# Laboratorio 1 ~ Sistemas Operativos

\_Integrantes\_

\* Joaquín Origlia

\* Isabel Rivadero isarivadero@gmail.com

\* Ignacio Todeschini

---

## Introducción

Comenzamos el proyecto descargando y estudiando la estructura del esqueleto

otorgado por la cátedra, encontrándonos con algunas dificultades que fueron

resueltos por los docentes y otras buscando un poco de información adicional en internet.

## TAD's Scommand y Pipeline

TAD Scommand:

Comenzamos inspeccionando en primer lugar lo que se especificaba en en el pdf.

Para el `TAD Scommand` utilizamos la estructura conformada por dos punteros de tipo char de los cuales serán para las redirecciones de input y output. Y para la lista de comandos utilizamos la implementación `GList` que pertenece a la librería `GLib` que estaba entre las recomendadas dentro del pdf.

Decidimos utilizar esa ya que para la implementación del TAD a nuestro parecer fue la más amigable y comprensible tanto a nivel de las funciones predeterminada que trae consigo como la estructura del mismo.

En principio una vez, decidida las herramientas a utilizar, comenzamos a inspeccionar el archivo `command.h` ya que es el archivo que contiene las definición de las funciones requeridas.

Para el `TAD Pipeline` utilizamos el mismo método que con `Scommand` a diferencia

que en vez de utilizar dos punteros, este necesitaba contar con una variable

`Booleana` que representara el wait del comando.

Y continuamos con el mismo procedimiento, nos guiamos con las funciones requeridas

que se encontraban en `command.c`.

##Execute

¿Refleja el documento todo lo que trabajaron?

¿Muestra el infome tanto una visión general, como ciertos aspectos específicos?

¿Cómo desarrollaron el programa?

¿Qué funcionalidades pedidas faltan?

¿Qué funcionalidades agregaron ustedes?

¿Qué TADs usan, cual es su interface y cómo están implementados?

¿Qué funcionalidades destacables de bibliotecas externas utilizaron?

¿Utilizaron herramientas especiales para el desarrollo, edición, compilación, debugging y profiling?

Módulo final encargado de invocar a las ​ syscalls fork()​; execvp()​ necesarias para

ejecutar los comandos, donde ademas armamos un pipeline para conectar dos o mas comandos

La primera tarea del módulo de ejecución es reconocer entre comandos internos y externos y para decidir si invocar a una función interna utilizando

builtin\_is\_internal(cmd), o a la ejecución de procesos de manera externa.

un punto importante es la correcta interacción entre ​ fork​ , ​ pipe​ , ​ close​ , a fin de ​ cerrar todas

las puntas innecesarias. Si esto es así, solo el proceso hijo estará apuntando a la entrada del pipeline ​ ​ pipe[1]​ y cuando

el comando termine se produce automáticamente el cierre de todos los file descriptors. Cuando el

sistema operativo recibe el último ​ close(pipe[1])​ , induce la lectura de un ​ EOF​ desde el proceso

que está colgado a ​ pipe[0]​ y éste puede terminar.

El síntoma más común de un ​ close​ olvidado es un ​ pipeline ​ que queda bloqueado para siempre,

esperando un ​ EOF​ que jamás llegará.

adaptación del ​ scommand​ a la estructura c

har \*\*argv​ que necesita ​ execvp​ .

Los permisos con los cuales se abren los archivos de redirección, especialmente el de

salida.