# 66:20 Organización de Computadoras
# Trabajo práctico 2: Profiling y optimización

## 1 Objetivos

Familiarizarse con las técnicas y herramientas de profiling y optimización de software, implementando y optimizando un programa que busca descubrir la contraseñas hasheadas en un archivo.

## 2 Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

## 3 Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 9), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido. Por este motivo, el día de la entrega deben concurrir todos los integrantes del grupo.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

## 4 Recursos

Utilizaremos la herramienta `gprof` [2] para evaluar la performance de las diversas áreas del programa y ver en cuáles se está empleando más tiempo.

## 5 Introducción

En los sistemas operativos tipo UNIX (Linux, *BSD, Minix, etc) una manera de manejar la autenticación consiste en almacenar un *hash* de la contraseña

de cada usuario en un archivo (típicamente `/etc/passwd` o `/etc/shadow`). Para autenticar a un usuario, el sistema calcula el hash de la contraseña dada por el usuario y lo compara con el que tiene almacenado. De esta manera, se evita guardar las contraseñas en texto plano. Esto, sin embargo, sigue teniendo riesgos: un usuario que consiga el archivo de contraseñas tendría potencialmente la manera de conseguirlas a todas, dado el suficiente tiempo. [1]

En este trabajo, vamos a suponer que hemos conseguido algunos archivos con contraseñas y queremos descubrir las contraseñas originales. Sabemos que el método usado para calcular el hash es MD5 [1], de manera que el método a seguir será ir probando cada combinación posible, comparando su hash MD5 con el de la contraseña que queremos descifrar. Sabemos que el primer archivo es de un sistema donde la gente sólo usa passwords de una letra, el segundo es de passwords de dos letras, el tercero es de passwords de tres letras, y el cuarto es de passwords de una letra, pero a los que se les agrega una 'sal' de dos caracteres, cuyo valor desconocemos, antes de calcular el hash. Se provee también una biblioteca que calcula el hash MD5 de un string dado, y un ejemplo de uso.

El trabajo a realizar es, entonces:

- Descubrir las contraseñas cifradas de los cuatro archivos, con el método propuesto.
- Buscar una manera (o más) de optimizar la tarea.

## 6 Programa

Se trata de escribir un programa que tome como entrada el nombre de un archivo y la longitud de las contraseñas que están hasheadas en él, encuentre qué contraseñas son buscando un string que dé el mismo hash MD5, y devuelva los valores de las contraseñas encontradas.

## 7 Archivos anexos

Los archivos provistos son:

- `md5.c, md5.h` : Implementación de MD5 por SolarDesigner
- `get_sum.c` : Función de ejemplo de uso de md5.c
- `getter.c` : Un main simple que toma un string como argumento y devuelve el hash MD5

[1] Esta es la razón de la existencia del archivo /etc/shadow: /etc/passwd tiene campos que deben ser legibles para todos, así que las contraseñas fueron mudadas a otro con más restricciones de acceso.

- `passwd_1` : Lista de hashes de once contraseñas de longitud 1
- `passwd_2` : Lista de hashes de once contraseñas de longitud 2
- `passwd_3` : Lista de hashes de once contraseñas de longitud 3
- `passwd_4` : Lista de hashes de once contraseñas de longitud 1 con sal de dos caracteres

# 8 Mediciones

## 8.1 Profiling

Si bien la medida más simple y más definitiva de evaluar las mejoras en desempeño de un programa es el tiempo de ejecución, es difícil saber cómo optimizarlo si no tenemos claro cómo se distribuye el tiempo de ejecución entre las diferentes partes de éste. Una manera recomendada de proceder a la optimización de un programa consiste en ver en qué emplea más tiempo, y si esa sección de programa se puede hacer más eficiente. Por eso será necesario entonces detectar y documentar los principales cuellos de botella. Para esto, usaremos la herramienta gprof [2].

## 8.2 Documentación

Es necesario que el informe incluya una descripción detallada de las técnicas y procesos de medición empleados, y de todos los pasos involucrados en el mismo, ya que forman parte de los objetivos principales del trabajo.

# 9 Informe

El informe deberá incluir:

- Este enunciado;
- Análisis de la performance del programa tal como está planteado, para los casos presentados.
- Las cuatro listas de contraseñas recuperadas por el programa.
- Optimizaciones planteadas y análisis de los cambios en performance.
- El código fuente completo del programa en sus dos versiones, en dos formatos: digital[2] e impreso en papel.

[2]No usar diskettes: son propensos a fallar, y no todas las máquinas que vamos a usar en la corrección tienen lectora. En todo caso, consultá con tu ayudante.

## 10 Fecha de entrega

- Primera entrega: Jueves 12 de Junio de 2014.
- Devolución: 19 de Junio.
- Última entrega: 26 de Junio.

## References

[1] `http://es.wikipedia.org/wiki/MD5`.

[2] GNU profiler, `http://sourceware.org/binutils/`.