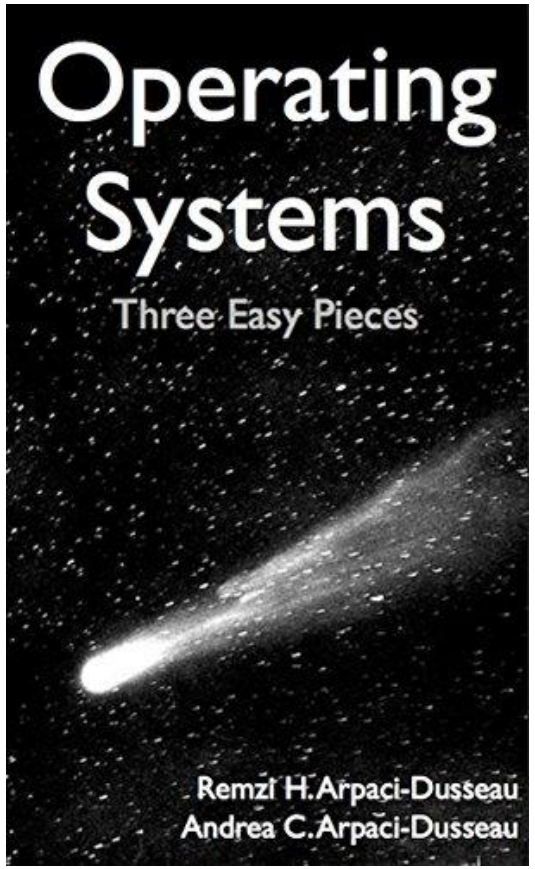
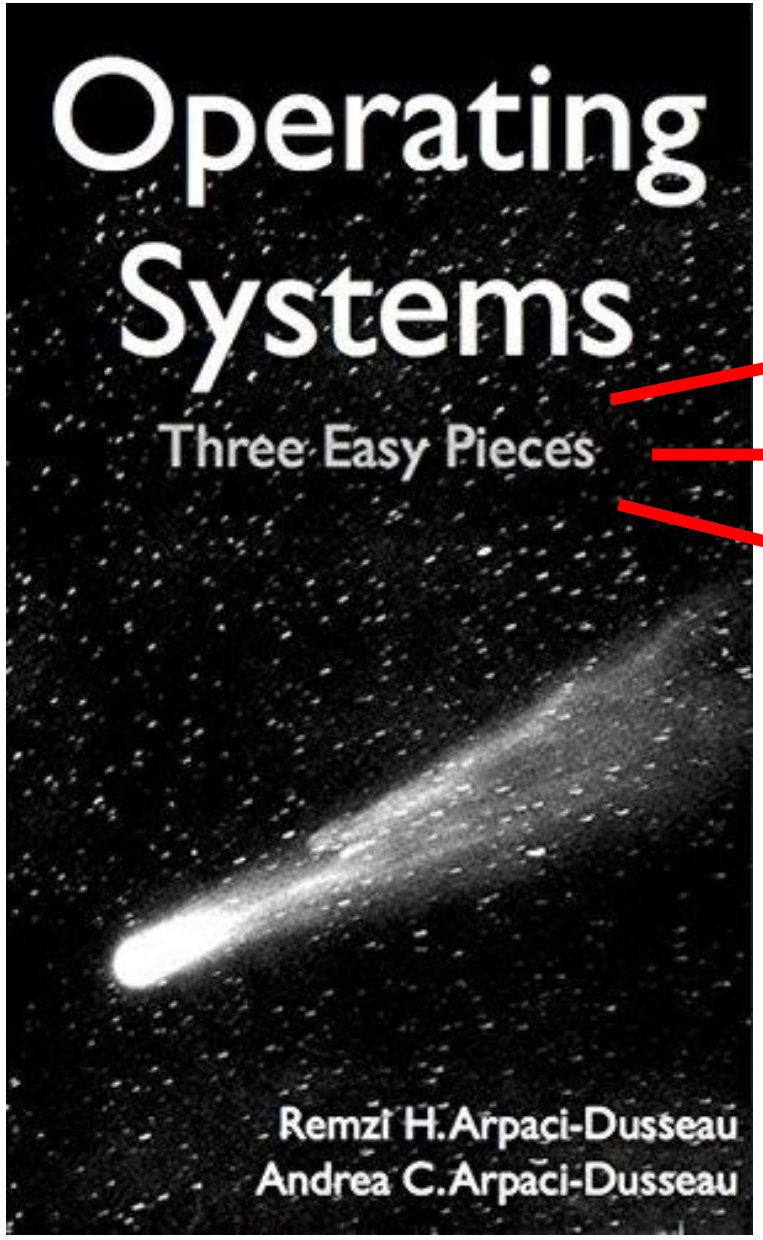


İşletim Sistemleri

Prof. Dr. Kemal Bıçakcı



Sanallaştırma (Virtualization)

Concurrency (Eşzamanlılık)

Kalıcılık (Persistency)

İşletim Sistemlerine Giriş

- Bir bilgisayar programı çalışınca neler olur?
- Çalışan bir program, sadece tek bir iş yapar. Nedir bu iş?

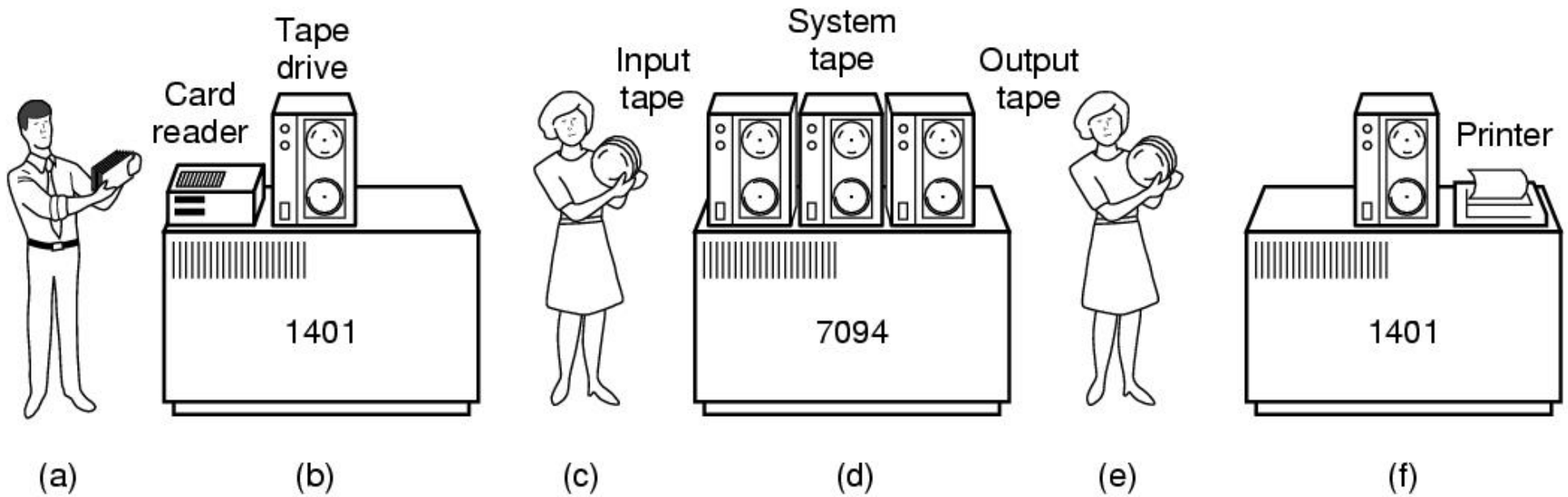
Bir Programın Çalışması

Çalışan bir program, buyrukları (instructions) yürütür:

1. Fetch: İşlemci bellekten bir buyruk alır.
2. Decode: Bunun hangi buyruk olduğunu bulur.
3. Execute: Örnek: iki sayıyı topla, belleğe eriş, bir koşulu kontrol et, işleve atla, vb.
4. İşlemci bir sonraki buyruğa geçer ve bu böyle program sonlanana kadar devam eder.

Von Neuman mimari modeli (Veri ve Program aynı bellekte tutulur)

Toplu (Batch) Sistem



İşletim Sistemi (İS)

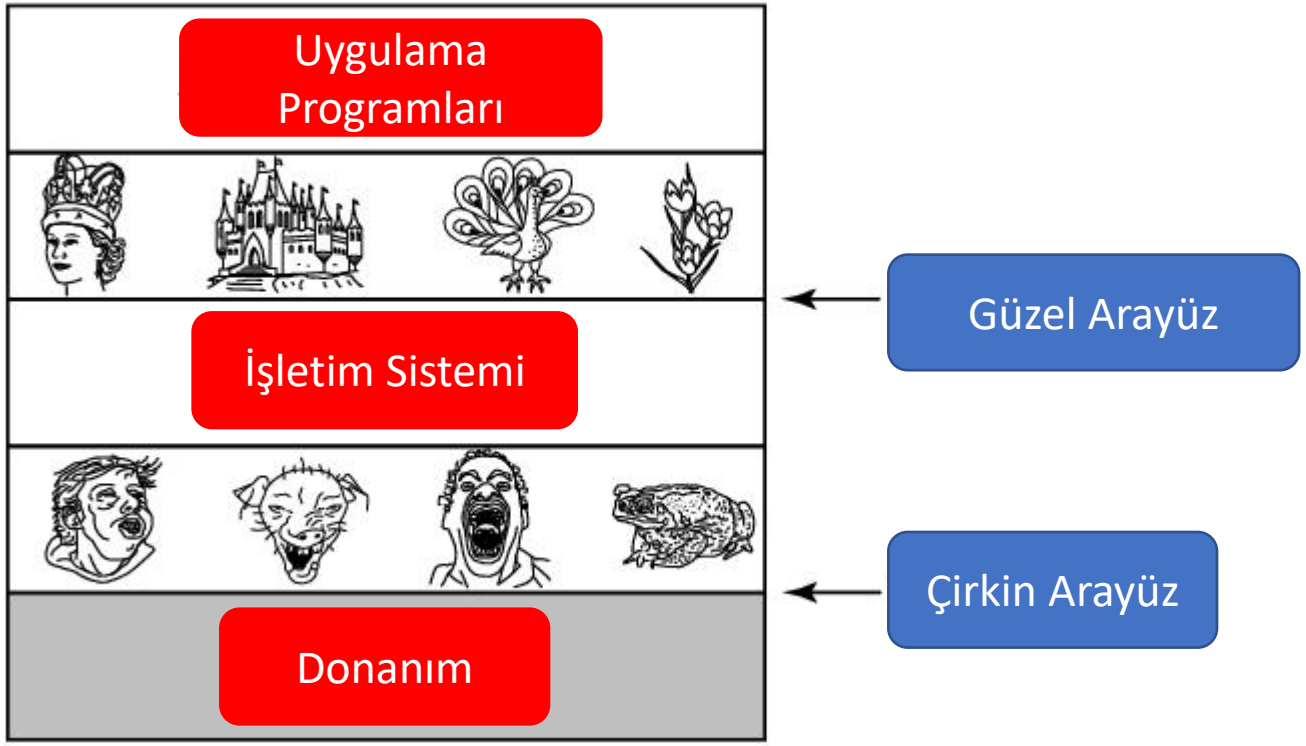
- Programları çalıştırmayı kolaylaştırır.
- Programların kaynakları paylaşmalarını sağlar.
- Programların G/Ç cihazları ile etkileşimini mümkün hale getirir.

İS, sistemin doğru ve verimli işlemesinden ve kolay kullanılmasından sorumludur.

İS = Kaynak Yöneticisi + Genişletilmiş (Sanal) Makine

İşletim Sistemi =
Kaynak Yöneticisi + Genişletilmiş (Sanal) Makine

Genişletilmiş Makine



Sanallaştırma (Virtualization)

- İşletim sistemi fiziksel bir kaynağı alır ve onu sanal bir biçime dönüştürür.
  - Fiziksel kaynak: İşlemci, Bellek, Disk …
- Sanal form, daha genel, güçlü ve kullanımı kolaydır.
- Bazen işletim sistemini sanal makine olarak adlandırırız.

(farklı kullanım!: bu terim, bir bilgisayarda birden fazla işletim sistemi çalıştırmak amacıyla geliştirilmiş fiziksel bir bilgisayarı emüle eden yazılım sistemleri (ör: VMware, Virtual Box) için de kullanılmaktadır.)

Sistem Çağrısı

- Sistem çağrısı, kullanıcının (programcının) işletim sistemine ne yapması gerektiğini söylemesine izin verir.
- Benzer terimler: Arabirim, API, standart kütüphane
- Tipik bir işletim sistemi, birkaç yüz sistem çağrısı sunar. Örneğin:
  - Programları çalıştırmak için
  - Belleğe erişim için
  - G/Ç cihazlara erişim için

Kaynak Yöneticisi

- İşletim sistemi CPU (işlemci), bellek ve disk gibi kaynakları yönetir.
- İşletim sistemi, çalıştırılan birçok programın CPU'yu paylaşmasına izin verir.
- Çalışan programların kendi buyruklarına ve verilerine erişmeleri için bellek paylaşımını da destekler.
- Farklı programlar aynı cihazları kullanır. Bu sebeple disk ve diğer G/Ç cihazları da paylaştırır.

İşlemciyi Sanallaştırma

- Sistem çok sayıda sanal işlemciye sahiptir.
- İşletim sistemi, tek bir işlemciyi görünüşte sonsuz sayıdaki işlemciye dönüştürür.
- Birçok programın görünüşte aynı anda çalışmasına izin verir.

(Not: Aksi belirtilmediği sürece bir bilgisayarda sadece bir işlemci olduğu varsayılabilir.)

İşlemciyi Sanallaştırma (2)

```
1       #include <stdio.h>
2       #include <stdlib.h>
3       #include <sys/time.h>
4       #include <assert.h>
5       #include "common.h"
6
7       int
8       main(int argc, char *argv[])
9       {
10              if (argc != 2) {
11                      fprintf(stderr, "usage: cpu <string>\n");
12                      exit(1);
13              }
14              char *str = argv[1];
15              while (1) {
16                      Spin(1); // Repeatedly checks the time and
                                 returns once it has run for a second
17                      printf("%s\n", str);
18              }
19              return 0;
20      }
```

Basit örnek (`cpu.c`): Döngü içinde devamlı ekrana değer yazan program

İşlemciyi Sanallaştırma (3)

Programı çalıştırmanın sonucu (1):

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
^C
prompt>
```

Sonsuza kadar çalışır, “Control-c” ile programı durdurabiliriz.

İşlemciyi Sanallaştırma (4)

Programı çalıştırmanın sonucu (2):

```
prompt> ./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
C
B
D
...
```

Sadece bir tane işlemciye sahip olmamıza rağmen, dört program da aynı anda çalışıyor gözüküyor.

Mekanizma ve Politika

- Programların aynı anda çalıştırılması pek çok soruyu gündeme getirir.
  - Örnek: Çalıştırmak için programlardan hangisini seçmeliyiz?
- Bu sorular İşletim Sisteminin uyguladığı politika ile cevap bulur.
- Öte yandan, İşletim Sisteminin bu yeteneği elde etmesi bazı mekanizmaları gerçekleştirmesiyle mümkün olur.

Belleği Sanallaştırma

- Fiziksel bellek, bir bayt dizisidir.
- Bir program, tüm veri yapılarını bellekte tutar.
- Belleği oku (Read Memory): Verilere erişebilmek için bir adres belirtilmesi gerekir.
- Belleğe yaz (Write Memory) (veya güncelle): Adrese ek olarak yazılacak veriler de belirtilir.
- Programın buyruklarının da bellekte tutulduğu unutulmamalıdır.
  - Bu yüzden program çalışırken belleğe pek çok defa erişilir.

Belleği Sanallaştırma (2)

- Belleğe erişen bir program (`mem.c`)

```
1       #include <unistd.h>
2       #include <stdio.h>
3       #include <stdlib.h>
4       #include "common.h"
5
6       int
7       main(int argc, char *argv[])
8       {
9               int *p = malloc(sizeof(int));  // a1: allocate some
                                                  memory
10              assert(p != NULL);
11              printf("(%d) address of p: %08x\n",
12                      getpid(), (unsigned) p);  // a2: print out the
                                                     address of the memory
13              *p = 0;  // a3: put zero into the first slot of the memory
14              while (1) {
15                      Spin(1);
16                      *p = *p + 1;
17                      printf("(%d) p: %d\n", getpid(), *p); // a4
18              }
19              return 0;
20      }
```

Belleği Sanallaştırma (3)

- `mem.c` programının çıktısı:

```
prompt> ./mem
(2134) memory address of p: 00200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

- Yeni ayrılan bellek `00200000` adresindedir.
- Değer güncellenir ve sonuç ekrana yazılır.

Belleği Sanallaştırma (4)

- `mem.c` programını birkaç defa çalıştırırsak:

```
prompt> ./mem &; ./mem &
[1] 24113
[2] 24114
(24113) memory address of p: 00200000
(24114) memory address of p: 00200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
...
```

- Sanki çalışan her programın **kendine özel bir belleği** var gibi!
  - Her programa aynı adresteki bellek alanı ayrılmış (gibi).
  - Öte yandan, her biri `00200000` adresindeki değeri birbirinden bağımsız güncellemekte.

Önemli Not:
Aynı adresin kullanıldığını gözlemlemek için «address-space randomization» özelliğinin kapatılması gerekir.

Belleği Sanallaştırma (5)

- Çalışan her program, kendine özel sanal adres alanına erişir.
- İşletim sistemi, adres alanını fiziksel belleğe eşler.
- Çalışan bir program içindeki bir bellek referansı (memory reference), diğer işlemlerin adres alanını etkilemez.
- Fiziksel belleğin, işletim sistemi tarafından yönetilen paylaşılan bir kaynak olduğunu tekrar hatırlayalım.

3 Basit Parça

- Sanallaştırma
- Eşzamanlılık
- Kalıcılık

Eşzamanlılık (Concurrency) Problemi

- İşletim sistemi birçok şeyi aynı anda yürütürken (önce bir işlemi, sonra diğerini çalıştırmak gibi) bazı problemler ortaya çıkar.
- Çok iş parçacıklı (multithreaded) programlar da eşzamanlılık problemlerine yol açmaktadır.

Eşzamanlılık Örneği

Çok İş Parçacıklı bir program (`thread.c`)

```
1       #include <stdio.h>
2       #include <stdlib.h>
3       #include "common.h"
4
5       volatile int counter = 0;
6       int loops;
7
8       void *worker(void *arg) {
9               int i;
10              for (i = 0; i < loops; i++) {
11                      counter++;
12              }
13              return NULL;
14      }
15
16      int
17      main(int argc, char *argv[])
18      {
19              if (argc != 2) {
20                      fprintf(stderr, "usage: threads <value>\n");
21                      exit(1);
22              }
```

Eşzamanlılık Örneği (2)

Çok İş Parçacıklı bir program (`thread.c`)

```
1       #include <stdio.h>
2       #include <stdlib.h>
3       #include "common.h"
4
5       volatile int counter = 0;
6       int loops;
7
8       void *worker(void *arg) {
9               int i;
10              for (i = 0; i < loops; i++) {
11                      counter++;
12              }
13              return NULL;
14  }
15  ...
```

Eşzamanlılık Örneği (3)

```
16      int
17      main(int argc, char *argv[])
18      {
19              if (argc != 2) {
20                      fprintf(stderr, "usage: threads <value>\n");
21                      exit(1);
22              }
23              loops = atoi(argv[1]);
24              pthread_t p1, p2;
25              printf("Initial value : %d\n", counter);
26
27              Pthread_create(&p1, NULL, worker, NULL);
28              Pthread_create(&p2, NULL, worker, NULL);
29              Pthread_join(p1, NULL);
30              Pthread_join(p2, NULL);
31              printf("Final value : %d\n", counter);
32              return 0;
33      }
```

- Ana program iki **iş parçacığı** üretir.
  - İş parçacığı: aynı bellek alanında çalışan ayrı akışlar. Her iki iş parçacığı da `worker()` isimli bir fonksiyonu çalıştırır.
  - `worker()` : sayaç değerini bir artırır.

Eşzamanlılık Örneği (4)

- `loops` değişkeni her iş parçacığının paylaşılan sayacı döngü içinde kaç kere artıracağını belirler.
  - `loops`: 1000.

```
prompt> gcc -o thread thread.c -Wall -pthread
prompt> ./thread 1000
Initial value : 0
Final value : 2000
```

  - `loops`: 100000.

```
prompt> ./thread 100000
Initial value : 0
Final value : 143012 // huh??
prompt> ./thread 100000
Initial value : 0
Final value : 137298 // what the??
```

Niye Böyle bir şey oldu?

- Paylaşılan bir sayacın değerini artırmak üç farklı buyruğun çalıştırılmasını gerektirir:
  - Sayacın değerini bellekten yazmaca yükle.
  - Değerini bir artır.
  - Belleğe geri yaz.
- Bu üç buyruk atomik (parçalanamaz) olarak yürütülmez.
- Sonuçta, Eşzamanlılık Problemi oluşur.

Kalıcılık (Persistence)

- DRAM gibi bellek üniteleri, değerleri geçici olarak depolar.
- Verileri kalıcı olarak depolamak için donanım ve yazılıma ihtiyaç vardır.

Donanım: Sabit sürücü, katı hal sürücüleri (SSD'ler) gibi G/Ç cihazları.

Yazılım: Dosya sistemi (File system) diski yönetir ve kullanıcının oluşturduğu dosyaları depolamaktan sorumludur.

Kalıcılık (2)

- “hello world” dizisini içeren bir dosya (`/tmp/file`) oluşturmak

```
1       #include <stdio.h>
2       #include <unistd.h>
3       #include <assert.h>
4       #include <fcntl.h>
5       #include <sys/types.h>
6
7       int
8       main(int argc, char *argv[])
9       {
10              int fd = open("/tmp/file", O_WRONLY | O_CREAT
                                | O_TRUNC, S_IRWXU);
11              assert(fd > -1);
12              int rc = write(fd, "hello world\n", 13);
13              assert(rc == 13);
14              close(fd);
15              return 0;
16      }
```

`open()`, `write()`, ve `close()` sistem çağrıları işletim sisteminin bir parçası olan dosya sistemine yönlendirilir.

Kalıcılık (3) – Diske Yazma İşlemi

Biraz bahsedelim ama önce gözlerini kapatacağınıza söz vermeniz gerekiyor; evet o kadar tatsız konular:

- Verilerin diskte nerede olduğunu bul
- Dosya sisteminin çeşitli yapılarında bu durumu kaydet.
- Depolama aygıtına uygun G/Ç isteklerini gönder.
- …

Cihaz sürücüsü (device driver) yazan herkesin bildiği gibi, bir cihazın sizin adınıza bir şeyler yapması karmaşık ve ayrıntılı bir sürecin işletilmesini gerektirir.

Şanslıyız ki, tüm bu tatsız işleri bizim yerimize yapan bir İşletim Sistemi var ☺