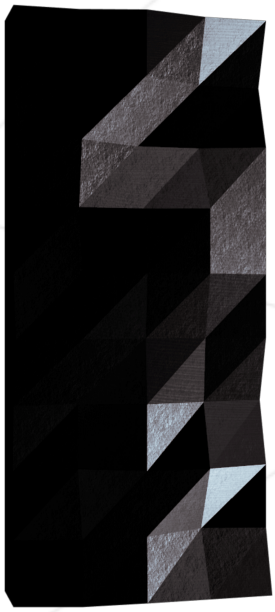

Get Next Line

Leer una línea de un file descriptor es demasiado aburrido

Resumen: El objetivo de este proyecto es simple: programar una función que devuelva una línea leída de un file descriptor.

Versión: 12.01

Índice general

Capítulo I

Objetivos

Este proyecto no solo te permitirá añadir una función bastante práctica a tu colección; también te hará aprender el increíble concepto de las variables estáticas en `C`.

Capítulo II

Instrucciones generales

- Tu proyecto deberá estar escrito en C.
- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) excepto en el caso de comportamientos indefinidos. Si esto sucede, tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en el heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra`, utilizar cc y por supuesto tu `Makefile` no debe hacer relink.
- Tu `Makefile` debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus de tu proyecto deberás incluir una regla `bonus` en tu `Makefile`, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos `_bonus.{c/h}`. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la `libft`, deberás copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` de tu proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo en tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus com-

pañeros. Si se encuentra un error durante la evaluación de Deepthought, esta habrá terminado.

Capítulo III

Parte obligatoria

Nombre de función	`get_next_line`
Prototipo	`char *get_next_line(int fd);`
Archivos a entregar	`get_next_line.c, get_next_line_utils.c, get_next_line.h`
Parámetros	`fd: File descriptor del que leer`
Valor devuelto	`Si todo va bien: la línea leída` `En caso de fallo o si la lectura termina: NULL`
Funciones autorizadas	`read, malloc, free`
Descripción	`Escribe una función que devuelva la línea leída de un file descriptor`

- Llamar a tu función `get_next_line` de manera repetida (por ejemplo, usando un bucle) te permitirá leer el contenido del archivo hacia el que apunta el file descriptor, **línea a línea**, hasta el final.
- Tu función deberá devolver la línea que se acaba de leer.
  Si no hay nada más que leer o si ha ocurrido un error, deberá devolver `NULL`.
- Asegúrate de que tu función se comporta adecuadamente cuando lea de un archivo y cuando lea de stdin.
- Ten en cuenta que la línea devuelta debe terminar con el caracter n, excepto si se ha llegado al final del archivo y esté no termina con un caracter n.
- En el header `get_next_line.h` deberás tener como mínimo el prototipo de la función `get_next_line`.
- Añade todas las funciones de ayuda que necesites en el archivo `get_next_line_utils.c`

Un buen comienzo sería saber qué es una variable estatica.

- Tu programa debe compilar con el flag `-D BUFFER_SIZE=xx`. Este flag se utilizará para determinar el tamaño del buffer de las lecturas de tu `get_next_line()`. Este parámetro será modificado por tus evaluadores y por Moulinette para probar tu programa.

Debemos ser capaces de compilar este proyecto con y sin el flag -D BUFFER_SIZE, junto a los flags habituales. Puedes elegir el valor por defecto que prefieras.

- El programa se compilará de la siguiente forma (se utiliza como ejemplo un tamaño de buffer de 42):
`cc -Wall -Werror -Wextra -D BUFFER_SIZE=42 <archivos>.c`.
- Se considera que `get_next_line()` tiene un comportamiento indeterminado si el archivo al que apunta el fd ha cambiado desde la última vez que se llamó, siempre que `read()` no haya llegado al final del archivo.
- Se considera que `get_next_line()` tiene un comportamiento indeterminado cuando lo que se lee es un archivo binario. Sin embargo, puedes implementar alguna manera lógica de sortear este problema, si quieres.

¿Funciona correctamente tu get_next_line si el BUFFER_SIZE es 9999? ¿Y si es 1? ¿Qué tal con 10000000? ¿Sabes por qué?

Intenta leer lo menos posible cada vez que se llame a get_next_line(). Si encuentras un salto de línea, deberás devolver la línea actual. No leas el archivo entero y luego proceses cada línea.

`Prohibido`

- No se permite la utilización de tu `libft` en este proyecto.
- Se prohibe la utilización de `lseek`
- Se prohibe la utilización de variables globales.

Capítulo IV

Parte bonus

Este proyecto es bastante directo y no deja mucho margen a los bonus. Sin embargo, confiamos en tu creatividad. Si has completado la parte obligatoria, inténtalo con estos bonus.

Aquí tienes los requisitos de la parte bonus:

- Desarrolla `get_next_line()` con una sola variable estática.
- Tu `get_next_line` tiene que ser capaz de gestionar múltiples fd a la vez. Es decir, si tienes tres fd disponibles para lectura (por ejemplo: 3, 4 y 5), debes poder utilizar `get_next_line` una vez sobre el fd 3, otra vez sobre el fd 4, y otra vez sobre el fd 5 de forma alterna. Y sí, no debe perder el hilo de lectura de cada uno de los fd.

Añade el sufijo `_bonus.[c\h]` a los archivos de esta parte bonus. Esto quiere decir que además de los archivos de la parte obligatoria, tienes que entregar los tres archivos siguientes:

- `get_next_line_bonus.c`
- `get_next_line_bonus.h`
- `get_next_line_utils_bonus.c`

La parte bonus solo será evaluada si la parte obligatoria está PERFECTA. Perfecta significa que se ha completado en su totalidad y funciona perfectamente, sin ningún fallo. Si no has completado TODOS los requisitos de la parte obligatoria, tu parte bonus no será evaluada de ninguna manera.

Capítulo V

Entrega y evaluación

Entrega tus ejercicios el tu directorio `Git`, como haces habitualmente. Solo se evaluará el trabajo que haya dentro de tu repositorio. Comprueba varias veces que el nombre de tus archivos es el correcto.

Cuando hagas los ejercicios, recuerda:
1) Tanto el tamaño del buffer como el de la línea pueden tener valores muy diferentes.
2) Un fd no solo apunta a archivos normales.
Sé inteligente y compara y verifica tus ejercicios con tus compañeros. Prepara una batería de pruebas de cara a la evaluación.

Una vez superado, no dudes en añadir tu `get_next_line()` a tu `lift`.

/=∂/\/\[](_)§ /\/\@|V †|-|@¯|¯ /-/!570®1<|-\£1_`/ ¢@/\/\ε vv!7}{ ???