Bölüm 3: İşlemler

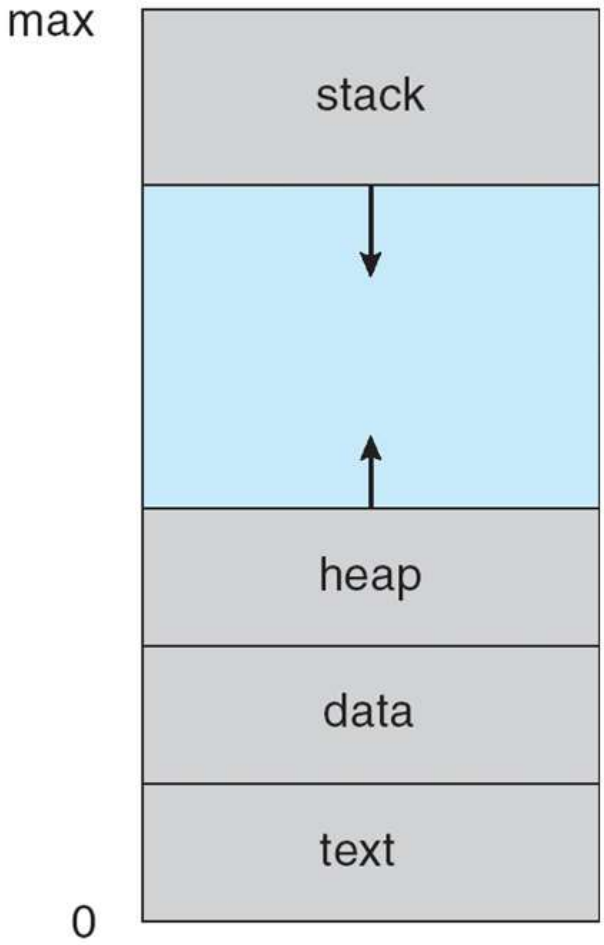
Bölüm 3: İşlemler

- İşlem Kavramı
- İşlem Zamanlaması (Process Scheduling)
- İşlemler Üzerindeki Faaliyetler
- İşlemler Arası İletişim (Interprocess Communication)
- IPC Sistemi Örnekleri
- İstemci-Sunucu Sistemlerde İletişim

İşlem Kavramı

- İşletim sistemleri farklı tipte programlar çalıştırabilir:
  - Toplu iş sistemleri - iş
  - Zaman paylaşımlı sistemler (time-shared systems) – kullanıcı programları veya görevleri
- Ders kitabı **iş (job)** ve **işlem (process)** kelimelerini kabaca birbirleri yerine kullanabilmekte
- İşlem – çalışmakta olan bir programdır
- İşlemler aşağıdakileri içermelidir:
  - **program sayacı (program counter)**
  - **yığın (stack)**
  - **veri bölümü (data section)**

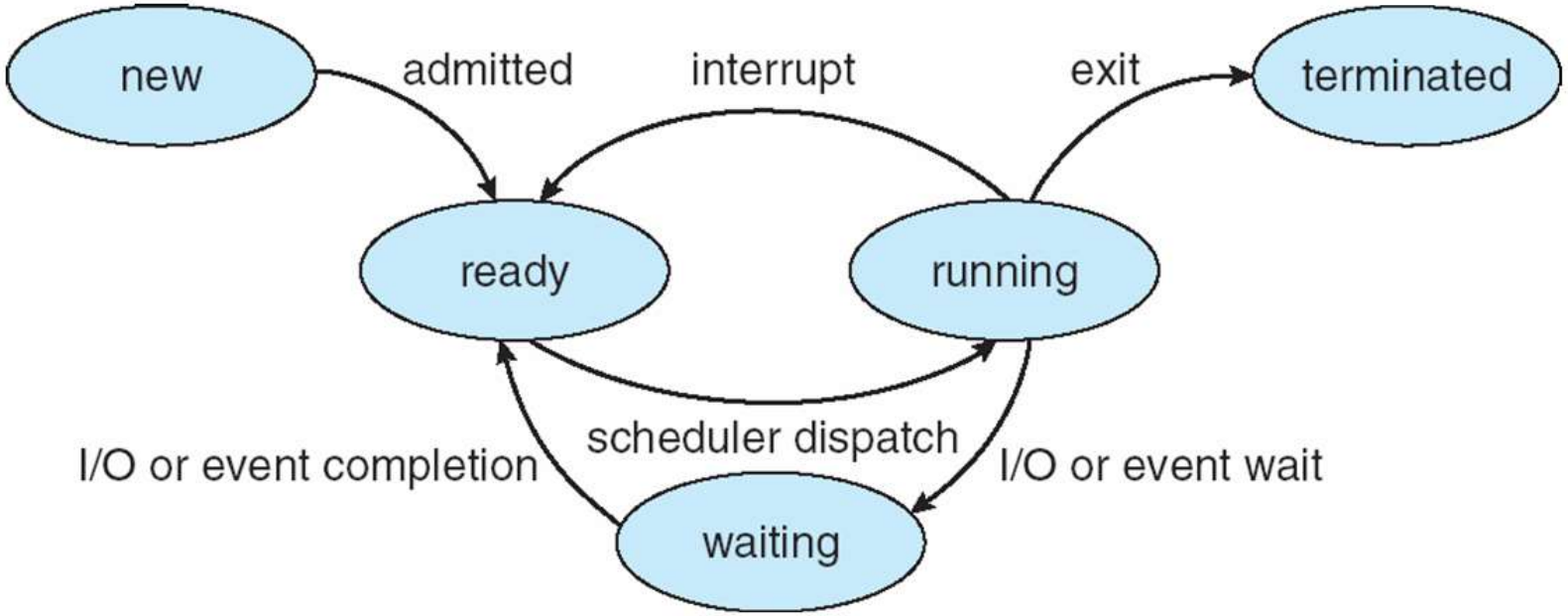
Hafızadaki İşlemler



İşlem Durumu

- Bir işlem çalıştırılırken durumunu (state) değiştirir
  - **yeni (new):** İşlem oluşturuldu
  - **çalışıyor (running):** İşlem komutları çalıştırılıyor
  - **bekliyor (waiting):** İşlem bir olayın gerçekleşmesini bekliyor
  - **hazır (ready):** İşlem bir işlemciye atanmayı bekliyor
  - **sonlandırılmış (terminated):** İşlem çalışmayı bitirmiş

İşlem Durumlarını Gösteren Şema



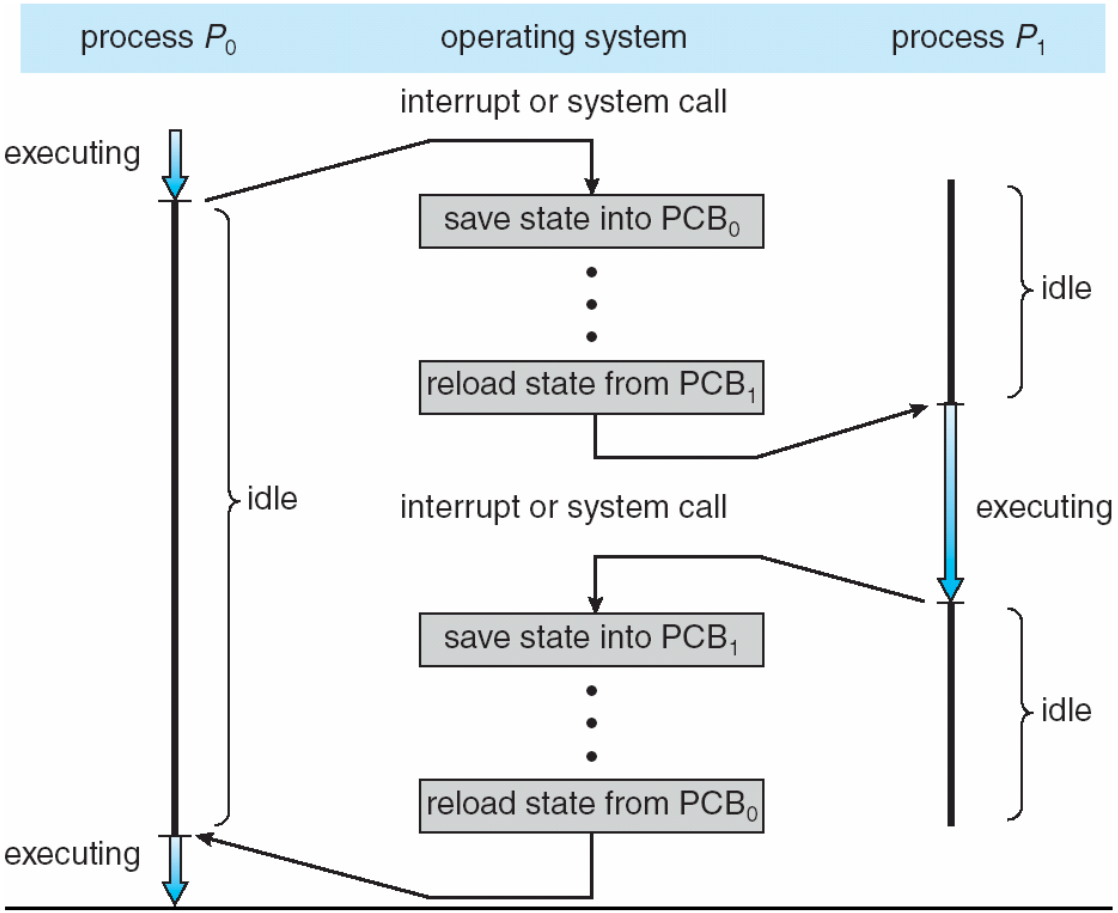
İşlem Kontrol Bloğu (PCB)

Herhangi bir işlem ile ilişkili bilgiler **(process control block)**

- İşlem durumu
- İşlem sayacı
- CPU yazmaçları (CPU registers)
- CPU zamanlama bilgileri
- Hafıza yönetim bilgileri
- Hesap (accounting) bilgileri
- I/O durum bilgileri

İşlem Kontrol Bloğu (PCB)

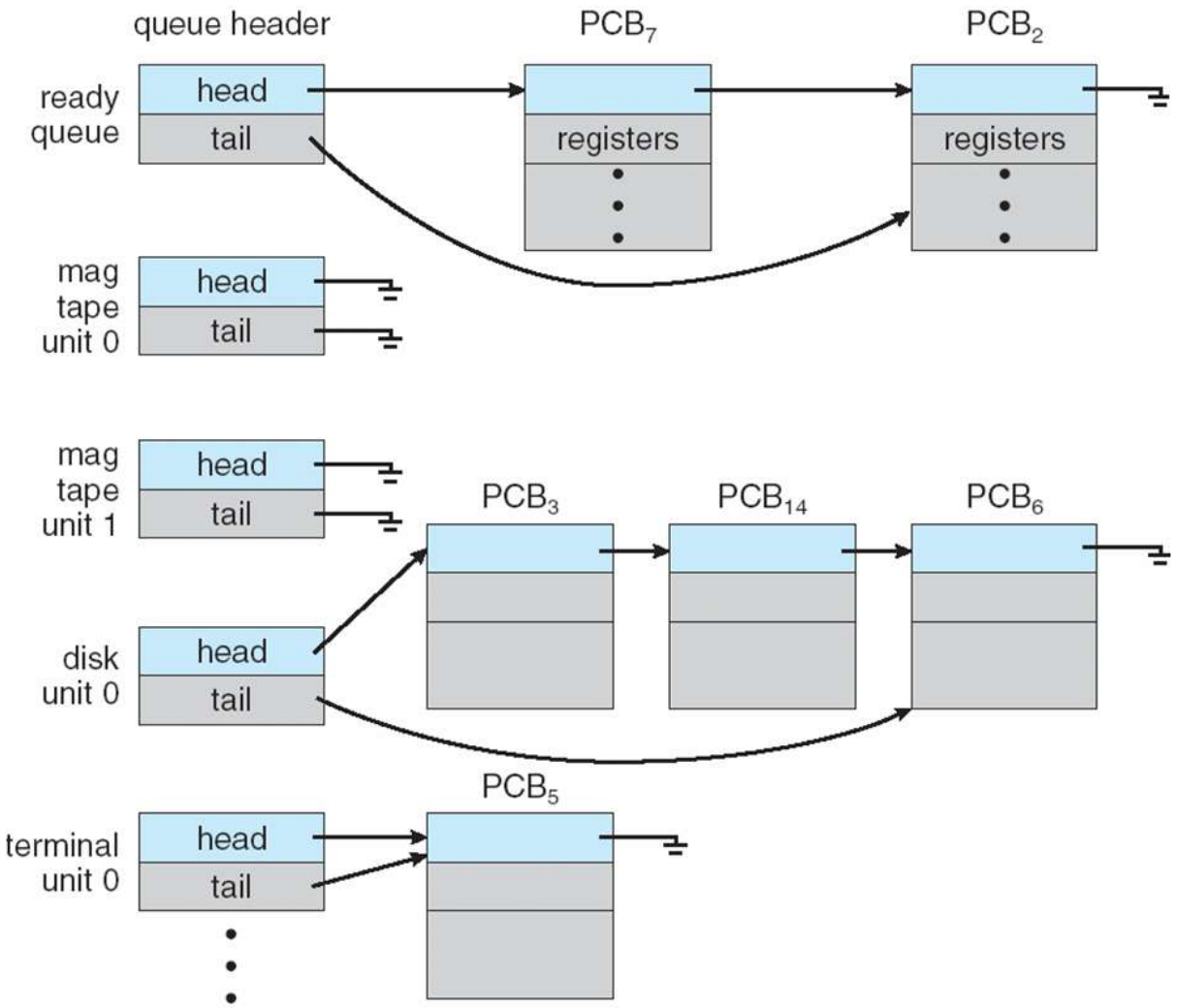
CPU İşlemden İşleme Geçiş Yapar



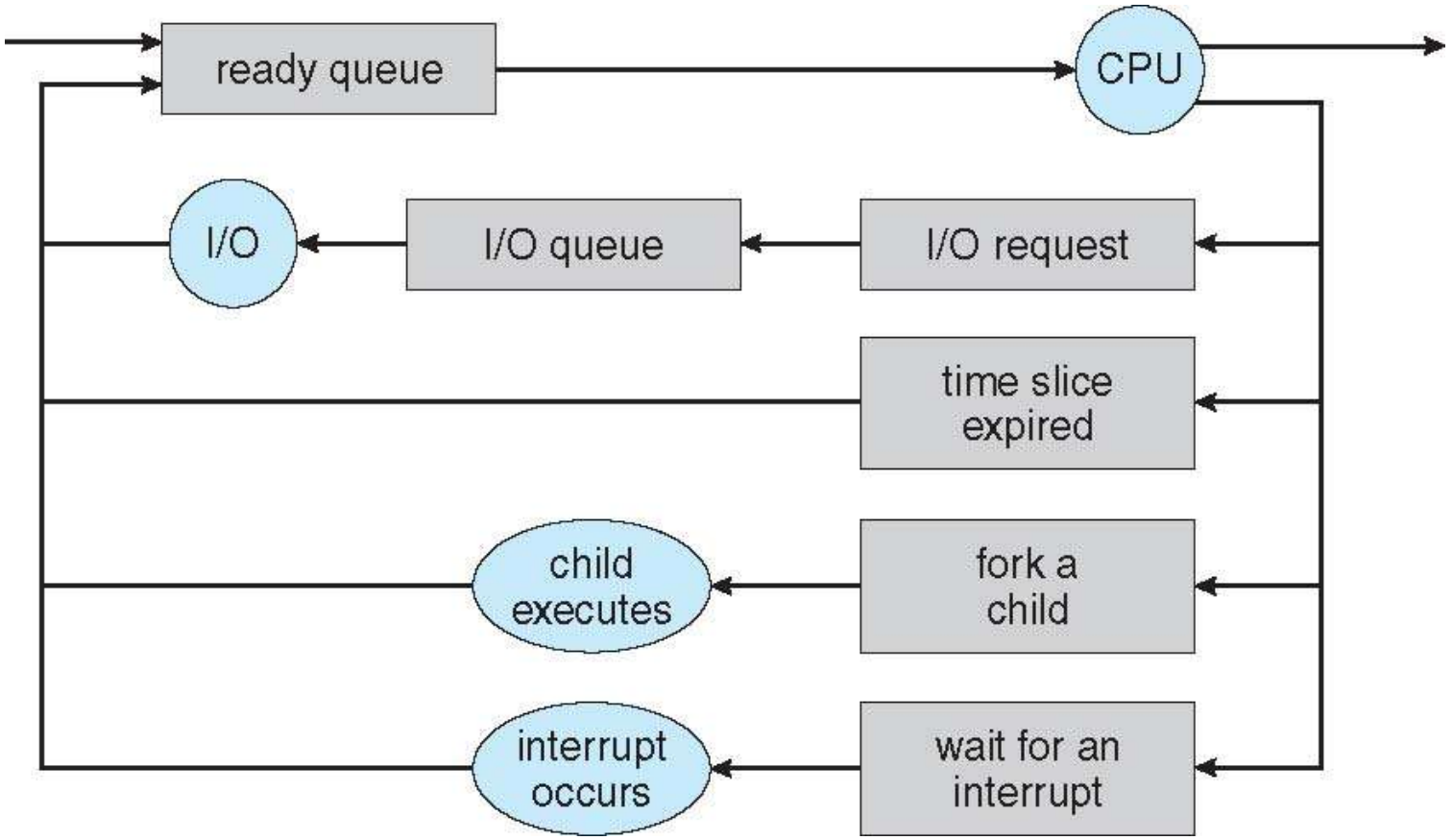
İşlem Zamanlama Kuyrukları

- Process Scheduling Queues
- **İş kuyruğu (job queue)** – sistemdeki tüm işlemlerin kümesi
- **Hazır kuyruğu (ready queue)** – ana hafızadaki, tüm işlemlerin kümesi - çalışmaya hazır ve çalıştırılmayı bekleyen
- **Cihaz kuyrukları (device queues)** – bir I/O cihazını kullanmayı bekleyen işlemler kümesi
- İşlemler değişik kuyruklar arasında geçiş yapar

Hazır Kuyruğu ve Bazı I/O Cihaz Kuyrukları



İşlem Zamanlaması Gösterimi



Zamanlayıcılar

- **Uzun vadeli zamanlayıcılar (long-term scheduler)**
  - veya iş zamanlayıcısı (job scheduler)
  - hangi işlemlerin hazır kuruğuna (ready queue) alınması gerektiğine karar verir
- **Kısa vadeli zamanlayıcılar (short-term scheduler)**
  - veya CPU zamanlayıcısı
  - sıradaki hangi işlemin CPU tarafından çalıştırılacağına karar verir

Zamanlayıcılar (devam)

- Kısa vadeli zamanlayıcı çok sık çalıştırılır (milisaniye) ⇒ hızlı çalışmalı
- Uzun vadeli zamanlayıcı çok sık çalıştırılmaz (saniye, sakika) ⇒ yavaş olabilir

Bağlamsal Anahtarlama

- **Context Switch**
- CPU başka bir işleme geçerken, sistem eski işlemin durumunu kaydetmeli ve yeni işlemin kaydedilmiş durumunu yüklemelidir
- Bir işlemin **bağlamı (context)** PCB'de tutulur
- Bağlamsal Anahtarlama için harcanan zaman ek yüktür. Değişim sırasında işlemci kullanıcının işine direk yarayan bir iş yapmamaktadır
- Harcanan zaman donanım desteğine bağlıdır

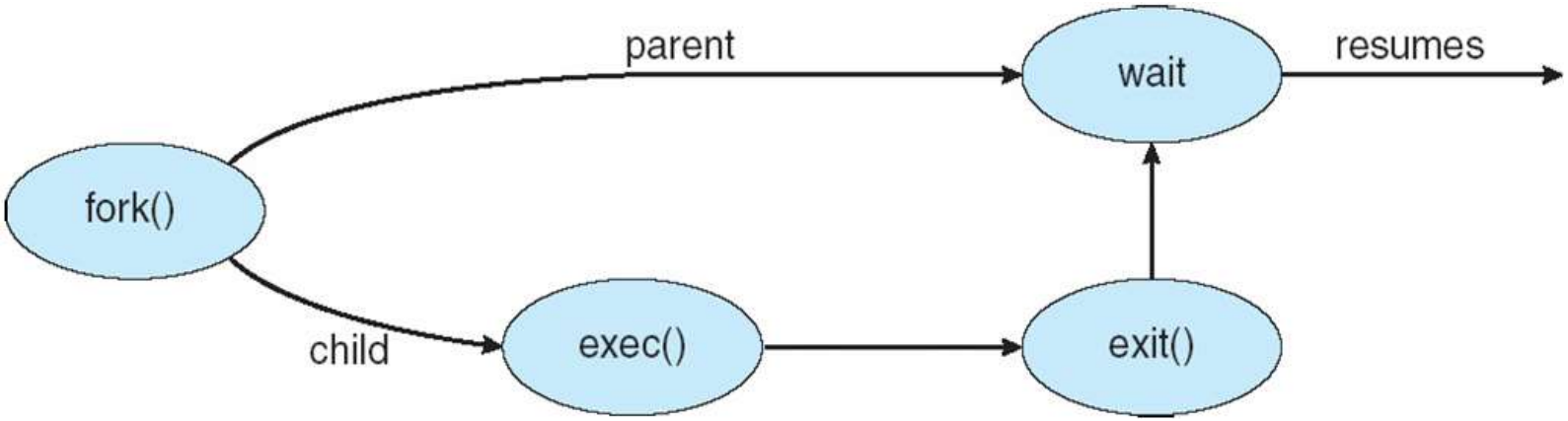
İşlem Oluşturma

- **Ana işlem** (parent process) **çocuk işlemleri** (children processes) oluşturur
- Çocuk işlemlerde yeni işlem oluşturabileceğinden, sistemde işlemlerin bir ağacı oluşur
- Genellikle işlemler **işlem belirteci (process identifier - pid)** kullanılarak birbirlerinden ayrılırlar ve yönetilirler
- Kaynak paylaşımı: 3 olasılık
  - Ana işlem ve çocuklar tüm kaynakları paylaşırlar
  - Çocuklar ana işlemin kaynaklarının belli bir alt kümesini paylaşır
  - Ana işlem ve çocuklar kaynak paylaşmazlar
- Çalıştırma
  - Ana işlem ve çocuklar aynı anda çalışır
  - Ana işlem, çocuk işlemler sonlanana kadar bekler

İşlem Oluşturma (devam)

- **Hafıza alanı (address space)**
  - Çocuk ana işlemin kopyasına sahip
  - Çocuk adres alanına yeni bir program yükler
- UNIX örnekleri
  - **fork** sistem çağrısı yeni bir işlem oluşturur
  - **exec** sistem çağrısı, **fork** sistem çağrısı ile yeni işlem oluşturulduktan sonra, işlemin hafıza alanına yeni bir program yüklemek için kullanılır

İşlem Oluşturma Şeması



Yeni İşlem Oluşturan C Programı

```
int main()
{
pid_t  pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
          fprintf(stderr, "Fork Failed");
          exit(-1);
    }
    else if (pid == 0) { /* child process */
          execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
          /* parent will wait for the child to
    complete */
          wait (NULL);
          printf ("Child Complete");
          exit(0);
    }
}
```

POSIX'te İşlem Oluşturma

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       exit(-1);
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
       exit(0);
    }
}
```

Win32'de İşlem Oluşturma

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

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    // allocate memory
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // create child process
    if (!CreateProcess(NULL, // use command line
     "C:\\WINDOWS\\system32\\mspaint.exe", // command line
     NULL, // don't inherit process handle
     NULL, // don't inherit thread handle
     FALSE, // disable handle inheritance
     0, // no creation flags
     NULL, // use parent's environment block
     NULL, // use parent's existing directory
     &si,
     &pi))
    {
       fprintf(stderr, "Create Process Failed");
       return -1;
    }
    // parent will wait for the child to complete
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    // close handles
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

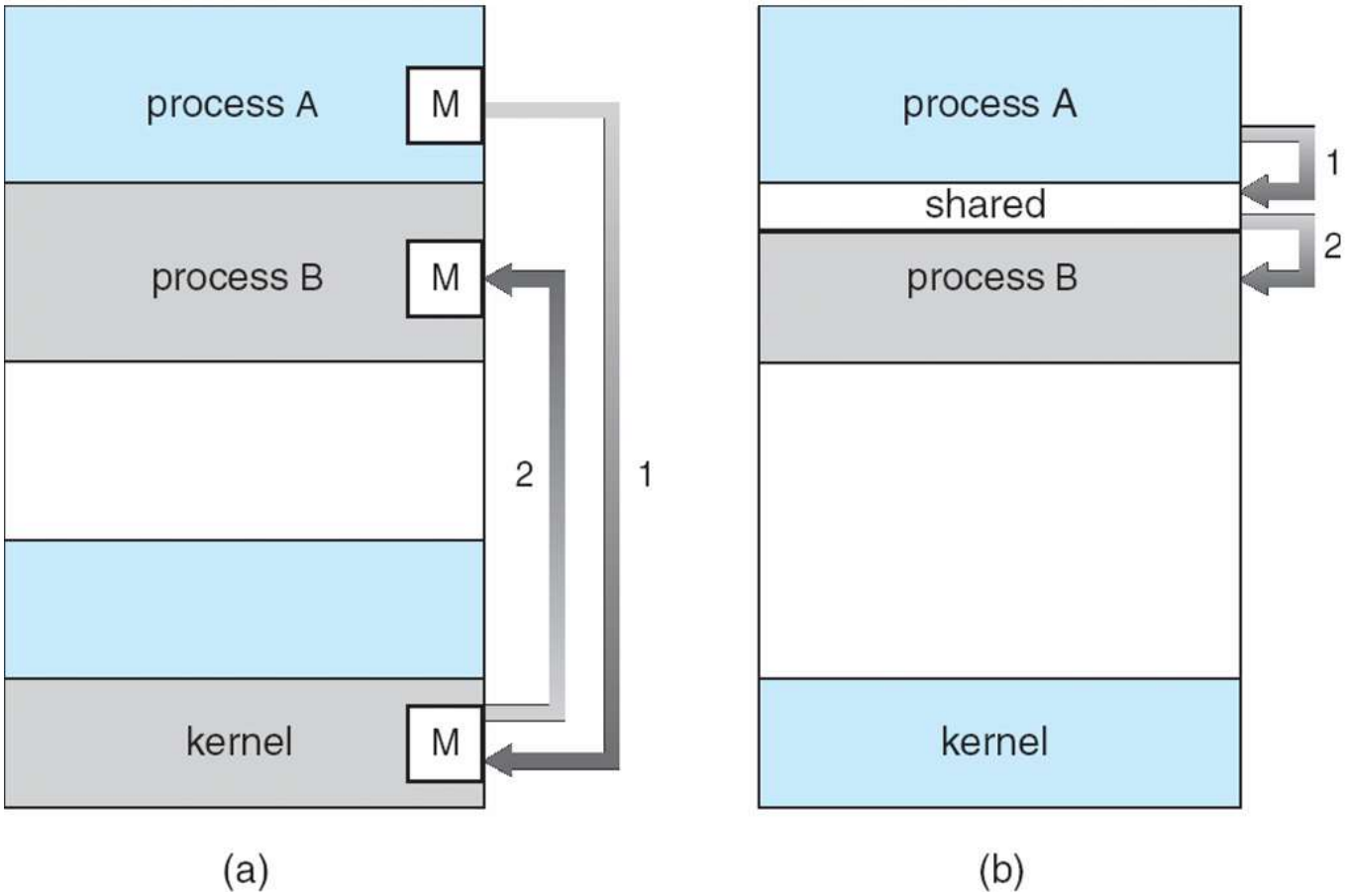
İşlem Sonlandırma

- Process Termination
- İşlemler son komutlarını çalıştırdıktan sonra işletim sistemine kendilerini silmelerini isterler (**exit**)
  - İşlemin dönüş değeri çocuktan ana işleme döndürülür (**wait** ile)
  - İşlemin kaynakları işletim sistemi tarafından geri alınır
- Ana işlem çocuk işlemlerin çalışmasını sonlandırabilir (**abort**)
  - Çocuk kendine ayrılan kaynakları aşmışsa
  - Çocuk işleme atanan göreve artık ihtiyaç yoksa
  - Eğer ana işlem sonlandırılıyorsa
    - Bazı işletim sistemleri ana işlem sonlandırıldıktan sonra çocuklarının çalışmaya devam etmesine izin vermez
      - **Peşpeşe sonlandırma (cascading termination)** ile tüm çocuk işlemler sonlandırılır

İşlemler Arası İletişim

- Interprocess Communication
- Bir sistemdeki işlemler **bağımsız (independent)** ya da **işbirliği yapıyor (cooperating)** olabilir
- İşbirliği yapan işlemler birbirlerini etkileyebilirler veya veri paylaşabilirler
- Neden işlemler işbirliği yapar?
  - Bilgi paylaşımı
  - İşlem hızını arttırmak
  - Modülerlik sağlamak
- İşbirliği yapan işlemler **işlemler arası iletişime (interprocess communication - IPC)** ihtiyaç duyar
- IPC'nin iki modeli
  - **Paylaşımlı bellek** (shared memory)
  - **Mesaj gönderme** (message passing)

İletişim Modelleri



Üretici-Tüketici Problemi

- Producer-Consumer Problem
- İşbirliği yapan işlemler için problem örneği: **üretici (producer)** işlem bilgi üretirken, **tüketici (consumer)** işlem üretilen bilgiyi tüketir
  - **sınırsız tampon bellek** *(unbounded-buffer):* tampon bellek için herhangi bir pratik sınır getirmez
  - **sınırlı tampon bellek** *(bounded-buffer*): sınırlı boyutta bir tampon bellek kullanıldığını varsayar

İşlemler Arası İletişim – Mesaj Gönderme

- İşlemlerin iletişim kurması ve eylemlerini senkronize etmelerini sağlayan mekanizma
- Mesaj sistemi – işlemler birbirleriyle paylaşımlı değişkenlere bağlı kalmaksızın haberleşir
- IPC mekanizması iki işlemi sağlar:
  - **send** (*mesaj gönderme*) – mesaj boyutu sabit ya da değişken olabilir
  - **receive** (*mesaj alma*)
- Eğer *P* ve *Q* işlemleri iletişim kurmak isterse:
  - birbirleri arasında bir **iletişim bağlantısı (communication link)** sağlamalıdırlar
  - send/receive kullanarak mesajlaşmalıdırlar

Direk İletişim

- İşlemler birbirlerini açıkça isimlendirmelidir:
  - **send (P, message)** – P işlemine bir mesaj gönder
  - **receive(Q, message)** – Q işleminden bir mesaj al
- İletişim bağlantısının özellikleri
  - Bağlantılar otomatik olarak sağlanır
  - Bir bağlantı sadece bir çift işlemci arasında oluşturulur
  - Her bir çift için sadece bir bağlantı oluşturulur
  - Bağlantı tek yönlü olabilir, ama genellikle çift yonlüdür

Dolaylı İletişim

- Mesajlar **posta kutusuna (messagebox)** yönlendirilir ve post kutusundan alınır
  - Posta kutusu yerine **port** terimi de kullanılır
  - Her bir posta kutusu özgün bir ada sahiptir (**unique id**)
  - İşlemler sadece bir posta kutusunu paylaşıyor ise iletişime geçebilir
- İletişim bağlantısının özellikleri
  - Bağlantı sadece işlemler ortak bir posta kutusunu paylaşıyor ise oluşturulur
  - İkiden fazla işlem tek bir posta kutusu ile ilişkilendirilebilir
  - Bağlantı tek yönlü veya çift yönlü olabilir

Dolaylı İletişim (devam)

- İşlemler
  - yeni bir posta kutusu oluşturma
  - posta kutusu aracılığıyla mesaj gönderme veya alma
  - posta kutusunun yok edilmesi
- Temel bileşenler:

  send(A, message) – A posta kutusuna bir mesaj gönder

  receive(A, message) – A posta kutusundan bir mesaj al

Senkronizasyon

- **Synchronization**
- Mesaj gönderme **bloklanan (blocking)** veya **bloklanmayan (non-blocking)** şekilde gerçekleşebilir
- Bloklanan mesajlaşma senkrondur (synchronous)
  - **Bloklanan gönderimde**, alıcı taraf mesajı alana kadar, gönderici taraf bloklanır
- Bloklanmayan mesajlaşma asenkrondur (asynchronous)
  - **Bloklanmayan gönderimde**, gönderici taraf mesajı gönderdikten sonra beklemeksizin çalışmaya devam eder

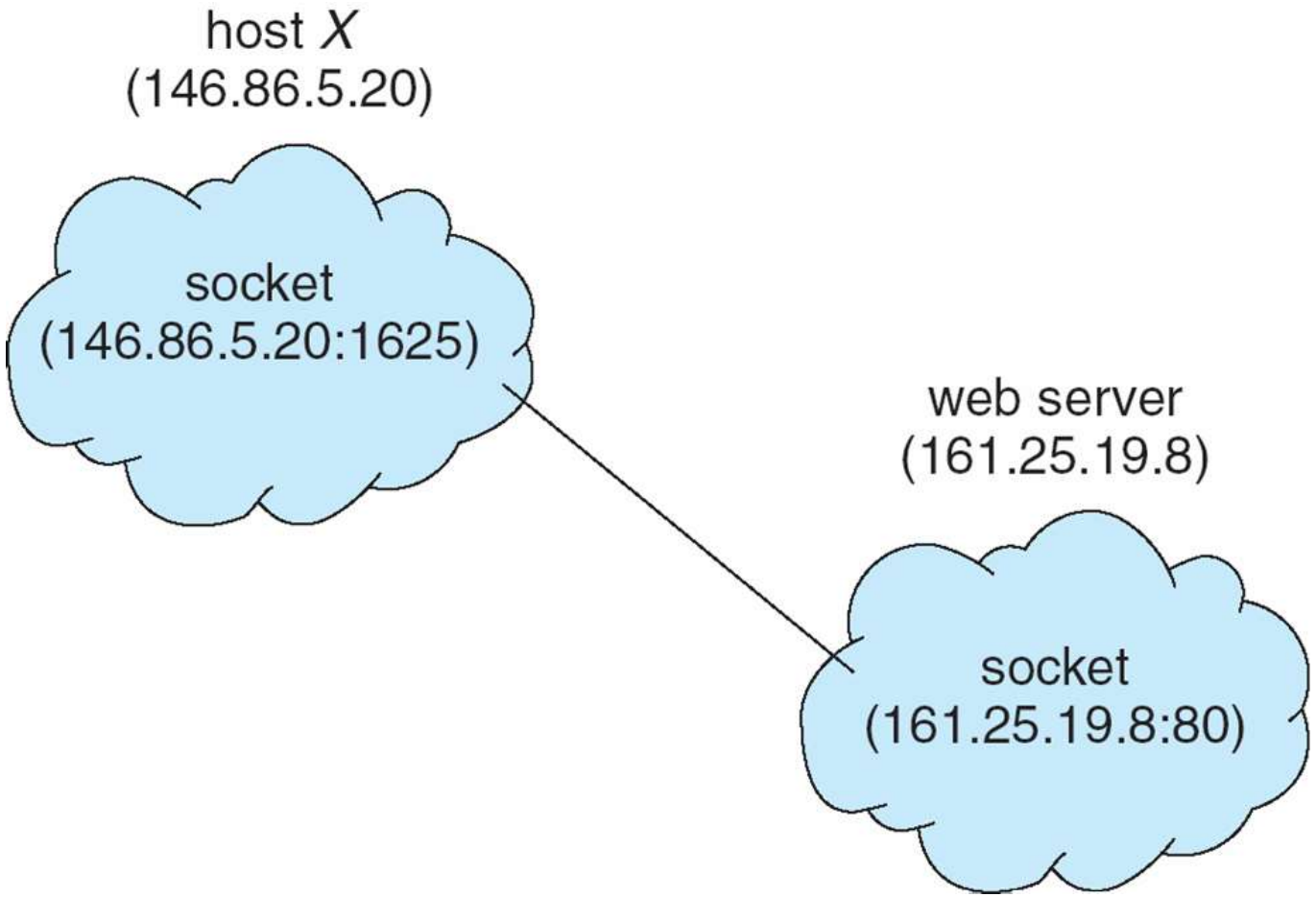
İstemci-Sunucu Sistemlerinde İletişim

- **Soketler** (Sockets)
- **Uzak Prosedür Çağrıları** (Remote Procedure Calls)
  Uzak Metot Çağırma (Remote Method Invocation)(Java)

Soketler

- Soketler iletişim amaçlı bağlantı noktaları olarak tanımlanır
- IP adresi ve portun birleşimidir
- **161.25.19.8:1625** soketi **161.25.19.8** IP'li makinanın **1625** numaralı portuna arşılık gelmektedir
- İletişim bir çift soket arasında gerçekleşir

Soket İletişimi



Java'da Soket İletişimi - Sunucu

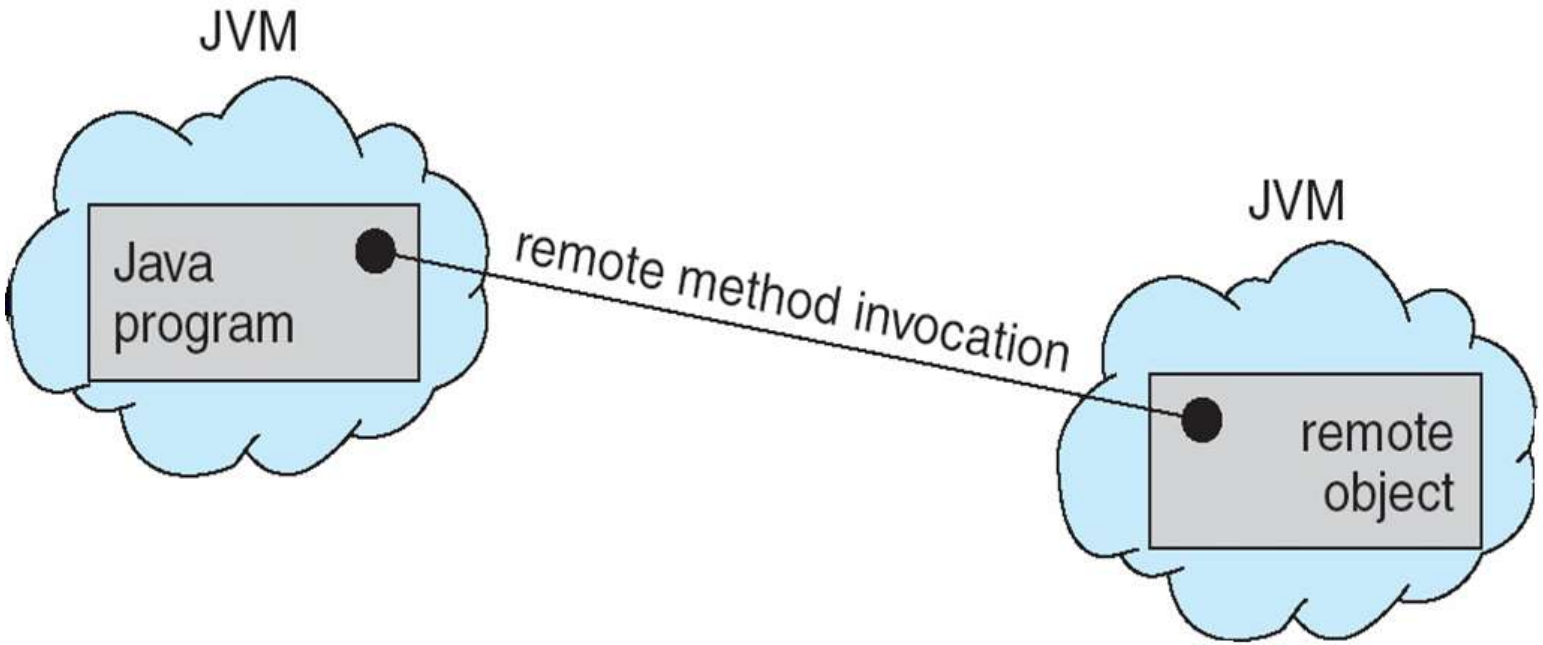
```
public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         // now listen for connections
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            // write the Date to the socket
            pout.println(new java.util.Date().toString());

            // close the socket and resume
            // listening for connections
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Java'da Soket İletişimi - İstemci

```
public class DateClient
{
   public static void main(String[] args) {
      try {
         //make connection to server socket
         Socket sock = new Socket("127.0.0.1",6013);

         InputStream in = sock.getInputStream();
         BufferedReader bin = new
            BufferedReader(new InputStreamReader(in));

         // read the date from the socket
         String line;
         while ( (line = bin.readLine()) != null)
            System.out.println(line);

         // close the socket connection
         sock.close();
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Uzak Metot Çağırma (RMI)

- **Remote Method Invocation (RMI)**
- RPC'ye benzer Java mekanizmasıdır
- RMI, bir Java programının, uzak bir nesnede bulunan bir metotu çağırmasını sağlar



RMI Registry