Curso 2020 - 2021
Práctica Final – Decodificación de mensajes

Objetivos

- Practicar la creación y gestión de hilos en UNIX.
- Sincronizar el funcionamiento de hilos en UNIX.
- Practicar la gestión y control de acceso a memoria compartida mediante semáforos.

Descripción general

El objetivo de la práctica es diseñar un sistema concurrente para la decodificación de un mensaje oculto.

El sistema deberá procesar una sucesión de cadenas de caracteres que vendrán dadas en un fichero de texto. En cada línea del fichero vendrá un *token*, que podrán tener longitud entre 2 y 6 caracteres. Entre esos *tokens,* sólo algunos portarán información sobre el mensaje a decodificar y cuando lo porten será un único carácter del mensaje oculto. Además, el token oculta la posición del carácter dentro del mensaje. El mensaje está distribuido y desordenado entre todos los *tokens* que contenga el fichero. Un fichero de ejemplo puede obtenerse en la ruta: /home/javramo/tmp/decodificacorto.dat y /home/javramo/decodificalargo.dat

El sistema deberá constar de una serie de hilos que deben lanzarse al comienzo del proceso de forma concurrente:

- **Productor**: Leerá del fichero de datos **mensaje.dat** un número desconocido de *tokens* de entre 2 y 6 caracteres por token. De éstos sólo se quedará con los que tengan 3 caracteres, ya que sólo los *tokens* de longitud 3 pueden contener información del mensaje oculto. Estos *tokens* los almacenará en un buffer circular (**buffer1**) con el siguiente formato:
  - Decena: primer carácter del *token*.
  - Unidad: segundo carácter del *token*.
  - Posible letra del mensaje: tercer carácter del *token*.

- **Consumidor/es**: Los consumidores se encargarán de leer del buffer **buffer1** los *tokens* que el productor vaya escribiendo. Por cada toquen leído se deberá procesar si el toquen es válido y si lo es, decodificar la letra del mensaje que contiene y el orden que ocupa en el mensaje:

  Para saber si el *token* es correcto hay que tener en cuenta que los *tokens* tienen la siguiente información:

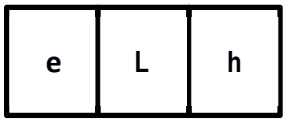
  Donde:

  - **Decena** será un carácter ascii comprendido entre ‘d’ (ascii 100) y ‘m’ (ascii 109) ambos incluidos, donde ‘d’ se corresponde con el número 0, ‘e’ con 1, …, y ‘m’ con 9. Si el carácter no estuviera en ese intervalo, se le considerará un *token* de tres caracteres erróneo y no formará parte del mensaje oculto. Codificará la decena del orden de la **letra** en el mensaje.
  - **Unidad** será un carácter ascii comprendido entre ‘F’(ascii 70) y ‘O’ (ascii 79) ambos incluidos, donde ‘F’ se corresponde con el número 0, ‘G’ con 1, …, y ‘O’ con 9. Si el carácter no estuviera en ese intervalo, se le considerará un *token* de tres caracteres erróneo y no formará parte del mensaje oculto.Codificará la unidad del orden de la letra del mensaje.

- ***Letra*** *será un carácter ascii comprendido entre los caracteres '!' (ascii 33) el carácter '{' (ascii 123) (ver una tabla ascii).* La letra está codificada con un retardo de 1, es decir, el carácter almacenado en **Letra** codificará el carácter anterior en la tabla ascii. Es decir, que si en **Letra** aparece el carácter 'B' el carácter decodificado será 'A', el carácter 'C' codificará 'B',etc. Si el carácter no estuviera en ese intervalo, se le considerará un *token* de tres caracteres erróneo y no formará parte del mensaje oculto.

Mediante los caracteres **Decena** y **Unidad**, permite decodificar el orden que el carácter **Letra** ocupa en el mensaje oculto. Por lo que el mensaje oculto no podrá tener una longitud superior a 100.

Por ejemplo.



- Decena: 'e' codifica el número 1.
- Unidad: 'L' codifica el número 6.
- Letra: 'h' codifica el carácter 'g' (carácter anterior).

Este *token* codifica el carácter 'g' que ocupa la posición 16 del mensaje oculto.

Todos los consumidores competirán por leer del **buffer1** de forma concurrente y una vez leído un *token* validarán y decodificarán el token, calculando el carácter que esconde y la posición que ocupará.

Con los datos recogidos del buffer circular (**buffer1**), cada consumidor realiza los cálculos necesarios para obtener:

- El número de tokens de tres caracteres procesados.
- El número de tokens válidos procesados.
- El número de tokens inválidos procesados.
- La posición más alta en el mensaje decodificada (maxIndex).

Los consumidores terminarán cuando el productor indique que se han leído todos los datos y una vez realizados los cálculos cada uno de los consumidores almacenará el resultado de estos cálculos en un **vector** de tamaño igual al número de consumidores, de tal forma que cada consumidor tiene una posición del **vector** asignada para almacenar sus resultados.

Además, introducirá en una lista enlazada ordenada, el carácter decodificado en el orden que establece la decodificación de Decena y Unidad, o en la versión más sencilla que opta a 8 puntos, se insertarán los caracteres en un vector estático de 100 caracteres.

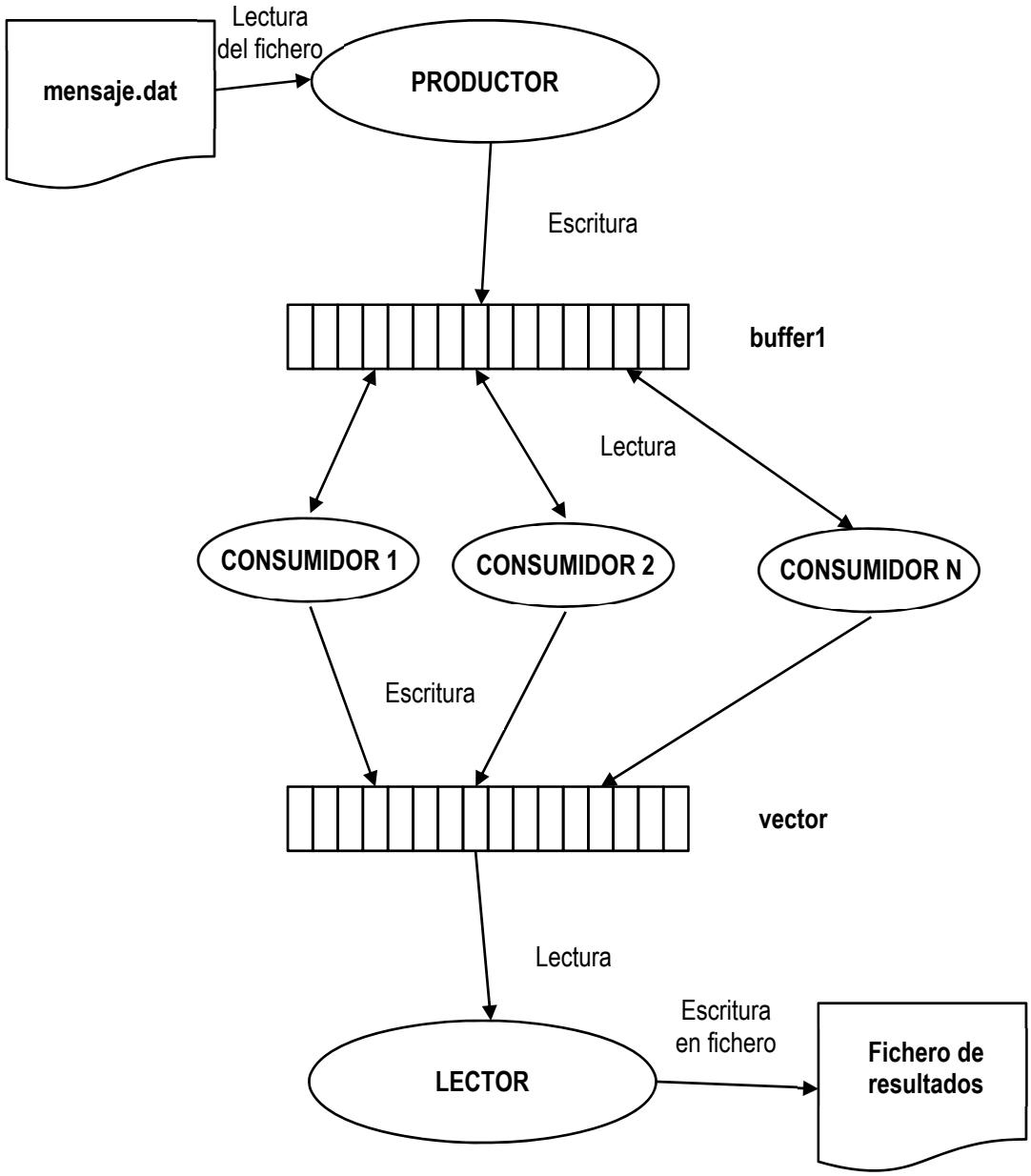
- **Consumidor Final (Lector)**: recogerá la información almacenada en el **vector** y la hará persistente en un fichero de texto plano una vez que algún consumidor termine su proceso. No deberá esperar a que todos los consumidores terminen, sino que cada vez que un consumidor termine y escriba en su posición los resultados, el **Lector** recogerá los datos almacenados en esa posición y los escribirá en el fichero de salida. El formato del fichero de salida será el siguiente:

```
Hilo 1
        Tokens procesados: 28
        Tokens correctos: 19
        Tokens incorrectos: 9
        MaxIndex: 26
Hilo 2
        Tokens procesados: 5
        Tokens correctos: 4
        Tokens incorrectos: 1
        MaxIndex: 25
Hilo 0
        Tokens procesados: 7
        Tokens correctos: 5
        Tokens incorrectos: 2
        MaxIndex: 27

Resultado final (los que procesa el consumidor final).
        Tokens procesados: 40
        Tokens correctos: 28
        Tokens incorrectos: 12
        MaxIndex: 27
        Mensaje: Correcto

Mensaje traducido: Este es un mensaje de prueba
```

Para determinar si el mensaje es correcto, deberá asegurarse que el número de *tokens* correctos del mensaje es igual al `maxIndex` menos 1.

Si el mensaje es correcto, se leerá en orden la lista enlazada que contiene el mensaje o del vector estático según la versión, y se escribirá también en el fichero de texto de resultados.

En la siguiente imagen se muestra de manera esquemática los diferentes hilos y recursos de los que consta el sistema.



Desarrollo de la práctica:

Parte 0 (OBLIGATORIA): ***1 único consumidor – Nota máxima alcanzable 4.0 puntos sobre 10.***

La aplicación a realizar tendrá la siguiente sintaxis a la hora de ser invocada para su ejecución:

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

IMPORTANTE - El programa realizará las siguientes tareas:

- El ***programa principal (main)*** se encargará de validar los parámetros pasados en la línea de comandos, reservar la memoria necesaria y crear hilos, asignando a éstos la tarea/función que les corresponda, y esperará a que estos hilos creados finalicen su ejecución de manera correcta. Se valorará el uso de estructuras y memoria dinámica.

- El hilo ***Productor*** deberá***:***

  - Leer los tokens almacenados en cada línea del fichero de entrada `<inputFile>`, pasado como primer argumento al programa, validarlas y estructurarlos.
  - Almacenar las estructuras en el **buffer1**, cuyo tamaño será el del tercer argumento pasado al programa `<tamBuffer>`.

- Existirá un único hilo ***Consumidor*** que*:*
  - Realizará los cálculos siguientes, leer el token del buffer, decodificar el token (posición y letra), número de tokens de tres caracteres procesados, el número de tokens válidos procesados, número de tokens descartados y la posición más alta del mensaje oculto calculada (maxIndex).
  - Introducir los caracteres en el vector estático de 100 caracteres en la posición determinada por **orden**.

- Todos los hilos deben lanzarse a la vez y han de ejecutarse de manera concurrente, coordinando, sus operaciones mediante semáforos. Se lanzarán primero los consumidores y por último el productor.

- El tamaño del buffer circular **buffer1** será `<tamBuffer>` (tercer argumento al programa).

- Los resultados obtenidos deben mostrarse por pantalla en el formato explicado anteriormente por el programa principal una vez terminen todos los hilos.

Parte 1 (OBLIGATORIA): Varios consumidores y lector final - Nota máxima alcanzable 8.0 puntos sobre 10.

La aplicación será una extensión de la parte 0 donde

- El número de consumidores se obtendrá del cuarto argumento `<numConsumidores>`, que además almacenará en el **vector** los mismos resultados calculados en la parte 0.

- Existirá un único hilo ***Lector*** que recogerá los datos del **vector** y las irá almacenando en el fichero según terminen los consumidores y cuyo nombre aparece como segundo argumento al programa `<outputFile>`.

- Todos los hilos deben lanzarse a la vez y han de ejecutarse de manera concurrente, coordinando, si fuera necesario, sus operaciones mediante semáforos.

- El tamaño del **vector** para depositar los resultados de cada consumidor será igual al número de consumidores que se proporcionan como argumento del programa. Cada consumidor almacenará sus resultados en una posición fija, correspondiente al identificador del hilo.

- El mensaje decodificado se insertará en una vector estático fijo de 100 caracteres.

En las pruebas es importante probar diferentes tamaños del **buffer1**, para comprobar que la coordinación con semáforos es correcta. Se pedirá que se prueben diferentes tamaños del buffer durante la defensa.

Parte 2: Decodificación del mensaje - Nota máxima alcanzable 10.0 puntos sobre 10.

Los diferentes hilos consumidores deberán introducir de manera ordenada cada uno de los caracteres decodificados en una lista enlazada.

El hilo Lector deberá recorrer la lista para poder leer el mensaje decodificado.

Seguimiento y entrega parcial

La práctica se debe realizar tanto en las sesiones de prácticas como fuera de estas. Cada grupo de prácticas debe dejar constancia al terminar cada sesión de prácticas del trabajo que lleva realizado y de cómo va a afrontar los próximos pasos.

La parte 0 de la práctica debe entregarse en el Moodle de la asignatura el día 9 de diciembre de 2002 antes de las 23:59. Se entregará lo que se haya hecho hasta ese momento independientemente de si se han conseguido todos los requisitos de esa fase.

El día 27 de diciembre de 2020 antes de las 23:59 se deberá realizar la entrega definitiva de la práctica, cumpliendo al menos los requisitos de la parte 1 y la 2 en caso de optar a los 10 puntos.

Cada grupo defenderá la práctica ante el profesor, donde se evaluará entre otras cosas que sea coherente con el trabajo realizado a lo largo de las sesiones de prácticas previas. Ente los días 11 de Enero de 2021 y el 29 de Enero de 2021 se deberá concertar una hora para realizar la defensa de la práctica según se acuerde con el profesor.

Condiciones que deben cumplir las entregas

- La entrega se realizará mediante la web de la asignatura en la plataforma Moodle aulas.inf.uva.es
- Se entregará el código fuente del programa en lenguaje C realizado.
- El código debe estar desarrollado en C estándar y compilar en la máquina jair.lab.inf.uva.es.
- El programa debe ser concurrente.
- Para una mejor identificación, el nombre del programa indicará el grupo de laboratorio y primer apellido de cada miembro de la pareja; será algo así como: X2_GonzalezMerino_ParteX.c. IMPORTANTE: si los apellidos tienen acento no ponerlo en el nombre del programa.
- En las primeras líneas de cada fichero fuente enviado, debe aparecer como comentario el nombre, dos apellidos y DNI de cada uno de los autores.
- 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.