

Sistemas Operativos 2/2020

Laboratorio 1

Profesores:

Cristóbal Acosta (cristobal.acosta@usach.cl)

Fernando Rannou (fernando.rannou@usach.cl)

Ayudantes:

Benjamín Muñoz (benjamin.munoz.t@usach.cl)

Marcela Rivera (marcela.rivera.c@usach.cl)

I. Objetivos Generales

Este laboratorio tiene como objetivo aplicar los conceptos de creación de procesos y envío de señales a través del uso de la función `exec`, mediante el soporte de un sistema operativo basado en el núcleo Linux y el lenguaje de programación C.

II. Objetivos Específicos

- Recibir como parámetros de entradas: nombre del archivo de entrada, número de procesos a crear, cantidad de caracteres en una línea del archivo, cadena a buscar y finalmente una bandera que indica si se deben mostrar los resultados por pantalla.
- Dentro de la función `main`, crear `n` procesos hijos. Éste valor corresponde al entero que se debe recibir como parámetro de entrada.
- Cada proceso deberá ejecutar un nuevo programa, el cual debe recibir como parámetro una porción del archivo de texto, para luego verificar si en cada línea de texto se encuentra la cadena de caracteres recibido como parámetro de entrada.
- Escribir en un archivo de texto la respuesta final. Es decir, indicar si en cada línea se encuentra o no la cadena.

III. Conceptos

III.A. Funciones `exec`

La familia de funciones `exec()` reemplaza la imagen de proceso actual con una nueva imagen de proceso. A continuación se detallan los parámetros y el uso de dichas funciones:

- `int execl(const char *path, const char *arg, ...);`
 - Se le debe entregar el nombre y ruta del fichero ejecutable
 - Ejemplo:
`execl("/bin/ls", "/bin/ls", "-l", (const char *)NULL);`
- `int execlp(const char *file, const char *arg, ..., (const char *)NULL);`
 - Se le debe entregar el nombre del fichero ejecutable (implementa búsqueda del programa).

- Ejemplo:
`execlp("ls", "ls", "-l", (const char *)NULL);`
- **int execl(const char *path, const char *arg,..., char * const envp[]);**
 - Se le debe entregar el nombre y ruta del fichero ejecutable, recibe además un arreglo de strings con las variables de entorno
 - Ejemplo:
`char *env[] = {"PATH=/bin", (const char *)NULL};
execl("ls", "ls", "-l", (const char *)NULL, char *env[]);`
- **int execlv(const char *path, char *const argv[]);**
 - Se le debe entregar el nombre y ruta del fichero ejecutable, recibe además un arreglo de strings con los argumentos del programa
 - Ejemplo:
`char *argv[] = {"bin/ls", "-l", (const char *)NULL};
execlv("ls", argv);`
- **int execlvp(const char *file, char *const argv[]);**
 - Se le debe entregar el nombre del fichero ejecutable (implementa búsqueda del programa), recibe además un arreglo de strings con los argumentos del programa
 - Ejemplo:
`char *argv[] = {"ls", "-l", (const char *)NULL};
execlvp("ls", argv);`
- **int execlvpe(const char *file, char *const argv[], char *const envp[]);**
 - Se le debe entregar el nombre del fichero ejecutable (implementa búsqueda del programa), recibe además un arreglo de strings con los argumentos del programa y un arreglo de strings con las variables de entorno
 - Ejemplo:
`char *env[] = {"PATH=/bin", (const char *)NULL};
char *argv[] = {"ls", "-l", (const char *)NULL};
execlvpe("ls", argv, env);`

Tomar en consideración que por convención se utiliza como primer argumento el nombre del archivo ejecutable y además **todos** los arreglos de *string* deben tener un puntero *NULL* al final, esto le indica al computador que debe dejar de buscar elementos.

Otra cosa importante es que la definición de estas funciones vienen incluidas en la biblioteca **unistd.h**, que algunos compiladores la incluyen por defecto, pero en ciertos sistemas operativos deben ser incluidas explícitamente al comienzo del archivo con la sentencia **#include <unistd.h>**.

III.B. Función fork

Esta función crea un proceso nuevo o “proceso hijo” que es exactamente igual que el “proceso padre”. Si `fork()` se ejecuta con éxito devuelve:

Al padre: el PID del proceso hijo creado. Al hijo: el valor 0.

Es decir, la única diferencia entre estos procesos (padre e hijos) es el valor de la variable de identificación PID.

A continuación un ejemplo del uso de `fork()`:

```

1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <stdio.h>
4
5  int main(int argc, char *argv[])
6  {
7      pid_t pid1, pid2;
8      int status1, status2;
9
10     if ( (pid1=fork()) == 0 )
11     { /* hijo */
12         printf("Soy el primer hijo (%d, hijo de %d)\n", getpid(), getppid());
13     }
14     else
15     { /* padre */
16         if ( (pid2=fork()) == 0 )
17         { /* segundo hijo */
18             printf("Soy el segundo hijo (%d, hijo de %d)\n", getpid(), getppid());
19         }
20         else
21         { /* padre */
22             /* Esperamos al primer hijo */
23             waitpid(pid1, &status1, 0);
24             /* Esperamos al segundo hijo */
25             waitpid(pid2, &status2, 0);
26             printf("Soy el padre (%d, hijo de %d)\n", getpid(), getppid());
27         }
28     }
29
30     return 0;
31 }

```

Figure 1. Creando procesos con fork().

IV. El problema

Una secuencia de ADN, secuencia de nucleótidos o secuencia genética es una sucesión de letras que representa la estructura primaria de una molécula real o hipotética de ADN o banda, con la capacidad de transportar información.

Las posibles letras son A, C, G, y T, que simbolizan las cuatro subunidades de nucleótidos de una banda de ADN - adenina, citosina, guanina, timina, que son bases covalentemente ligadas a cadenas fosfóricas. Una sucesión de cualquier número de nucleótidos mayor a cuatro es posible de llamarse una secuencia.

Es decir, se busca encontrar una sucesión de éstos nucleótidos en distintos archivos de texto.

A continuación un ejemplo del archivo de entrada y un ejemplo de sucesión.



```

ejemplo.txt
1  AGAAAGGCATAAATATATTAGTATTTGTGTACATCTGTTCTTCCTGTGTGACCCTAAGT
2  GGTCCATTTTGTGAAAAACATAAAAAAAGAAGTGTACATCTTAATTTAAAAAATATATG
3  CTTAGTGGTAAGGAGATATATGTCAACTTTTAAGAGGTTGAAAAACAAACGCCTCCCATT
4  GGAGTGAGTGGAAGAGTAAGTTTGCCAATGTGAAATGTGCCTTCTAGGTCTAGACATCT
5  TGGCACAAAGTATATTACTTGGATCCATCTATGTCATTTTCCATGGTTAATGTTTAAAC
6  TGATTTTGGGCACATTACGTAGCTTTTCATGAGCTTTAGTTTCTACATTTATAAACAGGAG
7  TTAACAAAGCCAAAAGTTTCAAACCTTTACTTTTTCCCAACATTCTTGTGAAATATGACAC
8  TAAGAAAAAAGGAATCTAGTTTGTCAAAATGTGACTTGAATTAATAGATAAGGAGAGTCA
9  GATGATAAGAGGGTCAAAATTATGTTTATCTTAGGAAAAGTAGAATAGAAAATTTATAAG
10 ACAGTTAAACTTACAGTTCATACATAAGGAGAATCAGTCTTTTTTTTTTTTTTACAGTT

```

Figure 2. Archivo de entrada

En el contenido de este archivo, se debe buscar línea a línea si existe una sucesión que cumpla con la que busco.

Un ejemplo de sucesión: AAAA, TTTA, CATT, etc.

Es decir que en la primera linea busco si se encuentra AAAA, si es verdadero, escribo en un archivo la linea revisada y un SI a su lado, en caso contrario, se escribe NO. Este paso se repite hasta recorrer todo el archivo.

V. Enunciado

Se solicitan dos programas. Uno para realizar la tarea de comparación y el otro para coordinar los procesos y así juntar los resultados.

V.A. Programa comparador

El primer programa debe realizar la tarea de encontrar las coincidencias de las cadenas en una sección de un archivo, para esto se le debe indicar el archivo a ocupar, desde que posición de cursor debe comenzar a leer este archivo, la cadena a encontrar, en cuantas cadenas (lineas) debe hacer la comparación y un numero identificador.

Los resultados de estas comparaciones deben guardarse en un archivo de salida, este archivo debe tener un nombre estandarizado, siguiendo el siguiente patrón:

`rp_<cadena a buscar>_<numero de proceso>.txt`

El prefijo *rp_*, de resultado parcial, junto con el resto del patrón es para que el programa coordinador pueda encontrarlos sin problemas.

V.B. Programa coordinador

El segundo programa es el encargado de coordinar las ejecuciones de los procesos del programa comparador. Es el que calcula e indica cuales lineas debe leer cada uno de los procesos, para esto se le debe indicar el archivo a leer, el numero de procesos y el largo de las cadenas a leer, de este modo al conocer el largo del archivo, el programa coordinador puede calcular la posición del cursor del archivo para pasarlo como argumento a los programas coordinadores.

Finalmente cuando todos los procesos comparadores hayan terminado, el programa coordinador debe reunir los resultados en un solo archivo de salida, para la unión de los archivos. Se recomienda usar la función *fread*, indicándole el tamaño completo del archivo y con *fwrite* escribir esta información al final del archivo de salida de resultados completo.

Del mismo modo que el programa anterior se pide un nombre estandarizado para el archivo de resultados completo, este debe seguir el siguiente patrón:

`rc_<cadena a buscar>.txt`

A continuación un ejemplo del archivo de salida:

Ejemplo del archivo de salida:



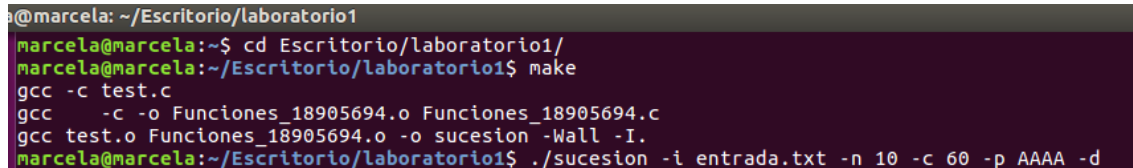
1	AGAAAGGCATAAATATATTAGTATTTGTGTACATCTGTTCTTCCTTGTTGACCCTAAGT	NO
2	GGTCCATTTTGTGAAAAACATAAAAAAGAAGTGTACATCTTAATTTAAAAAATATATG	SI
3	CTTAGTGGTAAGGAGATATATGTCAACTTTTAAGAGGTTGAAAAACAAACGCCTCCCATT	SI
4	GGAGTGAGTGGAAGAGTAAGTTTGCCAAATGTGAAATGTGCCTTCTAGGTCCTAGACATCT	NO
5	TGGCACAAAGTATATTACTTGGATCCATCTATGTCATTTTCCATGGTTAATGTTTAAAC	SI
6	TGATTTTGGGCACATTACGTAGCTTTTCATGAGCTTTAGTTTCTACATTTATAAACAGGAG	NO
7	TTAACAAAGCCAAAAGTTTCAAACCTTTACTTTTCCCAACATTCTTGTAATATGACAC	SI
8	TAAGAAAAAAGGAATCTAGTTTGTCAAATGTGACTTGAATTAATAGATAAGGAGAGTCA	SI
9	GATGATAAGAGGGTCAAATTATGTTTATCTTAGGAAAAGTAGAATAGAAAATTTATAAG	SI
10	ACAGTTAAACTTACAGTTCATACATAAGGAGAATCAGTCTTTTTTTTTTTTTTACAGTT	NO

Figure 3. Archivo de salida

Las entradas del programa coordinador deben ser utilizando getopt, con la siguiente lista de entradas:

- -i: nombre del archivo de entrada
- -n: numero de procesos de tipo comparador
- -c: cantidad de líneas de archivo de entrada.
- -p: cadena a buscar
- -d: bandera que indica si se deben mostrar los resultados por pantalla

Ejemplo:



```

marcela@marcela: ~/Escritorio/laboratorio1
marcela@marcela:~$ cd Escritorio/laboratorio1/
marcela@marcela:~/Escritorio/laboratorio1$ make
gcc -c test.c
gcc -c -o Funciones_18905694.o Funciones_18905694.c
gcc test.o Funciones_18905694.o -o sucesion -Wall -I.
marcela@marcela:~/Escritorio/laboratorio1$ ./sucesion -i entrada.txt -n 10 -c 60 -p AAAA -d

```

Figure 4. Como recibir los parámetros de entrada

V.C. Parámetros de entrada

VI. Entregables

Debe entregarse un archivo comprimido que contenga al menos los siguientes archivos:

1. *Makefile*: archivo para make que compile los programas.
2. dos o mas archivos con el proyecto (*.c, *.h)

El archivo comprimido debe llamarse: RUTESTUDIANTE1_RUTESTUDIANTE2.

Ejemplo: 19689333k_189225326.zip

VII. Fecha de entrega

Domingo 08 de Noviembre hasta las 23:55 hrs.