

Trabajo Práctico 0

Versión 2.3.0

El TP0 es una práctica inicial para empezar a familiarizarse con algunas de las herramientas necesarias para el trabajo práctico cuatrimestral como es la configuración del entorno, el lenguaje C, etc. Es un ejercicio que sirve como base para empezar el TP luego.

El TP0 va a ser realizado en **etapas**, cada una de ellas con un entregable que servirá de base para la siguiente. La idea de este ejercicio es que lo realicen de manera individual o grupal (no más de cinco, idealmente los mismos con los que harán el TP cuatrimestral), y en unas semanas tendremos una entrega **obligatoria**.

No es necesario tener el ejercicio completo y la entrega no lleva nota, pero presentarse con lo que tengan en condición necesaria para la continuidad de la materia. Más adelante publicaremos junto al enunciado del TP la fecha de entrega de este ejercicio.

Objetivo

El objetivo de este TP0 es empezar a familiarizarse con el entorno en el que desarrollarán el TP de la materia, aprendiendo en el proceso cómo utilizar las commons por su cuenta. La idea es que siguiendo este documento logren completar las funciones vacías y comentarios que les dejamos en el código.

TIP

Pueden hacernos cualquier pregunta que tengan sobre el enunciado, C, o cualquier otro concepto en los [medios de consulta de la práctica](#).

Requisitos

- Contar con un entorno Linux
- Tener configuradas las commons y git
- Tener a mano el [repo del TP0](#)

Etapas 1: Setup inicial

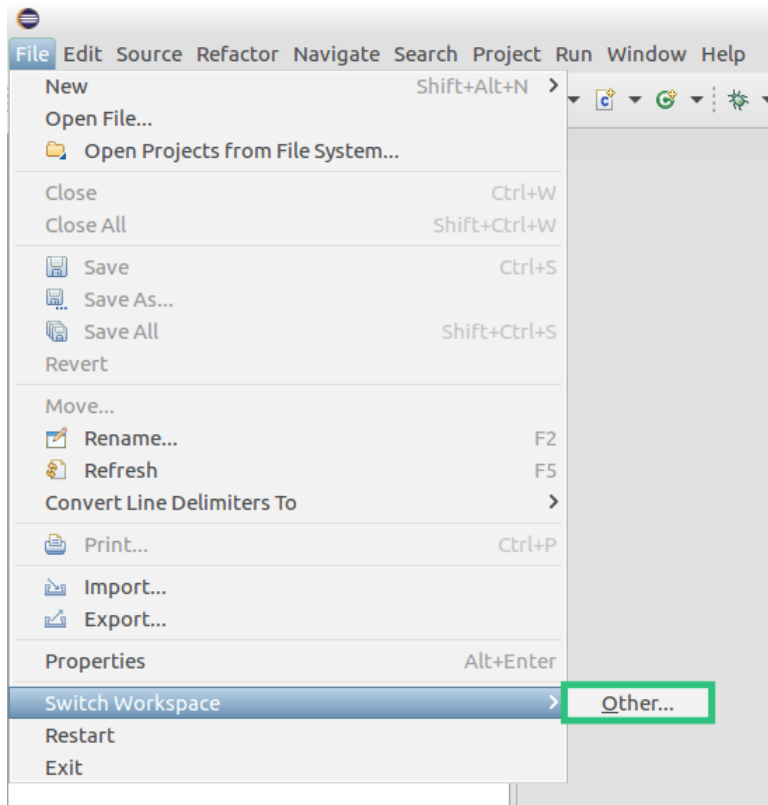
Crear un workspace

Primero, abramos una consola y, de la misma forma que bajamos el proyecto de las commons, bajemos el del TP0:

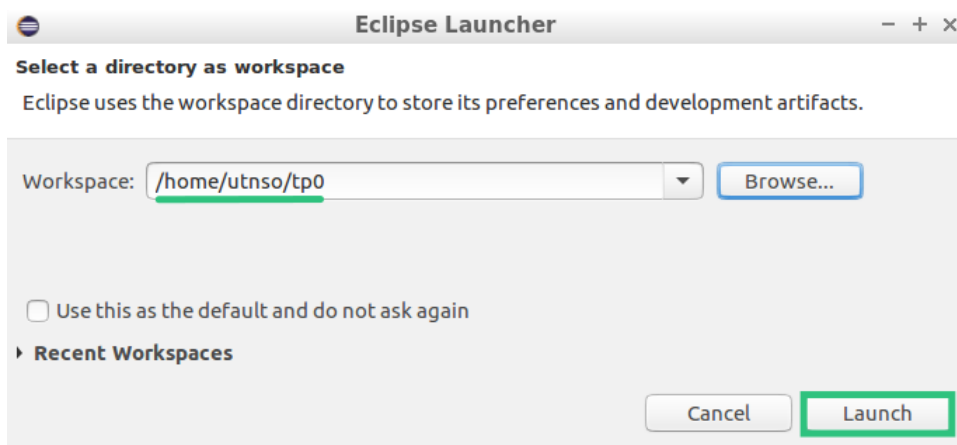
```
git clone https://github.com/sisoputnfrba/tp0
cd tp0
```

bash

Luego, en el Eclipse vamos a movernos hacia `File > Switch Workspace > Other...` :

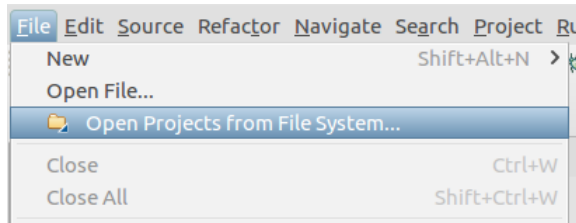


Y vamos a seleccionar la carpeta donde clonamos el repositorio del TP0:

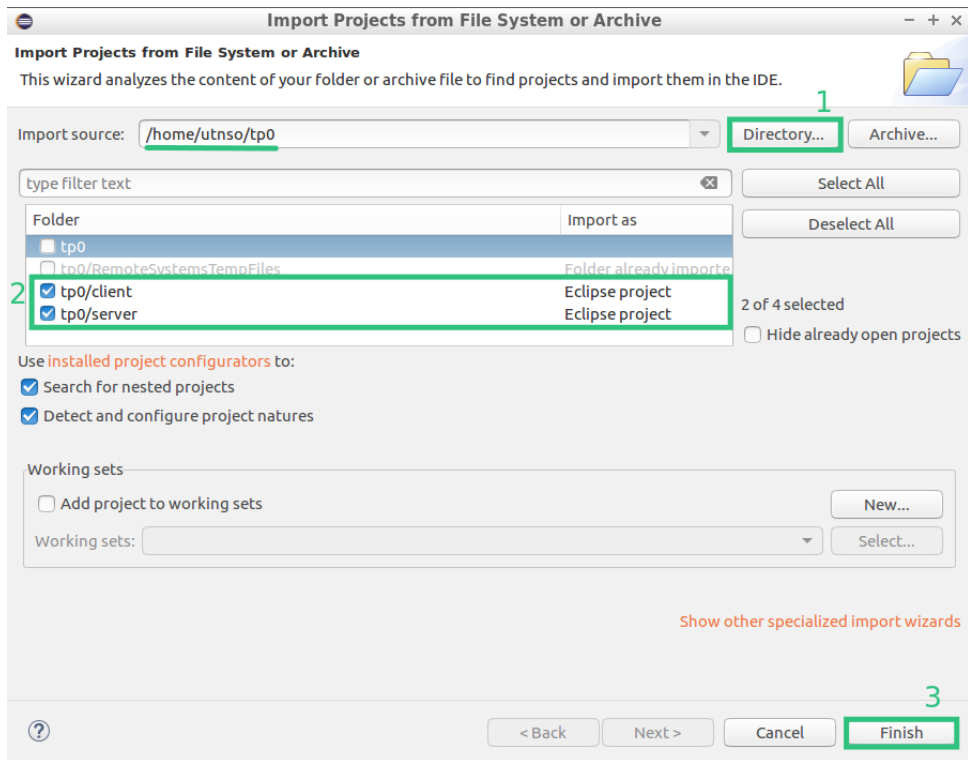


Agregar los proyectos al workspace

Ahora, para agregar los proyectos al workspace, nos vamos a mover hacia `File > Open Projects From File System...` :

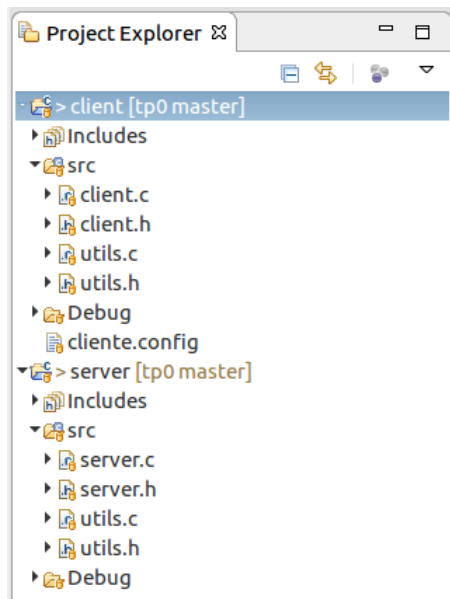


Nos aparecerá el siguiente menú, sobre el cual...



1. Elegiremos de nuevo la carpeta del workspace desde el botón `Directory...`
2. Dejaremos tildados solamente los proyectos `server` y `client`
3. Haremos click en `Finish` para continuar

A la izquierda, en la pestaña `Project Explorer`, nos van a aparecer ambos proyectos listos para arrancar con el TP0:



Etapa 2: Comandos básicos

El objetivo de esta etapa es aprender un par de funcionalidades que utilizaremos bastante durante todo el desarrollo del trabajo práctico cuatrimestral.

Logging



Durante todo el TP iremos logueando en un archivo de texto las diferentes acciones que el programa vaya realizando, tanto las correctas, como los errores. Para ello, utilizaremos las funciones de logging que proveen las commons.

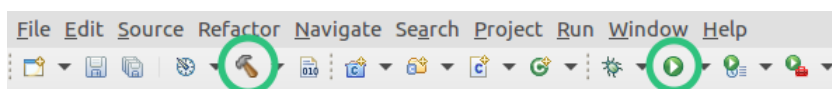
Parados en el archivo `cliente.c`, si revisamos el **header de log de las commons** vamos a encontrar la función `log_create()`, que nos devuelve un logger listo para usar.

Ayudándonos con la descripción que aparece en el header, vamos a configurarla para que:

- Loguee en el archivo "tp0.log"
- Muestre los logs por pantalla (y no solo los escriba en el archivo)
- Muestre solo los logs a partir del nivel "info".

Creado nuestro logger, usemos `log_info()` para loggear el string "Soy un Log" y cerremos el logger al final del programa con `log_destroy()`.

Compilemos usando el  (o apretando `Ctrl + B`) y démosle Run al programa con  (o `Ctrl + F11`).



Archivos de configuración

Estaría bueno que ese valor que logueamos no esté hardcodeado en el código, sino que podamos configurarlo para que varíe sin tener que recompilar todo el proyecto, por lo que vamos a leerlo a partir de un archivo de configuración y lo vamos a loggear usando nuestro logger.

Para ello vamos a usar **las config de las commons**. Siguiendo su header, creemos una config que levante el archivo `tp0.config` y obtengamos el valor de la key `CLAVE` en formato string.

TIP

Para saber dónde guardar el archivo config y cómo hace Eclipse para leerlo, podés consultar la [guía de paths](#).

Usemos el logger anterior para mostrar el valor que obtuvimos. Compilamos, corramos el programa y evaluemos los resultados.

¡No se olviden de destruir el config al final del programa!

IMPORTANTE

Para todas las funciones de biblioteca que uses, recuerden chequear los valores de retorno de las mismas para poder manejar los casos de error.

En este caso, si llegamos a tener algún error al crear el config vamos a querer terminar con la ejecución:

```
if (config == NULL) {  
    // ¡No se pudo crear el config!  
    // Terminemos el programa  
}
```

C

Leer de consola

De los comandos básicos, nos queda leer de consola. Si bien existen muchas formas de hacerlo, vamos a usar la biblioteca `readline`.

Necesitamos incluirla usando:

```
#include <readline/readline.h>
```

C

Una vez incluida, la función `readline("> ")` va a hacer que el programa espere a que se ingrese una línea y devolverla en un string ya listo para loggear.

La misma ya se encuentra agregada en el TP0, por lo que no hace falta hacer este include.

IMPORTANTE

Recuerden que `readline()` no te libera la memoria que devuelve, por lo que es necesario liberarla usando `free(1)`.

TIP

Para más info sobre algunas features más avanzadas de `readLine()`, pueden consultar la [guía de readline](#).

Strings

Terminando con esta etapa, nos gustaría que el TP0 lea de consola todas las líneas que se ingresen, las loguee y, si se ingresa una línea vacía, termine con el programa.

Si ejecutamos el comando `man readline` en la consola (o visitamos el [manual en internet](#)), podemos ver en la sección "Return Value" que, ante una línea vacía, el valor de retorno de la función es un string vacío.

Pero... ¿cómo hacemos para revisar eso?

Los strings son cadenas de caracteres terminadas con `'\0'` (el caracter nulo de la [tabla ASCII](#)).

<code>char str[6] = "Hello";</code>						
index	0	1	2	3	4	5
value	H	e	l	l	o	\0
address	1000	1001	1002	1003	1004	1005

Eso implica que un string vacío va a tener, en su contenido, ese caracter como primer valor, por lo que podemos usar una comparación como **condición de corte**.

También podemos usar la función `strcmp()` de la biblioteca estándar de C para comparar strings. En este caso, compararíamos lo que nos devuelva `readline()` con un string vacío `""` para saber si debemos salir del bucle o no.

IMPORTANTE

Si en alguna etapa del TP el programa no se comporta como esperaban, pueden intentar ejecutarlo línea por línea siguiendo el [tutorial de debugging en Eclipse](#).

TIP

Las commons también proveen funciones para simplificar el manejo de strings. Pueden consultar su documentación leyendo los [headers](#).

Etapas 3: Programar el Cliente-Servidor

IMPORTANTE

Para poder ayudar con los conceptos y aspectos técnicos de esta etapa tienen disponible la [guía de sockets](#).

A partir de esta etapa, vamos a plantear una arquitectura **cliente-servidor**. Para esta sección, ambos `client` y `server` en sus respectivas carpetas tienen un archivo `utils.c` con comentarios sobre lo que debemos hacer para poder conectar ambos procesos mediante la red.

Consigna

El entregable de esta etapa es enviar al servidor el valor de **CLAVE** en el archivo de configuración, y luego enviarle todas las líneas que se ingresaron por consola juntas en un paquete.

Simplificando un poco, una conexión por socket hacia otro programa va a requerir de realizar lo siguiente:

- Iniciar el servidor en la función `iniciar_servidor()` del `utils` del `server`.
- Esperar a que el cliente se conecte mediante la función `esperar_cliente()`
- Crear una conexión contra el servidor en la función `crear_conexión()` del `utils` del `client`.
- Enviar como mensaje el valor de CLAVE.
- Ir juntando las líneas que se leen por consola para luego enviarlas como paquete.
- Cerrar la conexión.

Funciones

Para simplificar el TP0, tenemos ya pre implementadas un par de funciones para comunicarnos con el servidor en el archivo fuente `utils`, que deberán consultarlo de manera similar al log y config de la etapa anterior:

- `enviar_mensaje(3)` : Recibe el socket, el tamaño de los datos y los datos a mandar.
- `liberar_conexion(1)` : Termina la conexión y libera los recursos que se usaron para gestionar la misma.

La única limitación es que estas funciones no nos sirven para enviar las líneas de consola todas juntas, por lo que vamos a crear un paquete. Este paquete nos va a asegurar que toda la información que mandemos se envíe junta. Para ello, les proveemos otro conjunto de funciones o "API" para crear, rellenar y enviar paquetes:

- `crear_paquete()` : Nos crea el paquete que vamos a mandar.
- `agregar_a_paquete(2)` : Dado un **stream** y su tamaño, lo agrega al paquete.
- `enviar_paquete(2)` : Dada una conexión y un paquete, lo envía a través de ella.
- `eliminar_paquete(1)` : Elimina la memoria usada por el paquete.

Strings vs streams

Es importante recalcar que hay que agregar al paquete un string (y no un stream). La diferencia entre ambos radica en que, si bien son una seguidilla de "bytes", el string termina en el caracter `'\0'`, mientras que un stream puede contener cualquier tipo de dato, por lo que hace falta especificar su tamaño utilizando un parámetro extra.

Para calcular el tamaño de ese string vamos a consultar la documentación de una función amiga llamada `strlen()`. Veremos que **nos devuelve el tamaño de un string, sin contar el caracter centinela**, por lo que hay que "sumarle 1".

Usando `enviar_mensaje()` para enviar nuestro valor de config y `enviar_paquete()` para enviar las líneas de consola, deberíamos poder resolver el entregable.

TIP

En el TP van a necesitar enviar estructuras más complejas que un string. Para hacerlo, pueden consultar nuestra [guía de serialización](#).

Notas finales

El transcurso de esta guía de primeros pasos fue un poco largo, ¡pero aprendimos un montón! Recapitulemos un poco:

- Pudimos configurar nuestro entorno de desarrollo.
- Aprendimos a usar funciones de las commons que nos van a ser muy útiles.
- Aprendimos sobre reservar memoria, liberarla y leer por consola.
- Pudimos mandar mensajes por red a otro programa.

Esto fue todo, pero recuerden que el TP0 es solo una introducción a todas las herramientas que podemos usar.

Por lo tanto, les pedimos que consulten las guías y video tutoriales linkeados en la pestaña de "Guías" de la barra de navegación de esta página para mejorar constantemente y llegar bien holgados a fin de cuatrimestre.

¡Hasta la próxima amigos!