



Game & Watch - Enunciado

¿De qué se trata?

Este ejercicio 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 corto y sirve como base para empezar el TP luego.

La idea de este ejercicio es que intenten realizarlo solos en sus casas para luego poder hacer una resolución en vivo durante el segundo seminario.

El TP0 va a ser realizado en **etapas**, cada una de ellas con un entregable que servirá de base para la siguiente.

Así mismo, si bien el workshop va a ser de forma pseudo-guiada con esta consigna, pueden consultarnos lo que necesiten.

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. Luego haremos una puesta en común y procederemos a explicar la solución

Requisitos

- La VM con Lubuntu: <https://www.utnso.com.ar/recursos/maquinas-virtuales/>

Etapa 1: Setup Inicial

El objetivo de esta etapa es tener el **entorno bien configurado** para poder desarrollar sin problemas las etapas siguientes. También te va a servir para ya tener el entorno configurado para el resto del TP, por lo que no esperes al seminario, hazlo en la compu que vas a usar para desarrollar.

Para ello, prendamos la Máquina Virtual e instalemos las Guest Additions. Pero, **¿qué son las Guest Additions?** Son unas herramientas extras que permiten que Virtualbox nos provea de ciertas funcionalidades; como, por ejemplo, **poder hacer pantalla completa**.

Cómo hacerlo en este documento es un poco aburrido, ¡tenemos un [video](#) enseñando como se hace!

Una vez instaladas las Guest Additions, necesitamos instalar el repo de las “commons”, la biblioteca con funciones útiles y necesarios que utilizarán a lo largo de todo el desarrollo del TP cuatrimestral, que va a ser una dependencia en nuestro tp0. A partir de acá, todo se va a hacer con la consola, así que preparen su Ctrl+T para abrirla. Protip: Por lo general, siempre vamos a estar en la “home” del usuario utnso: `/home/utnso` (o `~/`, para los amigos).



Para bajar el repo de las commons, vamos a usar git, la herramienta de versionado que usamos por defecto en la cátedra. Con **git clone**, bajemos las commons e instalémoslas usando **make install**.

```
git clone https://github.com/sisoputnfrba/so-commons-library
cd so-commons-library
sudo make install
cd ..
```

Ya tenemos la Máquina Virtual lista para poder arrancar con el TP0!

De la misma forma que bajamos el proyecto de las commons, bajemos el del tp0:

```
git clone https://github.com/sisoputnfrba/Game-watch-client
```

Recuerden bien la ubicación donde se está bajando, porque la vamos a necesitar para **configurar Eclipse!**

De la misma manera que las Guest Additions, tenemos [un video para saber como Importar un Proyecto con "makefile"](#).

¿Por qué con makefile? Porque make es una herramienta de gestión de dependencias y compilación que vamos a estar usando en la materia y nos permite distribuirles un proyecto ya armado, sin necesidad de usar el Eclipse de antemano.

Configurado el proyecto en Eclipse, **ya tenemos la máquina virtual preparada** para programar el ejercicio junto con el mismo.

Etapa 2: Comandos básicos

El objetivo de esta etapa es aprender una 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 **tp0.c**, si revisamos [el header de logs de las commons](#), habla de la función **log_create()**, que nos devuelve un logger listo para usar. Usaremos esa función para configurarlo 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 del nivel info para arriba.



Luego, al final del programa, deberemos usar la función **log_destroy()** para liberar la memoria que ocupa nuestro logger.

Archivos de Configuración

Estaría bueno poder parametrizar lo más posible toda información estática que un programa necesita para poder arrancar, como es en nuestro caso la IP y puerto del servidor al que deberemos conectarnos.

Para ello vamos a usar [las config de las commons](#). Usando el link, creamos una config sobre el archivo "tp0.config". Las configs de las commons utilizan el formato de "CLAVE=VALOR" sin comillas, así que deberemos crear el archivo y agregar lo siguiente:

- IP=127.0.0.1
- PUERTO=4444

Nota: no se olviden de destruir el config al final del programa!

Etapa 3: Codear el Cliente

A partir de esta etapa, vamos a plantear una arquitectura [cliente-servidor](#). El servidor, **ya lo tenemos hecho** en el repositorio <https://github.com/sisoputnfrba/Game-watch-server>, solo falta compilarlo con **make** y ejecutarlo en otra consola como `~/Game-watch-server/Server`. Por lo que solo nos falta adaptar nuestro **tp0.c** para que se convierta en el Cliente que se conectará al Servidor.

El entregable de esta etapa es enviar al servidor strings que este devolverá sin modificar, y luego recibirlas y loguearlas.

Para simplificar el ejercicio, 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. Algunas de ellas sin embargo es parte de este ejercicio realizar su desarrollo **sin modificar las firmas de las funciones y utilizando los struct definidos en dicho .h**:

- **crear_conexion(2)**: Nos devuelve el socket que representa la conexión.
- **enviar_mensaje(2)**: Recibe el socket y los datos a mandar.
- **liberar_conexion(1)**: Termina la conexión y libera los recursos que se usaron para gestionar la misma.
- **recibir_mensaje(1)**: Recibe el socket por el que el servidor devolverá el string que le enviamos, y lo retorna

Es importante recalcar, que hay que enviar un string (y no un stream). La diferencia entre ambos radica en que, si bien ambos son una seguidilla de "bytes", **el string termina en el carácter \0**.

Dado que nuestra función recibe un string, es necesario incluir este carácter (denominado centinela) al calcular el tamaño del string a mandar. Esto a que viene? Si consultan la documentación de una función amiga llamada `strlen` verán que **nos devuelve el tamaño de un string, sin contar el carácter centinela, por lo que hay que "sumarle 1"**.



Sin embargo, a la hora “enviar mensaje” lamentablemente no resulta siempre tan sencillo como realizar el send o el recv de un struct. Para ayudarlos con el desarrollo de las funciones de envío de mensajes es que redactamos para este ejercicio [el siguiente documento](#) con una explicación detallado de cómo **serializar un mensaje**.

Notas Finales

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 y, ¡liberarla!
- 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 que les pedimos que hagan uso de los links de [Links Útiles](#), del [redireccionario](#), de las [Guías](#) y los [Videos Tutoriales](#) para mejorar constantemente y llegar bien holgados a fin de cuatrimestre.