

Sistem Programlama

Ders 7

Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Standart I/O kütüphanesi

- Bu bölümde standart girdi çıktı kütüphanesini tanıtacağız.
- Bu kütüphane ISO C standartlarının parçasıdır ve UNIX dışında birçok işletim sisteminde tanımlanmıştır.
- Standart I/O kütüphanesindeki fonksiyonlar önbellek ayrılması ve girdi çıktı işlemlerini en uygun büyüklükteki önbellek ile yapılması gibi konuları bizim yerimize halleder.
 - Blok büyüklüğü konusunda endişelenmenize gerek kalmaz.
- Önceki bölümde gördüğümüz girdi çıktı fonksiyonları dosya belirteçleri üzerinden çalışıyordu.
 - Bir dosya açılınca dosya belirteci dönüyordu ve bu dosya belirtecini kullanarak girdi çıktı işlemleri yapılıyordu.
- Standart I/O kütüphanesinde ise fonksiyonlar veri akışı (stream) üzerinden çalışır.
 - Bir dosyayı standart I/O fonksiyonları ile açtığımızda, o dosya ile alakalı bir veri akışı başlatırız.

Veri akışı ve FILE nesneleri

- ASCII karakter kümesi ile her karakter bir byte ile temsil edilir.
- Milletlerarası karakter kümesi ile her karakter birden fazla byte ile temsil edilebilir.
- Standart I/O veri akışı tekli byte veya çoklu byte veri akışı olabilir.
 - Bir veri akışının oryantasyonu verinin tekli byte mı çoklu byte mı okunacağını belirler.
 - Bir veri akışı oluşturulduğunda, oryantasyonu belirsizdir.
 - Veri akışı üzerinde çoklu-byte fonksiyonu kullanılırsa, veri akışının oryantasyonu çoklu byte olarak ayarlanır.
 - Veri akışı üzerinde tekli-byte fonksiyonu kullanılırsa, veri akışı oryantasyonu tekli byte olarak ayarlanır.
 - Ayrıca freopen ve fwide fonksiyonları ile bir veri akışının oryantasyonunu değiştirebilir.

Veri akışı ve FILE nesneleri

- Bir veri akışı açtığımızda, standart I/O fonksiyonları bir FILE nesnesine işaretçi döner.
- Bu nesne normalde aşağıdaki bilgileri içeren bir yapıdadır.
 - I/O operasyonları için kullanılan dosya işaretçisi
 - Veri akışı için kullanılan önbelleğe bir işaretçi
 - Kullanılan önbelleğin büyüklüğü
 - Arayüzde bulunan karakter sayısı.
 - Bir hata bayrağı
- Yazdığınız uygulama FILE nesnesinin içeriğini incelemek zorunda değildir.
 - Tek yapılması gereken I/O fonksiyonlarına bu FILE işaretçisi argüman olarak verilmesidir.

Standart girdi, standart çıktı ve standart hata

- Üç veri akışı her işlem için otomatik olarak erişilebilir. Bunlar standart girdi, standart çıktı ve standart hatadır.
- Bunlar daha önce belirttiğimiz `STDIN_FILENO`, `STDOUT_FILENO` ve `STDERR_FILENO` dosya işaretçileri ile erişilebilen dosyalardır.
- Bu üç veri akışı önceden tanımlı `stdin`, `stdout` ve `stderr` işaretçileri ile erişilebilir.

Önbellekleme

- Standart I/O kütüphanesinin önbellekleme yapmaktaki amacı:
 - Mümkün olduğunca az sayıda okuma ve yazma çağrısı yapmak.
 - Önbellekleme işlemini otomatik olarak yapmak.
 - Bu sayede önbellek büyüklüğü konusunu düşünmek zorunda kalmazsınız.
- Üç farklı önbellekleme yöntemi mevcuttur:
 - Tam önbellekli
 - Asıl I/O işlemi standart I/O önbelleği dolduğunda gerçekleşir.
 - Diskte bulunan dosyalara ait veri akışı oluşturulduğunda bu işlem tam önbelleklidir.
 - Kullanılan önbellek veri akışı üzerinde ilk kez işlem yapıldığında bu işlemi yapan fonksiyon tarafından malloc çağırılmasıyla oluşur.
 - Önbellek standart fonksiyonlar tarafından otomatik olarak boşaltılabilir (örneğin önbellek dolduğunda).
 - Yada fflush fonksiyonu ile önbellek boşaltılabilir.

Önbellekleme

- Üç farklı önbellekleme yöntemi mevcuttur:
 - Satır önbellekli
 - Bu durumda asıl I/O işlemi girdi ve çıktıda yeni satır karakteri karşılaşıldığında olur.
 - Bu sayede birer birer karakterleri çıktı yapabiliriz (fputc fonksiyonu ile) ve asıl I/O satır tamamlandığında olur.
 - Satır önbellekleme genellikle bir veri akışı terminale bağlantılı ise kullanılır: standart girdi ve standart çıktı.
 - Önbelleksiz
 - Bu durumda girdi ve çıktı önbelleğe alınmaz
 - Karakterler mümkün olduğunca kısa zamanda çıktıya geçer.
 - Standart hata genellikle önbelleksizdir.
 - Bu sayede hata mesajları mümkün olan en kısa zamanda gösterilebilir.

Önbellekleme

- ISO C standartları önbellekleme ile ilgili aşağıda görülen kuralları içerir.
 - Standart girdi ve standart çıktı, interaktif bir araca yönlendirilmemişse tam önbelleklidir.
 - Standart hata hiçbir zaman tam önbellekli değildir.
- Çoğu sistem aşağıdaki şekilde dizayn edilmiştir.
 - Standart hata önbelleksizdir.
 - Diğer bütün veri akışları terminal araca yönelmişse satır önbellekli, aksi takdirde tam önbelleklidir.
- Belli bir veri akışının önbellekleme metodu setbuf ve setvbuf fonksiyonları ile değiştirilebilir.

Önbellekleme

- Herhangi bir zamanda bir veri akışının yazılmasını istersek fflush fonksiyonunu kullanabiliriz.

```
#include <stdio.h>
```

```
int fflush(FILE *fp)
```

Dönüş: OK ise 0, hata ise EOF.

Bir veri akışı açmak

```
#include <stdio.h>
```

```
FILE *fopen(const char *restrict pathname, const char *restrict type);
```

```
FILE *freopen(const char *restrict pathname, const char *restrict type, FILE  
*restrict fp);
```

```
FILE *fdopen(int filedes, const char *type);
```

Dönüş: OK ise dosya işaretçisi, hata ise NULL.

- fopen belirtilen bir dosyayı açar.
- freopen fonksiyonu belirtilen veri akışında belirtilen dosyayı açar, veri akışı önceden açık ise önce kapatır.
 - Genellikle bu fonksiyon önceden tanımlı veri akışları için kullanılır: standart girdi, standart çıktı ve standart hata.
- fdopen fonksiyonu bir dosya belirteci alır ve bunu bir standart I/O veri akışı ile eşler.
 - Bu fonksiyon genellikle pipe veya ağ iletişim kanalı oluşturan fonksiyonlar tarafından dönen dosya belirteçleri ile kullanılır.

Bir veri akışı açmak

- ISO C'ye göre 15 farklı type argümanı vardır.

• r veya rb	Okumak için aç
w veya wb	Uzunluğu 0 yap veya yazmak için yarat
a or ab	Sona ekle, sona yazmak için aç veya yazmak için yarat
r+ veya r+b veya rb+	Okuma ve yazma için aç
w+ veya w+b veya wb+	Uzunluğu 0 yap veya yazma ve okuma için aç
a+ veya a+b veya ab+	Okumak ve sonuna yazmak için aç veya yarat.

Bir veri akışı açmak

- Bir veri akışı açmak için 6 farklı yol ve her yol için kısıtlamaları şu şekilde özetleyebiliriz.

Kısıtlama	r	w	a	r+	w+	a+
Dosya bulunmalı	*			*		
Dosya içeriği silinir		*			*	
Veri akışı okunabilir	*			*	*	*
Veri akışı yazılabilir		*	*	*	*	*
Veri akışı sonuna yazılabilir			*			*

Bir veri akışı açmak

- w veya a argümanı ile yeni bir dosya oluşturduğumuzda dosyanın izinlerini ayarlamamız mümkün değildir (open ve creat fonksiyonları ile bunu yapmak mümkündür).
- Açık olan bir veri akışı fclose fonksiyonu ile kapatılır.

```
#include <stdio.h>
```

```
int fclose(FILE *fp) ;
```

Dönüş: OK ise 0, hata ise EOF.

- Önbelleklenmiş çıktı verisi dosya kapatılmadan yazılır.
- Önbelleklenmiş girdi verisi silinir.
- Veri akışı için otomatik olarak önbellek oluşturulmuşsa, önbellek yokedilir.
- Bir işlem normal olarak sonlanırsa (exit veya return ederek) önbellekteki yazılmamış veriler yazılır, açık olan I/O veri akışları kapanır.

Bir veri akışını okumak ve yazmak

- Veri akışını açtıktan sonra, üç farklı tip formatsız I/O yapabiliriz:
 - Karakter-karakter I/O: bir karakter okuyup bir karakter yazabiliriz.
 - Satır-satır I/O: Bir satır okuyup bir satır yazabiliriz.
 - Direk I/O: fread ve fwrite fonksiyonları ile belli büyüklükte belli sayıda nesne okuyup yazabiliriz.
 - Bu fonksiyonlar genellikle binary dosyalar ile kullanılır ve okuma yazma yaparken belli yapıda veri yazılır ve okunur.
- Formatlı I/O fonksiyonlarını (printf ve scanf gibi) daha sonra göreceğiz.

Bir veri akışını okumak ve yazmak

- Girdi fonksiyonları

- Aşağıdaki fonksiyonlar ile karakter karakter okuyabiliriz.

```
#include <stdio.h>
```

```
int getc(FILE *fp) ;
```

```
int fgetc(FILE *fp) ;
```

```
int getchar(void) ;
```

Dönüş: OK ise bir sonraki karakter, hata veya dosya biterse EOF.

- getchar ile getc(stdin) aynı işi yapar.
- İlk iki fonksiyon arasındaki fark, getc bir makro olarak tanımlanabilirken fgetc makro olarak tanımlanamaz.

Bir veri akışını okumak ve yazmak

- Önceki slaytta görüldüğü üzere bu fonksiyonlar hata verdiğinde veya dosyanın sonuna gelindiğinde aynı değeri döner.
 - İki durumu birbirinden ayırabilmek için `ferror` veya `feof` fonksiyonları kullanılmalıdır.

```
#include <stdio.h>
```

```
int ferror(FILE *fp) ;
```

```
int feof(FILE *fp) ;
```

Dönüş: koşul doğru ise 0 dışı (doğru) değer, yanlış ise 0 (yanlış).

```
void clearerr(FILE *fp) ;
```


Bir veri akışını okumak ve yazmak

- Bir veri akışından karakter okuduktan sonra, bu karakteri geri koymak istersek `ungetc` kullanılabilir.

```
#include <stdio.h>
```

```
int ungetc(int c, FILE *fp);
```

Dönüş: OK ise `c`, hata ise EOF.

Bir veri akışını okumak ve yazmak

- Çıktı fonksiyonları

- Karakter karakter çıktı için aşağıdaki fonksiyonlar kullanılabilir.

```
#include <stdio.h>
```

```
int putc(int c, FILE *fp);
```

```
int fputc(int c, FILE *fp);
```

```
int putchar(int c);
```

Dönüş: OK ise C, hata ise EOF.

- putchar(c) ile putc(c,stdout) aynı işlemi yapar.
 - putc makro olarak tanımlanabilirken, fputc makro olarak tanımlanamaz.

Bir veri akışını okumak ve yazmak

- Satır satır I/O

- Satır satır okuma işlemi aşağıdaki fonksiyonlar ile yapılabilir.

```
#include <stdio.h>
```

```
char *fgets(char *restrict buf, int n, FILE *restrict fp);
```

```
char *gets(char *buf);
```

Dönüş: OK ise buf, dosya sonu ve hata ise NULL.

- Her iki fonksiyon satırın okunacağı önbelleği belirtir.
- gets standart girdiden okurken fgets belirtilen veri akışından okur.

Bir veri akışını okumak ve yazmak

- Satır satır I/O

- Satır satır çıktı yapabilmek için aşağıdaki fonksiyonlar kullanılabilir.

```
#include <stdio.h>
```

```
int fputs(const char *restrict str, FILE *restrict fp);
```

```
int puts(const char *str);
```

Dönüş: OK ise negatif olmayan değer, hata ise EOF.

- fputs fonksiyonu null ile biten metni belirtilen veri akışına yazar.
- Sondaki null byte yazılmaz.
 - Yazılan metnin son karakteri yeni satır karakteri olmak zorunda değildir.
- puts fonksiyonu null ile biten metni standart çıktıya yazar, null byte yazılmaz.
 - puts son olarak yeni satır karakterini standart çıktıya yazar.

Standart I/O verimliliği

- Bu bölümde tanıttığımız fonksiyonları kullanarak standart I/O fonksiyonlarının verimliliği konusunda bilgi edinebiliriz.

```
#include "apue.h"

int
main(void)
{
    int    c;

    while ((c = getc(stdin)) != EOF)
        if (putc(c, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

Standart I/O verimliliği

```
#include "apue.h"

int
main(void)
{
    char    buf[MAXLINE];

    while (fgets(buf, MAXLINE, stdin) != NULL)
        if (fputs(buf, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

Standart I/O verimliliği

Fonksiyon	Kullanıcı CPU (saniye)	Sistem CPU (saniye)	Saat zamanı (saniye)	Byte olarak program metni
File I/O bölümünde en iyi	0.01	0.18	6.67	
fgets, fputs	2.59	0.19	7.15	139
getc, putc	10.84	0.27	12.07	120
fgetc, fputc	10.44	0.27	11.42	120
File I/O bölümünde tek byte	124.89	161.65	288.64	

98.5 mb büyüklüğünde 3 milyon satırlık bir dosya ile test yapılmıştır. Standart I/O bölümündeki sayılar ile karşılaştırma yapılmış.

Bir veri akışında pozisyonlama

- Standart I/O veri akışında pozisyon belirlemenin üç yolu vardır.

- ftell ve fseek

```
#include <stdio.h>
```

```
long ftell(FILE *fp);
```

Dönüş: OK ise mevcut pozisyon, hata ise -1.

```
int fseek(FILE *fp, long offset, int whence);
```

Dönüş: OK ise 0, hata ise 0 dışı değer.

```
void rewind(FILE *fp);
```

- whence değeri SEEK_SET, SEEK_CUR veya SEEK_END olabilir.

Bir veri akışında pozisyonlama

- Bir veri akışında pozisyon ayarlama için üç farklı yöntem vardır.
- ftello ve seeko

```
#include <stdio.h>
```

```
off_t ftello(FILE *fp);
```

Dönüş: OK ise pozisyon, hata ise (off_t) -1.

```
int fseeko(FILE *fp, off_t offset, int whence);
```

Dönüş: OK ise 0, hata ise 0 dışı değer.

Bir veri akışında pozisyonlama

- Bir veri akışında pozisyon belirlemek için üç farklı yöntem vardır:
- fgetpos ve fsetpos

```
#include <stdio.h>
```

```
int fgetpos(FILE *restrict fp, fpos_t *restrict pos);
```

```
int fsetpos(FILE *fp, const fpos_t *pos);
```

Dönüş: OK ise 0, hata ise 0 dışı değer.

Binary I/O

- Buraya kadar gördüğümüz fonksiyonlar satır veya karakter okuyabiliyor.
- Bazı durumlarda bütün bir yapıyı okumamız ve yazmamız gerekebilir.
 - Bunu sizce getc veya fputs ile yapabilir miyiz?
- Aşağıdaki fonksiyonlar ile binary I/O yapabiliriz.

```
#include <stdio.h>
```

```
size_t fread(void *restrict ptr, size_t size, size_t nobj, FILE  
*restrict fp);
```

```
size_t fwrite(const void *restrict ptr, size_t size, size_t nobj,  
FILE *restrict fp);
```

Dönüş: okunan veya yazılan nesne sayısı.

Binary I/O

- Bu fonksiyonların iki kullanımı vardır
 - Bir dizinin okunması ve yazılması için kullanılabilir. Örneğin 10 elemanlı bir dizinin 2 ile 5 arası elemanlarını yazdırabiliriz.

```
float data[10];
```

```
if(fwrite(&data[2], sizeof(float), 4, fp) != 4
```

```
    err_sys("fwrite error");
```

- Bir yapıyı okuyabilir ve yazabiliriz.

```
struct {
```

```
    short count;
```

```
    long total;
```

```
    char name[NAMESIZE];
```

```
} item;
```

```
if(fwrite(&item, sizeof(item), 1, fp) != 1
```

```
    err_sys("fwrite error");
```

Binary I/O

- Binary I/O ile ilgili temel sorun sadece aynı sistem üzerinde yazma ve okuma işlemi yapılabilmesidir.
 - Bir yapı içerisindeki ofset değeri farklı derleyiciler ve sistemler üzerinde, hizalama yöntemi nedeniyle, farklılık gösterebilir.
 - Multibyte tam sayılar ile noktalıklı sayıların saklanma formatı farklı makine mimarilerinde farklılık gösterir.