**get\_next\_line** adlı programın **BUFFER\_SIZE** değerinin 9999, 1 veya 10000000 gibi değerler almasının nedeni, programın içinde bellekte ne kadar yer ayrılmasının isteneceğini belirlemek için kullanılan bir kontrol mekanizmasıdır. Daha büyük bir **BUFFER\_SIZE** değeri, daha büyük bir bellek alanı için daha fazla veri okunmasına ve daha az okuma işlemi gerektiğine işaret eder, ancak daha küçük bir **BUFFER\_SIZE** değeri, daha sık okuma işlemi gerektiği anlamına gelebilir. Her iki durumda da program çalışır, ancak performans ve bellek kullanımı açısından farklı sonuçlar doğurabilir.

Buffer, bellekte bir veri depolama alanıdır. Buffer, verilerin bellekte toplu olarak saklanmasını ve daha sonra bir seferde okunmasını sağlar. Böylece verilerin sık sık diskten okunması gerekmiyor ve performans artar.

Buffer\_size ise bufferin bellekte kaç bayt veri saklayabileceğini belirleyen bir değerdir. Yani buffer\_size, bellekte kaç bayt veri saklanabileceğini tanımlayan bir değerdir. Aynı şekilde get\_next\_line gibi programlarda buffer\_size, bellekte kaç bayt veri okunabileceğini belirler.

Static değişkenler, bir fonksiyon veya bir blok içinde tanımlanır ve sadece o fonksiyon veya bloğun içinde geçerlidir. Ancak, fonksiyon veya bloğun içinde tanımlanmasına rağmen, bellekte programın sonunda bile yer içinde tutulur ve programın başka bir yerinde erişilebilir. Bu, fonksiyonlar arasında veya birden fazla fonksiyon çağrısı arasında değerleri koruyan değişkenler olarak kullanılabilir.

"lseek" Unix sistemlerinde işletim sistemi tarafından kullanılan ve dosya işlemleri için bir dosya pozisyon göstericisi sağlayan bir fonksiyonun adıdır. Bu fonksiyon, dosyadan okuma ve yazma işlemlerinin başlaması gereken konumu belirlemek için kullanılır. lseek fonksiyonu, dosya içinde bir pozisyon belirlemek için bir offset değerini (dosyanın bir belirli noktasına olan mesafeyi) kullanır. Offset, dosya başından itibaren belirli bir bayt sayısı olarak tanımlanır. lseek fonksiyonu, dosya okuma ve yazma işlemlerinin başlaması gereken pozisyonu belirlemek için kullanılır ve okuma veya yazma işlemleri sırasında dosya pozisyonunun değiştirilmesine olanak tanır.

Lseek örnek:

**lseek** fonksiyonu, bir dosya açıkken dosya içinde bulunan okuma/yazma konumunu değiştirmek için kullanılır. Aşağıdaki örnek, dosya içinde belirli bir konuma gitmek ve oradan bir veri okumak için **lseek** fonksiyonunun nasıl kullanılabileceğini gösterir

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main()

{

int fd;

char buffer[100];

fd = open("test.txt", O\_RDONLY);

if (fd == -1)

{

perror("open");

return 1;

}

lseek(fd, 10, SEEK\_SET);

read(fd, buffer, 5);

buffer[5] = '\0';

printf("Read data: %s\n", buffer);

close(fd);

return 0;

}

Bu örnekte, **test.txt** isimli bir dosya açılır ve dosya içinde bulunan okuma/yazma konumu **lseek** fonksiyonu ile 10. bayt'tan itibaren değiştirilir. Sonra, okuma işlemi yapılarak 5 baytlık veri okunur ve ekrana yazdırılır.

**perror()** fonksiyonu, sistem çağrılarında oluşan hata durumlarını raporlamak için kullanılan bir standart C fonksiyonudur. Fonksiyon, belirtilen hata mesajını standart hatalı çıkış akışına (stderr) yazdırır ve bu hata mesajına ": " eklenmiş "System Error: " şeklinde bir ön-ek yapar. **perror()** fonksiyonu, hata durumlarını takip etmek ve belirli bir problemi tanımlamak için yararlıdır.

"buff" değişkenin başlangıç değeri "1" olarak ayarlanmıştır. Daha sonra "read()" fonksiyonu ile fd dosyasından okuma yapılır. Eğer okuma işlemi başarılı ise "read()" fonksiyonu verilen "BUFFER\_SIZE" değerine eşit veya daha küçük bir değer döndürür ve "buff" değişkenine atanır.

Eğer okuma işlemi başarısız ise, "read()" fonksiyonu "-1" döndürür ve "buff" değişkeni "-1" olur. İşte bu durumda "if" bloğunda "buff" değişkeninin "-1" eşit olup olmadığı kontrol edilir. Eğer "buff" değişkeni "-1" ise bu bir hata durumunun işaretidir ve bu durumda program "0" döndürür.

"while (!ft\_strchr(str, '\n') && buff)" bu while döngüsünde, "buff" değişkeni okuma işlemi sonucunu ifade eder. "buff" değişkeninin değeri "-1" olduğunda okuma işlemi başarısız oldu ve program "free(s1)" komutunu çalıştırarak bellekte ayrılan yeri serbest bırakır ve "0" döndürür. Eğer "buff" değişkeni "0" ise, okuma işlemi sona ermiştir ve bu while döngüsü sonlandırılır.

**buff** değişkeni 1 olarak atanır çünkü **read** fonksiyonu döndürdüğü veri miktarını gösterir. İlk atamada 1 verilmesinin nedeni **read** fonksiyonunun döndürdüğü veri miktarının 0'dan büyük olup olmadığının kontrol edilmesidir. Eğer **read** fonksiyonu 0 döndürürse, dosya sonuna gelinmiş olduğu anlamına gelir ve daha fazla veri okunamaz. Eğer **read** fonksiyonu 1 den büyük bir değer döndürürse, okunan verinin miktarı gösterir ve fonksiyon devam edebilir.

Leaks Kontrolü

system(“leaks a.out”);