

GET NEXT LINE

A função deve retornar a linha que foi lida incluindo o caractere `\n`, exceto se o final do arquivo foi atingido e não termine com um caractere `\n`.

Resumo da estratégia usada para leitura por linha:

1. Retorne uma linha lida a partir de um file;
2. Cada linha lida será armazenado na str;
3. Esta str será acrescentada a cada linha lida (incluindo o caractere `\n` ou no final (`\0`)).
4. Enquanto a str é acrescentada, deve-se apagar o buffer (a memória ocupada pela leitura)
5. 2 funções além da `get_next_line`:
 - a. Uma função irá ler, criar str, passar para str o que foi lido e apagar o buffer; → `Read_file`
 - b. A segunda função será criada para pegar tudo que foi lido e retornar a linha. → `Get_line`

```

static void read_file(int fd, char *buffer, char **str)
{
    char *temp;
    int i;

    if (!*str || !ft_strchr(*str, '\n'))
    {
        i = read(fd, buffer, BUFFER_SIZE);
        while (i > 0)
        {
            buffer[i] = '\0';
            if (!*str)
                *str = ft_substr(buffer, 0, i);
            else
            {
                temp = *str;
                *str = ft_join(*str, buffer);
                free(temp);
            }
            if (ft_strchr(buffer, '\n'))
                break ;
            i = read(fd, buffer, BUFFER_SIZE);
        }
        free(buffer);
    }
}

```

Por que ponteiro duplo? Como essa variável será manipulada pelo programa, ela é um ponteiro de char duplo, ou seja, ao invés do ponteiro apontar para um endereço de memória que contenha um dado, na verdade nesse endereço contém outro endereço de memória.

Se str for vazia ou não encontrar a quebra de linha até o tamanho do buffer (BUFFER_SIZE), irá ler o arquivo fd.

Irá ler e armazenar no buffer a leitura no tamanho BUFFER_SIZE. Read irá retornar o número de bytes lidos.

Irá colocar o byte nulo na posição final do buffer.

Se str for vazia (na 1ª lida), faz uma substring a partir do buffer: "copia" do buffer para str o tamanho i de bytes, a partir do ponto 0.

Se str não for vazia (estiver passando após a 1ª lida), irá juntar a str com o lido em buffer, na str. E será limpa a versão anterior da str, que foi armazenada em temp.

Caso encontre a quebra de linha no buffer, irá dar um break para usar apenas até a primeira quebra de linha encontrada e irá ler. Por que usar o break? Caso haja outro \n dentro do tamanho do buffer, só irá ler até o primeiro \n.

```

static char *get_line(char **str)
{
    char *line;
    char *temp;
    int after_n;

    if (!*str)
        return (NULL);
    if (!ft_strchr(*str, '\n'))
    {
        line = ft_substr(*str, 0, ft_strlen(*str));
        free(*str);
        *str = '\0';
        if (!*line)
        {
            free(line);
            return (NULL);
        }
        return (line);
    }
    after_n = ft_strlen(ft_strchr(*str, '\n'));
    line = ft_substr(*str, 0, ft_strlen(*str) - after_n + 1);
    temp = *str;
    *str = ft_substr(ft_strchr(*str, '\n'), 1, after_n - 1);
    free(temp);
    return (line);
}

```

Se não for encontrada a quebra de linha, copiará a str para line e irá esvaziar a str (é necessário atribuir o byte nulo para não ter lixo e/ou o espaço de memória poder ser usado por outro programa).

Se contém quebra de linha na str, irá passar pelos seguintes passos para poder retornar line.

Exemplo com a frase ola-cadete\n-da-42\0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
o	l	a	-	c	a	d	e	t	e	\n	-	d	a	-	4	2	\0
										0	1	2	3	4	5	6	7

Str = ola-cadete\n-da-42\0

Strchr(ola-cadete\n-da-42) = \n-da-42\0

After_n = strlen(\n-da-42) = 7

Line = substr(ola-cadete\n-da-42, 0, 17 - 7 + 1) = (ola-cadete\n-da-42, 0, 11) = ola-cadete\n\0

Temp = str → servirá para esvaziar a str anterior

Str = substr(\n-da-42, 1, 7 - 1) = (\n-da-42, 1, 6) = -da-42

Free(temp) → libera espaço

Return (line) = ola-cadete\n\0

Formar a linha: ola-cadete\n\0

Formar a str do que sobrou da linha: -da-42

```

char *get_next_line(int fd)
{
    static char *str;
    char *buffer;
    char *line;

    buffer = malloc(sizeof (char) * (BUFFER_SIZE + 1));
    if (!buffer)
        return (NULL);
    if (fd == -1 || BUFFER_SIZE <= 0)
    {
        free(buffer);
        return (NULL);
    }
    read_file(fd, buffer, &str);
    line = (get_line(&str));
    return (line);
}

```

Tratamento para erros

Leia o arquivo, acessando o endereço de str (origem)

Gere a linha com quebra

Retorne a linha

Passo A Passo

GET NEXT LINE

```
get_next_line.c
16 {
17     char    *temp;
18     int     i;
19
20     if (!*str || !ft_strchr(*str, '\n'))
21     {
22         i = read(fd, buffer, BUFFER_SIZE);
23         while (i > 0)
24         {
25             buffer[i] = '\0';
26             if (!*str)
27                 *str = ft_substr(buffer, 0, i);
28             else
29             {
30                 temp = *str;
31                 *str = ft_join(*str, buffer);
32                 free(temp);
33             }
34             if (ft_strchr(buffer, '\n'))
35                 break ;
36             i = read(fd, buffer, BUFFER_SIZE);
37         }

```

native process 10528 In: read_file L34 PC: 0x555555553e9

```
(gdb) n
(gdb) n
(gdb) n
(gdb) p *str
$1 = 0x0
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) p *str
$2 = 0x5555555592c0 "ola-o"
(gdb)
```

Rodando pela 1ª vez, irá parar na 22 e seguir até 34, onde fará um break e irá para 36 →

```
get_next_line.c
26         if (!*str)
27             *str = ft_substr(buffer, 0, i);
28         else
29         {
30             temp = *str;
31             *str = ft_join(*str, buffer);
32             free(temp);
33         }
34         if (ft_strchr(buffer, '\n'))
35             break ;
36         i = read(fd, buffer, BUFFER_SIZE);
37     }
38     free(buffer);
39 }
40
41 static char    *get_line(char **str)
42 {
43     char    *line;
44     char    *temp;
45     int     after_n;
46
47

```

native process 10528 In: read_file L36 PC: 0x555555553ff

```
(gdb) p *str
$1 = 0x0
(gdb) n
(gdb) n
(gdb) n
(gdb) p *str
$2 = 0x5555555592c0 "ola-o"
(gdb) n
(gdb) p *str
$3 = 0x5555555592c0 "ola-o"
(gdb)
```

Da linha 36, irá pular para o while da 23. →

```
get_next_line.c
13      #include "get_next_line.h"
14
15      static void    read_file(int fd, char *buffer, char **str)
16      {
17          char        *temp;
18          int          i;
19
20          if (!*str || !ft_strchr(*str, '\n'))
21          {
22              i = read(fd, buffer, BUFFER_SIZE);
23              while (i > 0)
24              {
25                  buffer[i] = '\0';
26                  if (!*str)
27                      *str = ft_substr(buffer, 0, i);
28                  else
29                  {
26                  temp = *str;
27                  *str = ft_join(*str, buffer);
28                  free(temp);
29                  }
30                  if (ft_strchr(buffer, '\n'))
```

native process 10528 In: read_file L25 PC: 0x555555555379

```
(gdb) n
(gdb) n
(gdb) p *str
$2 = 0x5555555592c0 "ola-o"
(gdb) n
(gdb) p *str
$3 = 0x5555555592c0 "ola-o"
(gdb) n
(gdb) n
(gdb) p buffer
$4 = 0x5555555592a0 "la-ca"
(gdb) █
```

Do while 23, irá colocar o byte nulo no que foi lido com quebra de linha, formando: la-ca\0 →

```
get_next_line.c
13      #include "get_next_line.h"
14
15      static void    read_file(int fd, char *buffer, char **str)
16      {
17          char        *temp;
18          int          i;
19
20          if (!*str || !ft_strchr(*str, '\n'))
21          {
22              i = read(fd, buffer, BUFFER_SIZE);
23              while (i > 0)
24              {
25                  buffer[i] = '\0';
26                  if (!*str)
27                      *str = ft_substr(buffer, 0, i);
28                  else
29                  {
30                      temp = *str;
31                      *str = ft_join(*str, buffer);
32                      free(temp);
33                  }
34                  if (ft_strchr(buffer, '\n'))
```

native process 10528 In: read_file L32 PC: 0x5555555553dd

```
(gdb) p *str
$3 = 0x5555555592c0 "ola-o"
(gdb) n
(gdb) n
(gdb) p buffer
$4 = 0x5555555592a0 "la-ca"
(gdb) n
(gdb) n
(gdb) n
(gdb) p *str
$5 = 0x5555555592e0 "ola-ola-ca"
(gdb) █
```

Após passar por join, será formada a nova str: ola-ola-ca. Como não encontrará ainda a quebra de linha, voltará ao while da linha 23. →

```
get_next_line.c
13  #include "get_next_line.h"
14
15  static void    read_file(int fd, char *buffer, char **str)
16  {
17      char    *temp;
18      int        i;
19
20      if (!*str || !ft_strchr(*str, '\n'))
21      {
22          i = read(fd, buffer, BUFFER_SIZE);
23          while (i > 0)
24          {
25              buffer[i] = '\0';
26              if (!*str)
27                  *str = ft_substr(buffer, 0, i);
28              else
29              {
30                  temp = *str;
31                  *str = ft_join(*str, buffer);
32                  free(temp);
33              }
34              if (ft_strchr(buffer, '\n'))
```

native process 10528 In: read_file L32 PC: 0x555555553dd

\$6 = 0x5555555592e0 "ola-ola-ca"

(gdb) n

(gdb) n

(gdb) n

(gdb) p buffer

\$7 = 0x5555555592a0 "dete\\"

(gdb) n

(gdb) n

(gdb) n

(gdb) p *str

\$8 = 0x5555555592c0 "ola-ola-cadete\\"

(gdb) █

```
get_next_line.c
13  #include "get_next_line.h"
14
15  static void    read_file(int fd, char *buffer, char **str)
16  {
17      char    *temp;
18      int        i;
19
20      if (!*str || !ft_strchr(*str, '\n'))
21      {
22          i = read(fd, buffer, BUFFER_SIZE);
23          while (i > 0)
24          {
25              buffer[i] = '\0';
26              if (!*str)
27                  *str = ft_substr(buffer, 0, i);
28              else
29              {
30                  temp = *str;
31                  *str = ft_join(*str, buffer);
32                  free(temp);
33              }
34              if (ft_strchr(buffer, '\n'))
```

native process 10528 In: read_file L32 PC: 0x555555553dd

\$8 = 0x5555555592c0 "ola-ola-cadete\\"

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) p *str

\$9 = 0x5555555592e0 "ola-ola-cadete\\n-da-"

(gdb) █

Encontrou uma quebra de linha, parou nela (break), leu e juntou a str na linha 31. Formando a str: ola-ola-cadete\\n-da- . Na próxima passagem →


```
get_next_line.c
13     #include "get_next_line.h"
14
15     static void    read_file(int fd, char *buffer, char **str)
16     {
17         char        *temp;
18         int          i;
19
20         if (!*str || !ft_strchr(*str, '\n'))
21         {
22             i = read(fd, buffer, BUFFER_SIZE);
23             while (i > 0)
24             {
25                 buffer[i] = '\0';
26                 if (!*str)
27                     *str = ft_substr(buffer, 0, i);
28                 else
29                 {
30                     temp = *str;
31                     *str = ft_join(*str, buffer);
32                     free(temp);
33                 }
34                 if (ft_strchr(buffer, '\n'))
```

native process 10528 In: read_file L32 PC: 0x5555555553dd

\$9 = 0x5555555592e0 "ola-ola-cadete\\n-da-"

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) p *str

\$10 = 0x555555559300 "ola-ola-cadete\\n-da-42\\0\\n"

(gdb)

Foi encontrado o byte final.

```
get_next_line.c
24         {
25             buffer[i] = '\0';
26             if (!*str)
27                 *str = ft_substr(buffer, 0, i);
28             else
29             {
30                 temp = *str;
31                 *str = ft_join(*str, buffer);
32                 free(temp);
33             }
34             if (ft_strchr(buffer, '\n'))
35                 break ;
36             i = read(fd, buffer, BUFFER_SIZE);
37         }
38     }
39     free(buffer);
40 }
41
42     static char    *get_line(char **str)
43     {
44         char        *line;
45         char        *temp;
```

native process 10528 In: read_file L39 PC: 0x555555555425

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb) p *str

\$10 = 0x555555559300 "ola-ola-cadete\\n-da-42\\0\\n"

(gdb) n

(gdb) n

(gdb) n

(gdb) n

(gdb)

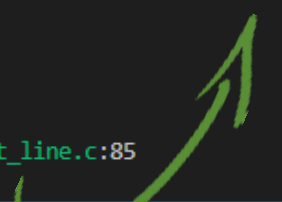
E será limpo o buffer.

```
get_next_line.c
75
76     buffer = malloc(sizeof (char) * (BUFFER_SIZE + 1));
77     if (!buffer)
78         return (NULL);
79     if (fd == -1 || BUFFER_SIZE <= 0)
80     {
81         free(buffer);
82         return (NULL);
83     }
84     read_file(fd, buffer, &str);
>85     line = (get_line(&str));
86     return (line);
87 }
```

native process 10528 In: get_next_line L85 PC: 0x5555555555e6

(gdb) n
(gdb) n
(gdb) n
(gdb) p *str
\$10 = 0x555555559300 "ola-ola-cadete\\n-da-42\\0\\n"

(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
get_next_line (fd=3) at get_next_line.c:85
(gdb) |



Juntou o buffer com str e passou para a penúltima linha de get_next_line

```
get_next_line.c
52     line = ft_substr(*str, 0, ft_strlen(*str));
53     free(*str);
54     *str = '\0';
55     if (!*line)
56     {
57         free(line);
58         return (NULL);
59     }
60     return (line);
61 }
>62     after_n = ft_strlen(ft_strchr(*str, '\n'));
63     line = ft_substr(*str, 0, ft_strlen(*str) - after_n + 1);
64     temp = *str;
65     *str = ft_substr(ft_strchr(*str, '\n'), 1, after_n - 1);
66     free(temp);
67     return (line);
68 }
69
70 char *get_next_line(int fd)
71 {
72     static char *str;
73     char
```

native process 10528 In: get_line L62 PC: 0x55555555554e2

(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
get_next_line (fd=3) at get_next_line.c:85
(gdb) s
get_line (str=0x5555555555431 <read_file+278>) at get_next_line.c:43
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) |

Como foi encontrado \n, irá gerar after_n, e então, criar a linha →

```
get_next_line.c
52         line = ft_substr(*str, 0, ft_strlen(*str));
53         free(*str);
54         *str = '\0';
55         if (!*line)
56         {
57             free(line);
58             return (NULL);
59         }
60         return (line);
61     }
62     after_n = ft_strlen(ft_strchr(*str, '\n'));
63     line = ft_substr(*str, 0, ft_strlen(*str) - after_n + 1);
64     temp = *str;
65     *str = ft_substr(ft_strchr(*str, '\n'), 1, after_n - 1);
66     free(temp);
67     return (line);
68 }
69
70 char *get_next_line(int fd)
71 {
72     static char *str;
73     char *buffer;
```

native process 10528 In: get_line L66 PC: 0x555555555574

```
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) p line
$11 = 0x555555559330 "ola-ola-cadete\n-da-42\n"
(gdb) n
(gdb) n
(gdb) p *str
$12 = 0x5555555592a0 ""
(gdb)
```

A linha retornada foi ola-ola-cadete\n-da-42\n e na linha 65 irá retornar o que sobrou após \n, que neste caso foi nada. →

```
get_next_line.c
76     buffer = malloc(sizeof (char) * (BUFFER_SIZE + 1));
77     if (!buffer)
78         return (NULL);
79     if (fd == -1 || BUFFER_SIZE <= 0)
80     {
81         free(buffer);
82         return (NULL);
83     }
84     read_file(fd, buffer, &str);
85     line = (get_line(&str));
86     return (line);
87 }
```

native process 10528 In: get_next_line L87 PC: 0x5555555555fa

```
(gdb) p *str
$12 = 0x5555555592a0 ""
(gdb) n
(gdb) n
(gdb) p line
$13 = 0x555555559330 "ola-ola-cadete\n-da-42\n"
(gdb) n
get_next_line (fd=3) at get_next_line.c:86
(gdb) n
(gdb) p line
$14 = 0x555555559330 "ola-ola-cadete\n-da-42\n"
(gdb)
```

Após retornar a linha na linha 67 de get_line, irá voltar a função get_next_line para retornar a linha da função principal. Terminando a leitura.