# Introducción

Miguel Ángel Solinas 11 mar 2022 (Última modificación: 11 mar 2022)

Los sistemas compuestos por hardware y software utilizan arquitecturas de capas para desarrollar aplicaciones complejas. En las capas superiores se trabaja con lenguajes de alto nivel, mas "amigables" con el programador. En la capa inferior, mas baja, siempre está el hardware puro y duro. Inmediatamente encima está la capa de lenguaje de bajo nivel, podríamos decir más amigable con el hardware.

Recordemos que los lenguajes de bajo nivel están entre uno de los primeros intentos de la humanidad de despegar de la programación directa en lenguaje de máquina. Así el "ensamblador" es un lenguaje propio de la arquitectura y un intento de construir un lenguaje más accesible con el programador.

Los lenguajes de alto nivel, para controlar el hardware y su interacción con los sistemas físicos que lo rodean, necesitan acceder al hardware a través de los lenguajes de bajo nivel. Para ello utilizan convenciones de llamadas.

Entender cómo funciona una convención de llamada nos acercará a un conocimiento de sumo interés para áreas de desarrollo de sistemas críticos, seguridad y también para profundizar sobre el conocimiento de la interacción entre software y hardware.

# Enunciado y condiciones de aprobación

Para aprobar el TP#1 se debe diseñar e implementar una Calculadora de Cotización de Criptomonedas. La capa superior recuperará la cotización de al menos dos criptomonedas que pueden ser obtenidas de alguna de "Las 12 mejores API del mercado de valores para crear productos financieros". Se recomienda el uso de API Rest y python. Los datos de consulta realizados deben ser entregados a un programa en C que convocará rutinas en ensamblador para que hagan los cálculos de conversión y devuelvan las cotizaciones en pesos, u$s y euros. Luego el programa en C o python mostrará los cálculos obtenidos.-

Se debe utilizar el stack para convocar, enviar parámetros y devolver resultados. O sea utilizar las convenciones de llamadas de lenguajes de alto nivel a bajo nivel.-

La defensa del trabajo es GRUPAL (mínimo 2 estudiantes, máximo 3).

Las presentaciones de los trabajos se realizarán utilizando GitHub (una cuenta privada). Cada grupo debe asignar un responsable (con email institucional) como usuario de la cuenta de GitHub. Se debe realizar un commit brevemente comentado con cada funcionalidad implementada y validada.-

# Preparación

En este trabajo haremos una inmersión en los detalles de la interacción entre lenguajes de alto nivel ( C ) y lenguajes de bajo nivel ( Assembler ).

Trabajaremos sobre una arquitectura X86 (la que revisamos en el Teórico). Sobre ella tendremos que disponer de un Sistema Operativo Linux.

Ambiente de trabajo:

Una arquitectura X86SO ( Probado con Ubuntu 16 32 bits)

Visual Studio Code, Sublime Text (Instalar Package Control para "view" sintaxis de & ) o cualquier otro editor.

Netwide Assembler: Ensamblador libre para X86.

Linkeditor, generalmente viene instalado por default en el SO.

Compilador de lenguaje C, también viene instalado por default

Depurador (sudo apt install build-essential)

La propuesta es que antes de la primer clase presencial realice las siguientes tareas:

Leer los Capítulos 1 a 4 de la Bibliografía (pcasm-book-spanish.pdf).

Compilar, ejecutar y revisar los programas ejemplos de dichos Capítulos.

Ejecutar y depurar los programas del Cap 4 en el ambiente de trabajo.

# Que es una api REST

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. El informático Roy Fielding es el creador de la transferencia de estado representacional (REST).

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones.Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.

En otras palabras, las API le permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla.

Ejemplo

<http://api.open-notify.org/iss-now.json>

## Cómo consumir una api rest en C

la libreria mas popular es libcurl

<https://curl.se/libcurl/c/simple.html>

sudo apt install libcurl4-openssl-dev

jjorge@pluma:~$ gcc simple.c -lcurl -o simpleget

jjorge@pluma:~$ ./simpleget

## Cómo consumir una api rest en python

pip install requests

import requests

res = requests.get('https://scotch.io')

if res:

print('Response OK')

else:

print('Response Failed')

print(res.status\_code)

print(res)

print(res.text)



## Como llamar código de c en python

<https://es.stackoverflow.com/questions/102982/puedo-insertar-c%C3%B3digo-c-en-python>

Tienes muy buenos manuales de ambos métodos en la documentación estándar, para completar la respuesta voy a dejar dos ejemplos simplificados de como usar una función muy simple de C en Python.

**Nota:** En el ejemplo se va a usar un entorno Linux con gcc 7.2.0 como compilador y Python 3.6.

Imaginemos nuestra función de C para calcular el factorial, algo como:

unsigned long long \_factorial(int n) {

unsigned long long result = 1;

for (int i = 1; i <= n; ++i)

result \*= i;

return result;

}

Bien, vamos a ver como extender Python con este código:

**Usando ctypes:**

1. **Creamos un archivo factorial.c** que contiene nuestra función.
2. **Compilamos nuestro código** para crear una librería de enlace dinámico (situándonos en el directorio del archivo fuente):

 $ gcc -shared -W -o libfactorial.so factorial.o

Esto nos crea un fichero *shared object* llamado libfactorial.so.

1. Ahora que hemos terminado con C vamos a **crear un wrapper con ctypes** para poder usar la librería de C con Python y poder llamar sus funciones:

# Importamos la librería ctypes

import ctypes

# Cargamos la libreria

libfactorial = ctypes.CDLL('./libfactorial.so')

# Definimos los tipos de los argumentos de la función factorial

libfactorial.factorial.argtypes = (ctypes.c\_int,)

# Definimos el tipo del retorno de la función factorial

libfactorial.factorial.restype = ctypes.c\_ulonglong

# Creamos nuestra función factorial en Python

# hace de Wrapper para llamar a la función de C

def factorial(num):

return libfactorial.factorial(num)

