

LABORATORIO 2

1. Introducción

En esta tarea se desea implementar una herramienta para la administración de procesos y recursos en un sistema operativo tipo Linux. Denominaremos a esta herramienta, el **adminso** (Administrador Sistemas Operativos)

En el contexto de este laboratorio un **proceso** consiste en una secuencia acotada de instrucciones de cómputo y solicitudes de recursos al sistema. Los distintos usuarios pueden solicitar **crear**, **eliminar**, **iniciar** o **matar** procesos a lo largo de su sesión en el sistema, utilizando las funcionalidades expuestas por el adminso.

Una vez que un proceso se encuentra en ejecución, deberá ser posible monitorear la actividad del mismo. A través de adminso, interesa conocer:

- Fecha de inicio de ejecución
- Recursos que tiene asignados
- Nombre del binario que con el que se ejecuta

Los procesos, a lo largo de su vida, pueden requerir recursos, como por ejemplo, archivos, memoria, E/S a dispositivos, canales de comunicación, etc. Estos recursos serán administrados por **adminso** también. En el caso particular del laboratorio, los recursos a manejar serán archivos de solo lectura.

2. Componentes del sistema

A continuación se detallan los elementos que componen la herramienta **adminso**

2.1 Registry de procesos

Los procesos que **adminso** puede ejecutar, están asociados a archivos binarios ejecutables. Antes que los usuarios puedan ejecutar los procesos a través del administrador, se deberá registrar los datos completos del binario. Por ejemplo, si se dispone de un programa que imprime el clásico mensaje "Hola Mundo", en la ruta "/tmp/hola-mundo", deberían registrarse los datos presentados en la siguiente tabla en **adminso**.

La estructura que almacenara los datos de los binarios a ejecutar, será de la forma:

Clave de Proceso	Ruta	Descripción
HolaMundo	/tmp/hola-mundo	Proceso que escribe "Hola mundo"
Fibonacci	/home/usr/fibo	Proceso que calcula fibonacci

2.2 Registry de recursos

Los distintos recursos que se deseen utilizar en los procesos iniciados desde **adminso**, deberán ser registrados previamente, en una forma similar a como se describió para los procesos. El registro se hará en una estructura similar a la anterior, donde tendremos la **clave** del recurso, la **ruta**, el **tipo**, la **descripción** del mismo y el tiempo de vida del recurso

Clave de Recurso	Ruta	Tipo	Descripción	Tiempo de vida
Archivo1	/etc/archivo1.txt	A	Archivo de texto con datos	-1
hosts	/etc/hosts	A	Archivo con direcciones IP	600

Dado un recurso, el tiempo de vida del mismo indica el tiempo máximo que este puede permanecer en la registry. El servidor realiza periódicamente un proceso de barrido de los recursos. Si se detecta un recurso que supera su tiempo de vida y no está en uso, será eliminado. El valor -1 indica que el tiempo de vida es infinito. El tiempo de vida se mide en segundos.

2.3 Administración de procesos y recursos (Servidor)

Una de las funciones principales del **adminso**, es administrar recursos y procesos. Las tareas que deberán realizarse a través del administrador son:

- **Crear y Eliminar** un proceso
- **Iniciar y Matar** un proceso.
- **Consultar** el estado de un proceso.

Todas las operaciones son realizadas a través de la clave de proceso registrado en el sistema.

Las tareas que deberán realizarse a través del administrador relativas a recursos son:

- **Crear y Eliminar** un recurso
- **Adquirir y Liberar** un recurso.
- **Consultar** el estado de un recurso.

Para poder utilizar un recurso, un proceso debe primero solicitarlo a través de las funciones anteriores. La adquisición de un recurso, es siempre **exclusiva**.

En la consola del servidor, se deberá poder ejecutar dos comandos:

- **exit**
Este comando finaliza la ejecución del servidor. Una vez emitido el mismo, deberán matarse todos los procesos registrados en el servidor, que estén ejecutando.
- **stat**
Se deberá desplegar en la pantalla información de estado del servidor (procesos que están en ejecución y recursos en uso).

2.4 Pedido de servicios (Cliente)

A fin de invocar los servicios provistos por el servidor, se deberá implementar un módulo cliente, que reciba los parámetros de la línea de comando, y pida el servicio al servidor (ver punto anterior). Denominaremos a este cliente **cliensso**.

El **cliensso** recibirá los parámetros de la siguiente forma:

- **Crear un proceso:**
`cliensso creproc <clave proceso> <ruta binario> <descripcion>`
- **Eliminar un proceso**
`cliensso eliproc <clave proceso>`
- **Iniciar un proceso**
`cliensso iniproc <clave proceso>`
- **Matar un proceso.**
`cliensso matproc <clave proceso>`
- **Consultar el estado de un proceso.**
`cliensso conproc <clave proceso>`

- **Crear un recurso archivo**

```
clienseo crerecu <clave recurso> <ruta recurso> <descripcion> <tiempo vida>
```

- **Eliminar un recurso**

```
clienseo elirecu <clave recurso>
```

- **Consultar el estado de un recurso.**

```
clienseo conrecu <clave recurso>
```

2.5 API

Los procesos que ejecuta el administrador pueden hacer uso de los recursos a través de la siguiente API

```
// Retorna un puntero al contenido del recurso solicitado
char* pedir_recurso(
    char* clave_proceso,    // clave de proceso (entrada)
    char* clave_recurso,   // clave de recurso (entrada)
    int* tamano_recurso,   // tamaño del curso (salida)
    int* codigo_retorno    // código de retorno (salida)
)
// Libera el recurso adquirido por el proceso
void liberar_recurso(
    char* clave_proceso,   // clave de proceso (entrada)
    char* clave_recurso,   // clave de recurso (entrada)
    char* recurso,        // recurso asignado (entrada)
    int* codigo_retorno    // código de retorno (salida)
)
```

Esta API deberá implementarse en un archivo API.c. Deberá aclararse con la documentación correspondiente, que archivos objetos (*.o) deben incluirse en la linkedición para utilizar esta API.

2.6 Tarea

La tarea consiste en **diseñar, implementar y documentar las herramientas adminso y clienseo**, asegurando el correcto funcionamiento de las mismas. Se deberán hacer casos de pruebas para verificar la correctitud del producto entregado, mostrando el uso del API presentada.

3. Aclaraciones

- Los recursos manejados en esta tarea son archivos de texto de solo lectura
- Los procesos lanzados por el **adminso**, corresponden a programas que se encuentran en el sistema de archivos local
- Se deberán definir las funciones necesarias para realizar las tareas anteriores, explicitando la semántica de las mismas.
- Puede asumirse que la cantidad máxima de recursos a registrar esta acotada. Este valor máximo será definido con el nombre **MAX_RECURSOS**.
- Puede asumirse que la cantidad máxima de procesos a registrar esta acotada. Este valor máximo será definido con el nombre **MAX_PROCESOS**.
- Puede asumirse que la cantidad máxima de recursos asignados a un proceso esta acotada. Este valor máximo será definido con el nombre **MAX_RECURSOS_PROCESO**. Puede asumirse que este valor es pequeño (Por ejemplo, **MAX_RECURSOS_PROCESO = 5**)
- El servidor realiza el proceso de barrido de recursos cada **TIEMPO_BARRIDO_SEGUNDOS**.
- Los procesos que **adminso** puede ejecutar no deben recibir parámetros por la línea de comandos

4. Consideraciones generales

4.1 Implementación

El trabajo debe realizarse utilizando el lenguaje de programación C y debe correr sobre el sistema operativo instalado en las PCs de la sala 502.

- Puede utilizarse sintaxis o funciones de biblioteca de C++, pero no clases.
- Compile utilizando make, con un archivo Makefile (sin extensión). No se aceptarán entregas sin Makefiles, o que compilan con scripts o similares.
- Deberá compilarse con la opción `-Wall` (un parámetro de `g++`). Esta opción permite ver errores y advertencias (warnings) de compilación más detalladas. El trabajo entregado no debe producir ningún warning al compilarlo.

A continuación se enumeran algunos de los comandos con los cuales se deben familiarizar para resolver el problema planteado:

- | | | |
|-------------------------------|-------------------------------|--------------------------------|
| <input type="checkbox"/> gcc | <input type="checkbox"/> ps | <input type="checkbox"/> ipcs |
| <input type="checkbox"/> make | <input type="checkbox"/> kill | <input type="checkbox"/> ipcrm |
| <input type="checkbox"/> man | | |

La siguiente es una lista con algunas de las llamadas al sistema que se deben conocer para entender la sincronización entre procesos en el sistema y el mecanismo de memoria compartida.

- | | | |
|----------------------------------|---------------------------------|---------------------------------|
| <input type="checkbox"/> fork | <input type="checkbox"/> shmctl | <input type="checkbox"/> semget |
| <input type="checkbox"/> wait | <input type="checkbox"/> shmget | <input type="checkbox"/> semctl |
| <input type="checkbox"/> waitpid | <input type="checkbox"/> shmat | <input type="checkbox"/> semop |
| <input type="checkbox"/> exit | <input type="checkbox"/> shmdt | |

Otras funciones útiles:

- | | | |
|--------------------------------|---------------------------------|--------------------------------|
| <input type="checkbox"/> fgets | <input type="checkbox"/> signal | <input type="checkbox"/> sleep |
| <input type="checkbox"/> execv | | |

4.2 Documentación

Debe entregarse toda la documentación impresa y en versión electrónica. En este caso se requiere el diseño de la solución tomada (las estructuras de datos, el pseudo código).

No incluir la letra ni parte de ella en la documentación. Se penalizarán documentaciones que repitan los requerimientos o no describan lo implementado. Los fuentes deben estar impresos al final de la documentación, en un anexo.

Debe especificarse en el sobre de la entrega el número de grupo e integrantes.

4.3 Plazo y Entrega

La fecha de entrega es el día **31 de Mayo de 2005 en el horario de 18:00 a 20:00 hs.**, en el buzón correspondiente en la entrada del INCO (5º piso). También deberá entregarse a través de la pagina web del curso.

De ninguna forma se aceptarán entregas fuera de la fecha de entrega estipulada.

5. Condiciones del Laboratorio

El trabajo es obligatorio, la no entrega del trabajo o el mal funcionamiento implican la pérdida del mismo y por lo tanto la pérdida del curso.

Cada grupo debe realizar su trabajo en forma individual, y la detección de trabajos hechos en forma no individual (entre los grupos) implica la pérdida del mismo para los grupos involucrados. En este caso también implica la pérdida del curso para todos los participantes de los grupos.

Cada grupo es responsable de proteger su trabajo contra copia. No olvide diskettes en la sala de máquinas, ni envíe correos o mensajes al newsgroup que permitan a otros copiar su código o parte de él.

5.1 Consultas

Se atenderán consultas durante los últimos 45 minutos de las clases prácticas y mediante el newsgroup del curso.

6. Bibliografía

- [1] El entorno de programación Unix. (Kernighan/Pike)
- [2] El lenguaje de programación C (Kernighan / Ritchie)
- [3] Red Hat inc, The Official Red Hat Linux Getting Started Guide
<https://www.redhat.com/docs/manuals/linux/RHL-9-Manual/getting-started-guide/>
- [4] Home Page del Fedora Project (<http://fedora.redhat.com/>)
- [5] Ayuda de los comandos Unix. (en el archivo Lab1/Documentacion/ComandosUnix.txt)
- [6] Cualquier tutorial básico de Unix/Linux
(por ejemplo - <http://ie.fing.edu.uy/~vagonbar/unixbas/tutorial.htm>)
- [7] Páginas del manual de Unix on-line
(por ejemplo - <http://man.he.net/>)
- [8] Advanced Linux Programming (<http://www.advancedlinuxprogramming.com>)