# Sistemas Operativos 1/2023
# Laboratorio 2

Profesores:
Cristóbal Acosta (cristobal.acosta@usach.cl)
Fernando Rannou (fernando.rannou@usach.cl)
Ayudantes:
Ricardo Hasbun (ricardo.hasbun@usach.cl)
Leo Vergara (leo.vergara@usach.cl)

## I. Objetivos Generales

Este laboratorio tiene como objetivo aplicar técnicas de programación imperativa mediante lenguaje C, como la recepción de parámetros mediante getopt y compilación mediante Makefile. Además de aplicar conocimiento adquiridos en cátedra sobre la creación de procesos y la comunicación entre ellos, en un sistema operativo Linux.

## II. Objetivos Específicos

1. Usar las funcionalidades de `getopt()` como método de recepción de parámetros de entradas.
2. Uso del Makefile para compilación por partes de programas.
3. Crear procesos utilizando la función `fork()`.
4. Utilizar `pipe()` para la comunicación entre procesos.
5. Hacer uso de la familia `exec()` para ejecutar procesos.
6. Duplicar descriptores con `dup()` o `dup2()`.
7. Construir funciones de lectura y escritura de archivos.
8. Practicar técnicas de documentación de programas.

# III. Enunciado

## III.A. Expresiones Regulares

Estás expresiones es otro tipo de notación para la definición de lenguajes. También puede pensarse en las expresiones regulares como un "Lenguaje de programación", en el que es posible escribir algunas aplicaciones importantes, como por ejemplo aplicaciones de búsqueda de texto o componentes de compilador. Las expresiones regulares están estrechamente relacionadas con los autómatas finitos no deterministas y pueden considerarse una alternativa, que el usuario puede comprender fácilmente, a la notación de los AFN (autómata finito no determinista) para describir componentes de software.

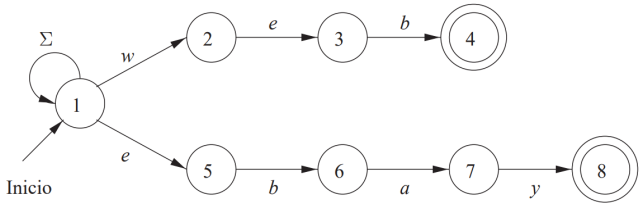


**Figure 1. Ejemplo de un AFN que busca las palabras "web" y "ebay".**

Las expresiones regulares ofrecen algo que los autómatas no proporcionan: una forma declarativa para expresar las cadenas que deseamos aceptar. Por tanto, las expresiones regulares sirven como lenguaje de entrada de muchos sistemas que procesan cadenas. Como por ejemplo: Generadores de analizadores léxicos, como "Lex" o "Flex". Recordando que un analizador léxico es el componente de un compilador que divide el programa fuente en unidades lógicas o sintácticas formadas por uno o más caracteres que tienen un significado.

# IV. El reconocedor

La expresión regular a reconocer es:

$$(A + C + G + T)^* G T^+ C (A + C + G + T)^*$$

Donde:

1. + significa: un "or" lógico
2. $X^*$ significa: cero o más de los caracteres.
3. $X^+$ significa: uno o más de los caracteres.

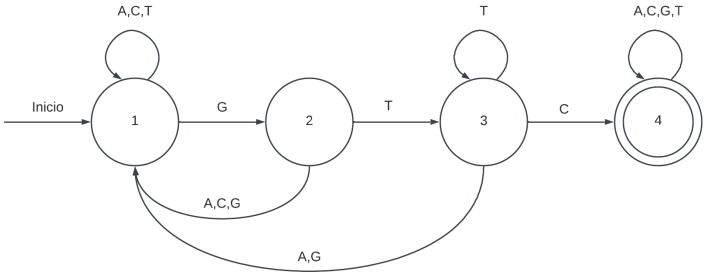


**Figure 2. Ejemplo del AFN que busca la expresión regular dada.**

# V. Cálculos necesarios

El siguiente es una ejemplo de salida:

```
GATCTTATATAACTGTGAGATTAATCTCAGATAATGACACAAAATATAGTGAAGTTGGTA no
AGTTATTTAGTAAAGCTCATGAAAATTGTGCCCTCCATTCCCATATAATTTATTAATTGT no
CTAGGAACTTCCACATACATTGCCTCAATTTATCTTTCAACAACTTGTGTGTTATATTTT no
GGAATACAGATACAAAGTTATTATGCTTTCAAAATATTCTTTTGCTAATTCTTAGAACAA no
AGAAAGGCATAAATATATTAGTATTTGTGTACATCTGTTCCTTCCTGTGTGACCCTAAGT si
TTAGTAGAAGAAAGGAGAGAAAATATAGCCTAGCTTATAAATTTAAAAAAAAATTTATTT no
GGTCCATTTTGTGAAAAACATAAAAAAAGAACTGTCACATCTTAATTTAAAAAATATATG si
CTTAGTGGTAAGGAGATATATGTCAACTTTTAAGAGGTTGAAAAACAAACGCCTCCCATT si
ATAAGTTTATACTTCACCTCCCACCACTATAACAACCCAGAATCCATGAGGGCATTATCA no

Total de expresiones que Si son regulares:3
Total de expresiones que No son regulares:6
Total de lineas leídas:9
```

### V.A. Cálculos necesarios

Los cálculos necesarios para el desarrollo de este laboratorio son los siguientes:

(a) Obtener la cantidad de expresiones que Si son regulares.

(b) Obtener la cantidad de expresiones que No son regulares.

(c) Obtener la cantidad total de lineas leídas.

# VI. El programa del lab

### VI.A. Lógica de solución

En este laboratorio crearemos un "reconocedor", que en base a una expresión regular (ver sección IV) y a un archivo de texto, diga si cada expresión es una expresión regular o no .

1. El proceso principal recibirá como argumentos por línea de comando el nombre del archivo de entrada, el nombre del archivo de salida,la cantidad de workers que serán generados y el número de chunks con el cuál se trabajara.
2. El proceso principal validará los datos entregados por pantalla y, una vez validados, ejecutará el proceso broker con `fork()` y algún miembro de la familia `exec()` entregándole los elementos ya validados.
3. El broker recibirá los argumentos del proceso padre y los utilizara para las tareas necesarias.
4. El broker creará la misma cantidad de workers que la ingresada por pantalla.
5. El broker ejecutara los procesos worker utilizando algún miembro de la familia `exec()` y se comunicará con ellos mediante el uso de `pipes`
6. Cuando no queden líneas por leer, el broker avisará a cada uno de los workers con la palabra "FIN", indicando que no debe esperar más líneas del broker.
7. Cada worker se encargará de obtener y procesar las líneas que entregue el broker.
8. Cuando un worker recibe una línea este se encargará de realizar cada `calculo necesario` para dicha línea los cuales deben ser almacenados en caso de que lo amerite (ej: si la línea es regular o no). La estructura que se use para almacenar estos resultados queda a elección del desarrollador.

9. Cuando un worker lea la palabra clave "FIN" este deberá dejar de esperar líneas del broker y enviar a este último la cantidad de líneas que procesó durante su ejecución. Luego, se finaliza su proceso.

10. Una vez que todos los workers hayan terminado sus cálculos, el broker retomará el control y escribirá en el archivo de salida los resultados siguiendo el formato entregado. Si se utiliza la bandera -b, estos resultados deberán además aparecer en la salida estándar (stdout) del terminal del Sistema Operativo incluyendo con ellos la cantidad de líneas que procesó cada worker (basta con mostrar el PID del worker o con una ID arbitraria asignada al mismo). Con esto, el padre finaliza su ejecución.

11. Tanto proceso padre, proceso broker y proceso worker debe encontrarse finalizado a la hora de terminar el programa.

### VI.B. Implementación de la Lógica

Para implementar esta lógica se deben generar los siguientes programas independientes.

(a) lab2.c: Programa padre, el cual obtendrá los argumentos, los validará y los enviará al proceso broker.

(b) `fbroker.c y fbroker.h/fworker.c y fworker.h`: Archivo con funciones, en el .c el desarrollo de la función y en el .h las cabeceras. Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje.

(c) `broker.c`: archivo con el código del proceso broker. Debe utilizar las funciones de `fbroker` y puede incluir otros archivos fuente. Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje.

(d) `worker.c`: archivo con el código del proceso worker. Debe utilizar las funciones de `fworker` y puede incluir otros archivos fuente. Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje.

### VI.C. Línea de comando

El programa se ejecutará usando los siguientes argumentos (ejemplo):

```
$ ./lab2 -i prueba_10.txt -o salida.txt -c 10 -n 10 -b
```

- **-i**: nombre del archivo de entrada.
- **-o**: nombre de archivo de salida.
- **-n**: cantidad de workers a generar.
- **-c**: tamaño del chunk, es decir, cantidad de líneas a leer por chunk.
- **-b**: bandera o flag que permite indicar si se quiere ver por consola la cantidad de expresiones regulares encontradas por cada proceso worker. Es decir, si se indica el flag, entonces se muestra la salida por consola.

### VI.D. Requerimientos

Como requerimientos no funcionales, se exige lo siguiente:

- Debe funcionar en sistemas operativos con kernel Linux.
- Debe ser implementado en lenguaje de programación C.
- Se debe utilizar un archivo Makefile para compilar los distintos targets.
- Realizar el programa utilizando buenas prácticas, dado que este laboratorio no contiene manual de usuario ni informe, es necesario que todo esté debidamente comentado.
- Los programas se encuentren desacoplados, es decir, que se desarrollen las funciones correspondientes en otro archivo .c para mayor entendimiento de la ejecución.
- La solución debe implementar `fork()`, `excec()`, `dup2()` y `pipe()`. De lo contrario se considerara invalido.

## VII. Entregables

El laboratorio puede ser en parejas o de forma individual. Si se elije una pareja está no podrá ser cambiada durante el semestre es decir se comienza con una pareja o se realiza el siguiente laboratorio de forma individual. Se descontará 1 punto (de nota) por día de atraso con un máximo de tres días, a contar del cuarto se evaluará con nota mínima. Debe subir en un archivo comprimido ZIP (una carpeta) a USACH virtual con los siguientes entregables:

- `Makefile`: Archivo para compilar los programas.
- `lab2.c`: Archivo principal del laboratorio, contiene el main.
- `funciones.c y funciones.h`: Archivo con funciones, en el .c el desarrollo de la función y en el .h las cabeceras. Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje.
- `Otros:` Cualquier otro archivo fuente que se vea necesario para la realización del lab. Pueden ser archivos con funciones, estructuras de datos, etc.
- Trabajos con códigos que hayan sido copiados de un trabajo de otro grupo serán calificados con la nota mínima.

Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje. Se deben comentar todas las funciones de la siguiente forma:

```
// Entradas: explicar qué se recibe
// Salidas: explicar qué se retorna
// Descripción: explicar qué hace
```

El archivo comprimido (al igual que la carpeta) debe llamarse: RUTESTUDIANTE1_RUTESTUDIANTE2.zip
Ejemplo: 19689333k_186593220.zip
NOTA 1: El archivo debe ser subido a uvirtual en el apartado "Entrega Lab2".
NOTA 2: Cualquier diferencia en el formato del laboratorio que es entregado en este documento, significara un descuento de puntos.
NOTA 3: SOLO UN ESTUDIANTE DEBE SUBIR EL LABORATORIO.
NOTA 4: En caso de solicitar corrección del lab está será en los computadores del Diinf, es decir, si funciona en eso computadores no hay problema.

## VIII. Fecha de entrega

Jueves 22 de Junio, 2023.