Francisco Rubio Illán

frubio-i
frubio-i@student.42barcelona.com

GET_NEXT_LINE (+ BONUS)

Este proyecto consiste en desarrollar una **función personalizada**, **get_next_line(fd)**, que se encarga de **leer** una **línea** de texto desde un **file descriptor** de una forma eficiente. Como **ft_printf()**, también es una parte integral de la **librería** que estamos creando en 42, diseñada para proporcionar una serie de funciones de utilidad que facilitan el desarrollo de programas en C. Al igual que **ft_printf()** y las demás, está diseñada para ser **modular** y fácilmente **integrable** en **proyectos** de **C**, utilizándola como herramienta (función) **según** la **necesidad** del propio **proyecto**.

CARACTERÍSTICAS PRINCIPALES

La programación de **get_next_line(fd)** implica el manejo de un **file descriptor** proporcionado por nosotros, en el que **leemos línea a línea** mediante la **llamada** repetida de la **función**, en un **bucle**. La función debe **retornar** con cada llamada la **nueva línea** que se acaba de leer (**salto** de línea **incluido** “\n”). Si **no** hay nada **más** que **leer o** ha ocurrido algún **error**, deberá devolver **NULL**. También debe leer desde **stdin** (standard input). Todo esto hay que manejarlo correctamente, **permitiendo** en última instancia leer archivos grandes **sin leerlo todo** de una vez en **memoria**.

ARCHIVOS DEL PROYECTO

- **get_next_line.c:** Contiene la función principal del programa, ademas de las auxiliares de este.
- **get_next_line_utils.c:** Contiene las funciones que ayudan a la consecución de la nueva línea.
- **get_next_line.h:** Es el header. Contiene todas las funciones, librerías y macros necesarios en el programa.
- **get_next_line_bonus.c:** Casi mismo codigo que sin bonús, solo cambiada una parte para cumplir con el bonus.
- **get_next_line_utils_bonus.c:** Igual. Casi mismo codigo que sin bonús, solo cambiada una parte para cumplir con el bonus. En este caso, es el mismo.
- **get_next_line_bonus.h:** También es el header pero del bonus. Mismo código pero con un macro añadido (MAX_FD).

FUNCIONES CLAVE

El **proyecto** se estructura alrededor de varias **funciones auxiliares** que trabajan en conjunto con la **función principal** proporcionando la funcionalidad requerida en el **subject** de **get_next_line().**

get_next_line(): Esta es la función principal, donde devolvemos el resultado. Misma en BONUS pero la estática tiene que ser doble puntero, nada más.

gnl_check_static(): Se encarga de recoger si hay salto de línea en buffer (estática), llamar a otras funciones según la lógica y retornar la nueva línea a get_next_line() para su devolución final.

gnl_to_newline(): Se encarga de copiar en otro nuevo string el buffer hasta el len delimitado (donde encuentra salto de línea, que se incluye).

gnl_str_rest(): Se encarga de separar la linea leida (hasta \n, sin incluirlo) de resto. En buffer queda resto para la siguiente llamada de get_next_line();

FUNCIONES ÚTILES

check_str_n(): Se encarga de comprobar si hay salto de línea.

gnl_strlen_n(): Strlen modificado para que además de calcular el tamaño de un string, calcule según le pasemos un flag (control) u otro, el tamaño de un string hasta el propio salto de línea.

gnl_check_bf_join(): Se encarga de calcular el tamaño de buffer (resto), leer en un nuevo str (str_read), calcular también su tamaño y según el resultado, llamar a gnl_str_buffer_join() para unirlo en un string a devolver (str_buffer_join). Maneja todos los casos.

gnl_str_buffer_join(): Strjoin modificado para que libere lo necesario y retorne el resultado, str_buffer_join.

gnl_calloc_read_free(): Se encarga tanto de crear espacio en memoria mediante calloc (malloc + bzero), leer en un str mediante read y de liberar e igualar a NULL mediante free. Todos accesibles, de nuevo, con una flag (control) mandada según necesidad en cada llamada.

BONUS

Consiste en hacer nuestro programa capaz de gestionar varios múltiples a la vez sin perder el hilo de la lectura en cada uno de los fd mandados. Simplemente hay que hacer la estática doble puntero.

Consideraciones Adicionales

La implementación de get_next_line**()** permite un manejo efectivo de **grandes archivos** en C a través de los file descriptor (**fd**) y una interpretación precisa de gran parte de los casos posibles. Está supeditada a una **lectura** del tamaño de **BUFFER_SIZE** mandado o implementado en el header.

Es una **función** bastante más **compleja** de todo lo realizado hasta ahora. Por la **optimización necesaria** a la que obliga la propia **norma** (5 funciones en cada archivo y 25 líneas en cada una máximo). Sirve también como una excelente oportunidad para profundizar en las **variables estáticas**, el **manejo de memoria y su correcta liberación** y el diseño de código **modular** a un nivel más complejo del desarrollado hasta ahora en **C**.

Es recomendable igualmente explorar otras formas de hacer el programa ya que seguro que las hay mucho mejores, **óptimas** y **concisas**. Además, en el **examen** rank 03, uno de los ejercicios será conseguir **hacerlo** desde cero, completamente, **en** pocas **horas**. También podría ser usada para el **checker** de movimientos del **bonus** de **push_swap**, un proyecto a futuro.

Funciones usadas en el proyecto:

Funciones autorizadas:

- read
- malloc
- free

get_next_line

Declaración:

```
char *get_next_line(int fd);
```

Variables:

***str_buffer**: Variable **estática** donde guardamos el resto o lectura final al finalizar cada llamada.

***new_line**: Variable utilizada para recoger la línea a devolver, que llega hasta salto de línea.

Valor devuelto:

Retorna la nueva línea o el NULL si no se puede leer o queda un resto en la estática sin nueva lectura.

Descripción:

Función principal que primero comprueba que el BUFFER_SIZE, el fd y la próxima lectura, sea válido. Después, entra en bucle y comprueba si hay resto. Si no hay, lee directo en la estática y da otra vuelta al bucle. Si hay, llama a gnl_check_static() y dependiendo del retorno, devuelve new_line o NULL.

gnl_check_static

Declaración:

```
char *gnl_check_static(char **str_buffer, char *new_line, int fd);
```

Variables:

len: La longitud del string mandado a gnl_check_bf_join() antes y después de la llamada.

Valor devuelto:

new_line o NULL.

Descripción:

Determina si hay salto de línea. Si no hay, hace join entre resto y un nuevo read mediante la llamada a gnl_check_bf_join(). Si hay, crea nueva línea, determina el resto y devuelve dicha línea. Maneja también el doble puntero mandado desde la get_next_line() con la dirección en memoria de la estática.

gnl_to_newline

Declaración:

```
char *gnl_to_newline(char *str_buffer, int fd);
```

Variables:

***str_newline**: String donde recogemos la creación de este hasta salto de línea (incluido).

i: Contador general de carácteres introducido a new_line.

len_nl: Número de caracteres que tenemos que introducir hasta salto de línea (incluido).

Valor devuelto:

str_newline o NULL.

Descripción:

Se encarga de la creación del nuevo string con la nueva línea a devolver.

gnl_str_rest

Declaración:

```
char *gnl_str_rest(char *str_buffer, int fd);
```

Variables:

***str_tmp**: Puntero a la cadena de caracteres a imprimir.

len_rest: Número de caracteres que tenemos que introducir desde salto de línea (sin incluir).

len_nl: Número de caracteres que tenemos que introducir hasta salto de línea (incluido).

y: Contador general de carácteres introducidos al nuevo resto.

Valor devuelto:

str_tmp o NULL.

Descripción:

Se encarga de la creación del nuevo resto mediante una variable temporal. Al final liberamos la estática antes de meterle el resto de nuevo desde donde es llamada.

check_str_n

Declaración:

```
int  check_str_n(char *str_buffer);
```

Variables:

len_buffer: Número de caracteres que tenemos en el string mandado.

len_nl: Número de caracteres que tenemos hasta salto de línea (incluido).

Valor devuelto:

0 o 1.

Descripción:

Se encarga de la comprobación de si hay o no salto de línea en nuestro string. Básicamente, comprueba si el tamaño del string mandado y la longitud hasta salto de línea (si es que hay), sea el mismo. Dependiendo, retornará 0 (si hay salto) o 1 (no hay salto).

gnl_strlen_n

Declaración:

```
int  gnl_strlen_n(char *str, int control);
```

Variables:

i: Contador que recorre el string enviado.

Valor devuelto:

i.

Descripción:

Se encarga de determinar el numero de caracteres en un string. Es un strlen modificado para que además pueda determinar el largo hasta el salto de línea. Este comportamiento está controlado con un flag llamado control. 0 para hacer strlen y 1 para contar hasta salto de línea (si no hay, lee el tamaño completo, sin más).

gnl_check_bf_join

Declaración:

```
char *gnl_check_bf_join(char *str_buffer, int fd);
```

Variables:

***str_buffer_join**: string usado para unir resto con nueva lectura.

***str_read**: string usado para hacer la nueva lectura.

len_buffer: Número de caracteres que tenemos en la estática.

len_read: Número de caracteres que tenemos en la nueva lectura.

Valor devuelto:

str_buffer_join, str_buffer o NULL.

Descripción:

Se encarga de calcular el tamaño de buffer (resto), leer en un nuevo str (str_read), calcular también su tamaño y según el resultado, llamar a gnl_str_buffer_join() para unirlo en un string a devolver (str_buffer_join). Maneja todos los casos.

gnl_str_buffer_join

Declaración:

```
char *gnl_str_buffer_join(char *str_buffer_join, char *str_buffer,
char *str_read, int len_read);
```

Variables:

len_buffer: Número de caracteres que tenemos en la estática.

i: Contador usado para contar los caracteres introducidos en el nuevo string.

Valor devuelto:

str_buffer_join.

Descripción:

Se encarga de determinar la creación de un nuevo string que una el resto alojada en la estática con la nueva lectura, si la hubiera. Maneja cualquiera de las situaciones. Al final, libera el string donde se lee y la estática.

gnl_calloc_read_free

Declaración:

```
char *gnl_calloc_read_free(int size, int control, int fd,
char *str_free);
```

Variables:

i: Contador que recorre el string enviado. Usado para hacer bzero.

*__str__: String creado para hacer Calloc y Read, según flag (control) mandado.

len_read: Número de caracteres que tenemos en la nueva lectura.

Valor devuelto:

str o NULL.

Descripción:

Se encarga tanto de crear espacio en memoria mediante Calloc (Malloc + Bzero), leer en un str mediante Read y de liberar e igualar a NULL mediante Free. Todos accesibles, de nuevo, con una flag (control) mandada según necesidad en cada llamada. Es una función que recoge múltiples funcionalidades para ahorrar número de funciones debido a la norma.