# Tarea 1

Implementación de algoritmos paralelos

Publicada el 16 de agosto de 2018

**Fecha de entrega (parte 1):** viernes 31 de agosto de 2018, 23:59 hrs., en Git.
**Código base:** Ingresar a GitHub Classroom en `https://classroom.github.com/g/rSIYA8od`

## 1. Objetivo

En la actualidad los sistemas multiprocesador constituyen la mayor parte de la oferta de computadores personales y dispositivos móviles disponibles en el mercado, algo probablemente impensado hace un par de décadas por la mayoría de los ingenieros y desarrolladores de software. Sin embargo, en un primer curso de estructuras de datos y algoritmos en una carrera como la nuestra, se mantiene la tradición de enseñar algoritmos seriales que no aprovechan las ventajas de contar con sistemas multiprocesador. En esta tarea, esperamos despertar tu curiosidad para que investigues maneras de implementar – e implementes – versiones paralelas de algoritmos tradicionalmente estudiados en su forma serial en el pregrado de ingeniería. Para esto, aplicarás los conceptos y herramientas de concurrencia vistas en el curso, y comunicación inter-procesos.

El objetivo de esta primera tarea implementar versiones paralelas de los algoritmos de búsqueda binaria (*Binsearch*[1]) y *quicksort*[2], y realizar experimentos comparando sus rendimientos con las versiones seriales de éstos. Las comparaciones entre algoritmos seriales y paralelos deberán realizarse utilizando varios conjuntos de datos, los cuales debe ser generados para cada nuevo experimento. La métrica a comparar será el tiempo transcurrido desde el inicio hasta el término de la ejecución de cada algoritmo.

La implementación de versiones paralelas de los algoritmos requerirá el uso de POSIX *threads* (biblioteca `pthreads`).

**La primera parte de la tarea (50 % de la nota) consistirá en comparar versiones serial y paralela del algoritmo de búsqueda binaria**. La segunda parte consistirá en comparar las versiones serial y paralela de *quicksort*. El presente enunciado sólo contiene los requerimientos de la primera parte de la tarea.

## 2. Prerequisitos

Para esta tarea es importante repasar programación en C, incluyendo entrada y salida en consola, control de flujo, arreglos, strings, punteros y manejo de memoria. Para esto, puedes revisar tus

[1] Ver `https://en.wikipedia.org/wiki/Binary_search_algorithm`
[2] Ver `https://en.wikipedia.org/wiki/Quicksort`

apuntes del curso anterior de Programación de Bajo Nivel, y algún libro como "*Essential C*". El capítulo 14 del libro de Arpaci-Dusseau describe el funcionamiento de la API de memoria en C, el capítulo 27 la API de `threads`, y el 28 algunas primitivas de sincronización útiles.

Además, es indispensable contar con un ambiente de programación que incluya las herramientas de desarrollo GNU, como `gcc` (compilador de C), `valgrind` (análisis de memoria), `gdb` (depurador), `make` (scripts para compilación), y también `Git` para versionar adecuadamente tu código. El sistema operativo oficial para las tareas del curso será Ubuntu 16.04 LTS `http://www.ubuntu.com/download/desktop`. Puedes instalar Ubuntu rápidamente en tu computador como una máquina virtual con productos como Virtualbox, VMWare o Parallels. Sin embargo, es factible desarrollar la tarea en otros ambientes, como en Cygwin y en Mac OS X, mientras el código finalmente compile correctamente en Ubuntu 16.04 LTS.

## 3. Descripción y Requisitos

La tarea aquí pedida consistirá en desarrollar un programa que implemente una versión paralela del algoritmo de búsqueda binaria, y realizar un conjunto de experimentos que permita comparar el rendimiento empírico del algoritmo con la versión serial.

El diseño del algoritmo paralelo es parte de la tarea. Puedes investigar en la web descripciones teóricas de este algoritmo y decidir cuál implementar. Sin embargo, cuando implementes el algoritmo, deberás tener cuidado con verificar y limitar si fuese necesario la cantidad de *threads* que utiliza, dado que el paralelismo sólo aprovecha los núcleos de CPU efectivamente disponibles en el sistema.

Para realizar los experimentos, será necesario correr las versiones serial y paralela utilizando un mismo conjunto de datos, y comparar los tiempos obtenidos.

El programa principal, llamado `binsearch`, debe recibir como argumentos por línea de comando (¡usar `getopt` para esto![3]) un valor $E$ ($1 \leq E$) de experimentos (corridas) del algoritmo serial y del paralelo, un valor $T$ ($3 \leq T \leq 9$), que indica la potencia de 10 correspondiente al tamaño del conjunto de datos a generar para cada experimento (el tamaño $10^T$ se debe mantener para todos los experimentos en la misma ejecución de `binsearch`), y la posición $P$ ($0 \leq P \leq 10^T - 1$) del dato a buscar en el conjunto de datos. Con el valor de $P$ se garantizará que el algoritmo de búsqueda binaria siempre encuentre el dato buscado y se puede tener cierto control sobre la cantidad de pasos requeridos por el algoritmo para encontrar el dato buscado.

Para generar los datos para cada experimento, el proceso `binsearch` deberá comunicarse a través de un *domain socket* de Unix con otro proceso llamado `datagen` que le proveerá los datos. Un nuevo proceso para `datagen` debe ser iniciado por `binsearch` cada vez que ejecute. Además `binsearch` debe encargarse de terminar `datagen`, y esperar su terminación para cerrarse. Se provee la implementación del proceso `datagen`, el cual escucha conexiones en el socket, y atiende peticiones del proceso `binsearch`. Las comunicaciones entre `binsearch` y `datagen` se describen en la sección

[3] Ver `https://www.gnu.org/software/libc/manual/html_node/Example-of-Getopt.html#Example-of-Getopt`

siguiente.

# 4. Comunicación Inter-Procesos

El proceso `binsearch` deberá iniciar el proceso `datagen`, y luego comunicarse con éste a través de un Unix *domain socket* en `/tmp/dg.sock`. El proceso datagen acepta una conexión desde `binsearch`, y recibe sus instrucciones a través del socket. Las instrucciones son los siguientes mensajes (strings) con texto ASCII:

- `BEGIN S <T>`: El valor `<T>` corresponde al recibido por `binsearch` desde la línea de comando. El proceso `datagen` responderá con un string con el texto `OK\n\n`, y luego comenzará a generar un conjunto de enteros sin signo de 32 bits ordenados de menor a mayor, de tamaño $10^T$ que el proceso `binsearch` podrá leer desde su socket cliente. Notar que la lectura desde el socket se debe realizar utilizando un arreglo de bytes (de unos 1000 bytes sería recomendable), y desde allí se debe extraer el mensaje `OK\n\n`, y luego los enteros de 4 bytes sin signo.
- `END`: Este mensaje es enviado por `binsearch` a `datagen` para terminarlo.

# 5. Reporte de resultados

## 5.1. Implementación de la salida en `binsearch`

El programa `binsearch` debe presentar los resultados obtenidos luego de cada ejecución en una lista de valores separados por coma, pensando en que con la salida estándar del programa se pueda generar un archivo de tipo CSV, que luego pueda ser utilizado para análisis de datos con MS Excel, Python o R.

El formato de salida debe incluir $E + 1$ líneas; la primera línea debe contener rótulos para los encabezados de columna, los cuales son: $E$ (entero), $T$ (entero), `TIEMPO_SERIAL` (tiempo tomado por el algoritmo serial en milisegundos, usar precisión decimal doble para esto), `TIEMPO_PARALELO` (lo análogo para el algoritmo paralelo):

```
E,T,TIEMPO_SERIAL,TIEMPO_PARALELO
```

A partir de la segunda línea deben aparecer los resultados de los $E$ experimentos en el mismo orden que la primera línea de encabezado, separados por coma.

Recuerda que puedes redirigir la salida estándar a archivo utilizando el operador `>` en la consola. Ejemplo:

```
$./binsearch -T 8 -E 1000 -P 5000 > outputT9E1000P5000.csv
```

### 5.2. Informe con análisis

Junto con el código de su tarea, debe entregar un informe escrito que debe contener las siguientes secciones:

- Descripción del Algoritmo: Descripción del algoritmo de búsqueda binaria paralela implementada. Usar pseudocódigo y/o diagramas de flujo (con flujo serial y paralelo) para describirlo.
- Análisis de Resultados: Comparación de los tiempos medios de ejecución de los algoritmos serial y paralelo variando $T$ (4, 6 y 8; usar 9 sólo si resultara práctico en el harware utilizado) y $P$ (posiciones al inicio, medio y final del conjunto de valores) y manteniendo $E$ constante en 100. Para esto puede presentar un gráficos de línea, en donde aparezca el valor de $T$ en la abscisa, y Tiempo en ms en la ordenada, para cada valor de $P$. Serían en total nueve gráficos, o doce si resulta práctico el caso de probar $T = 9$.
- Conclusiones: Alguna discusión sobre los resultados obtenidos. ¿Mejora el rendimiento paralelizar el algoritmo serial?
- Problemas encontrados y limitaciones: Problemas encontrados en la implementación del programa y detalles que el programa no implemente. En particular, si el programa se cae en algún punto de la ejecución, describir dónde está la mayor sospecha del problema que causa la caída.

El informe debe encontrarse en un archivo de texto (o formato *markdown*) disponible en el directorio raíz del repositorio de código. Además, **el informe debe contener los nombres de los integrantes del grupo**.

## 6. Consideraciones Técnicas

El sistema operativo oficial para desarrollar esta y la siguientes tareas del curso es Ubuntu Linux 16.04 LTS. Si se ejecuta en una máquina virtual, se debe tener cuidado de asignar todos los núcleos de CPU disponibles al sistema operativo huesped (*guest*), a fin de que el algoritmo paralelo implementado utice todos los núcleos disponibles.

Para desarrollar esta tarea necesitarás usar funciones de la librería estándar de C, y también funciones de la interfaz de programación de llamadas de sistema[4]. Algunas de las llamadas de sistema y funciones relevantes de la biblioteca estándar de C a utilizar podrían ser las siguientes (revisar la documentación de cada una):

- `read`, `write`: operaciones para leer y escribir desde/a un socket.
- `pthread_create`, `pthread_join`, `pthread_broadcast`: biblioteca de *threads* POSIX.

[4] *POSIX system calls API*, hay una buena lista en `http://www.tutorialspoint.com/unix_system_calls/index.htm`

- `fork`, `wait`, `getpid`, `pipe`: creación y administración de procesos y *sockets*.
- `malloc`, `realloc`, `free`: manejo de memoria dinámica.
- `strnlen`, `strncmp`, `strncpy`, `snprintf`: manejo de *strings*.
- `perror`, `strerror`: manejo de errores.

# 7. Evaluación

## 7.1. Inspección de código y funcionamiento (50 %)

Se revisará el código de la tarea en forma exhaustiva y se realizarán pruebas de funcionamiento con los distintos parámetros involucrados, a fin de verificar que el programa desarrollado permite obtener los resultados reportados.

## 7.2. Manejo de errores (10 %)

Debe existir código con adecuado manejo de errores en situaciones como las siguientes:

- Error al crear socket.
- Invocación a `fork` retorna `-1`.
- Error al iniciar *thread*
- Invocación a `malloc` retorna `NULL`.

## 7.3. Manejo de memoria (15 %)

Es importante que toda la memoria pedida por el programa al sistema operativo a través de invocaciones a funciones como `malloc` y `realloc` sea correctamente liberada llamando a `free`, y que además, todo el acceso a la memoria a través de punteros sea implementado en forma correcta.

## 7.4. Informe (25 %)

La tarea debe incluir el informe descrito anteriormente en un archivo de texto de nombre `README`, ubicado en el directorio raíz del código de la tarea.

## 8. Modalidad de Trabajo

La tarea debe ser desarrollada en grupos de 2 (parejas). Cada integrante debe contar con una cuenta de usuario en GitHub. El desarrollo debe realizarse por ambos integrantes, y deben quedar claramente registradas sus operaciones de *commit* en el sistema, con comentarios descriptivos en cada una de estas operaciones.

El profesor publicará en Canvas el medio a través del cual se deben inscribir los grupos e informar las cuentas de usuario en GitHub.

Se proporcionará junto con este enunciado un código base con algún trabajo adelantado. Para compilar la tarea basta invocar `make` en el directorio en donde se encuentra el código. El nombre del programa ejecutable es `bank`.

## 9. Entrega

La tarea debe compilar; en caso contrario, la nota máxima será un 3,9 – apelable – si el error de compilación es un detalle mínimo.

El lenguaje C tiene una serie de dialectos y extensiones según la versión de las herramientas que se esté usando para desarrollar. Para evitar confusiones, la evaluación será hecha compilando con `gcc` (la versión incluída en la distribución de Ubuntu de referencia), usando los parámetros `-std=c11 -Wall -Werror`.

La entrega de la tarea debe hacerse hasta las 23:59 hrs. en el repositorio Git del grupo. Para hacer efectiva esta restricción, se evaluará la última revisión del repositorio del grupo previa a la hora de cierre. Si un grupo hace su primer *commit* y/o *push* después de la hora de cierre, tendrá derecho a nota máxima 4,0.