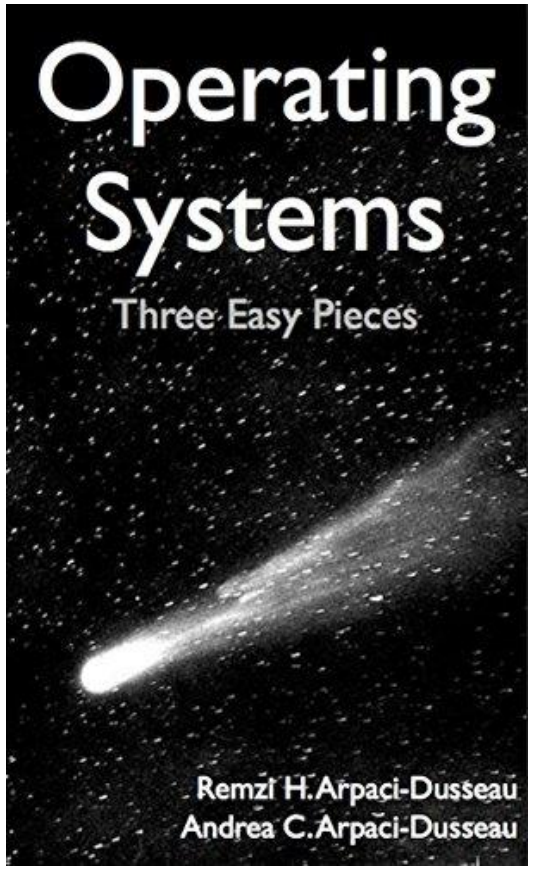
İşletim Sistemleri



2. Ders

Prof. Dr. Kemal Bıçakcı

Biraz Tarih

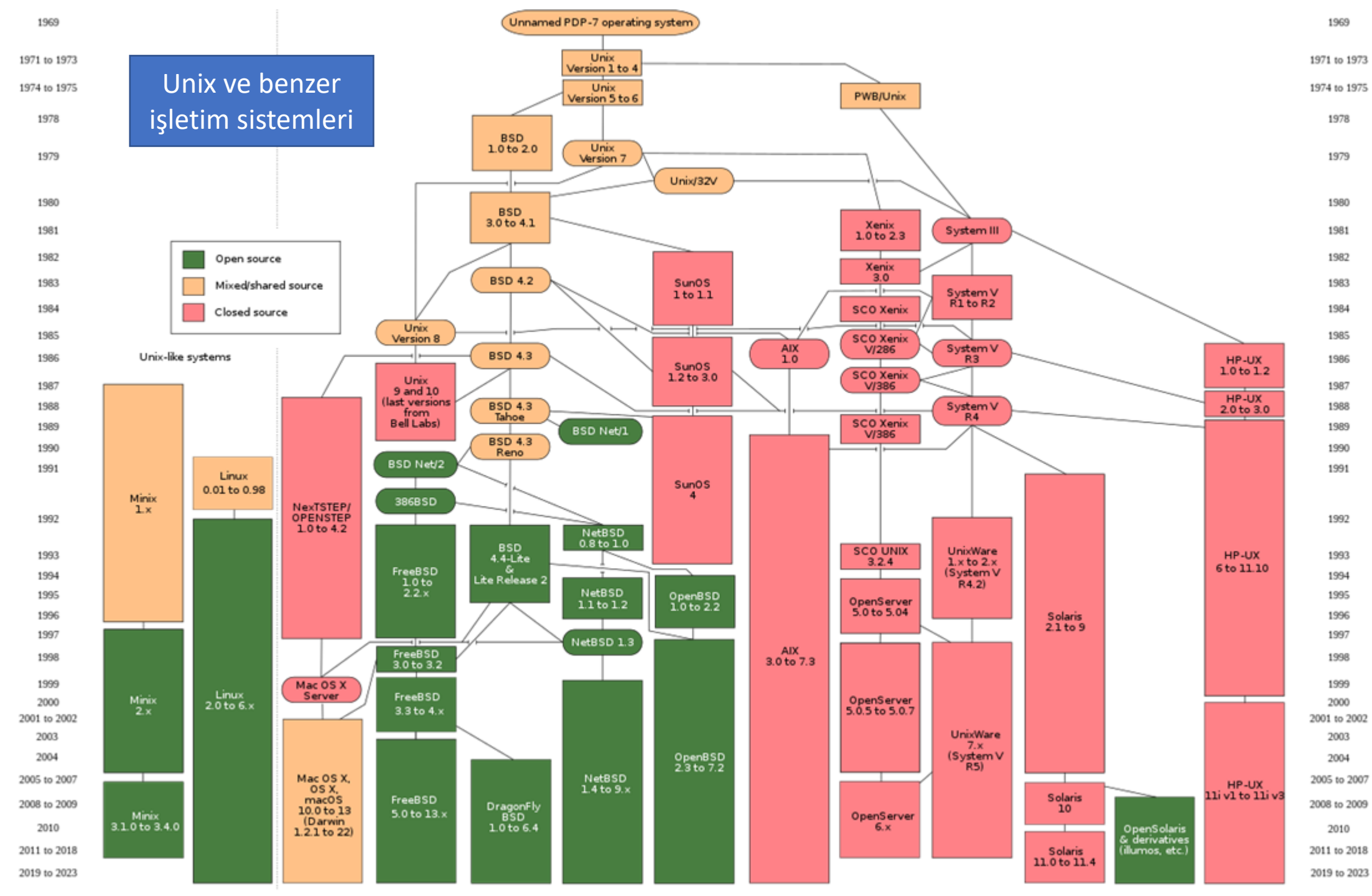
- İlk İşletim Sistemleri: Sadece Kütüphaneler
  - Toplu İşleme (Batch Processing)
- Kütüphanelerin Ötesindeki Konu: Koruma Problemi
  - Sistem çağrıları (trap ve return-from-trap buyrukları)
- Çoklu Programlama Çağı
  - Bellek Koruma
  - Eşzamanlılık sorunları
- Modern Çağa Giriş

  mainframe → minicomputer → kişisel bilgisayar (personal computer)

UNIX’in önemi

- Daha önceki MIT tarafından yürütülmüş Multics projesinden etkilenmiştir.
- Aynı zamanda C programlama dili için bir derleyici sağlanmıştır.
- UNIX'in yaygınlaşması, avukatların olaya dahil olması nedeniyle yavaşlamıştır.
- Windows tanıtılmış ve kişisel bilgisayar pazarını ele geçirmiştir (geriye doğru büyük bir sıçrama).
- Ardından, olaya Linux dahil olur.
- Mac OS X/macOS işletim sisteminin de temelinde UNIX vardır.
- Sonrasında, Windows da iyi fikirlerin çoğunu benimsemiştir.

Unix ve benzer işletim sistemleri



4. Soyutlama: İşlem

İşletim Sistemleri: Üç Basit Parça

«Birçok İşlemci» ilüzyonunu nasıl sağlarız?

- İşlemci sanallaştırma
  - İşletim sistemi, birçok sanal CPU'nun var olduğu yanılsamasını desteklemelidir.
- Zaman paylaşımı (Time sharing): Bir işlemi çalıştırmak, ardından onu durdurmak ve başka bir işlemi çalıştırmak.
  - Performans düşebilir.

İşlem (Process)

İşlem: Çalışan program

- Bir işlemi oluşturan parçalar:
  - Bellek (adres uzayı)
    - Buyruklar
    - Veri kısmı
  - Yazmaçlar (Registers)
    - Program sayacı (Program Counter) (PC)
    - Yığın işaretleyicisi (Stack pointer)

«Process» teriminin Türkçe karşılığı nedir?

Process API

- Bu tür bir API veya benzeri tüm modern işletim sistemlerinde mevcuttur (daha sonra daha detaylı incelenecektir)
  - **Oluşturma**
    - Bir programı çalıştırmak için yeni bir işlem oluşturma
  - **Yok etme**
    - Kontrolden çıkmış bir işlemi durdurma
  - **Bekleme**
    - Bir işlemin durmasını bekleme
  - **Çeşitli Kontroller**
    - Örneğin bir işlemi askıya alma ve sonra devam ettirme
  - **Durum bilgisi**
    - Bir işlemle ilgili durum bilgisi alma

İşlem Oluşturma

1. Belleğe (işlemin adres alanına) bir program kodu **YÜKLE (LOAD).**
   - Programlar başlangıçta *yürütülebilir* biçimde diskte bulunur.
   - İşletim sistemi yükleme işlemini aslında tembel olarak gerçekleştirir.
     - Kod veya veri parçalarını yalnızca programın yürütülmesi sırasında ihtiyaç duyulursa yüklemek.
2. Programın çalışma zamanı **yığını** tahsis edilir.
   - Yerel değişkenler, fonksiyon parametreleri ve dönüş adresi için yığını kullanın.
   - Yığın, argüman değerleri ile başlatılır → `main()` fonksiyonunun `argc` ve `argv` dizisi ile.

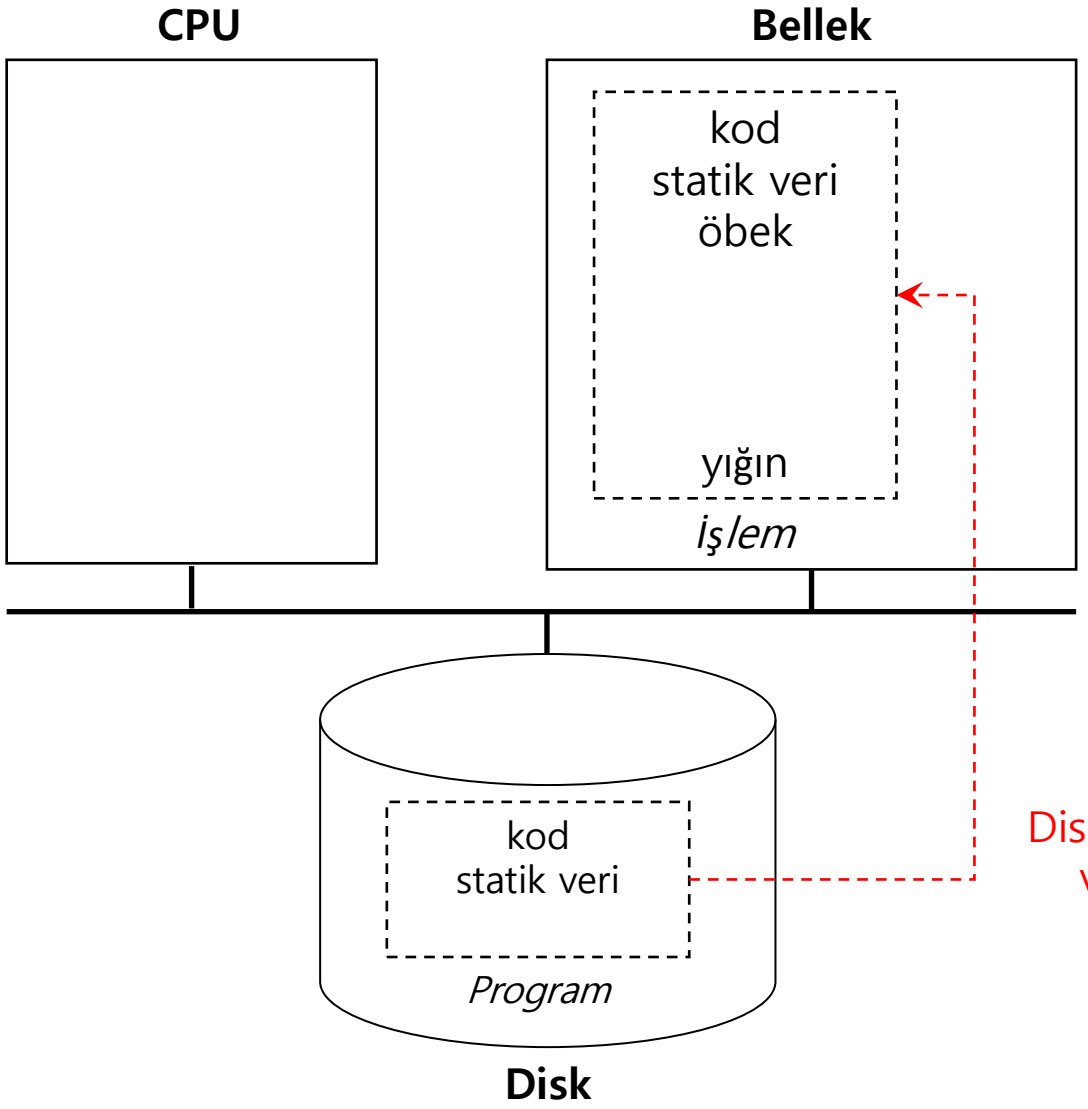
İşlem Oluşturma (Devam)

3. Programın öbeği (**heap**) oluşturulur.
   - Açıkça istenen dinamik olarak tahsis edilen veriler için kullanılır.
   - Programın `malloc()` çağrısı yaparak alan istemesi ve `free()` çağrısı yaparak alanı boşaltması.

4. İşletim sistemi diğer bazı başlatma görevlerini de yerine getirir.
   - girdi/çıktı (I/O) kurulumu
     - Varsayılan olarak her işlemin üç açık dosya tanıtıcısı (file descriptors) vardır.
     - Standart girdi, çıktı ve hata.

5. Giriş noktasında - yani `main()`'de - **programı başlat**.
   - İşletim sistemi, CPU'nun kontrolünü yeni oluşturulan işleme aktarır.

Yükleme: Program'dan İşleme

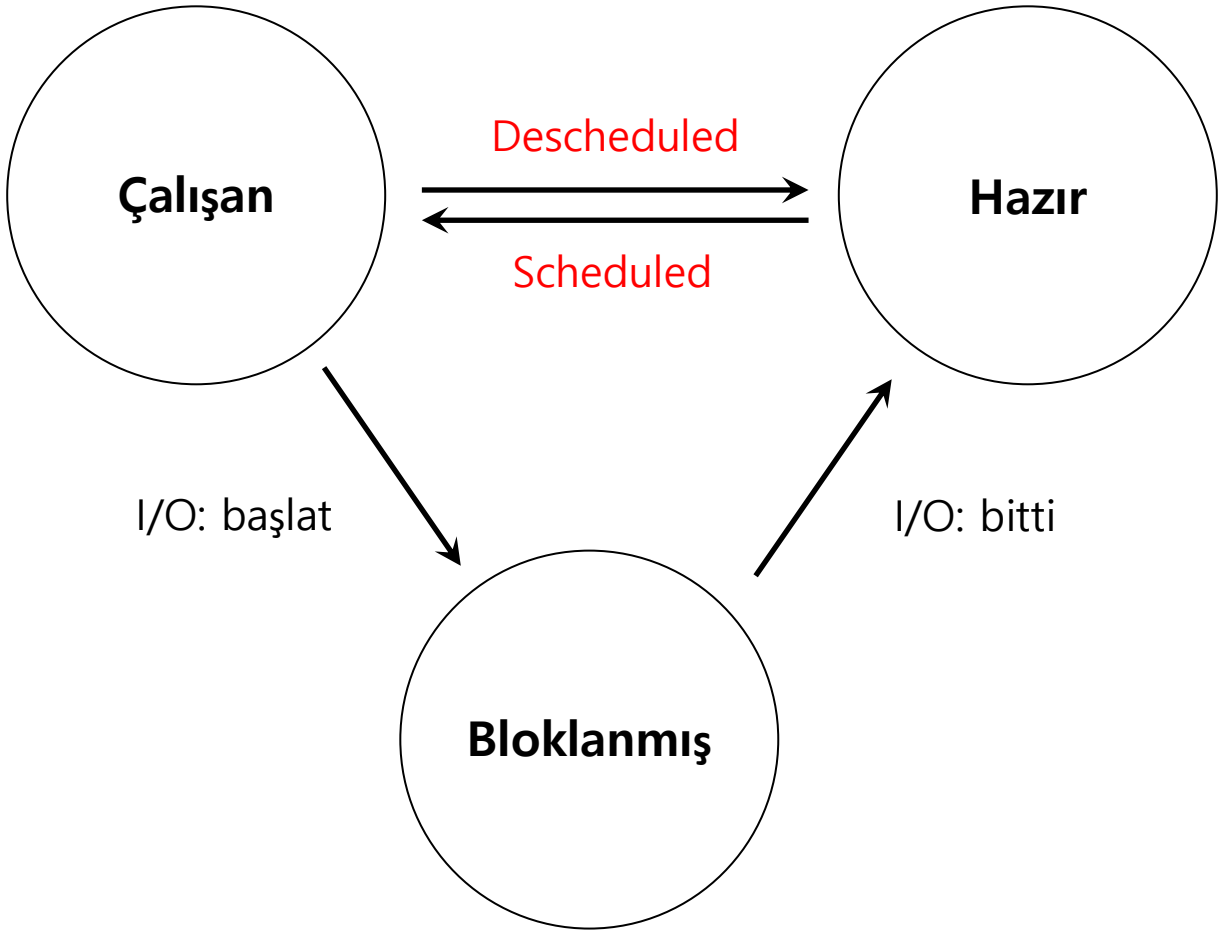


Yükleme:
Diskteki programı alır ve işlemin adres alanına yazar

İşlem Durumları

- Bir işlem üç durumdan (state) birinde bulunur:
  - **Çalışan**
    - İşlem CPU'da çalışıyordur.
  - **Hazır**
    - Bir işlem çalışmaya hazırdır, ancak bazı nedenlerden dolayı işletim sistemi onu şu anda çalıştırmamayı tercih etmiştir.
  - **Bloklanmış**
    - İşlem bir buyruk çalıştırmış örneğin bir I/O isteğinde bulunmuş ve bloklanmıştır. Bu sebeple başka bir işlem CPU'yu kullanabilir.

İşlem Durum Geçişleri



İşlem Kontrol Bloğu

- İşletim Sistemi, işlemlerle ilgili bir takım bilgileri tutmak için bazı temel veri yapılarına sahiptir.
  - **İşlem listesi**
    - Hazır işlemler
    - Bloklanmış işlemler
    - O an çalışan işlemler
  - **Yazmaç bağlamı**

- İşlem Kontrol Bloğu (PCB - Process Control Block)
  - C dilindeki struct yapısında her işlem ile ilgili bilgiler tutulur.

Örnek: xv6 işletim sistemi - proc

```
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
    int eip;   // Index pointer register
    int esp;   // Stack pointer register
    int ebx;   // Called the base register
    int ecx;   // Called the counter register
    int edx;   // Called the data register
    int esi;   // Source index register
    int edi;   // Destination index register
    int ebp;   // Stack base pointer register
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };
```

Örnek: xv6 işletim sistemi - proc (devam)

```
// the information xv6 tracks about each process
// including its register context and state
struct proc {
    char *mem;                  // Start of process memory
    uint sz;                    // Size of process memory
    char *kstack;               // Bottom of kernel stack
                                // for this process
    enum proc_state state;      // Process state
    int pid;                    // Process ID
    struct proc *parent;        // Parent process
    void *chan;                 // If non-zero, sleeping on chan
    int killed;                 // If non-zero, have been killed
    struct file *ofile[NOFILE]; // Open files
    struct inode *cwd;          // Current directory
    struct context context;     // Switch here to run process
    struct trapframe *tf;       // Trap frame for the
                                // current interrupt
};
```

5. Ara oyun: İşlem API’si

İşletim Sistemleri: Üç Basit Parça

fork() Sistem Çağrısı

- Yeni bir işlem oluşturma
  - Yeni oluşturulan işlem kendine ait adres uzayı, yazmaç ve program sayacına sahiptir.

p1.c

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

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}
```

fork() çağrısı örneği (devam)

(deterministik olmayan) sonuç

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

veya

```
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
```

wait() sistem çağrısı

- Bu sistem çağrısı yapılınca çocuk işlem çalışıp sonlanmadan işlem dönmez.

p2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
        rc, wc, (int) getpid());
    }
    return 0;
}
```

wait() sistem çağrısı (devam)

(deterministik) sonuç

```
prompt> ./p2
hello world (pid:29266)
hello, I am child (pid:29267)
hello, I am parent of 29267 (wc:29267) (pid:29266)
prompt>
```

exec() sistem çağrısı

- Çağıran programdan farklı bir programın çalıştırılması

p3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {                    // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {            // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
        char *myargs[3];
        myargs[0] = strdup("wc");           // program: "wc" (word count)
        myargs[1] = strdup("p3.c");         // argument: file to count
        myargs[2] = NULL;                   // marks end of array
        …
```

exec() sistem çağrısı (devam)

p3.c (devam)

```
…
        execvp(myargs[0], myargs); // runs word count
        printf("this shouldn't print out");
    } else {                       // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
            rc, wc, (int) getpid());
    }
    return 0;
}
```

Sonuç

```
prompt> ./p3
hello world (pid:29383)
hello, I am child (pid:29384)
29 107 1030 p3.c
hello, I am parent of 29384 (wc:29384) (pid:29383)
prompt>
```

Niye fork()/exec() birleşimini kullanıyoruz?

- Bir işlem oluşturmak gibi basit bir eylem için niye böyle garip bir arayüz tasarlanmış?

Yeniden yönlendirme ile önceki program

p4.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>

int
main(int argc, char *argv[]){
    int rc = fork();
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child: redirect standard output to a file
        close(STDOUT_FILENO);
        open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
        …
```

Yeniden yönlendirme ile önceki program (devam)

p4.c

```
        …
        // now exec "wc"...
        char *myargs[3];
        myargs[0] = strdup("wc");           // program: "wc" (word count)
        myargs[1] = strdup("p4.c");         // argument: file to count
        myargs[2] = NULL;                   // marks end of array
        execvp(myargs[0], myargs);          // runs word count
    } else {                          // parent goes down this path (main)
        int wc = wait(NULL);
    }
    return 0;
}
```

Sonuç

```
prompt> ./p4
prompt> cat p4.output
32 109 846 p4.c
prompt>
```

6. Mekanizma: Sınırlı Doğrudan Yürütme

CPU’yu verimli ve «**kontrollü**» bir şekilde nasıl sanallaştırırız?

- İşletim Sisteminin fiziksel CPU’yu zaman paylaşımı yöntemi ile paylaşması gerekir.
- İki sorun var:
  - **Performans**: Sisteme fazlaca bir miktar yük bindirmeden sanallaştırmayı nasıl gerçekleştirebiliriz?
  - **Kontrol**: CPU üzerindeki kontrolü kaybetmeden işlemleri nasıl çalıştırabiliriz?

Doğrudan Çalıştırma

- Doğrudan programı CPU üzerinde çalıştırmak.

İS	Program
1. İşlem listesine yeni bir girdi ekle 2. Program için bellek ayır 3. Programı bellekten yükle 4. Yığını `argc` / `argv` ile kur 5. Yazmaçları temizle 6. `main()` çağrısını yap	
	7. `main()` çalıştır 8. `return` ile `main()`'den dön
9. İşlem belleğini boşalt 10. İşlem listesinden kaldır	

Çalışan programlar için bir sınırlama yok ise, İşletim Sistemi hiçbir şeyi kontrol edemez ve sadece kütüphane olur.

Problem 1: Kısıtlanmış Operasyon

- Bir işlem kısıtlanmış (ayrıcalıklı) bir operasyon yapmak isterse ne olur?
  - Disk'e bir I/O isteğinde bulunmak
  - CPU ve bellek gibi sistem kaynaklarına daha fazla erişim hakkı isteği

- **Çözüm**: Korumalı olarak kontrol modunu değiştir
  - Kullanıcı modu: Uygulamalar donanım kaynaklarına tam erişime sahip değildir.
    Çekirdek (kernel) modu: İşletim sistemi makine kaynaklarına tam erişime sahiptir.

Sistem Çağrısı

- Çekirdeğin bazı kilit işlevselliği kullanıcı programlarına dikkatli bir şekilde sunması.
- Örneğin:
  - Dosya sistemine erişim
  - İşlem oluşturma ve yok etme
  - Diğer işlemler ile iletişim
  - Daha fazla bellek tahsis etme

Sistem Çağrısı (devam)

- **Trap** buyruğu
  - Çekirdeğe atla
  - Ayrıcalık seviyesini çekirdek moduna yükselt

- **Return-from-trap** buyruğu
  - Çağıran kullanıcı programına geri dön
  - Ayrıcalık seviyesini tekrar kullanıcı moduna düşür

Sınırlı Doğrudan Yürütme Protokolü

İS @ boot ederken (çekirdek mod)	**Donanım**
trap tablosunu ayarla	
	syscall işleyicisinin adresini hatırla

İS @ çalışırken (çekirdek mod)	**Donanım**	**Program (kullanıcı mod)**
İşlem listesinde girdi oluştur		
Program için bellek tahsis et		
Programı belleğe yükle		
Yığını argv ile kur		
Çekirdek yığınını reg/PC ile doldur		
return-from -trap		
	regs değerlerini çek.yığından getir	
	Kullanıcı moduna geç	
	main' e atla	
		main() çalıştır
		…
		trap çağrısı yap

Sınırlı Doğrudan Yürütme Protokolü (devam)

İS @ çalışırken (çekirdek mod)	Donanım	Program (kullanıcı mod)
	(Devam)	
	regs değerlerini çek.yığına sakla Çekirdek moda geç trap tablosuna bak	
trap'ı işle Syscall işini yerine getir **return-from-trap**		
	regs değerlerini çek.yığından getir Kullanıcı moda geç Jtrap sonrasındaki PC'ye atla	
		... Main'den dön trap (`exit()`)
İşlem belleğini sil İşlem listesinden kaldır		

Problem 2: İşlemler Arasında Değişim

- İşletim Sistemi çalışan işlemleri değiştirmek için CPU'nun kontrolünü nasıl tekrar eline alır?
  - İşbirlikçi yaklaşım: **Sistem çağrılarını bekle**
  - İşbirlikçi olmayan yaklaşım: **İşletim Sistemi kontrolü kendi ele alır**

İşbirlikçi yaklaşım: Sistem çağrılarını bekle

- `yield` veya benzer **sistem çağrıları** yaparak işlemler düzenli olarak CPU'yu bırakırlar.
  - İşletim sistemi başka bir işlemi çalıştırmak için seçer.
  - Uygulamalar yanlış bir şey yaptıklarında da kontrolü işletim sistemine bırakmak zorunda kalırlar.
    - Sıfır ile bölme
    - Erişim izni olmayan bir bellek alanına yazma isteğinde bulunma

Bir işlem sonsuz döngüde kalırsa ne yaparız?
→ Makineyi tekrar başlatmak

İşbirlikçi Olmayan Yaklaşım: İşletim Sistemi Kontrolü Alır

- **Zaman Kesmesi (Timer Interrupt)**
  - Henüz boot edilirken İşletim Sistemi saati (zamanı) başlatır.
  - Saat belirlenmiş her milisaniyede bir kesme (interrupt) üretir.
  - Kesme olduğunda:
    - O an çalışan işlem durdurulur.
    - Gerekli durum bilgileri kaydedilir.
    - Daha önce ayarlanmış bir kesme işleyicisi (interrupt handler) çalıştırılır.

Zaman kesmesi İşletim Sisteminin CPU kontrolünü tekrar eline almasını sağlar.

Bğlam Bilgisini Kaydetmek ve tekrar iade etmek

- Scheduler (planlayıcı) bir karar verir:
  - O an çalışan işleme mi devam edilecek yoksa başka bir ileme mi geçilecek?
  - Eğer karar başka bir işleme geçmek ise, context switch gerçekleştirilir.

Context Switch

- Düşük seviye assembly kodu
  - O anki işleme ait yazmaç değerlerini çekirdek yığınına kaydet
    - Genel amaçlı yazmaçlar
    - PC
    - Çekirdek yığın işaretleyicisi
  - Biraz sonra çalıştırılacak işleme ait yazmaç değerlerini çekirdek yığınından geri yükle.
  - Biraz sonra çalıştırılacak işleme ait çekirdek yığınına geçiş yap.

Sınırlı Doğrudan Yürütme Protokolü (Timer interrupt)

İS @ boot ederken (çekirdek modu)	Donanım
trap tablosunu ayarla	syscall işleyicisinin ve zaman kesmesi işleyicisinin adresini hatırla
kesme saatini başlat	saati başlat CPU'yu X ms sonra kes

İS @ çalışırken (çekirdek mod)	Donanım	Program (kullanıcı mod)
		İşlem A ...
	zaman kesmesi regs(A)'yı çek-yığın(A)'a kaydet çekirdek moda geç trap işleyiciye atla	

Sınırlı Doğrudan Yürütme Protokolü (Timer interrupt)

İS @ çalışırken (çekirdek mod)	Donanım	Program (kullanıcı mod)
	(Devam)	
Trap'ı işle switch() rutinini çağır regs(A)'yı proc-struct(A)'a kaydet regs(B)'yi proc-struct(B)'den yükle çek.yığın(B)'ye geç **return-from-trap (B'ye)**		
	regs(B)'yi çek.yığın(B)'den yükle kullanıcı moduna geç B'nin PC'sine atla	
		İşlem B ...

xv6 İşletim Sistemi
Context Switch Kodu

```
1  # void swtch(struct context **old, struct context *new);
2  #
3  # Save current register context in old
4  # and then load register context from new.
5  .globl swtch
6  swtch:
7          # Save old registers
8          movl 4(%esp), %eax          # put old ptr into eax
9          popl 0(%eax)                # save the old IP
10         movl %esp, 4(%eax)          # and stack
11         movl %ebx, 8(%eax)          # and other registers
12         movl %ecx, 12(%eax)
13         movl %edx, 16(%eax)
14         movl %esi, 20(%eax)
15         movl %edi, 24(%eax)
16         movl %ebp, 28(%eax)
17
18         # Load new registers
19         movl 4(%esp), %eax          # put new ptr into eax
20         movl 28(%eax), %ebp         # restore other registers
21         movl 24(%eax), %edi
22         movl 20(%eax), %esi
23         movl 16(%eax), %edx
24         movl 12(%eax), %ecx
25         movl 8(%eax), %ebx
26         movl 4(%eax), %esp          # stack is switched here
27         pushl 0(%eax)               # return addr put in place
28         ret                         # finally return into new ctxt
```

Eşzamanlılık ile ilgili Endişeler?

- Kesme veya trap işlenirken başka bir kesme olursa ne olur?
- İşletim Sistemi bu durumları ele alır:
  - Kesme işlenirken kesmeleri **devre dışına** alabilir
  - Kendi veri yapılarına erişim sağlanırken sofistike bazı **kilitleme** yöntemlerini kullanabilir.