Práctica Final – Procesamiento de cadenas de DNA

FUNDAMENTOS DE SISTEMAS OPERATIVOS

Curso 2018 - 2019

Resumen

Vamos a crear un sistema concurrente de procesamiento de cadenas de DNA. Estas cadenas de DNA se forman a través de combinaciones de las 4 bases nitrogenadas, Adenina, Tiamina, Citosina, Guanina, (caracteres `A`, `T`, `C`, y `G`), y en este caso su longitud no superará los 100 caracteres. Sin embargo, estas cadenas de DNA normalmente traen más información de la que necesitamos, y hay que procesarlas para quedarnos sólo con la secuencia contenida entre 2 patrones "`AAAAA`" (patrón *primer*).

Ejemplo 1: para la cadena de DNA dada "`ATGD`**`AAAAA`**`CGACTGCTGCTGC`**`AAAAA`**`TTAA`" conseguimos `CGACTGCTGCTGC`

Ejemplo 2: para la cadena de DNA dada "`TTTCCCD`**`AAAAA`**`AAACTTTTC`**`AAAAA`**`AATAA`" conseguimos `AAACTTTTC`

Suponemos que este proceso de limpieza y procesamiento es más costoso que el propio proceso de secuenciación, por lo que necesitaremos un sistema donde haya varios consumidores (cleaners) que trabajen sobre la salida del productor (secuenciador). Por lo tanto, nuestro sistema procesador estará basado en el siguiente esquema de hilos en paralelo:

- Productor (*DNA sequencer/Reader*): Leerá de un fichero de texto plano un número desconocido de cadenas de DNA y simplemente los irá almacenando en forma de string (cadena de caracteres) en un buffer circular (buffer1) compartido con los Consumidores.
- Consumidor/es (*Cleaners*): recoge/n las cadenas de DNA del buffer1, detectan si existen los 2 patrones primer (`AAAAA`) en la cadena, y generan un string con la cadena contenida entre estos 2 patrones. Esta cadena de DNA ya procesada ha de ser almacenada en un segundo buffer circular (buffer2) compartido con el Consumidor Final. Si durante esta fase de limpieza no se encontraran los 2 patrones primer, se enviará al buffer2 la frase "NO PRIMERS". Si en cualquier momento de este procesamiento se detecta un carácter diferente a los 4 válidos (`A`, `T`, `C`, `G`) se enviará al buffer2 la frase "`ALIEN DNA`".
- Consumidor Final (*Writer*): recoge la información almacenada en el buffer2 y la hace persistente en un fichero de texto plano.
- Vamos a dividir la práctica en dos partes de dificultad incremental. Las dos primeras partes valen el 80% de la calificación total de la práctica. Para optar al 20% restante habrá que realizar también la segunda parte. Se aconseja empezar por la primera parte y una vez que se tenga, modificarla para la realización de las siguientes.

Objetivos

- Comprender el funcionamiento de cadenas en C, familiarizarse con las funciones de la librería string.h
- Practicar la lectura y escritura en ficheros
- Practicar la creación y gestión de hilos en UNIX.
- Practicar la gestión de los recursos mediante directivas malloc de reserva de memoria
- Practicar la gestión y control de acceso a memoria compartida mediante semáforos.

Seguimiento y entrega parcial

Se va a realizar un seguimiento de la realización de la práctica, así como varias entregas parciales de la misma, que irán seguidas de una evaluación cruzada entre pares. En otras palabras el alumno tendrá que evaluar y proponer las mejoras que considere necesarias sobre el trabajo realizado sobre sus compañeros y sobre su propio trabajo, de modo que esto le permita completar las carencias de su propio código. La nota de cada parte se obtiene así:

30% por la valoración del profesor del trabajo realizado en la parte.

70% por las aportaciones realizadas por el grupo en la evaluación de otros grupos de laboratorio.

Evidentemente esto exige la anonimización de los archivos, tanto en el nombre como en el contenido. Además, el profesor tiene la potestad de preguntar a cualquier grupo de prácticas en cualquier momento sobre el trabajo y la evaluación realizada. Las fechas de entrega y evaluación son las siguientes:

Fecha límite	Entrega	% sobre la calificación	tipo
3-dic (lunes) - 23:55	**Entrega** código Parte 0	5	
5-dic (miércoles) – 23:55	**Evaluación** pares y **autoevaluación**, Parte 0	10	
10-dic (lunes) – 23:55	**Entrega** código Parte 1	10	OB
13-dic (jueves) – 23:55	**Evaluación** pares y **autoevaluación**, Parte 1	20	OB
17-dic (lunes) – 23:55	**Entrega** código Parte 2	15	OB
21-dic (viernes) – 23:55	**Evaluación** pares y **autoevaluación**, Parte 2	40	OB
		100	

OB: significa actividad obligatoria vía Portal Moodle de la asignatura.

En esos días de entrega se subirá a la plataforma Moodle lo que se tenga realizado. No importa que no funcione, aunque obviamente, se recomienda que sí que funcione.

Entregas

- La entrega se realizará mediante la web de la asignatura.
- Se entregará el código fuente del programa en lenguaje C realizado.
- La entrega será **completamente anónima.** Tu trabajo será evaluado independientemente de quien seas. Todo trabajo que contenga información

personal será **penalizado** con la exclusión del sistema de revisión y deberá ser presentado personalmente frente al profesor.

- Deberá incluir obligatoriamente el código fuente de la aplicación. **Si se han realizado ambas partes habrá que incluir en el envío sólo el código de la versión de la segunda parte.**
- Se podrá incluir cualquier información adicional que se considere que puede ser útil para la evaluación de la práctica.
- Se valorará la claridad del código y la inclusión de comentarios aclaratorios.

Parte 0 (OBLIGATORIA): Consumidor único

La aplicación a realizar tendrá la siguiente sintaxis en su ejecución:

```
./<program> <inputFile> <outputFile> <tamBuffer>
```

Lista de funciones que deberá cumplir la entrega:

- El **programa principal** creará el búfer circular `buffer1`, de tamaño <*tamBuffer*> y posteriormente creará hilos, asignándoles la tarea o función que les corresponda. Finalmente esperará a que estos hilos finalicen su ejecución de manera correcta.
  - El programa principal creará esto hilos lanzando primero el hilo consumidor y posteriormente el hilo productor.
- 1 hilo **Productor**…
  - ✓ leerá cadenas de DNA almacenadas en el fichero de entrada <*inputFile*>, pasado como primer argumento al programa, y que el profesor suministrará a través de Moodle o durante las sesiones de prácticas; y
  - ✓ guardará en el buffer elementos que responden a una *estructura* (*struct*) que contiene un entero y un NTString (de tamaño apropiado) y almacenará esos elementos en el buffer1 que tendrá la capacidad indicada en <*tamBuffer*>.
- 1 hilo **Consumidor**…
  - ✓ recogerá los elementos almacenados en el buffer1, comprobará si existen los 2 patrones primer (`AAAAA`). Si fuera este el caso devolverá una estructura con un campo entero y un string donde figura la cadena contenida entre los mismos;
  - ✓ en caso de no encontrar los primers, devolverá "`NO PRIMERS`" en el campo de texto;
  - ✓ en caso de encontrar un carácter no válido devolverá "`ALIEN DNA`", y no válido es cualquier letra como "`BDEFHIJKLMNOPQRSUVWXYZ`", suponiendo que sólo se reciben mayúsculas;
  - ✓ para aumentar la verosimilitud de la ejecución, se detendrá el programa durante10000 microsegundos por cada símbolo que se procesa (mediante la llamada `usleep()`); y

- ✓ finalmente mostrará por pantalla cada cadena para que podamos ver el efecto en modo animado. En el Anexo III se presenta la función `xprintf()` que puede ser muy útil al respecto.

~~Es especialmente importante que el alumno compruebe la importancia de la relación de tamaños entre el valor <*tamBuffer*> y la talla de `buffer2`. Podrá verse el funcionamiento correcto de la solución basada en semáforos en los distintos casos.~~

Parte 1 (OBLIGATORIA): Añadimos el Consumidor final

La aplicación será una extensión de la Parte 0 donde el programa principal creará, además, un nuevo búfer (`buffer2`) donde se almacenarán los resultados filtrados. Cada elemento será también un registro de la estructura (int, NTString) exactamente igual que en la Parte 0. Este búfer será utilizado para comunicar los resultados de un solo hilo de tipo **Consumidor**, con el hilo **Consumidor Final**.

El programa principal, tras lanzar el hilo **Consumidor Final**, iniciará el hilo **Consumidor** y después el hilo **Productor**. Finalmente esperará a que concluyan todos ellos de la forma correcta.

- `buffer2` tiene tamaño 5, por lo que únicamente puede alojar hasta 5 registros del tipo estructurado indicados en la Parte 0.
- El hilo **Consumidor** funcionará igual que en la Parte 0, pero ahora copiará los registros en el segundo búfer (`buffer2`) sin tener en cuenta el ordenamiento original de la entrada.
- El hilo **Consumidor Final** recuperará los registros de `buffer2` y las almacenará secuencialmente por orden de recuperación a `buffer2` en el archivo <*outFile*>.

Es especialmente importante que el alumno compruebe la importancia de la relación de tamaños entre el valor <tamBuffer> y la talla de buffer2. Podrá verse el funcionamiento correcto de la solución basada en semáforos en los distintos casos.

Parte 2: Varios Consumidores

La aplicación a realizar tendrá ahora la siguiente sintaxis en su ejecución:

```
./<program> <inputFile> <outputFile> <tamBuffer> <numCleaners>
```

En este caso, en lugar de un *único* hilo **Consumidor** filtrando cadenas de DNA se deberán lanzar <*numCleaners*> Consumidores. Resultará especialmente interesante que se observe el cambio en tiempo en función del tamaño del archivo y el número de 'cleaners'.

Anexo I – Ejemplos de archivos

Se proveerá al alumno con varios casos de ficheros de entrada que contienen cadenas de DNA para la prueba y test del programa con sus correspondientes soluciones. Nótese que debido al carácter concurrente que adquirirán los hilos en la Parte 2 de la práctica, no se puede asegurar un orden fijado para las cadenas resultantes. Por lo que para comprobar contra la solución hay que ordenar el fichero de salida por orden alfabético.

- * Ficheros `dnaXX.fasta`: ficheros de entrada sin procesar
- * Ficheros `sol_dnaXX.fasta`: ficheros solución para su correspondiente fichero de entrada ordenadas las cadenas alfabéticamente.

Anexo II – <string.h>

En la librería C `<string.h>` encontramos diferentes funciones para el procesamiento de cadenas de caracteres. Se recomienda el estudio, porque facilitará la programación de la práctica, de las siguientes funciones:

```
size_t strlen(const char *str)
       Computes the length of the string str up to but not including the terminating null
       character.

char *strcpy(char *dest, const char *src)
       Copies the string pointed to, by src to dest.

int strcmp(const char *str1, const char *str2)
       Compares the string pointed to, by str1 to the string pointed to by str2.

char *strstr(const char *haystack, const char *needle)
       Finds the first occurrence of the entire string needle (not including the terminating
       null character) which appears in the string haystack.

size_t strcspn(const char *str1, const char *str2)
       Calculates the length of the initial segment of str1 which consists entirely of
       characters not in str2.

char *strpbrk(const char *str1, const char *str2)
       Finds the first character in the string str1 that matches any character specified in
       str2.
```

Anexo III - “xprintf.h”

Librería personalizada para imprimir cadenas de caracteres por la salida estándar con colores, útil para tareas de depuración y comprobación de funcionalidad del código.

```
#define RED   "\x1B[31m"
#define GRN   "\x1B[32m"
#define YEL   "\x1B[33m"
#define BLU   "\x1B[34m"
#define URED  "\x1B[4m\x1B[31m"
#define UGRN  "\x1B[4m\x1B[32m"
#define UYEL  "\x1B[4m\x1B[33m"
#define UBLU  "\x1B[4m\x1B[34m"
#define MAG   "\x1B[1m\x1B[35m"
#define CYN   "\x1B[1m\x1B[36m"
#define WHT   "\x1B[37m"
#define RESET "\x1B[0m"


void xprintf(char* dna){
    int i,j,inprimer=0;
    for(i=0; i<strlen(dna); i++)
        switch(dna[i]){
```

```
            case 'T': inprimer?printf(UGRN "T" RESET):printf(GRN "T" RESET); break;
            case 'C': inprimer?printf(UYEL "C" RESET):printf(YEL "C" RESET); break;
            case 'G': inprimer?printf(UBLU "G" RESET):printf(BLU "G" RESET); break;
            case 'A': if(&dna[i] == strstr(&dna[i], "AAAAA")){
                        for(j=0; j<5; j++,i++)
                            printf(CYN "A" RESET);
                        if(inprimer)  inprimer=0;
                        else          inprimer=1;
                      }else
                        inprimer ? printf(URED "A" RESET) : printf(RED "A" RESET);
                      break;
            default:  printf(MAG "%c" RESET, dna[i]);
        }
}
```

Anexo IV - <sys/time.h>

```
#include <sys/time.h>


int main(int argc, char** argv){
    struct timeval  tv1, tv2;
    gettimeofday(&tv1, NULL);


    … /* código */


    gettimeofday(&tv2, NULL);
    printf ("Total time = %f seconds\n",
             (double) (tv2.tv_usec - tv1.tv_usec) / 1000000 +
             (double) (tv2.tv_sec - tv1.tv_sec));
    return 0;
}
```