get_next_line,

▼ `FONKSİYONLAR`

1. `get_next_line()` fonksiyonu, bir dosya belirtecisinden bir satır okur. Dosya belirtecisini bir argüman olarak alır ve okunan satırın bir işaretçisini döndürür. Okumadıysa `NULL` döndürür.
2. `ft_copy_to_stash()` fonksiyonu, `stash` değişkenine bir buffer'dan metin kopyalar. `stash` değişkeni `NULL` ise, buffer'ı döndürür. Değilse, `stash` ve buffer'ı birleştirir ve yeni metnin bir işaretçisini döndürür.
3. `ft_have_nl()` fonksiyonu, bir dizi içinde yeni satır karakteri olup olmadığını kontrol eder. Dizi `NULL` ise `0` döndürür. Değilse, dizi içinde yeni satır karakteri varsa `1` döndürür, yoksa `0` döndürür.
4. `ft_extract_line()` fonksiyonu, bir dizide yeni satır karakterine kadar olan metni çıkarır ve yeni bir dizi olarak döndürür. Dizi `NULL` ise `NULL` döndürür. Değilse, yeni dizinin bir işaretçisini döndürür.
5. `ft_recreate_stash()` fonksiyonu, bir dizinin yeni satır karakterinden sonraki metnini alır ve yeni bir dizi olarak döndürür. Dizi `NULL` ise `NULL` döndürür. Değilse, yeni dizinin bir işaretçisini döndürür.
6. `ft_free_stash()` fonksiyonu, `stash` göstergesi tarafından işaret edilen dizgenin belleğini boşaltır. `create_line` 0 ise, fonksiyon belleği basitçe boşaltır ve `stash`'i NULL olarak ayarlar. `create_line` 1 ise, fonksiyon önce `stash` tarafından işaret edilen dizenin bir kopyasını oluşturur ve ardından orijinal dizgeyi boşaltır. Dizgenin kopyası daha sonra `stash`'e atanır.

`read()` fonksiyonu, bir dosya belirtecisinden bir miktar bayt okur. Dosya belirtecisini, okunacak bayt sayısını ve okunacak baytların depolanacağı bir buffer'ı bir argüman olarak alır. Okunan bayt sayısını döndürür. Okunamadıysa -1 döndürür.

`get_next_line()` fonksiyonu, `read()` fonksiyonunu kullanarak bir dosya belirtecisinden bir satır okur. `read()` fonksiyonundan okunan baytları `stash`

değişkenine kopyalar. `stash` değişkeninde yeni satır karakteri bulursa, yeni satır karakterine kadar olan metni çıkarır ve yeni bir dizi olarak döndürür. `stash` değişkeninde yeni satır karakteri bulamazsa, `read()` fonksiyonunu tekrar çağırır.

▼ `STATİK DEĞİŞKEN`

Statik değişkenler programın `data segment` bölümünde tanımlanan ve program işlediği sürece bellekte kendisine belirli bir yer ayrılan değişken türüdür.

- Statik değişkenler, sadece tanımlandıkları fonksiyon içinde kalıcı olarak değer taşırlar. Fonksiyon dışından bu değişkenlere erişim sağlanamaz.
- Statik olan elemanlar uygulama çalıştığı sürece kendilerine en son atanan değeri tutarlar.

▼ **Data Segment nedir? Stack ile arasındaki fark nedir?**

- `Data Segment`, programın başlangıcında ayrılan ve global değişkenlerin (static ve non-static) ve sabit verilerin saklandığı bellek bölümüdür.
- `Stack Segment`, fonksiyon çağrıları ve yerel değişkenlerin yönetildiği bellek bölümüdür.

Yani Data Segment ve Stack Segment arasındaki fark, sakladıkları verilerin türü ve yaşam döngüsüdür. Data Segment, programın başlangıcında ayrılır ve programın çalışma süresi boyunca sabit bir boyuta sahip olan verileri içerir. Stack Segment ise işlev çağrıları ve yerel değişkenler için kullanılan bellek bölümüdür ve işlevin çalışma süresi boyunca değişen yerel değişkenleri ve işlev çağrı bilgilerini içerir. Bu segmentlerin farklı amaçları ve özellikleri vardır.

▼ **Bu komutta Statik değişken ne işimize yarıyor?**

Fonksiyonlarımızda bir `static char *stash` değişkenimiz bulunmakta. Bu değişken aracılığıyla program çalıştığı sürece son değerlerimizi koruyarak `get_next_line` fonksiyonunu yeniden çağırabiliyoruz.

▼ `FILE DESCRİPTOR (FD)`

`File descriptor` (dosya tanımlayıcısı), çeşitli işlemler yapmak için kullanılan, dosya yolu yerine dosya sistemi çağrıları tarafından temsil edilen pozitif bir tamsayıdır. Her

işlemin kendi dosya tanımlayıcı tablosu vardır ve bu tablo, işlem ve Linux çekirdeği içindeki dosya nesneleri arasında bir bağlantı kurar.

Dosya erişimi sağlayan `open` fonksiyonu bize bir file descriptor değeri döndürür. Bu bir unsigned int değerdir. Yani fd negatif bir değer olamaz. Ancak fd değerimiz standart file descriptor değerleri olan 0,1 ve 2 ile de başlayamaz. Çünkü bu değerler sistem tarafından önceden tutulmuştur.

File Descriptor	Name	<unistd.h>	<stdio.h>
0	Standard Input	STDIN_FILENO	stdin
1	Standard Output	STDOUT_FILENO	stdout
2	Standard Error	STDERR_FILENO	stderr

Standart file descriptor'lar, C dilinde dosya giriş/çıkış işlemlerini kolaylaştırmak ve standart giriş, çıkış ve hata akışlarına erişimi sağlamak için önceden tanımlanan üç dosya tanımlayıcısıdır. C dilinde, bu üç standart file descriptor şunlardır:

1. `stdin` (Standart Giriş): File descriptor numarası `0` olan standart giriş, genellikle klavyeden kullanıcının girdilerini okumak için kullanılır. Örneğin, `scanf` fonksiyonu standart girişten veri okumak için kullanılabilir.
2. `stdout` (Standart Çıkış): File descriptor numarası `1` olan standart çıkış, programın çıktılarını yazmak için kullanılır. Örneğin, `printf` fonksiyonu standart çıkışa veri yazmak için kullanılabilir.
3. `stderr` (Standart Hata): File descriptor numarası `2` olan standart hata, programın hata mesajlarını veya hata çıktılarını yazmak için kullanılır. Bu sayede, normal çıktılar ve hata mesajları ayrı akışlar üzerinden yönlendirilebilir ve ayrı tutulabilir.

▼ Ayrıntılı bilgi için ekteki kaynaklara başvurulabilir;

Bir Dosyayı C - codequoi'deki Tanımlayıcısına Göre İşleme

Dosya tanımlayıcısı ve açık dosya açıklaması | Viyaçeslav Biriukov

▼ `LEAKS ve VALGRİND`

Programımızda sürekli dinamik olarak bellek alanı açtığımızda bunu iyi kontrol etmemiz gerekiyor. Yani açtığımız bu bellek alanlarını tekrardan `free` işlemi yapmamız lazım.

Eğer bunu tam olarak yapamzsak programda ciddi `bellek sızıntıları` (leaks) oluşabilir. Bu da sistemin veya programın bellek sorunları nedeni ile düzgün çalışmasını engelleyebilir.

Bellek sızıntılarını aşağıdaki iki farklı araçla kontrol edebiliriz.

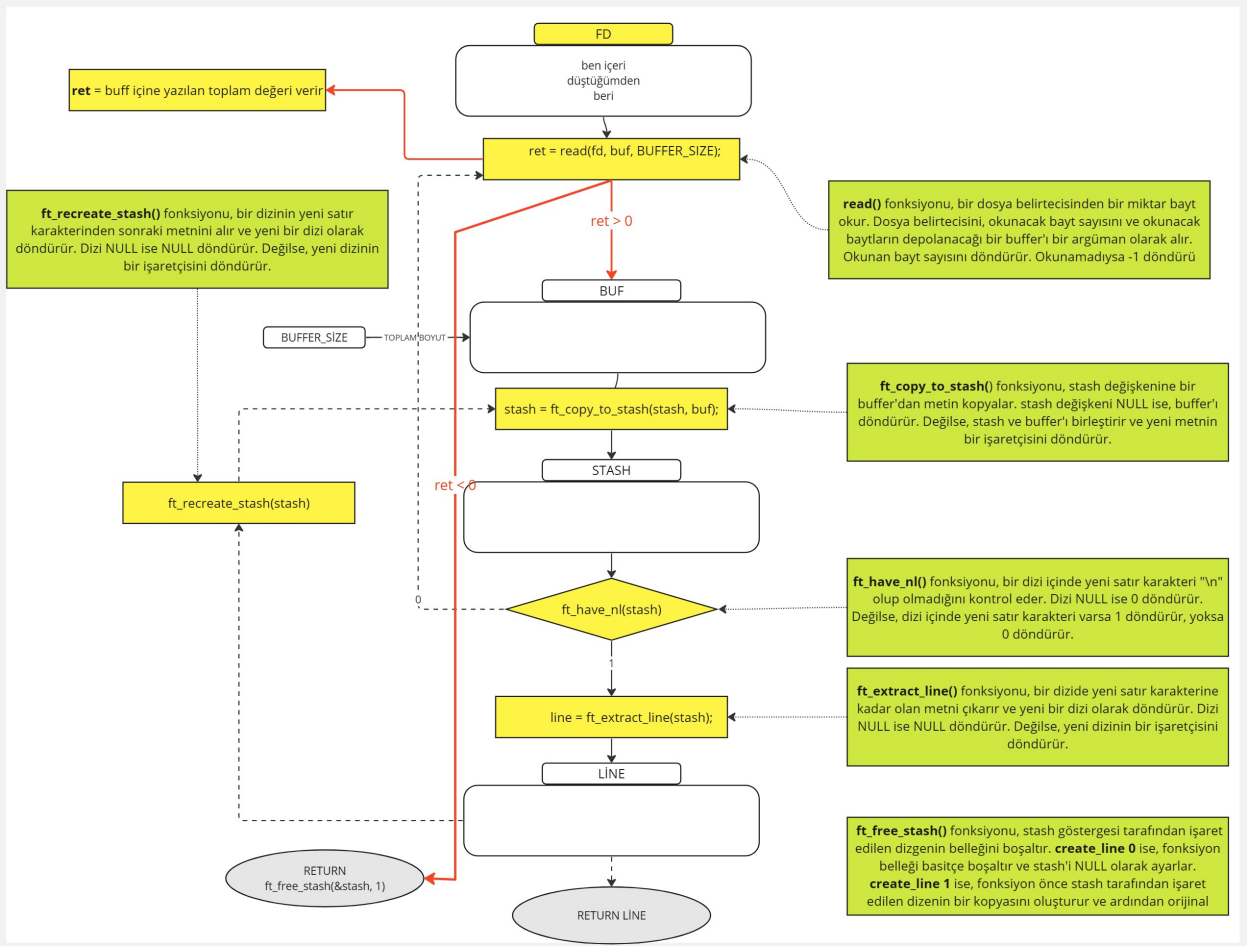
```
leaks a.out
valgrind ./a.out
```

▼ `MAİN`

```
#include <fcntl.h>
#include <stdio.h>
#include "get_next_line.h"

int main(void)
{
  int fd;
  char  *line;

  fd = open("test.txt", O_RDONLY);
  int i = -1;
  while (++i < 3)
  {
    line = get_next_line(fd);
    printf("%s", line);
    free(line);
  }
  return (0);
}
```



▼ MİRO

https://miro.com/app/live-embed/uXjVM1TYQzI=/?moveToViewport=-1776,180,3781,1846&embedId=189538131532