador VDA, formateado (si/no), etc)

**FSS**

- Configuración inicial recibida por archivo
- Al iniciar ejecución: VDAs formateados (info: identificador VDA, cantidad de archivos VDA, espacio libre VDA, etc)
- Comienzo y finalización de cada una de las llamadas a su API (info: parámetros de las llamadas, valor de retorno)
  - crearTablaSectoresLibres(): indicar el tamaño **en memoria** que ocupará el correspondiente bitmap

**VDA**

- Configuración inicial recibida por archivo
- Información especial definida en la sección "Comportamientos definidos" del VDA
- Comienzo y finalización de cada una de las llamadas a su API (info: parámetros de las llamadas, valor de retorno)
  - getSectores(): el tiempo que le insumiría ubicar dichos sectores si se tratara de un disco duro convencional



## **Anexo B :: Systems Calls del KSS:**

### **OPERACIÓN SYS\_OPEN()**

*Esta operación será utilizada para abrir archivos en modo lectura, escritura o para eliminar archivos*

#### **PARÁMETROS**

Tipo de apertura: *LECTURA (0), ESCRITURA (1) o ELIMINAR (2)*  
Ruta y nombre del archivo: */[VDA]/[nombre de archivo]*

El KSS realizará al menos las siguientes operaciones:

- 1) Verificará que el archivo no esté actualmente abierto.
- 2) Consultará con el FSS que el archivo exista. Primitiva del FSS: *existeArchivo()*.

#### **EN CASO DE ABRIR PARA LECTURA:**

- 3) Solicitará al FSS la lista de bloques que componen dicho archivo. *Primitiva del FSS: infoArchivo()*.
- 4) Creará el registro de la TDD y almacenará dicha lista en el campo de sectores.

#### **EN CASO DE ABRIR PARA ESCRITURA:**

- 3) En caso de existir el archivo, solicitará al FSS que lo elimine. *Primitiva del FSS: eliminarArchivo()*.
- 4) Solicitará al FSS que cree el archivo. *Primitiva crearArchivo()*.
- 5) Creará el registro de la TDD.

#### **EN CASO DE ABRIR PARA ELIMINAR:**

3. Creará el registro de la TDD

Por último responderá el éxito de la operación o su correspondiente mensaje de error el cual también será impreso en el archivo log del proceso.

### **OPERACIÓN SYS\_READ ()**



*Esta operación permite leer un bloque de un determinado descriptor de archivo abierto para lectura. Es importante recordar que el kernel trabaja con bloques fijos de **1 KB**.*

### **PARÁMETROS**

Descriptor de archivo: Número identificador del descriptor a leer.

El KSS realizará al menos las siguientes operaciones:

1. Verificará en la TDD que dicho descriptor corresponda a un archivo abierto en modo lectura.
2. Obtendrá de la lista de sectores de la TDD los dos próximos sectores a leer y los solicitará al dispositivo de almacenamiento correspondiente. *Primitiva del VDA getSectores()*. Temporalmente utilizará el buffer de 1KB como memoria intermedia.

*Nota: En caso que la cantidad de sectores a leer sea menor a dos, o alguno de dichos sectores se deba leer parcialmente (dado el tamaño máximo del archivo) el buffer de 1KB deberá ser rellenado con caracteres '\0'.*

3. Enviará el buffer correspondiente al solicitante junto con un valor correspondiente al éxito de la operación o su correspondiente mensaje de error el cual también será mostrado por pantalla.

### **OPERACIÓN SYS\_WRITE()**

*Esta operación permite escribir un bloque de un determinado descriptor de archivo abierto para escritura. Es importante recordar que el KSS trabaja con bloques fijos de **1 KB**.*

### **PARÁMETROS**

Descriptor de archivo: Número identificador del descriptor a escribir.  
Buffer: Bloque de hasta 1KB a escribir en un archivo.

El KSS realizará al menos las siguientes operaciones:

1. Verificará en la TDD que dicho descriptor corresponda a un archivo abierto para **escritura**.
2. Escribirá en el buffer de la TDD el buffer recibido rellenado hasta 1KB en caso necesario.
3. Ejecutará su propia operación `sys_flush()`.
4. Actualizará el tamaño del archivo en el FSS.
5. Responderá el éxito de la operación al solicitante o su correspondiente mensaje de error el cual también será mostrado por pantalla.

### **OPERACIÓN SYS\_FLUSH ()**



*Esta operación persiste en el correspondiente dispositivo de almacenamiento el buffer acumulado para dicho archivo.*

*Si bien forma parte de los system calls del KSS, es interna y por ende **no está liberada** para el uso de las aplicaciones.*

### **PARÁMETROS**

Descriptor de archivo: Número identificador del descriptor a escribir.

El KSS realizará al menos las siguientes operaciones:

1. Verificará en la TDD que dicho descriptor corresponda a un archivo abierto para escritura.
2. Solicitará al FSS dos sectores libres del VDA correspondiente.  
*Primitiva: dosSectoresLibres().*
3. Escribirá ambos sectores del correspondiente VDA utilizando la *primitiva: putSectores().*
4. Asignará los sectores al archivo en cuestión. *Primitiva: asignarSectores()*
5. Responderá el éxito de la operación o su correspondiente mensaje de error el cual también será mostrado por pantalla.

### **OPERACIÓN SYS\_CLOSE ()**

*Esta operación es la forma de concluir una transacción sobre un archivo y registrar el éxito de la misma.*

### **PARÁMETROS**

Descriptor de archivo: Número identificador del descriptor a cerrar.

El KSS realizará al menos las siguientes operaciones:

1. Verificará en la TDD que dicho descriptor corresponda a un archivo abierto.

#### **EN CASO DE LECTURA:**

2. Ir directo al paso 3.

#### **EN CASO DE ESCRITURA:**

2. Enviará al FSS el tamaño final del archivo. *Primitiva: actualizarTamano().*

#### **EN CASO DE ELIMINAR:**

2. Solicitará al FSS que elimine el archivo. *Primitiva: eliminarArchivo().*

#### **POR ÚLTIMO, EN TODOS LOS CASOS**



3. Eliminará el registro correspondiente de la TDD y confirmará el éxito de la operación o su correspondiente mensaje de error el cual también será impreso en el archivo log del proceso.

### OPERACIÓN SYS\_LIST()

*Esta operación lista el contenido de un directorio. Es importante recalcar que **al no existir subdirectorios existen solamente dos opciones de listado***

#### PARÁMETROS:

Directorio de destino: Ruta absoluta al directorio a listar

#### SI EL PARÁMETRO ES “/”

Esta operación devolverá la lista de VDAs montados, cada uno de ellos, en un directorio separado.

**SI EL PARÁMETRO ES “/[VDA]”** (donde [VDA] es el nombre de un dispositivo de almacenamiento válido y montado

Esta operación devolverá la lista de archivos del VDA enviado como parámetro.

Enviará al FSS la solicitud de listar los archivos de un VDA.  
*Primitiva: listarDirectorio()*.

## Anexo C :: Protocolos de comunicación

### Protocolo MPS – (ex IPC/IRC)

Se utilizara un mensaje de protocolo interno. Estos son los campos mínimos que todo mensaje interno debe utilizar.

Descriptor ID	PayloadDescriptor	Payload Length	Payload
0	15	16	17
			20 21
			...

#### REQUEST:

##### **Descriptor ID:**

Identificador de 16 bytes único descriptor en la red.

##### **PayloadDescriptor:**

Identificador de nro de protocolo.



***Payload Length:***

La longitud del descriptor inmediatamente seguido del header.

***Payload:***

La carga de datos que se necesite pasar. Queda libre al usuario del protocolo.

**RESPONSE:**

***Descriptor ID:***

Identificador de 16 bytes correspondiente al Request.

***PayloadDescriptor:***

Identificador de nro de protocolo.

***Payload Length:***

La longitud del descriptor inmediatamente seguido del header.

***Payload:***

La carga de datos que se necesite pasar. Queda libre al usuario del protocolo.

## ***Handshake***

Para realizar los handshake entre los procesos se deberá enviar un mensaje de request del protocolo MPS con los siguientes valores:

### **FORMATO DEL REQUEST**

***Descriptor ID:***

Identificador de 16 bytes único descriptor en la red.

***PayloadDescriptor:***

Nro. identificando el proceso que se está conectando, reservado para iniciar la conexión.

***Payload Length:***

Cero (0), indicando que no hay payload.

### **FORMATO DEL RESPONSE:**

***Descriptor ID:***

El identificador del request para determinar a qué mensaje pertenece.

***PayloadDescriptor:***

Nro. reservado con 2 valores posibles: ok y fail (a continuación de un response fail se deberá cerrar la conexión).

***Payload Length:***

Cero (0), indicando que no hay payload.





## Anexo D :: Persistencia del FSS

Para la persistencia de las estructuras del FSS se usará el propio sistema de archivos del sistema operativo. De esta manera, se podrá reflejar de manera simple y clara el estado de sus estructuras e inclusive hacerle cualquier modificación manual que se desee (por ejemplo: eliminar un archivo).

Continuando con el ejemplo de la figura presentada previamente en la definición del FSS, el sistema de archivos debería contener lo siguiente:

/home/grupo-so/tp-2011/fss/data/

| -DA1/

- Por cada disco o archivo existente en el FSS, un directorio será creado con ese nombre.
- Dentro de los directorios se crearán archivos sin contenido, cuyo nombre contendrá la información en sí misma.
- Existirá un archivo “\_free-sectors”, cuyo contenido será el bitmap de sectores libres (Un string conteniendo un char 0 ó 1 por cada sector del disco). Ejemplo: 0101110110

```
|- archivo1/
    |- size-2048
    |- 00001-sector-1
    |- 00002-sector-2
    |- 00003-sector-5
    | ...
|- archivo2/
    |-size-1024
    |-00001-sector8
    | ...
|-_free-sectors
```

El material de soporte para poder llevar a cabo este trabajo práctico se encuentra en su mayoría en la Web. A continuación se enumeran enlaces a páginas con la información necesaria para cada plataforma.

**MSDN de Microsoft** <http://msdn.microsoft.com>.

### Threads y Processes

<http://msdn.microsoft.com/en-us/library/ms686937%28VS.85%29.aspx>

### Scheduling

[http://msdn.microsoft.com/en-us/library/ms685096\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms685096(VS.85).aspx)

### Winsocks

<http://msdn.microsoft.com/en-us/library/ms740673%28VS.85%29.aspx>

### Memory Management

<http://msdn.microsoft.com/en-us/library/aa366779%28VS.85%29.aspx>



### **Thread Synchronization**

<http://msdn.microsoft.com/en-us/library/ms682584%28VS.85%29.aspx>

### **Time Functions**

[http://msdn.microsoft.com/en-us/library/ms724962\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms724962(VS.85).aspx)

### **Cryptography**

[http://msdn.microsoft.com/en-us/library/aa380255\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa380255(VS.85).aspx)

### **Security**

<http://msdn.microsoft.com/en-us/library/cc527452.aspx>

Se recomienda el siguiente material bibliográfico como soporte teórico:

- Programming Applications for Microsoft Windows, 4<sup>th</sup> edición de Jeffrey Richter.
- Microsoft Windows Internals, 4<sup>th</sup> Edición de Mark E. Russinovich y David A. Solomon.
- Understanding the Linux Kernel, 3<sup>rd</sup> Edición de Daniel P. Bovet, Marco Cesati

### *Links recomendados:*

#### **FORO DE CONSULTA:**

<http://www.tpsosutnfrba.com.ar/foro>

#### **WIKI DE LA CÁTEDRA**

<http://www.tpsosutnfrba.com.ar/wiki>