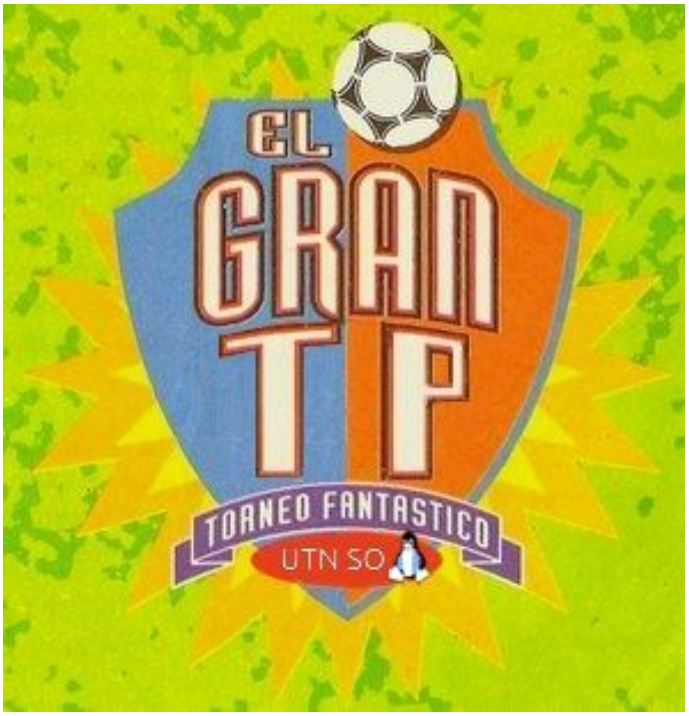
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en Sistemas de Información

# El Gran TP

***No tiene espera activa, señorita, tiene gatorei.***




Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

- 2C2018 -
Versión [1.1]

# Versión de Cambios

## Versión v1.1

1. Desambiguamos un poco el texto sobre **el funcionamiento del DTB-Dummy**, dado que se prestaba a confusión por la forma que se encontraba redactado. *El funcionamiento sigue siendo el mismo*, idéntico al expuesto en la charla y el Anexo II.

2. La operación **abrir** del lenguaje *Escriptorio* **no crea los archivos**. Sin embargo, como el FileSystem FIFA hacía alusión a eso en su especificación. Para solucionar esta inconsistencia, se agregaron las operaciones **crear** y **borrar**, ambas detalladas en el Anexo I.

# Índice

# Introducción

El trabajo práctico consiste en simular ciertos aspectos de un sistema multiprocesador, con la capacidad de interpretar la ejecución de scripts escritos en un lenguaje diseñado para el trabajo práctico. Este sistema planificará y ejecutará estos scripts (en adelante "Programas") controlando sus solicitudes de I/O, administrando los accesos a recursos tanto propios como compartidos.

Para ello, se tomará un contexto "futbolero" donde diferentes procesos, que simularán Directores Técnicos, tomarán control de un Equipo en particular, permitiendo la compra/venta de jugadores y competir contra otros Equipos.

Los scripts utilizados en el trabajo práctico estarán escritos en el *lenguaje Escriptorio*, el cual fue inventado y diseñado por la cátedra para fines didácticos. En el Anexo I encontrarán la especificación de este lenguaje. Se recomienda realizar una lectura detenida del mismo.

## Objetivos del Trabajo Práctico

Mediante la realización de este trabajo se espera que el alumno:

- Adquiera conceptos prácticos del uso de las distintas herramientas de programación (API) que brindan los sistemas operativos.
- Entienda aspectos del diseño de un sistema operativo.
- Afirme diversos conceptos teóricos de la materia mediante la implementación práctica de algunos de ellos.
- Se familiarice con técnicas de programación de sistemas, como el empleo de makefiles, archivos de configuración y archivos de log.
- Conozca con grado de detalle la operatoria de Linux mediante la utilización de un lenguaje de programación de relativamente bajo nivel como C.

## Características

- Modalidad: grupal (5 integrantes +- 0) y obligatorio
- Tiempo estimado para su desarrollo: 90 días
- Fecha de comienzo:  01 de Septiembre de 2018
- Fecha de primera entrega: 01 de Diciembre de 2018
- Fecha de segunda entrega: 15 de Diciembre de 2018
- Fecha de tercera entrega: 22 de Diciembre de 2018
- Lugar de corrección: Laboratorio de Medrano

## Evaluación del Trabajo Práctico

El trabajo práctico consta de una evaluación en 2 etapas.

La primera etapa consistirá en las pruebas de los programas desarrollados en el laboratorio. Las pruebas del trabajo práctico se subirán oportunamente y con suficiente tiempo para que los alumnos puedan realizar sus pruebas con antelación. Queda aclarado que para que un trabajo práctico sea considerado evaluable, el mismo debe proporcionar registros de su  funcionamiento de la forma más clara posible.

La segunda etapa se dará en caso de aprobada la primera y constará de un coloquio, con el objetivo de afianzar los conocimientos adquiridos durante el desarrollo del trabajo práctico y terminar de definir la nota de cada uno de los integrantes del grupo, por lo que se recomienda que la carga de trabajo se distribuya de la manera más equitativa posible.

Cabe aclarar que el trabajo equitativo no asegura la aprobación de la totalidad de los integrantes, sino que cada uno tendrá que defender y explicar tanto teórica como prácticamente lo desarrollado a lo largo de la cursada.

## Deployment y Testing del Trabajo Práctico

Al tratarse de una plataforma distribuida, los procesos involucrados podrán ser ejecutados en diversas computadoras. La cantidad de computadoras involucradas y la distribución de los diversos procesos en estas será definida en cada uno de los tests de la evaluación y **es posible cambiar la misma en el momento de la evaluación**. Es responsabilidad del grupo automatizar el despliegue de los diversos procesos con sus correspondientes archivos de configuración para cada uno de los diversos tests a evaluar.

Todo esto estará descrito en el documento de pruebas que se publicará cercano a la fecha de Entrega Final. Archivos y programas de ejemplo se pueden encontrar en el repositorio de la cátedra.

Finalmente, recordar la existencia de las Normas del Trabajo Práctico donde se especifican todos los lineamientos de cómo se desarrollará la materia durante el cuatrimestre.

## Aclaraciones

Debido al fin académico del trabajo práctico, los conceptos reflejados son, en general, versiones simplificadas o alteradas de los componentes reales de hardware y de sistemas operativos modernos, a fin de resaltar algún aspecto de diseño. En algunos casos los aspectos no fueron tomados de manera literal, por lo que invitamos a los alumnos a leer las notas y comentarios al respecto que haya en el enunciado, como así también a reflexionar y discutir con sus compañeros, ayudantes y docentes al respecto.

# Arquitectura del Sistema

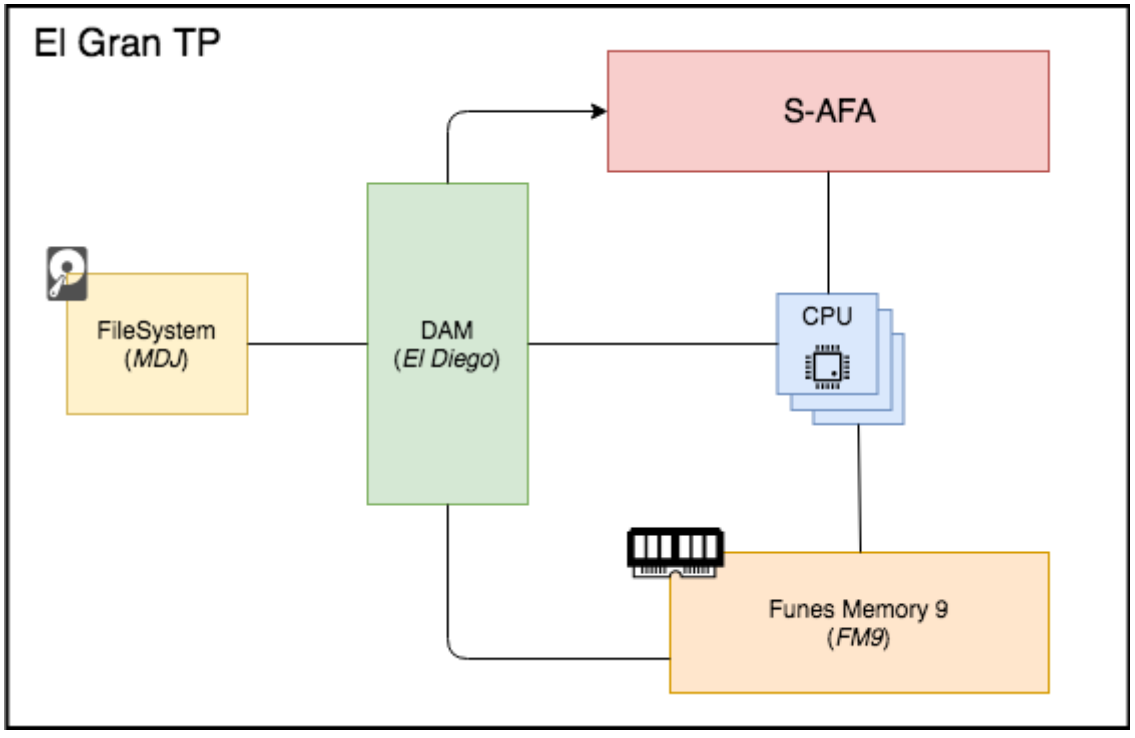
Como se mencionó anteriormente, el sistema simulará un sistema distribuido para el procesamiento de los programas escritos en lenguaje de scripting **Escriptorio** ingresados al sistema mediante la consola del proceso S-AFA. Dicho *proceso central "**S-AFA**"* se encargará de la planificación y creación de esos programas, denominados "programas G.DT", en las diferentes CPUs entre otras actividades.

El proceso DAM (Ehhh... DMA), o como llamaremos desde ahora, "***El Diego***"; cumplirá la función de DMA entre el proceso FileSystem y Memoria. Se encargará de obtener, cargar y actualizar la información desde FileSystem a Memoria a medida que las CPU o el sistema la requiera.

El proceso FileSystem, o ***Mercado de Jugadores (MDJ)***, será el encargado de persistir ***todos*** los datos del sistema. Los mismos se clasifican en: Equipos, Jugadores, Partidos *y los mismos scripts Escriptorios*.

Por último, el proceso ***Funes Memory 9 (o FM9)***, será el encargado de contener la información de los datos que requieran los distintos procesos Escriptorios en ejecución, así como los propios scripts de los procesos, para operar en el sistema.

A continuación se adjunta un esquema de la Arquitectura del sistema:



# Contexto a usar en el Trabajo Práctico

## Contexto Informal

El contexto dado a continuación tiene por objetivo generar una forma más amena y menos genérica de entender los diferentes contenidos del trabajo práctico, haciendo uso de escenarios conocidos.

Con el sistema en funcionamiento, cada programa G.DT dentro del sistema deberá, en su script, haber elegido un "**Equipo**" sobre el cual trabajar. Este Equipo será un archivo en el MDJ que contendrá la información para que, si el programa G.DT se abortara o terminara, la información del Equipo siempre quede persistida. Esto permite que el Equipo pueda ser vuelto a tomar mediante otro programas G.DT que quiera operar con él.

Por otro lado, existe el concepto de "**Jugador**" que será otro tipo de archivo en el MDJ (que determinará su estado) en el cual se permitirá modificar las estadisticas del mismo. Lo mismo vale para los scripts Escriptorio que crean los programas G.DT, como para los archivos de equipo.

Además, los archivos anteriormente mencionados, se crearán de antemano para la prueba, con el formato especificado en las próximas secciones. Estos archivos serán provistos por la cátedra para cada prueba a realizar en las entregas finales.

Se recomienda una distribución equitativa del trabajo. Para esto, bajo línea/recomendación de cátedra adjuntamos la siguiente distribución de trabajo que creemos que es óptima para que todos los integrantes trabajen equitativamente.

- Proceso S-AFA: 1,5 Integrantes
- Proceso CPU: 0,5 Integrantes
- Proceso FM9: 1,5 integrantes
- Proceso DAM: 0,5 integrantes
- Proceso MDJ: 1 integrante

Cabe destacar que esta distribución ***no asegura la aprobación***, sino que la misma está dada por la ***defensa del trabajo de cada integrante sobre lo trabajado***.

## DTB vs G.DT

En las próximas hojas se encontrarán con las siglas DTB y G.DT y la denominación Escriptorio. Es importante tener en cuenta que la identidad y diferencias entre el Script Escriptorio, el programa G.DT y el DTB son idénticas a las vistas en la teoría con a los "Código de Programa", Procesos y Bloques de Control de Procesos.

# Proceso S-AFA

El proceso S-AFA o “Sisop AFA” es el encargado de funcionar como un elemento coordinador de los procesos CPU. El mismo tendrá dos funciones principales: la de **Planificador** de los distintos programas G.DT y la de **Gestor de programas G.DT**.

La función **Planificador** de S-AFA estará compuesta por dos subsistemas principales, un planificador de largo plazo (*PLP*) y un planificador de corto plazo (*PCP*), siendo su funcionamiento similar al visto en la teoría.

Cada uno de estos dos subsistemas deberá administrarse ***independiente*** y en ***simultáneo***. Además, es menester mencionar que las conexiones con otros procesos no necesariamente están ligadas a uno de estos subsistemas, sino que ambos trabajan de forma colaborativa para que ante externos sean vistos como una única entidad.

Cuando el proceso S-AFA se inicie, comenzará en un estado ***corrupto***. En dicho estado ***no podrá aceptar el ingreso de ningún run para un programa G.DT***. Para que salga de dicho estado deberán pasar dos cosas:

1. La conexión del proceso “El Diego”.
2. La conexión de por lo menos un Proceso CPU.

Una vez que se cumplan ambas condiciones se dice que el proceso S-AFA se encuentra en un estado ***operativo,*** en el cual puede comenzar a iniciar programa G.DT. Una vez que este proceso alcanza este estado ***no volverá al estado anterior***.

## Funcionamiento como Planificador

### El DT Block (DTB)

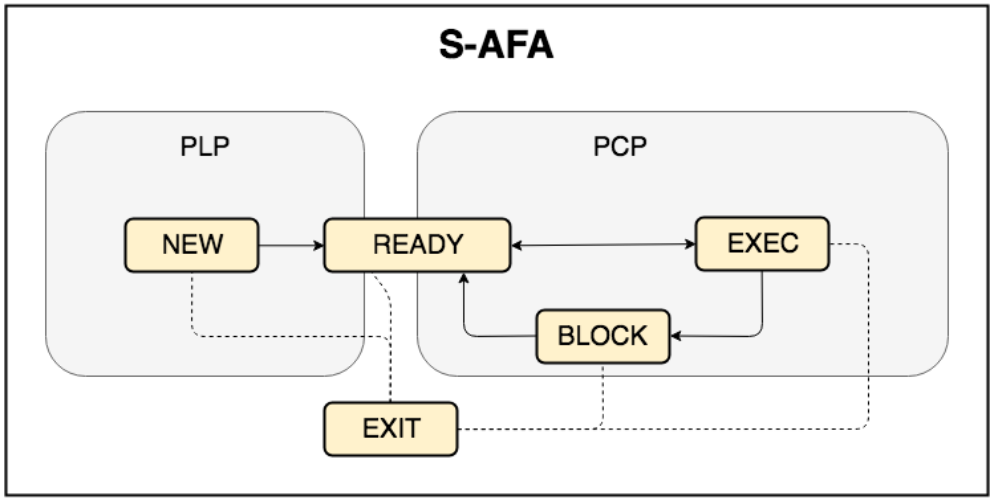
Cada ejecución de un nuevo Script *Escriptorio* generará un programa G.DT que tendrá asociado un Bloque, denominado DT Block o DTB, que será el elemento planificable dentro del sistema, identificando al “proceso” de la misma forma que lo haría un PCB. En él se deberá guardar toda la información referente al programa asociado al G.DT, conteniendo, como mínimo:

- El ID de G.DT
- El Escriptorio a abrir
- El Program Counter (PC)
- Flag G.DT inicializado
- Tabla de direcciones de archivos abiertos
- etc.

Todos los cambios que se consideren para el DTB deberán ser validados con el ayudante.

### Estados de Planificación

El sistema de planificación de S-AFA utiliza un sistema de colas de planificación de 5 estados:




### Planificador de Largo Plazo

El subsistema de PLP será el punto de entrada, encargado de crear los DTB asociados a los programas G.DT y agregarlos a la cola NEW. Además, será el encargado de mantener el grado de multiprogramación del sistema, planificando los pases a READY según el algoritmo **FIFO (First In, First Out)**. Estos pases a READY implicarán la carga en memoria del script Escriptorio asociado, responsabilidad del Gestor de programas G.DT descrito más adelante.

### Planificador de Corto Plazo

El subsistema de PCP será el encargado de distribuir los distintos DTB en las CPUs, interactuando con las colas READY y EXEC. Este recibirá las conexiones de los procesos CPUs y quedará a la espera de DT Blocks en la cola READY. Finalmente, al concluir la ejecución de un DT Block, deberá mover el mismo a la cola EXIT.

#### Algoritmos de Planificación

El planificador de corto plazo poseerá 3 tipo algoritmos diferentes. Los primeros dos serán **Round Robin** y **Virtual Round Robin**, cuyos parámetros serán leídos desde el archivo de configuración. El funcionamiento de los mismos será idéntico al visto en la teoría.

Por último, el tercer algoritmo de planificación deberá ser creado y diseñado por cada grupo, diferente a los dados en teoría, en base a métricas que se darán más adelante en la cursada (a partir del Hito Presencial). En caso de que algún grupo no se presente al Hito Presencial se le asignará restricciones de forma aleatoria sin tener en consideración el estado del TP, informando el mismo por email.

## Gestor de programas G.DT

El proceso S-AFA tendrá un subsistema de control y gestor de programas G.DT que cumplira dos tareas:

1. Iniciar nuevos G.DT
2. Verificar todo tipo de estadísticas y administración de los mismos.

Este subsistema se administra bajo una consola que tendrá que proveer el mismo planificador y mínimamente dispondrá las siguientes operaciones:

1. ejecutar
2. status
3. finalizar
4. metricas

### Ejecutar

Parámetros
*La ruta del script escriptorio que se desea ejecutar*

Funcionalidad
*Permitirá el ingreso de un nuevo programa G.DT al sistema. Pasada la ruta del script Escriptorio, el planificador creará su DTB asociado y colocará el proceso en la cola de NEW para que el PLP lo administre cuando lo permita el grado de multiprogramación. Cuando suceda esto, el PLP se comunicará con este submódulo para comenzar la ejecución del DTB dummy de Iniciar Escriptorio definida en el Proceso CPU (Ir a la sección).*

*El DTB dummy a enviar contendrá el archivo Escriptorio que se desea ejecutar y el flag de inicialización en 0, siendo el único DTB que contenga este flag con dicho valor (cualquier otro DTB lo tendrá en 1).*

### Status

Parámetros
*Ninguno o el ID correspondiente a un DT Block*

Funcionalidad
*Se deberá mostrar el estado de cada cola, así como la información complementaria a las mismas.*

*En caso de tener un parámetro, deberá informar todos los datos almacenados en el DT Block (tanto mínimos como agregados por el grupo).*

### Finalizar

Parámetros
*El ID correspondiente a un DT Block*

Funcionalidad
*Obligará a un DTB a pasar a la cola de EXIT para poder destrabar la ejecución y dar lugar a otro G.DT a operar sobre dicho equipo. Si el G.DT se encuentra en la cola EXEC se deberá esperar a terminar la operación actual, para luego moverlo a la cola EXIT.*

### Metricas

Parámetros
*Opcional: El ID correspondiente a un DT Block*

Funcionalidad

*Detalla las siguientes métricas:*

1. *Cant. de sentencias ejecutadas que esperó un DTB en la cola NEW*
2. *Cant.de sentencias ejecutadas prom. del sistema que usaron a "El Diego"*
3. *Cant. de sentencias ejecutadas prom. del sistema para que un DTB termine en la cola EXIT*
4. *Porcentaje de las sentencias ejecutadas promedio que fueron a "El Diego"*
5. *Tiempo de Respuesta promedio del Sistema*

## Archivo de Configuración y Logs

El proceso deberá poseer un archivo de configuración en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

| Campo | Tipo | Ejemplo |
|---|---|---|
| Puerto de Escucha | [numérico] | 8000 |
| Algoritmo de Planificación | [cadena] | "RR" \| "VRR" \|"PROPIO" |
| Quantum | [numérico] | 2 |
| Grado de multiprogramación | [numérico] | 3 |
| Retardo de planificación | [numérico - milisegundos] | 600 |

Queda a decisión del grupo el agregado de más parámetros al mismo. Además, el proceso deberá registrar toda su actividad mediante un archivo de log. Todos los campos deberán poder ser modificados en tiempo de ejecución, actualizando la operatoria del sistema (excepto los relacionados a conexiones).[1]

### Ejemplo de Archivo de Configuración

```
PUERTO=5003
ALGORITMO="RR"
QUANTUM=2
MULTIPROGRAMACION=3
RETARDO_PLANIF=600
```

[1] Investigar inotify()

# Proceso CPU

El proceso CPU será ejecutado múltiples veces, y cada una de estas instancias se encargará de realizar las siguientes operaciones:

1. Ejecutar las sentencias de los scripts Escriptorios.
2. Ejecutar la operación dummy de iniciar G.DT

Al ingresar al sistema el Proceso CPU se conectará al “El Diego” y a “S-AFA” realizando un handshake para obtener los datos necesarios para su funcionamiento a futuro (como es, la rafaga de planificación).
Cada operación que realice la CPU tendrá asociado un retardo de ejecución definido en el archivo de configuración.

Por otro lado, toda operación que requiera la intervención de El Diego provocará que el proceso sea desalojado de la CPU y posteriormente bloqueado por S-AFA a la espera que el primero indique que ya incluyó en FM9 el archivo solicitado.

## Ejecución de sentencias

Una vez que el proceso CPU recibe un DTB en el cual su Flag de inicialización se encuentra en 1, realizará la o las ejecuciones que correspondan (definidas por el quantum del S-AFA).

Cuando pase esto, por cada unidad de tiempo de quantum que posea disponible ejecutará una línea del Escriptorio asociado al DTB. Cada operación que requiera la intervención de FM9 (en base a las primitivas que se encuentran en el Anexo I) el CPU se comunicará directamente con él. En caso que se produzca un acceso inválido o un error en la FM9 se abortará el DTB informando a S-AFA para que sea pasado a la cola de Exit.

## Operación Dummy - Iniciar G.DT

Una vez que el proceso CPU recibe un DTB, en el cual su Flag de inicialización se encuentra en 0, realizara lo que llamamos la Operación Dummy de inicialización de G.DT.

Esta operación dummy consta de solicitarle a El Diego que busque en el MDJ el Escriptorio indicado en el DTB. Una vez realizado esto, el CPU desaloja a dicho DTB Dummy, avisando a S-AFA que debe bloquearlo. Si S-AFA recibe el DTB con el flag de inicialización en 0, moverá el DTB asociado al programa G.DT que se había ejecutado en la cola de Bloqueados.

Cuando El Diego finaliza su operatoria, le comunicará a S-AFA si pudo o no alojar el Escriptorio en FM9, para que el primero pueda pasarlo a la cola de Ready o Exit (según como corresponda).

## Archivo de Configuración y Logs

El proceso deberá poseer un archivo de configuración en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

| Campo | Tipo | Ejemplo |
|---|---|---|
| IP de S-AFA | [cadena] | “192.168.1.1” |
| Puerto de S-AFA | [numérico] | 8000 |
| IP de El Diego | [cadena] | “192.168.1.2” |
| Puerto de El Diego | [numérico] | 8001 |
| Retardo de Ejecución | [numérico - milisegundos] | 1000 |

Queda a decisión del grupo el agregado de más parámetros al mismo. Además, el proceso deberá registrar toda su actividad mediante un archivo de log.

### Ejemplo de Archivo de Configuración

```
IP_SAFA="192.168.1.1"
PUERTO_SAFA=8000
IP_DIEGO="192.168.1.2"
PUERTO_DIEGO=8001
RETARDO=1000
```

# Proceso El Diego (“DAM”/DMA)

El Proceso El Diego, ehhhh DMA, será el encargado de cumplir con dos funciones:

- Intermediario entre CPU y S-AFA
- Intermediario entre FM9 y MDJ

Para esto, al ejecutarse, se conectará con los procesos S-AFA, Memoria y MDJ y quedará a la espera de futuras conexiones de CPU y/o pedidos de S-AFA.

## Intermediario entre CPU y S-AFA

Cada vez que CPU requiere cargar un archivo a FM9 para un proceso G.DT determinado, El Diego recibira una solicitud con el path del archivo y el ID del DTB asociado del proceso. En este momento se deberá loguear “*Ehhh, voy a buscar [path] para [pid]*”

Una vez realizada la consulta a FM9 y cargados los archivos en MDJ (como indica la siguiente sección), El Diego se encargará de comunicarle a S-AFA la finalización de dicha operación. Esto es para que este último desbloquee al DTB asociado.

En este proceso se le comunicará al S-AFA que archivo fue abierto y los datos de memoria virtual necesarios, para poder consultarlos luego (estos datos deben ser cargados dentro del DTB, para luego utilizarlos cuando sean necesarios).

Por otro lado, si la ejecución de la consulta a MDJ falla por alguna razón o es imposible cargar en FM9 dicho archivo, se deberá informar al S-AFA dicho error para que este último envíe a la cola de EXIT al DTB asociado.

## Intermediario entre FM9 y MDJ

Cada vez que el El Diego reciba una solicitud de “Abrir” (ver Anexo 1) se deberá ejecutar una operación de escritura a FM9 que constará de ir a buscar el contenido del path recibido a MDJ y luego enviarlo a FM9.

En cambio, cuando se reciba una solicitud de “Flush” (ver Anexo 1) se deberá ejecutar una operación de escritura a MDJ obteniendo los datos desde FM9. Para esto, El Diego recibirá tanto el path donde debe guardar los datos en MDJ como los datos necesarios para saber dónde obtenerlos en FM9.

Toda interacción tanto entre FM9 como MDJ funciona bajo un tamaño de transferencia máxima (transfer size). Si el archivo a obtener/escribir es mayor que este transfer size se deberán obtener/escribir mediante múltiples solicitudes.

## Archivo de Configuración y Logs

El proceso deberá poseer un archivo de configuración en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

| Campo | Tipo | Ejemplo |
|---|---|---|
| Puerto de Escucha | [numérico] | 8001 |
| IP de S-AFA | [cadena] | “192.168.1.2” |
| Puerto de S-AFA | [numérico] | 8000 |
| IP de MDJ | [cadena] | “192.168.1.3” |
| Puerto de MDJ | [numérico] | 8002 |
| IP de FM9 | [cadena] | “192.168.1.4” |
| Puerto de FM9 | [numérico] | 8003 |
| Transfer Size (en bytes) | [numérico] | 16 |

Queda a decisión del grupo el agregado de más parámetros al mismo. Además, el proceso deberá registrar toda su actividad mediante un archivo de log.

### Ejemplo de Archivo de Configuración

```
PUERTO=8001
IP_SAFA="192.168.1.2"
PUERTO_SAFA=8000
IP_MDJ="192.168.1.3"
PUERTO_MDJ=8002
IP_FM9="192.168.1.4"
PUERTO_FM9=8003
TRANSFER_SIZE=16
```

# Proceso Funes Memory 9 (FM9)

El proceso memoria cumplira la funcion de disponibilizar los datos requeridos para la ejecución de cualquier G.DT. Cada vez que un G.DT requiera algún consultar o actualizar información siempre deberá existir previamente en la Memoria.

La memoria entenderá en su interfaz como “líneas” a lo que una memoria equivalente usaría como bytes, su unidad atómica de asignación, teniendo su tamaño máximo definido por el Tamaño de Línea. De esto se puede extraer que una línea no va a encontrarse en más de una página (en caso de usar un esquema con paginación), así como que no existirán tamaños anómalos y no compatibles entre el tamaño máximo de una línea y el tamaño de la página.

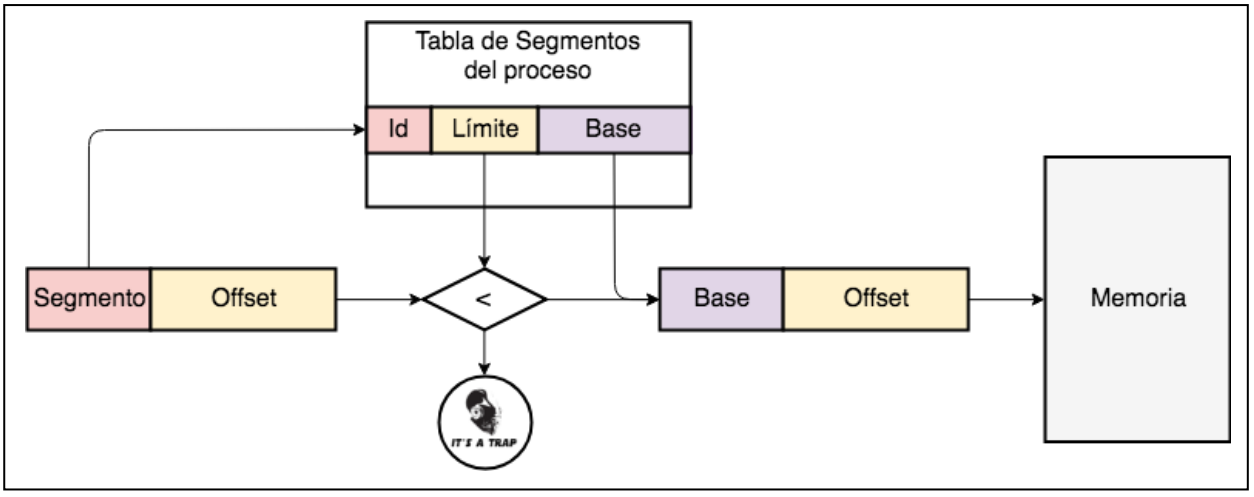
La memoria funcionará bajo tres conceptos. El concepto a usar de los definidos a continuación será designado desde el archivo de configuración:

1. Segmentación Pura
2. Tabla de Páginas Invertida
3. Segmentación Paginada

## Segmentación Pura

El concepto de segmentación pura funcionará como visto en la teoría[2]. De esta manera, dicho proceso mínimamente dispondrá de las siguientes estructuras administrativas:

- Tabla de segmentos
- Storage - Memoria Real



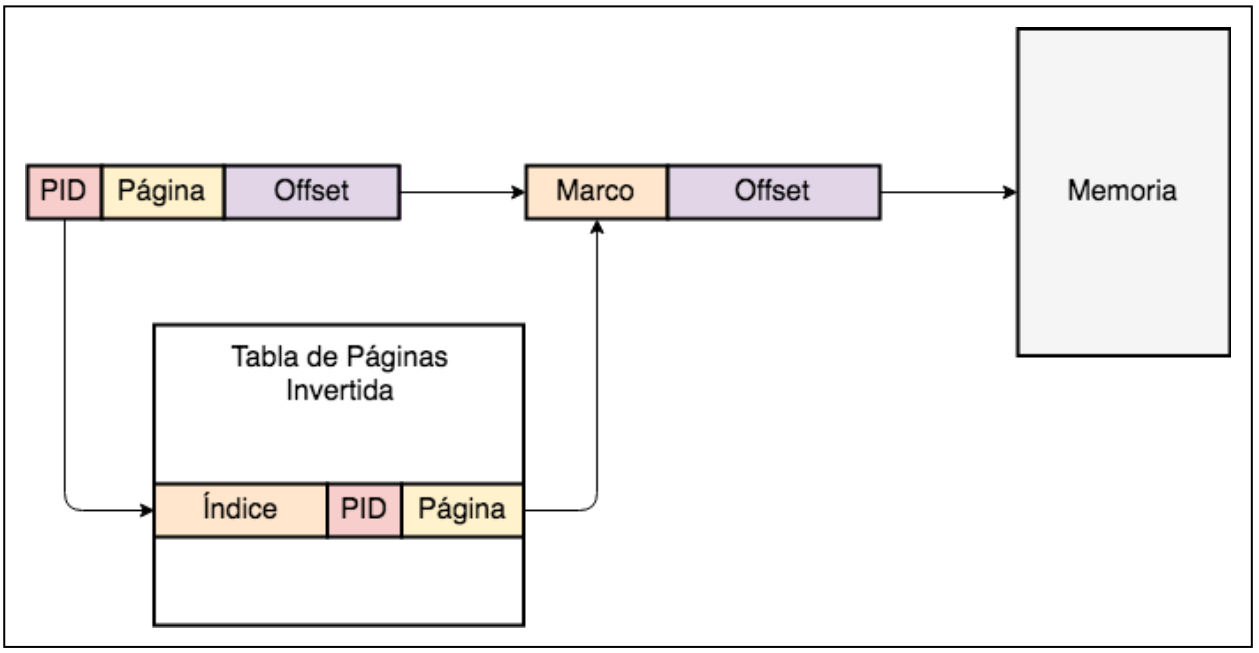

*Esquema modelo de una memoria con segmentación pura*

[2] Para más información se puede consultar el capítulo 8 del libro Fundamentos de Sistemas Operativos de Silberschatz (7ma Edición) o el capítulo 7 del libro Sistemas Operativos - Aspectos internos y principios de diseño de Stallings (5ta Edición).

## Tabla de Páginas Invertida

El concepto de Tabla de Páginas invertida funcionará como visto en la teoría[3]. De esta manera, dicho proceso mínimamente dispondrá de las siguientes estructuras administrativas:

- Tabla de Páginas Invertida
- Storage - Memoria Real




*Esquema modelo de una memoria con paginación invertida*

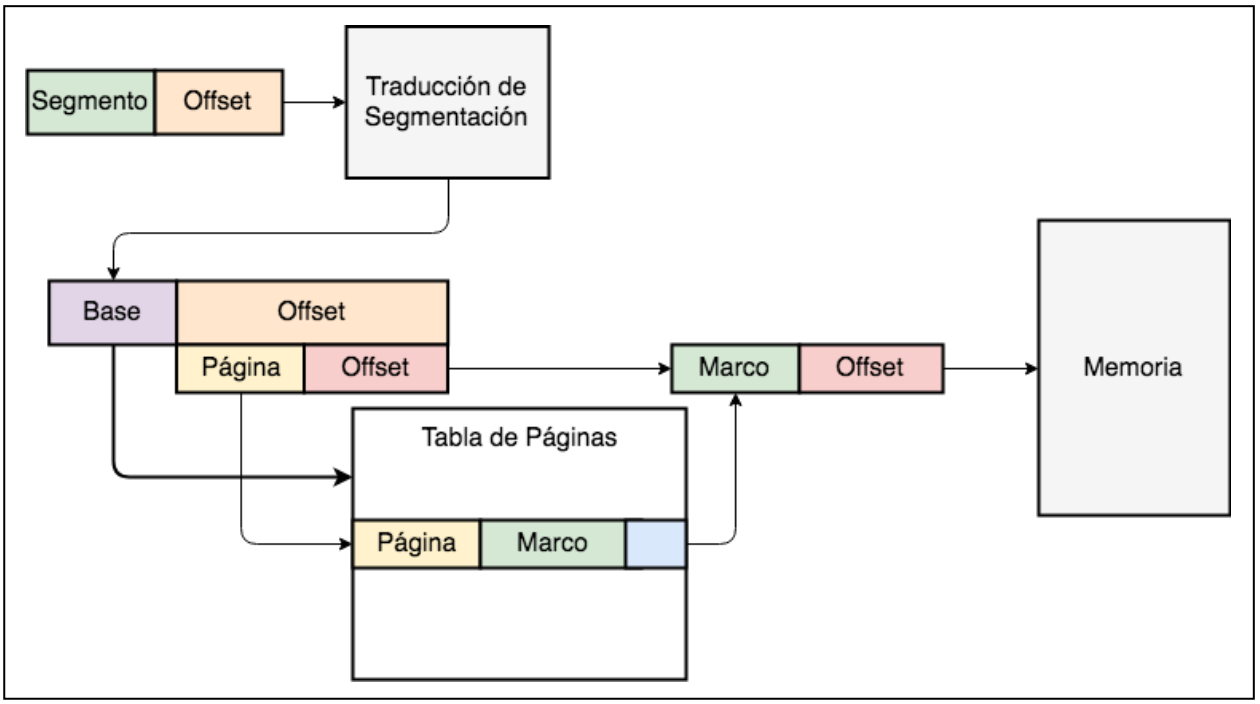
## Segmentación Paginada

El concepto de segmentación paginada funcionará como visto en la teoría[4]. De esta manera dicho proceso mínimamente dispondrá de las siguientes estructuras administrativas:

- Tabla de Procesos
- Tabla de Segmentos
- Tabla de Páginas
- Storage - Memoria Real

[3] Para más información se puede consultar el capítulo 8 del libro Fundamentos de Sistemas Operativos de Silberschatz (7ma Edición) o el capítulo 7 del libro Sistemas Operativos - Aspectos internos y principios de diseño de Stallings (5ta Edición).

[4] Para más información se puede consultar el capítulo 8 del libro Fundamentos de Sistemas Operativos de Silberschatz (7ma Edición) o el capítulo 7 del libro Sistemas Operativos - Aspectos internos y principios de diseño de Stallings (5ta Edición).




*Esquema modelo de una memoria con segmentación paginada*

## Storage - Memoria Real

El Storage o Memoria Real será inicializado una ***única vez*** al iniciar el proceso FM9. Esta inicialización consta de realizar una reserva de memoria por el total requerido para su administración. El tamaño del storage será definido por archivo de configuración.
Cualquier otra implementación de este Storage será ***inválido*** y condición de ***no aprobación***.

## Consola

La consola de FM9 tendrá mínimamente la siguiente operación de **Dump**:

El comando Dump constará del logueo de todos los datos asociados almacenados para un DTB indicado. De esta manera, en base a un ID de DTB pasado, se deberá imprimir por pantalla y en log tanto la totalidad de los datos administrativos guardados como los datos en memoria real que contienen estos últimos.

## Archivo de Configuración y Logs

El proceso deberá poseer un archivo de configuración en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

| Campo | Tipo | Ejemplo |
|---|---|---|
| Puerto de Escucha | [numérico] | 8000 |
| Modo de ejecución | [cadena] | "SEG" - "TPI" - "SPA" |
| Tamaño de la Memoria | [numérico] | 2048 |
| Tamaño Máximo de Línea | [numérico] | 128 |
| Tamaño de Página[5] | [numérico] | 1024 |

Todos los tamaños son en bytes. Queda a decisión del grupo el agregado de más parámetros al mismo. Además, el proceso deberá registrar toda su actividad mediante un archivo de log.

### Ejemplo de Archivo de Configuración

```
PUERTO=8000
MODO=SEG
TAMANIO=2048
MAX_LINEA=128
TAM_PAGINA=1024
```

[5] Solo para sistemas de administración que lo requieran

# Proceso MDJ (File System)

El proceso MDJ - File System será el proceso que interpretaremos como el acceso al File System de nuestro sistema. Este proceso se encargará de gestionar las peticiones que realicen los Procesos CPU sobre los archivos a través del Kernel. Para poder llevar adelante la gestión de las peticiones utilizará un sistema de archivos basado en el FileSystem FIFA.

## Funcionamiento

El Proceso MDJ será un proceso de tipo servidor, es decir, que estará a la espera de las conexiones de peticiones del DAM, validando por medio de un Handshake del protocolo la operación a realizar. Además, el File System deberá atender las peticiones de manera concurrente.

Finalmente, sobre cada operación realizada por el MDJ, se deberá aplicar un retardo antes de retornar dicha operación.

## Operaciones

El proceso MDJ manejara una interfaz reducida, que ***no puede ser ampliada***.

### Validar Archivo

- **Parámetros**: [Path]
- **Descripción**: Cuando el El Diego reciba la operación de abrir un archivo deberá validar que el archivo exista.

### Crear Archivo

- **Parámetros**: [Path]
- **Descripción**: Cuando el El Diego reciba la operación de crear un archivo deberá llamar a esta operación que creará el archivo dentro del path solicitado. El archivo creado deberá tener la cantidad de bloques necesarios para guardar las líneas indicadas por la operación crear con su contenido vacío.

### Obtener Datos

- **Parámetros**: [Path, Offset, Size]
- **Descripción**: Ante un pedido de datos File System devolverá del path enviado por parámetro, la cantidad de bytes definidos por el Size a partir del offset solicitado.

### Guardar Datos

- **Parámetros**: [Path, Offset, Size, Buffer]
- **Descripción**: Ante un pedido de escritura MDJ almacenará en el path enviado por parámetro, los bytes del buffer, definidos por el valor del Size y a partir del offset solicitado. En caso de que se soliciten datos o se intenten guardar datos en un archivo inexistente el File System deberá retornar un error de Archivo no encontrado.

## Consola

La consola del proceso File System deberá mostrar el directorio actual (por defecto, el directorio raíz) y permitir el uso, como mínimo, de los siguientes comandos:

| Comando | Parámetros | Comportamiento | Observaciones |
|---|---|---|---|
| **ls** | [path del directorio] *(Opcional)* | Lista los directorios y archivos dentro del directorio pasado por parámetro | Si no se pasa ningún parámetro, se usa el directorio actual |
| **cd** | [path del directorio] | Se cambia el directorio actual al pasado por parámetro | El directorio "." indica el directorio actual y ".." indica el directorio anterior |
| **md5** | [path del archivo] | Genera el md5 del archivo y lo muestra por pantalla | Realizarlo de forma similar al TP0 |
| **cat** | [path del archivo] | Muestra el contenido de un archivo por pantalla | El contenido debe ser mostrado como una cadena |

## FileSystem Interface For Academicals (FIFA)

El FIFA es un filesystem creado con propósitos académicos para que el alumno se interiorice y comprenda el funcionamiento básico de la gestión de archivos en un sistema operativo.

La estructura básica de FIFA se basa en el propio File System de linux, es decir, en una estructura de árbol de directorios para representar la información administrativa y los datos de los archivos. El árbol de directorios tomará su punto de partida del punto de montaje del archivo de configuración.

Durante las pruebas no se proveerán archivos que tengan estados inconsistentes respecto del trabajo práctico, por lo que no es necesario tomar en cuenta dichos casos y que su Sistema actúe de forma acorde.

### Metadata

#### FileSystem

Este archivo tendrá la información correspondiente a la cantidad de bloques y al tamaño de los mismos dentro del File System.

Dentro de cada archivo se encontrarán los siguiente campos:

- **Tamanio_Bloques**: Indica el tamaño en bytes de cada bloque
- **Cantidad_Bloques**: Indica la cantidad de bloques del File System
- **Magic_Number**: Un string fijo con el valor "FIFA\0"

Ej:

```
TAMANIO_BLOQUES=64
CANTIDAD_BLOQUES=5192
MAGIC_NUMBER=FIFA
```

Dicho archivo deberá encontrarse en la ruta `[Punto_Montaje]/Metadata/Metadata.bin`

### Bitmap

Este será un archivo de tipo binario donde solamente existirá un bitmap[6], el cual representará el estado de los bloques dentro del FS, siendo un 1 que el bloque está ocupado y un 0 que el bloque está libre.

La ruta del archivo de bitmap es: `[Punto_Montaje]/Metadata/Bitmap.bin`

### File Metadata

Los archivos dentro del FS se encontrarán en un path compuesto de la siguiente manera:

`[Punto_Montaje]/Archivos/[PathDelArchivo]`

Donde el path del archivo incluye también el nombre y la extensión del archivo.

```
Ej: /mnt/FS_FIFA/Archivos/passwords/alumnosSIGA.bin
```

Dentro de cada archivo se encontrarán los siguiente campos:

- **Tamaño**: indica el tamaño real del archivo en bytes.
- **Bloques**: es un array de números que contiene el orden de los archivos

Ej:

```
TAMANIO=250
BLOQUES=[40,21,82,3]
```

### Datos

Los datos estarán repartidos en archivos de texto nombrados con un número, el cual representará el número de bloque. (Por ej 1.bin, 2.bin, 3.bin),

Dichos archivos se encontraran dentro de la ruta:

`[Punto_Montaje]/Bloques/[nroBloque].bin`

```
Ej: /mnt/FS_FIFA/Bloques/1.bin
```

## Archivo de Configuración y Logs

El proceso deberá poseer un archivo de configuración en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

| Campo | Tipo | Descripción |
|---|---|---|
| PUERTO | [numerico] | Puerto TCP utilizado para recibir las conexiones de CPU y I/O |
| PUNTO_MONTAJE | [alfanumerico] | Valor del punto de montaje inicial de File System, a partir de este punto se creará el árbol para administrar el FS. |

[6] Se recomienda investigar sobre el manejo de los bitarray de las commons library.

| | | |
|---|---|---|
| RETARDO | [numérico - milisegundos] | Valor del Retardo de cada operación realizada. |

Queda a decisión del grupo el agregado de más parámetros al mismo. Además, el proceso deberá registrar toda su actividad mediante un archivo de log.

### Ejemplo de Archivo de Configuración

```
PUERTO=5003
PUNTO_MONTAJE="/mnt/FIFA_FS/"
RETARDO=500
```

# Anexo I - Especificación del Lenguaje Escriptorio

**Escriptorio** es un lenguaje de programación *interpretado* de propósito general y bastante bajo nivel. Su sintaxis es simple y de objetivo principalmente académico: ayudar a entender los conceptos y mecanismos que un sistema operativo debe tener en cuenta para manejar la ejecución de los programas.

Para la evaluación del trabajo práctico no se proveerán programas con errores de sintaxis y/o semántica. A su vez, la cátedra no proveerá un parser para interpretarlos, es decir, ***queda bajo responsabilidad de los grupos desarrollar un parser que comprenda y entienda dicho lenguaje***. Por esta razón, procederemos a explicar en profundidad el mismo.

## Sintaxis

- El lenguaje es **case-sensitive**; es decir, `hola` y `Hola` son diferentes
- Las sentencias finalizan con un salto de línea. Los saltos adicionales son ignorados.
- Toda línea comenzada por un caracter numeral (#) es un comentario y debe ser ignorado.
- Todo programa deberá terminar con una línea en blanco.
- Toda línea no comenzada por un caracter numeral (#), comenzará con una palabra reservada que indicará la operación a realizarse seguida por las diferentes palabras requeridas para el uso del mismo.

## Palabras Reservadas

Escriptorio contará con nueve palabras reservadas que consta de las palabras de inicio, cierre y las distintas operaciones:

1. abrir
2. concentrar
3. asignar
4. wait
5. signal
6. flush
7. close
8. crear
9. borrar

## Operaciones

Escriptorio contará con nueve operaciones que deberán ser interpretadas por los alumnos para el funcionamiento del mismo:

### Abrir

El comando abrir permitirá abrir un nuevo archivo para el escriptorio.

#### Sintaxis

Recibirá un parámetro con el Path del archivo a abrir:

```
Ej: abrir /equipos/Racing.txt
```

Funcionalidad

La funcionalidad del comando se realizará en los siguientes pasos:

1. Verificar si dicho archivo ya se encuentra abierto por el G.DT. Para esto se debe validar en las estructuras administrativas del DTB. En caso que ya se encuentre abierto, se procederá con la siguiente primitiva.
2. En caso que no se encuentre abierto, se enviará una solicitud a El Diego para que traiga desde el MDJ el archivo requerido. Cuando se realiza esta operatoria, el CPU desaloja al DTB indicando a S-AFA que el mismo está esperando que El Diego cargue en FM9 el archivo deseado. Esta operación hace que S-AFA bloquee el G.DT.
3. Cuando El Diego termine la operatoria de cargarlo en FM9, avisará a S-AFA los datos requeridos para obtenerlo de FM9 para que pueda desbloquear el G.DT.

Errores Esperables

Los errores esperados son:

- ***10001***: Path Inexistente
- ***10002***: Espacio insuficiente en FM9

### Concentrar

La primitiva concentrar será un comando de no operación. El objetivo del mismo es hacer correr una unidad de tiempo de quantum.

Sintaxis

No recibirá ningún parámetro.

```
Ej: concentrar
```

Funcionalidad

Esta primitiva realizará una espera de una unidad de tiempo (retardo de ejecución) y continuará con la siguiente primitiva. El objetivo es poder alargar la ejecución de los G.DT para poder realizar distintas operatorias de planificación.

### Asignar

La primitiva asignar permitirá la asignación de datos a una línea dentro de path pasado.

Sintaxis

Este comando recibe 3 parámetros:

- Path: Archivo que se desea escribir
- Línea: Valor numérico que indica la línea dentro del archivo que se desea escribir.
- Datos: Datos que se deben cargar en dicha línea

```
Ej: asignar /equipos/Racing.txt 9 GustavoBou
```

Funcionalidad

El comando asignar realizará los siguientes pasos:

1. Verificará que el archivo que se desea modificar se encuentre abierto por el G.DT. Para esto se utilizaran las estructuras administrativas del DTB. En caso que no se encuentre abierto, se abortará el G.DT informando el error a S-AFA.
2. Se enviará a FM9 los datos necesarios para actualizar el valor en memoria.

### Errores Esperables

Los errores esperados son:

- ***20001***: El archivo no se encuentra abierto.
- ***20002***: Fallo de segmento/memoria.
- ***20003***: Espacio insuficiente en FM9.

## Wait

La primitiva Wait permitirá la espera y retención de distintos recursos tanto existentes como no que administrá S-AFA.

### Sintaxis

Este comando recibe como parámetro el nombre del recurso que se requiere.

```
Ej: wait Conmebol
```

### Funcionalidad

Esta primitiva realizará los siguientes pasos:

1. Enviará a S-AFA una solicitud de retención del recurso requerido.
2. S-AFA, validará si existe dicho recurso. Si no existe en su tabla de recursos lo creará con el valor 1.
3. S-AFA, intentará asignarle dicho recurso al G.DT solicitante. Esta funcionalidad se realizará como los semáforos contadores vistos en teoría, decrementando su valor.
4. S-AFA responderá a la CPU tanto por sí o por no sobre el recurso. En caso que no pueda ser asignado, CPU detendrá su ejecución y S-AFA bloqueará el proceso esperando la liberación del recurso.

## Signal

La primitiva Signal permitirá la liberación de un recurso que son administrados S-AFA.

### Sintaxis

Este comando recibe como parámetro el nombre del recurso a liberar.

```
Ej: signal Conmebol
```

### Funcionalidad

Se ejecutarán los siguientes pasos:

1. Enviará a S-AFA una solicitud de retención del recurso requerido.
2. S-AFA, validará si existe dicho recurso. Si no existe en su tabla de recursos lo creará con el valor 1. En caso de existir, se le sumará uno a dicho recurso.
3. Se realizarán las operatorias necesarias para verificar y desbloquear recursos que estén pendientes de dicho recurso.

4. S-AFA responderá a CPU afirmativamente para que pueda continuar con la ejecución.

### Flush

La primitiva Flush permite guardar el contenido de FM9 a MDJ.

#### Sintaxis

Recibe como parámetro el path del archivo a guardar en MDJ.

```
Ej: flush /equipos/Racing.txt
```

#### Funcionalidad

Se ejecutarán los siguientes pasos:

1. Verificará que el archivo solicitado esté abierto por el G.DT. Para esto se utilizarán la estructura administrativa del DTB. En caso que no se encuentre, se abortará el G.DT.
2. Se enviará una solicitud a El Diego indicando que se requiere hacer un Flush del archivo, enviando los parámetros necesarios para que pueda obtenerlo desde FM9 y guardarlo en MDJ.
3. Se comunicará al proceso S-AFA que el G.DT se encuentra a la espera de una respuesta por parte de El Diego y S-AFA lo bloqueará.

#### Errores Esperables

Los errores esperados son:

- ***30001***: El archivo no se encuentra abierto.
- ***30002***: Fallo de segmento/memoria.
- ***30003***: Espacio insuficiente en MDJ.
- ***30004***: El archivo no existe en MDJ (fue borrado previamente).

### Close

La primitiva Close permite liberar un archivo abierto que posea el G.DT

#### Sintaxis

Recibe como parámetro el path del archivo a cerrar.

```
Ej: close /equipos/Racing.txt
```

#### Funcionalidad

Se ejecutarán los siguientes pasos:

1. Verificará que el archivo solicitado esté abierto por el G.DT. Para esto se utilizarán la estructura administrativa del DTB. En caso que no se encuentre, se abortará el G.DT.
2. Se enviará a FM9 la solicitud para liberar la memoria del archivo deseado. Para esto, se le pasaran los parámetros necesarios para que FM9 pueda realizar la operación.
3. Se liberará de las estructuras administrativas del DTB la referencia a dicho archivo.

#### Errores Esperables

Los errores esperados son:

- ***40001***: El archivo no se encuentra abierto.

- ***40002***: Fallo de segmento/memoria.

### Crear

La primitiva crear permitirá crear un nuevo archivo en el MDJ. Para esto, se deberá utilizar de intermediario a El Diego.

#### Sintaxis

Recibe como parámetro el path y la cantidad de líneas que tendrá el archivo.

```
Ej: crear /equipos/Racing.txt 11
```

#### Funcionalidad

Se ejecutarán los siguientes pasos:

1. Se deberá enviar a El Diego el archivo a crear con la cantidad de líneas necesarias. El CPU desalojará el programa G.DT y S-AFA lo bloqueará.
2. El Diego enviará los datos a MDJ para que este creé el nuevo archivo.
3. MDJ leerá dichos datos, verificará si tiene espacio necesario, por medio de sus estructuras administrativas, y creará el archivo. El archivo creado debe contener internamente la cantidad de líneas indicadas con cada línea vacía (con "\n")
4. MDJ informará a El Diego la finalización de dicha operación y este último se lo comunicará a S-AFA para que desbloquee el programa G.DT.

En caso de que suceda algún error en dicha operatoria se debe abortar el programa G.DT

#### Errores Esperables

Los errores esperables son:

- ***50001***: Archivo ya existente.
- ***50002***: Espacio insuficiente.

### Borrar

La primitiva borrar permitirá eliminar un archivo de MDJ.

#### Sintaxis

Recibe como parámetro el path del archivo a borrar.

```
Ej: borrar /equipos/Racing.txt
```

#### Funcionalidad

Se ejecutarán los siguientes pasos:

1. Se deberá enviar a El Diego el archivo a crear con la cantidad de líneas necesarias. El CPU desalojará el programa G.DT y S-AFA lo bloqueará.
2. El Diego enviará los datos a MDJ para que este elimine el archivo.
3. MDJ buscará si el archivo existe, eliminará el archivo de su carpeta de archivos y actualizará el bitmap con los nuevos bloques libres/disponibles.
4. MDJ informará a El Diego la finalización de dicha operación y este último se lo comunicará a S-AFA para que desbloquee el proceso G.DT.

En caso que suceda un error en dicha operación se debe abortar el programa G.DT.

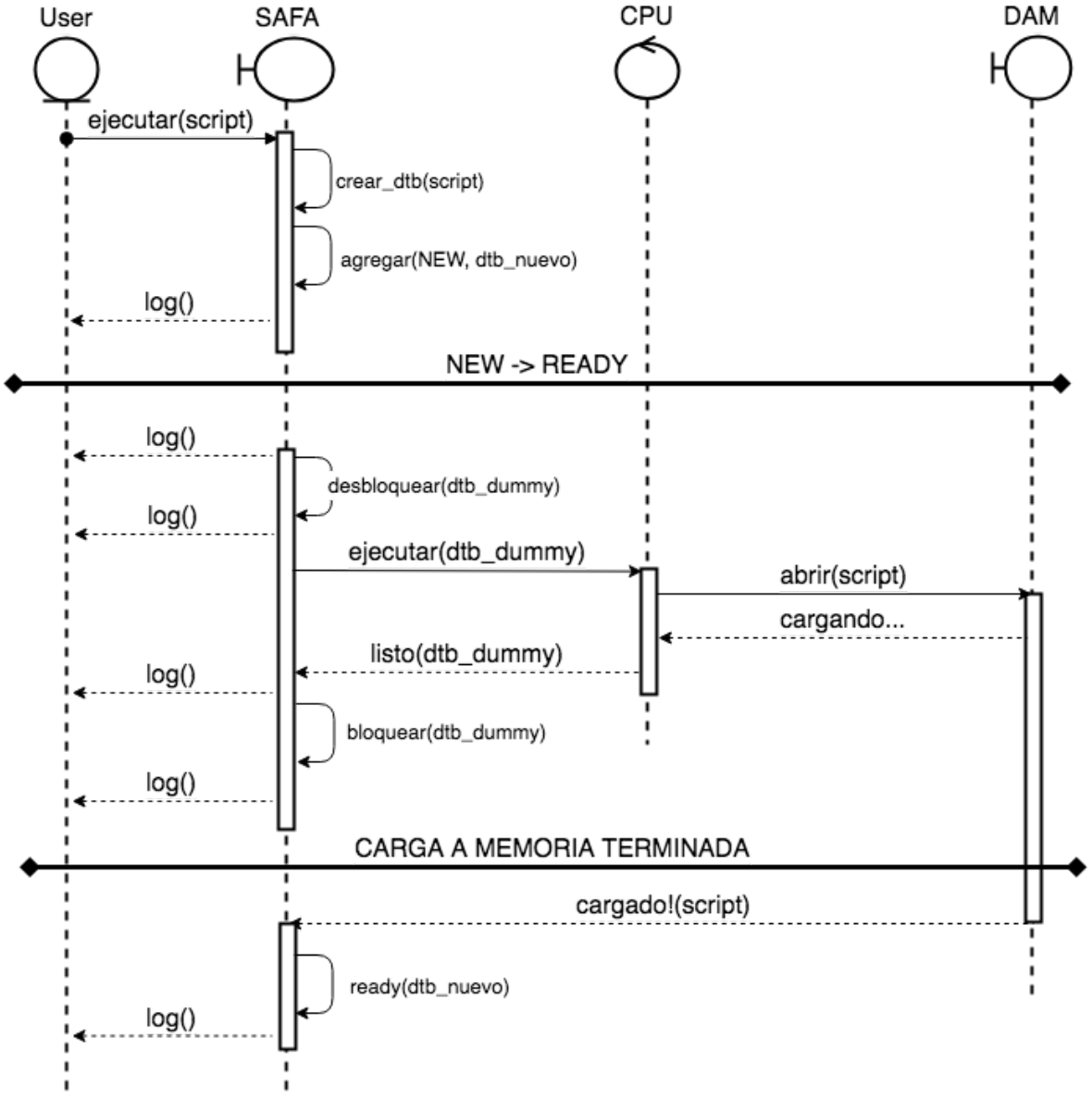
### Errores Esperables

Los errores esperables son:

- ***60001***: El archivo no existe.

# Anexo II - Flujos de Instrucciones

**¿Cómo se comunican los módulos del TP para iniciar un programa?**



# Descripción de las entregas

## Hito 1: Conexión Inicial

**Fecha**: 15/09/2018 **Tiempo Estimado**: 2 semanas

### Objetivos:

- ★ Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio
- ★ Aplicar las Commons Libraries, principalmente las funciones para listas, archivos de conf y logs
- ★ Definir el Protocolo de Comunicación
- ★ Familiarizarse con el desarrollo de procesos servidor

### Implementación mínima:

- ★ Creación de todos los procesos que intervienen
- ★ Desarrollar una comunicación simple entre los procesos que permita propagar un mensaje por cada conexión necesaria
- ★ Implementar la consola de S-AFA sin funcionalidades

**Lectura recomendada:**

- http://faq.utnso.com/arrancar
- Beej Guide to Network Programming - link
- Linux POSIX Threads - link
- SO UTN FRBA Commons Libraries - link
- Sistemas Operativos, Silberschatz, Galvin - Capítulo 3: Procesos
- Sistemas Operativos, Silberschatz, Galvin - Capítulo 4: Hilos

## Hito 2: Avance del Grupo

**Fecha**: 06/10/2018 **Tiempo Estimado**: 3 semanas

### Objetivos:

- ★ Implementación de la base del Protocolo de Comunicación
- ★ Comprender y aplicar mmap() si fuese necesario
- ★ Comprender con algo de profundidad cómo funcionan algunos algoritmos sencillos

### Implementación mínima:

- ★ Lectura de Escriptorios y primera versión del parseo de instrucciones. Instrucción "Ejecutar" lista.
- ★ S-AFA debe ser capaz de elegir a un G.DT utilizando un algoritmo sencillo (FIFO por ej)
- ★ El Diego debe poder reenviar mensajes a MDJ o FM9
- ★ MDJ deberá exponer su interfaz, devolviendo mensajes prefijados. Primera versión de las estructuras administrativas lista.
- ★ FM9 deberá poder guardar los datos de un único G.DT en su storage, sin manejar estructuras aún.

**Lectura recomendada:**

- ★ Sistemas Operativos, Silberschatz, Galvin - Capítulo 4: Hilos
- ★ Sistemas Operativos, Silberschatz, Galvin - Capítulo 5: Planificación
- ★ Sistemas Operativos, Silberschatz, Galvin - Capítulo 8: Memoria

## Hito 3: Checkpoint Presencial en el Laboratorio

**Fecha**: 03/11/2018 **Tiempo Estimado**: 4 semanas

### Objetivos:

- ★ Entender las implicancias de un algoritmo de planificación real
- ★ Entender el concepto de productor-consumidor y sus problemas de concurrencia
- ★ Implementar estructuras similares a las usadas en Memoria

### Implementación mínima:

- ★ CPU y El Diego completos
- ★ Planificador utilizando Round Robin y Virtual Round Robin
- ★ La consola de S-AFA deberá poder ejecutar los comandos "Status" y "Finalizar"
- ★ FM9 deberá implementar Segmentación Simple y Segmentación Paginada
- ★ MDJ deberá poder Validar y Crear Archivos así como también Obtener y Guardar Datos.

**Lectura recomendada:**

- Sistemas Operativos, Silberschatz, Galvin - Capítulo 8: Memoria
- Sistemas Operativos, Silberschatz, Galvin - Capítulo 10 y 11: File System

## Hito 4: Entregas Finales

**Fechas**: 01/12/2018 - 15/12/2018 - 22/12/2018

### Objetivos:

- ★ Probar el TP en un entorno distribuido
- ★ Realizar pruebas intensivas
- ★ Finalizar el desarrollo de todos los procesos
- ★ Todos los componentes del TP ejecutan los requerimientos de forma integral, bajo escenarios de stress.

### Implementación mínima:

- ★ Todos los módulos deberán estar completos.

**Lectura recomendada:**

- Guías de Debugging del Blog utnso.com - link
- MarioBash: Tutorial para aprender a usar la consola - link