Sistem Programlama

Bilgisayar donanımına direkt olarak erişim eden yazılım üretme sürecidir.

- ⊕ İşletim sistemlerinin oluşturulması
- ⊕ I/O işlemleri
- ⊕ Derleyici oluşturulması
- ⊕ Yükleme programları vb. diğer
- ⊕ Ağ programlama

UNIX SHELL

- Kabuk çalıştırıldığında kendisini başlatır.
- Sonra sorgu karakterini ekrana basar % veya $
- Kullanıcıdan girdiyi bekler
- İlk kelimeyi alır, çalıştırılacak programın ismi olarak
- Programı belirtilen yollarda içinde arar. Bulunursa çalıştırır
- Program çalışırsa, program bitene kadar kabuk shell kendini durdurur.

- Komutlar argüman alabilir

cp src dest

- cp → argv[0] programın ismi
- src → argv[1] argüman 2
- dest → argv[2] argüman 3

- Kabuklar terminale ihtiyaç duymaz.
- Kabuklar stdin, stdout ve stderr dosyalarına erişim sahibidir.

- Girdiyi yönlendirmek için "<"
- Çıktıyı yönlendirmek için ">"
- "|" pipe sembolü ile birbirini takip eden komutlar yazılır.

Dosya ve klasör · Filtreler · Editörler · Metin işleme · Sistem yönetimi · Diğer

UNIX

Üniversitelerde ve sunucularda sıklıkla kullanılır.

BSD benzeri : NETBSD, FreeBSD, Linux, MacOS X

System V benzeri : Solaris, HP-UX, IRIX

- ⊕ Çoklu kullanıcıya sahip, çoklu programları aynı anda yönetebilir.
- ⊕ Programcılar tarafından programcılar için oluşturuldu
- ⊕ BİR DOSYA SADECE BYTE TOPLULUĞUDUR
- ⊕ Sistemin parçası olan programlar sadece tek iş odaklanmış

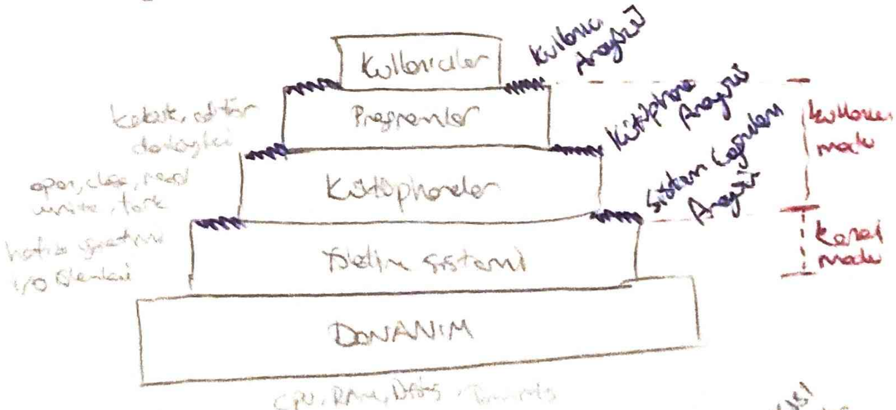
UNIX Programları

- Her biri basit işlemler yapar
- ⊕ MAN (Manual) kullanma kılavuzları bulunur.

- Stdin'den girdi alır
- Stdout'dan çıktı verir
- Anlamlı hata mesajı oluştururlar
- Anlamlı çıkış kodları verdirir.

UNIX Mimarisi

- İşletim sistemleri donanım kaynaklarını yönetir
- Programın çalışabileceği ortamı sağlar (KERNEL)
- Sistem çağrıları, çekirdek (KERNEL) üzerinde işlem yaparken kullanılabilir.
- Kütüphaneler, çok kullanılan fonksiyonların toplandığı bileşenlerdir.



UNIX SHELL

Bourne Shell	bin/sh
Bourne Again Shell	bin/bash
C shell	bin/csh

Satır yorumlayıcısı! Kullanıcıdan girdi alır verilen kodları gerçekleştirir

UNIX DOSYA SİSTEMİ

- Dosyalar ve klasörler hiyerarşik
- Her şey kök dizininden başlar (/)
- Klasörler de dosyadır (Diğer dosyaları içeren özel dosya tipi)
- Klasör içerdiği dosyaların bilgilerini saklayan bir yapıdır (struct)
  - Dosya tipi
  - Büyüklüğü
  - Sahibi
  - İzinleri
  - Son değişme zamanı

  stat
  fstat
- Dosyaya isim verirken boşluk (null) ve / karakterleri veremeyiz
- Yeni dosya oluşturulduğunda içinde "." ve ".." dosyaları otomatik oluşturulur.
- Uç uca birbirini takip eden dosyaların yoluna "pathname" "dosya yolu, yoladı" denir.

  / ile başlıyorsa mutlak (absolute)

  diğerleri kısmi, göreceli (relative)

Çalışma Klasörü — Working Directory

- Her programın, işlemin bir çalışma klasörü vardır
- chdir komutu ile bu çalışma dizini değiştirilebilir.
  - Sisteme giriş yaptığımızda çalışma klasörü olarak giriş dizinine gireriz.

Dosya Tanımlayıcıları — File Descriptors

- Kernel tarafından işlemlerin kullanmakta oldukları dosyaları tanımlayabilmek için kullanılan pozitif tam sayılardır.
- Açılan her dosyaya bir numara verilir.

STDIN → fd = 0
STDOUT → fd = 1
STDERR → fd = 2

bundan sonra gelenler 3, 4, 5 diye gider

Üst işlem, alt işlemin bitmesini bekler

Önbelleksiz (unbuffered) I/O işlemleri

open, read, write, lseek, close fonksiyonları ile yapılır

- read fonksiyonu okunan byte sayısını döndürür
- bu değer write fonksiyonunda ne kadar byte yazılacağı bilgisi için kullanılır

```
#define BUFFSIZE 4096
int n;
char buf[BUFFSIZE]
while(n = read(STDIN_FILENO, buf, BUFFSIZE)>0)
    if(write(STDOUT_FILENO, buf, n) != n)
        errsys("write error");
```

→ POSIX standarts
<unistd> kütüphanesindeki otomatik değerler
STDIN_FILENO "0"
STDOUT_FILENO "1"
STDERR_FILENO "2"

Önbellekli (buffered) I/O işlemleri

getc, putc

- getc fonksiyonu birer birer karakterleri okur ve okunan karakterler putc tarafından yazılır
- Son byte okununca EOF değeri döner

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

→ <stdio> kütüphanesinde tutulan standartlar

PROGRAM

Disk üzerindeki çalıştırılabilir dosyalar

Program çalıştığında işlem (görev) olur

her işlem için işlem numarası (process id) verilir (pozitif tam sayı)

fork, exec, waitpid

fgets standart girdiden satır okur

CTRL+D (satır sonu) (program sonlanır)

fork fonksiyonu yeni işlem yaratır

Yeni işlem fork çağrısı yapanın kopyasıdır.
Çağıran işlem üst, yeni yaratılan alt işlem
Fork, üst işleme, yeni yaratılan alt işlemin pid numarasını döner

DOSYA I/O İŞLEMLERİ

open — Dosya aç
read — Oku
write — Yaz
lseek — offseti ayarla
close — kapat

} Tamponsuz (unbuffered)

Dosyalara dosya belirteçleri sayesinde erişilir.

```
#include <fcntl.h>
int open(dosyanın ismi, bayrak, mode)
```

- Çalıştırsan dosya belirteci döner, çalışmazsa (−1)

Mode kısmı, yeni bir dosya oluşturulduğunda verilmelidir.

Bayrak — O_ flag kısmına

O_RDONLY	Sadece okumak için	1 tanesi zorunlu
O_WRONLY	Sadece yazmak için	
O_RDWR	İkisi de	

O_APPEND — Dosyanın sonundan başlar
O_CREAT — Dosya yoksa oluşturur
O_EXCL — O_CREAT yapılır, dosya varsa hata dönür
O_TRUNC — Eğer dosya mevcutsa ve O_WRONLY veya O_RDWR ile açılmışsa tümünü silerek başlar
O_DSYNC — Yazma operasyonu fiziksel olarak tamamlanır, dosya özellikleri beklenmez
O_RSYNC — Yazma tamamlanana kadar okuma yapılmaz
O_SYNC — Her yazma operasyonu dosya özellikleri dahil tamamlanır

Yeni dosya creat ile de yapılabilir

```
int creat(dosya ismi, mode);
```

open ile yapılışı böyle

```
open(dosya ismi, O_WRONLY | O_CREAT | O_TRUNC, mode);
```

```
#include <unistd.h>
ssize_t read(dosya belirteci, buff, boyut);
```

offsetten başlar istenilen byte kadar okur, okunmaya devam edildiğinde offseti büyütür.

istenen byte az ise dosyada kalanını döndürür

```
write(dosya belirteci, buff, boyut)
```

O_APPEND varsa sondan, yoksa başından başlar
offset değerinden başlar, yazılan byte kadar offseti arttırır.

```
close(dosya belirteci);
```

açık dosyayı kapatır

lseek

Her açılan dosyada "okunmakta olan dosya offseti" değeri saklanır.
Bu değer başlangıçtan itibaren okunan byte sayısıdır.
Var olan dosya açıldığında offset 0'dan başlar.

lseek açılan dosyanın offsetini değiştirir.

```
#include <sys/types.h>
off_t lseek(dosya belirteci, offset, whence);
```

offsetteki sayının anlamı whence kısmına göre farklılık gösterir

SEEK_SET — Başlangıçtan itibaren byte sayısı

SEEK_CUR — Bulunan noktadan itibaren byte sayısı

SEEK_END — Sondan itibaren byte sayısı

bulunduğumuz offseti şu şekilde bulabiliriz

```
lseek(fd, 0, SEEK_CURRENT)
```

Dosya işaretçileri bazen negatif tam sayılar alabilir

Dosyanın ofseti, dosyanın büyüklüğünden fazla ise

+ Dosyada delik oluşturulur.
+ Aradaki dosyanın okunmayan baytları 0 olur

Dosya Paylaşma

Birden fazla işlemin aynı dosya üzerinde çalışması olası

Her açık dosya için üç farklı değer saklanır

Her işlem tablosu bir dosya bağlantı tablosu tutar

+ Dosya bağlantısı bayrakları
+ Dosya tablosu kaydına bir işaretçi

Dosya tablosu (Her kayıt için)

+ Dosya Bayrakları (O_APPEND ...)
+ Şimdiki offset değeri
+ Dosyanın v_node bilgisine bir işaretçi

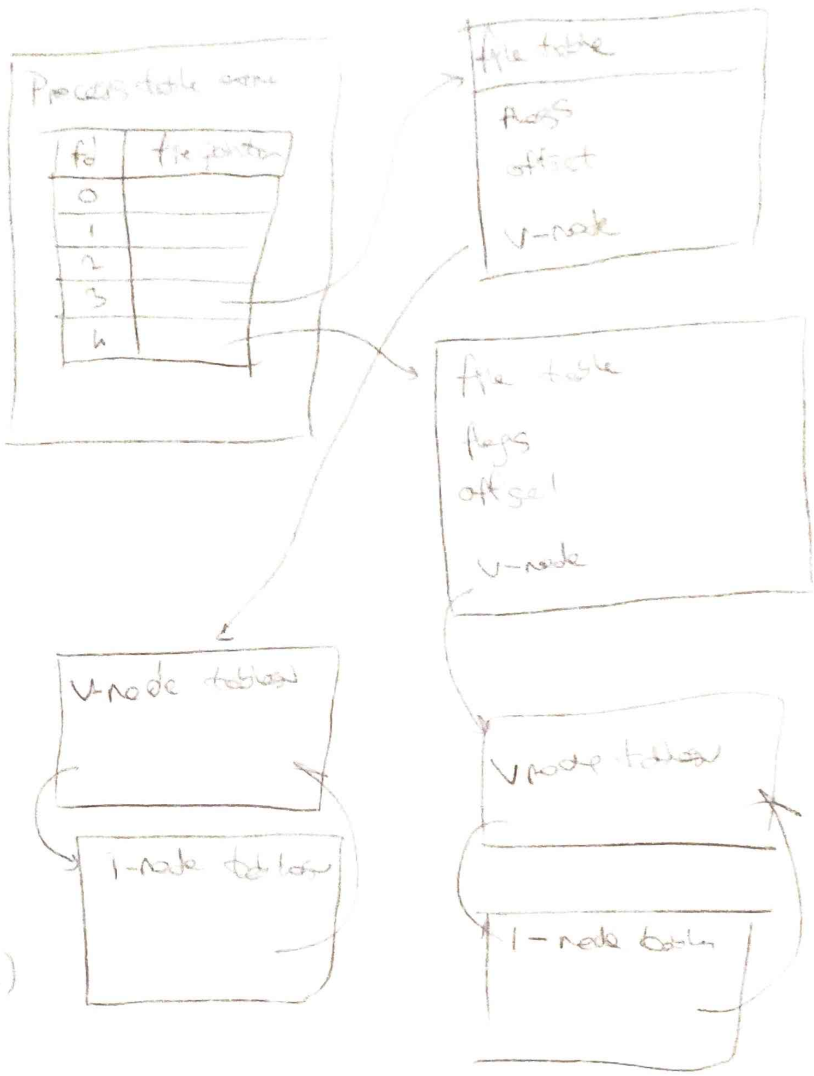
Her dosyanın v-node bilgisi bulunur

v-node

- Dosya tipi
- Dosyada çalışan fonksiyonlara işaretçi

i-node

- dosya sahibi, büyüklüğü, adresi



(İki farklı işlem bir dosyayı açtığında)

+ Her write'tan sonra offset değeri değiştirilir. Dosya büyüklüğü değiştiyse i-node da değişir

lseek sadece ilgili dosya tablosu kaydındaki offset değerini değiştirir

ATOMİKLİK

İki farklı işlemin bir dosyada işlem yapması tutarsızlıklara yol açabilir

İşlemin adımları bölünmeden yapılıyorsa atomik operasyondur

O_APPEND kullanarak atomiklik

ofset, lseek ile değiştirilirse atomik değildir

pread veya pwrite kullanılırsa bu operasyonları bölmek mümkün değildir

— işlem bitmeden ofset değişmez

HATA İŞLEME

Bir hata durumunda çoğu işlem negatif bir değer döner

errno değişkenine bir sayı atanır
Bu sayı hata hakkında bilgi verir

open fonksiyonunda 15 farklı errno değeri var

Hatalar numaraları ve açıklamaları <errno.h> kütüphanesinde

Hata mesajını yazdırmak için

```
#include <string.h>
char *strerror(int errnum);
```

bu fonksiyon hata değerini, hatayı anlatan bir açıklama döner

```
#include <stdio.h>
void perror(const char *msg)
```

bu fonksiyon otomatik errno'ya bakar ve hata mesajı döner.

<errno.h> hataları ikiye ayırıp.

- ölümcül olan
  - Tek yapılacak şey hata mesajı yazdırmak ve işlemi sonlandırmak
- ölümcül (geçici) olmayanlar
  - kaynak yetersizliği gibi bir süre bekleyip tekrar dene

KULLANICI NUMARASI

Kullanıcı No: Her kullanıcının bir numarası vardır. Bu kullanıcı numarasına göre diğer dosyalara izinler kontrol edilir. 0 olan kullanıcı root (süperuser)

Grup No: Grup numarası kullanıcıların katıldığı bir topluluktur. Grup izinleri kullanıcıların gruplarına tanımlanmıştır.

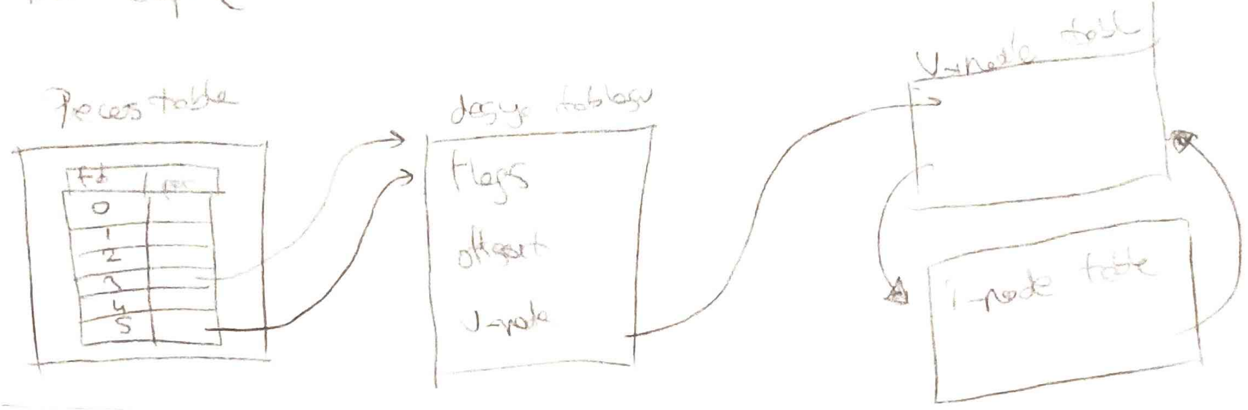
DOSYA BELİRTECİ KOPYALAMA

kullanılmakta olan bir dosya belirteci dup veya dup2 fonksiyonları ile kopyalanabilir.

```
#include <unistd.h>
int dup(int olddesc);
int dup2(int olddesc, int newdesc);
```

→ kullanılmakta olan dosya belirtecini kopyalar

→ kullanılmakta olan dosya belirtecini belli bir belirtece kopyalar



Diske veri yazmak istediğimizde kernel önce kendi yapıyor, önce önbelleğe kaydeder bir süre sonra yazmaya başlar.

⊕ Gecikmeli yazma

Disk ile önbellek arasında tutarlılık için sync, fsync, fdatasync fonksiyonları kullanılır.

sync

Üzerinde değişiklik yapılan tüm önbellek bloklarını yazılma sırasına sokar

fsync

Sadece bir dosya ile ilgili değişikliğe uygulanır. Bekler. Yazma işlemi bitinceye kadar bekler.

fdatasync

dosyanın sadece veri kısmını işlenir.

fcntl

Açık dosyanın özelliklerini değiştirme

```
int fcntl(dosya belirteci, command, arg);
```

başarısızsa cmd bağlı hata -1

- Dosya belirtecini kopyalayabilir
- dosya belirteci bayraklarını almak
- Dosya durum bayraklarını almak
- Asenkron I/O sahipliği almak
- Kayıt kilit almak için

cmd = F_DUPFD

cmd = F_GETFD F_SETFD

cmd = F_GETFL F_SETFL

cmd = F_GETOWN F_SETOWN

cmd = F_GETLK, F_SETLK F_SETLKW

ioctl

Çok amaçlı fonksiyon

stat fstat lstat

Belli bir dosya hakkında bilgi verir

#include <sys/stat.h>

int stat(dosya ismi, struct stat *restrict buf); Yol adı verilen dosyanın bilgilerini döner

int fstat(dosya betimleyicisi, struct stat *buf) açık olan dosyanın bilgilerini döner

int lstat(dosya ismi, struct stat *restrict buf); Sembolik linkleri de takip eder.

Dosya tipleri (st_mode)

- Normal Dosya
- Klasör Dosyası
- Blok Özel Dosyalar
- Karakter Özel Dosyalar
- FIFO
- Soket
- Sembolik Link