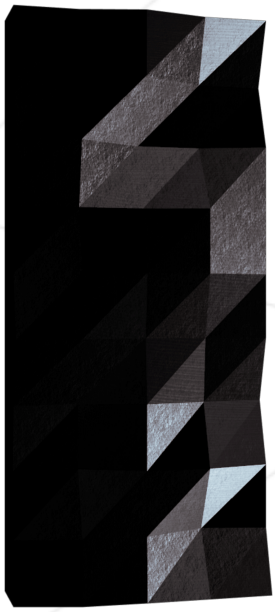

Get Next Line

Ler uma linha de um fd é entediante demais

Sumário:
O objetivo deste projeto é simples: programar uma função que retorne uma linha lida de um file descriptor.

Versão: 12

Conteúdo

Capítulo I

Objetivos

Este projeto não só permitirá que você adicione uma função muito prática à sua coleção, mas também fará com que você aprenda o incrível conceito de variáveis estáticas em programação em `C`.

Capítulo II

Regras gerais

- O seu projeto deve ser escrito em C.
- O seu projeto deve estar codificado dentro da Norma. Se você possui arquivos ou funções bônus, elas serão incluídas na verificação da norma e você receberá `0` no projeto se não seguir a norma.
- Suas funções não devem parar inesperadamente (falha de segmentação, erro bus, double free, etc.), exceto no caso de um comportamento indefinido. Se isso acontecer, o seu projeto será considerado não funcional e você receberá um `0` na avaliação.
- Qualquer memória alocada no heap deve ser liberada quando necessário. Nenhum leak será tolerado.
- Se o projeto pedir, você deve fazer um `Makefile` que compilará as suas fontes para criar a saída solicitada, utilizando as sinalizações `-Wall`, `-Wextra` e `-Werror`. O seu Makefile não deve ter relink.
- Se o seu projeto pedir um Makefile, o seu Makefile deve no mínimo conter as regras (NAME), `all`, `clean`, `fclean` e `re`.
- Para entregar o bônus, você deve incluir uma regra `bonus` no seu Makefile que vai adicionar os diversos headers, bibliotecas e funções que não são autorizadas na parte principal do projeto. Os bônus devem ficar em um arquivo `_bonus.{c/h}`. A avaliação da parte obrigatória e e da parte bônus são feitas separadamente.
- Se o projeto autorizar o uso do seu `libft`, você deve copiar suas fontes e o seu `Makefile` associado em uma pasta `libft` na raiz. O `Makefile` do seu projeto deve compilar a biblioteca usando o `Makefile` dela, depois compilar o projeto.
- Nós recomendamos criar programas de teste para o seu projeto, mesmo que esse trabalho **não seja entregue nem avaliado**. Isso te dará uma chance de testar facilmente o seu trabalho assim como o dos seus colegas. Isso será especialmente útil durante a sua avaliação. Inclusive, durante a avaliação, você fica livre para usar seus testes e/ou os testes da pessoa que você está avaliando.

- Você deve entregar o seu trabalho no repositório git que lhe foi atribuído. Somente o trabalho colocado no repositório git será avaliado. Se o Deepthought precisar corrigir o seu trabalho, isso será feito no fim do processo das avaliações dos colegas. Se um erro acontecer durante a avaliação Deepthought, ela será finalizada.

Capítulo III

Parte Obrigatória

Nome da função	`get_next_line`
Protótipo	`char *get_next_line(int fd);`
Ficheiros para entregar	`get_next_line.c, get_next_line_utils.c, get_next_line.h`
Parâmetros	`fd: O file descriptor a ser lido`
Valor de retorno	`A linha lida, se tudo correr bem` `NULL, se não houver nada a ser lido ou em caso de erro`
Funções externas autorizadas	`read, malloc, free`
Descrição	`Escreva uma função que retorne uma linha lida de um file descriptor`

- Chamadas repetidas (por exemplo, usando um loop) para a sua função `get_next_line()` devem permitir que você leia o arquivo de texto apontado pelo file descriptor, **uma linha de cada vez**, até o final.
- Sua função deve retornar a linha que foi lida.
  Se não houver mais nada para ler ou se ocorrer um erro, ela deve retornar `NULL`.
- Certifique-se de que sua função funcione corretamente tanto ao ler de um arquivo quanto ao ler da entrada padrão.
- **Por favor, note** que a linha retornada deve incluir o caractere terminador `\n`, exceto se o final do arquivo foi atingido e não termina com um caractere `\n`.
- O arquivo `get_next_line.h` deve conter pelo menos o protótipo da função `get_next_line()`.
- Adicione todas as funções auxiliares necessárias no arquivo `get_next_line_utils.c`.

Um bom começo seria saber o que é uma variável estática.

- Como você precisará ler arquivos na sua `get_next_line()`, adicione esta opção no comando que chama o compilador: `-D BUFFER_SIZE=n`
  Esta flag definirá o tamanho do buffer para a `read()`.
  O valor do tamanho do buffer será modificado por seus avaliadores e pela Moulinette para testar seu código.

Devemos ser capazes de compilar este projeto com e sem a flag -D BUFFER_SIZE além das flags usuais. Você pode escolher o valor padrão de sua preferência.

- Você compilará seu código da seguinte forma (um tamanho de buffer de `42` é usado como exemplo):
  `cc -Wall -Wextra -Werror -D BUFFER_SIZE=42 <arquivos>.c`

- Consideramos que `get_next_line()` tem um comportamento indefinido se o arquivo apontado pelo file descriptor mudou desde a última chamada, sempre que o `read()` não tiver atingido o final do arquivo.

- Também consideramos que `get_next_line()` tem um comportamento indefinido ao ler um arquivo binário. No entanto, você pode implementar uma maneira lógica de lidar com esse comportamento se desejar.

Seu get_next_line funciona corretamente se o você tentar ler um arquivo com um BUFFER_SIZE de 9999? E se for 1? E se for 10000000? Você sabe por quê?

Tente ler o mínimo possível cada vez que get_next_line() é chamado. Se você encontrar um nova linha, você deve retornar a linha atual. Não leia o arquivo inteiro e depois processe cada linha.

Proibido

- Você não pode usar sua `libft` neste projeto.

- O uso de `lseek()` é proibido.

- Variáveis globais são proibidas.

Capítulo IV

Parte Bônus

Este projeto é direto e não permite bônus complexos. No entanto, confiamos em sua criatividade. Se você completou a parte obrigatória, dê uma chance a esta parte bônus.

Aqui estão os requisitos da parte bônus:

- Desenvolva `get_next_line()` usando apenas uma variável estática.
- Seu `get_next_line()` pode gerenciar vários file descriptors ao mesmo tempo.
  Por exemplo, se você puder ler dos file descriptors 3, 4 e 5, você deve ser capaz de ler de um fd diferente por chamada sem perder a leitura de cada file descriptor ou retornar uma linha de outro fd.
  Isso significa que você deve ser capaz de chamar `get_next_line()` para ler do fd 3, depois do fd 4, depois do 5, depois novamente do 3, novamente do 4, e assim por diante.

Adicione o sufixo `_bonus.[c\h]` aos arquivos da parte bônus. Isso significa que, além dos arquivos da parte obrigatória, você entregará os 3 seguintes arquivos:

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

A parte bônus só será avaliada se a parte obrigatória estiver PERFEITA. Perfeito significa que todos os requisitos obrigatórios foram cumpridos integralmente e funcionam sem falhas. Se você não passou em TODOS os requisitos obrigatórios, sua parte bônus não será avaliada de forma alguma.

Capítulo V

Submissão e Avaliação entre Pares

Entregue seu projeto em seu repositório `Git` como de costume. Apenas o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes de seus arquivos para garantir que estejam corretos.

Ao escrever seus testes, lembre-se de que:
1) Tanto o tamanho do buffer quanto o tamanho da linha podem ter valores muito diferentes.
2) Um file descriptor não aponta apenas para arquivos comuns.
Seja inteligente e verifique seu projeto com seus colegas. Prepare um conjunto completo de testes diversos para a defesa.

Uma vez aprovado, não hesite em adicionar seu `get_next_line()` à sua `libft`.

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