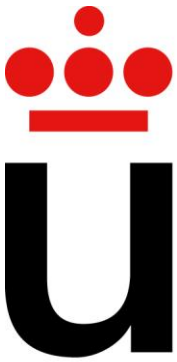

Universidad
Rey Juan Carlos

Sistemas Operativos

[PRÁCTICA 1 – PROGRAMACIÓN C]

MARIO ROJAS PADRÓN, ANA ACOSTA HERNÁNDEZ

TABLA DE CONTENIDO

Autores

Mario Rojas Padrón

m.rojas.2019@alumnos.urjc.es

Ana Cristina Acosta Hernández

ac.acosta.2017@alumnos.urjc.es

Descripción del Código

Diseño del Código

El código está estructurado en 4 funciones principales, alguna de ellas con métodos auxiliares para desempeñar su función. A su vez estas funciones se dividen en dos archivos:

- test.c : Contiene el *main* con la comprobación de argumentos y se encarga de la llamada de las funciones públicas de la librería.
- librería.c: En ella se implementa el código necesario para las funciones *head*, *tail* y *longlines*. Los elementos necesarios para ejecutar cada una de ellas son declarados inmediatamente encima y se implementan debajo de las mismas.

test.c

Está compuesto por dos funciones principales.

- *main:* Función en la que comienza la ejecución del programa. En ella se validan los parámetros y, en función del primero de ellos, hace que se ejecute la función correspondiente.
- *isNatural:* Comprueba si el carácter introducido es un número Natural (incluido 0).

main(int argc, char * argv[])

Parámetros:

La función puede recibir 1 o 2 parámetros a los cuales se va a poder acceder a través de las posiciones 1 y 2 de *argv*, respectivamente. Por otra parte, accedemos al número de parámetros mediante *argc*. Para mejorar la legibilidad del código, se han guardado todos los componentes necesarios en variables con nombres descriptivos.

- *number_of_parameters*: Almacena el número de parámetros, contiene *argc – 1* para que represente el número real ya que por defecto también incluye el nombre del programa.
- *function_name:* Contiene el nombre de la función que se ha de ejecutar.
- Posición 2 de *argv*: Contiene la cadena de caracteres con los parámetros de la función a testear.
- *N:* Contiene el parámetro que se tiene que pasar a las funciones que se están testeando. Por defecto tiene valor 10.

Comprobaciones:

- Que *number_of_parameters* sea 1 u 2. En caso contrario termina la ejecución con un error.
- Que *number_of_parameters* sea 2 y que la posición 2 de *argv* sea un número sea natural. Si tiene éxito, continua la ejecución reemplazando N por el valor introducido. En caso contrario, retorna un error.
- Que *function_name* sea una función válida, ejecutando la función correspondiente en caso de éxito y devolviendo un mensaje de error en caso contrario.

isNatural(char * number)

Parámetros:

Recibe el string del cual se quiere saber si es o no un número natural.

Ejecución:

Itera un bucle en el que, elemento por elemento, comprueba si son o no dígitos a través de la función *isdigit()*. Si alguno de los dígitos introducidos no es válido, devuelve false. Si todos han sido válidos, devuelve verdadero.

libreria.c

Está compuesta por 3 funciones públicas, declaradas en el archivo libreria.h y por las funciones auxiliares de estas.

Estructura General:

Tanto *head*, *tail* como *longlines* comparten una estructura muy similar. Ellas 3 utilizan la función *getline()* la cuál, dado un buffer (con valor nulo para que se aloje automáticamente), la longitud (con valor 0 para que se aloje automáticamente) y un fichero (en este caso la entrada estándar), permite recorrerlo línea a línea.

Pseudocódigo:

buffer = null

longitud = 0

Mientras (No se encuentre un EOF u ocurra un error)

buffer <- aloja memoria dinámicamente y se guarda la línea

longitud <- contiene el tamaño del buffer n

Libera memoria

Partiendo de esta base, en cada una de las funciones se imprime por pantalla en puntos diferentes de la ejecución, se reiniciarán buffer y la longitud para que se le vuelva a alojar memoria y la forma de liberar memoria será diferente. Eso se verá individualmente para cada una de las funciones.

head

head está basado en el mandato del mismo nombre. Esta, dado un número de líneas (opcional y por defecto 10), lee las primeras N líneas de la entrada estándar y las imprime por pantalla. Para imitar este comportamiento, usamos la estructura general descrita anteriormente y añadimos un contador para recorrer únicamente las N primeras líneas (o el EOF, lo que llegue primero) e irlas pintando conforme se recorren. Finalmente, se liberaría la línea.

Pseudocódigo:

buffer = null

longitud = 0

contador = 0

Mientras (El contador sea menor que N Y no se encuentre un EOF u ocurra un error)

buffer <- aloja memoria dinámicamente y se guarda la línea

 # (Solamente la primera vez)

longitud <- contiene el tamaño del buffer n

imprimir (buffer)

Liberar buffer

Estructuras de Datos

Para las siguientes dos funciones, se ha elegido utilizar dos estructuras de datos diferentes, cada una de ellas ajustada a las necesidades de las funciones. Debido a esto, las dos toman un enfoque casi idéntico, encapsulando dentro de las estructuras de datos las principales diferencias.

Esto se debe a que las dos, como se describirá más adelante, necesitan de almacenar varios elementos ya que tienen que esperar a que se termine el fichero antes de mostrar los datos. En este contexto, hemos preferido el uso de una ED antes de tener un array gigante que puede, o no, contener toda la entrada. Para así, dentro de lo posible, minimizar el la memoria del heap a reservar.

Estructura General con EEDD

Inicializar estructura de datos

buffer = null

longitud = 0

mientras (No se encuentre un EOF u ocurra un error)

> *buffer <- aloja memoria dinámicamente y se guarda la línea*
>
> *longitud <- contiene el tamaño del buffer n*
>
> *añadir buffer en ED*
>
> *buffer = null*
>
> *longitud = 0*
>
> *# Los vuelvo a inicializar para que cada vez que se repita el bucle aloje memoria*

Imprimir ED

Liberar ED

Componentes

Aunque la implementación sea diferente, las funciones públicas (realmente como están en el mismo archivo la única restricción de uso es que tan arriba o abajo se encuentren en el código) son prácticamente las mismas.

- Crear ED (a través de un struct que se debe inicializar)
- Añadir elemento
- Borrar elemento
- Mostrar ED
- Liberar ED

tail

tail está basado en el mandato de bash. Devuelve los N últimas líneas introducidas por entrada estándar por la salida estándar. La principal diferencia respecto a *head* es que necesitamos saber el tamaño entero de la entrada lo cuál es imposible hasta que el usuario no introduzca un EOF. Debido a esto es necesario almacenar las líneas para, una vez terminado el proceso, mostrar las líneas necesarias.

Stack

La ED elegida es una pila que está implementada a través de un array declarado dinámicamente que está guardado en un struct con los siguientes campos.

Stack:

- *store: Contiene un array de strings.*
- *len:* Contiene un entero con el tamaño del array.
- *init:* Tiene la posición del primer elemento insertado del array.
- *last:* Tiene la posición del último elemento insertado del array.

En un principio, al crearse, todos los elementos se inicializan a NULL y a partir de ahí funciona de la siguiente forma: Se insertan elementos y por cada uno se avanza el *last* a través del método *next* (para que sea capaz de dar la vuelta al array). Una vez da la primera vuelta, sigue insertando, desplazando al *init* y borrando el primer elemento insertado. De esta manera, se consigue siempre tener un array con la misma cantidad de elementos y que para mostrarse solo se deba recorrer desde el *init* hasta el *last*, dando la vuelta si es necesario.

longlines

longlines funciona de tal manera que lee la entrada estándar hasta que recibe un EOF y devuelve las N líneas con la mayor longitud. Y por las razones ya descritas hace uso de una ED.

Ordered List

La ED elegida es una Lista ordenada implementada a través de una Lista Doblemente Enlazada con Puntero al Principio y al Final. Esta elección se debe a que así se consigue complejidad de O(n) para insertar, complejidad O(1) para eliminar el último elemento y complejidad O(n) para mostrar los elementos.

El criterio de ordenación de la lista es de mayor a menor (mayor longitud de línea a menor longitud de línea) y se va ordenando conforme se introducen los elementos. Cada nuevo elemento recorre la lista y se coloca en la posición correspondiente.

Además, la operación para borrar el último es sencilla ya que con el puntero al final y el puntero de cada nodo al anterior, se puede borrar sin tener que hacer todo el recorrido.

Esta ED, a diferencia del Stack que se implementó, no tiene un tamaño máximo, por lo que cada vez que insertamos un elemento si el tamaño de la lista excede a la cantidad de elementos que se deberían imprimir, borra el último elemento.

Principales Funciones

	main	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**argc**	**int**	**Número de parámetros recibido**
	Argumento 2	**argv**	**char ****	**Argumentos introducidos al ejecutar el programa**
Variables Locales	Variable 1	**number_of_parameters**	**int**	**Equivale al número de parámetros introducido restándole uno para no contar el nombre de la función**
	Variable 2	**function_name**	**char ***	**Nombre de la función**
	Variable 3	**N**	**int**	**Inicializa por defecto a 10 el valor de N para usarlo en caso de que el usuario no introduzca ninguno**
	…			
Valor Devuelto	**int**			
Descripción de la Función	**Ejecuta el programa principal, controla el número de argumentos recibido**			

	head	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**N**	**int**	**Número de líneas a mostrar según inserción**
Variables Locales	Variable 1	**line**	**char ***	**Líneas recibidas por entrada estándar**
	Variable 2	**len**	**size_t**	**Guarda la longitud de cada línea**

	Variable 3	**read_len**	**ssize_t**	**Va guardando las líneas leídas**
	Variable 4	**amount_printed**	**int**	**Va guardando el número de líneas leídas**
Valor Devuelto	**int**			
Descripción de la Función	**Dado un entero N, muestra por pantalla las líneas recibidas por la entrada estándar hasta llegar a N**			

	tail	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**N**	**int**	**Número de líneas a mostrar en orden inverso a su inserción**
Variables Locales	Variable 1	**stack**	**struct Stack**	**Va almacenando las líneas**
	Variable 2	**line**	**char ***	**Líneas recibidas por entrada**
	Variable 3	**len**	**size_t**	**Guarda la longitud de cada línea**
	Variable 4	**read_len**	**ssize_t**	**Va guardando las líneas leídas**
Valor Devuelto	**int**			
Descripción de la Función	**Dado un entero N, muestra por pantalla las N últimas líneas recibidas por la entrada estándar**			

	longlines	**Nombre**	**Tipo**	**Descripción**

Argumentos	Argumento 1	**N**	**int**	**Número de líneas a mostrar según mayor tamaño**
Variables Locales	Variable 1	**list**	**Struct OrderedList**	**Va almacenando las líneas**
	Variable 2	**line**	**char ***	**Líneas recibidas por entrada**
	Variable 3	**len**	**size_t**	**Guarda la longitud de cada línea**
Valor Devuelto	**int**			
Descripción de la Función	**Dado un número N, muestra las N líneas más largas introducidas por teclado**			

Casos de Prueba

Para comprobar el correcto funcionamiento del programa, hemos testeado varios casos con texto escrito por el terminal y redirigiendo ficheros a la entrada estándar. Ambos casos pasaron correctamente obteniendo los resultados esperados. Además, se hizo uso del debugger para comprobar que internamente las estructuras de datos estaban correctamente compuestas.

Comentarios Personales

Los principales problemas que hemos encontrado han sido implementando las Estructuras de Datos y configurando el entorno de trabajo con los makefiles y entendiendo cómo debugear. Muchas veces planteábamos la estructura de datos de una determinada manera, pero conforme avanzábamos veíamos que no era la "óptima", por lo que acabábamos haciendo modificaciones que arrastraban demasiadas partes del código.

Para la siguiente práctica deberíamos quizás codificar al mismo tiempo que hacemos la memoria. Así, aunque quizás vayamos más lento no tendremos las prisas al final de rellenar rápido para tener la entrega.

Así mismo, el flujo de trabajo en general ha sido muy bueno. Hemos trabajado lo justo y necesario repartiéndonos las sesiones a lo largo de las semanas, programando los dos juntos y asegurándonos de que comprendíamos todas las partes.