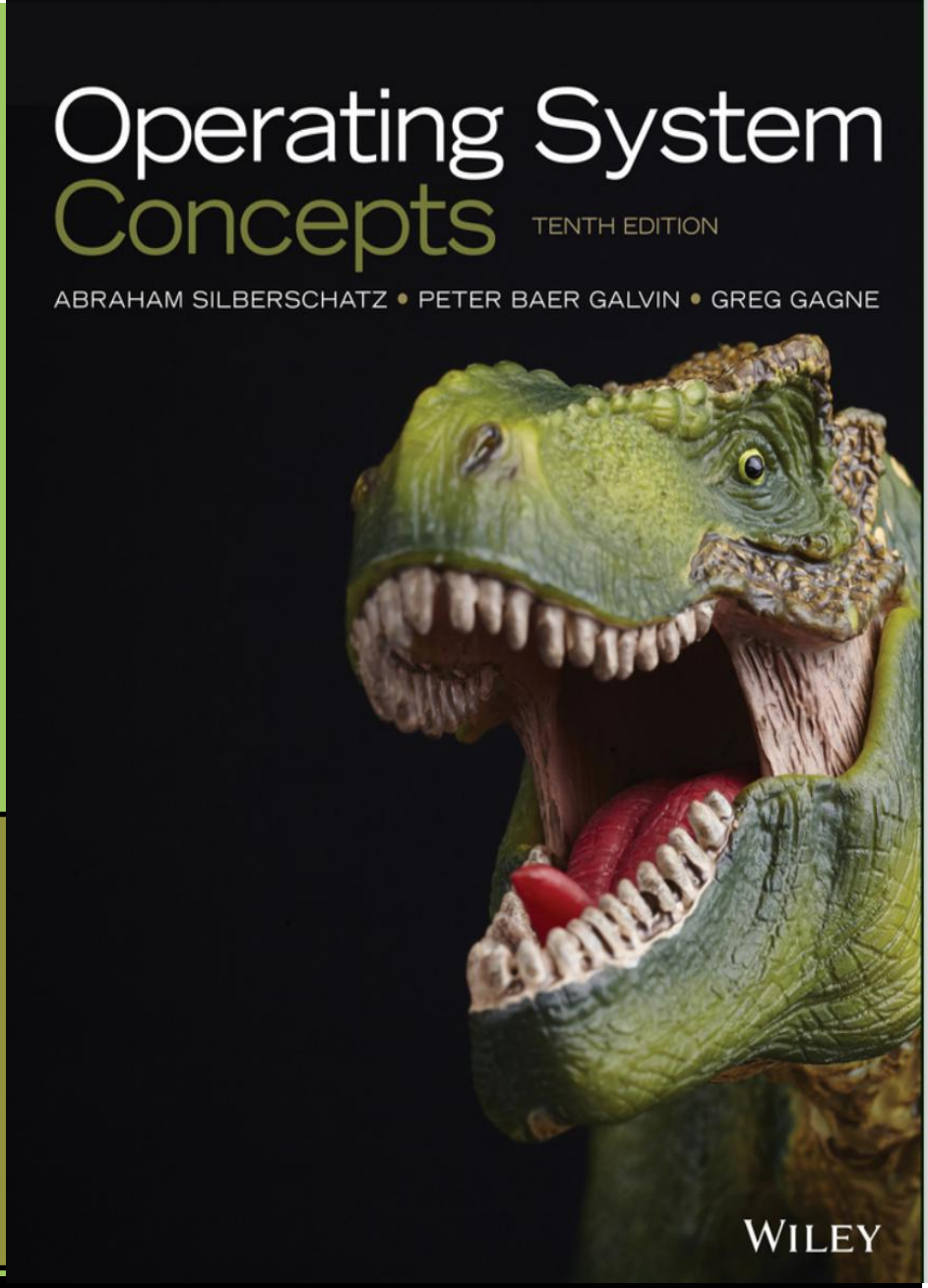
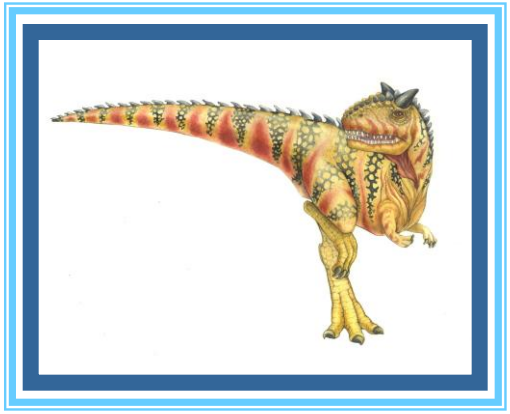
# Operation Systems

Dr. A. Taghinezhad




Website: ataghinezhad.github.io, Email:a0taghinezhad@gmail.com

# فصل ۳: فرآیندها



# Outline طرح کلی

- مفهوم فرایند (Process Concept):
- زمانبندی فرایند (Process Scheduling):
- عملیات روی فرایندها (Operations on Processes):
- ارتباط بین فرایندی (IPC - Interprocess Communication):
  - حافظه مشترک
  - ارسال پیام (Message Passing):
- ارتباط بین فرایندی در سیستم های حافظه مشترک (IPC in Shared-Memory Systems):
- ارتباط بین فرایندی در سیستم های ارسال پیام (IPC in Message-Passing Systems):
- نمونه هایی از سیستم های IPC (Examples of IPC Systems):
- ارتباط در سیستم های سرویس گیرنده–سرور (Communication in Client-Server Systems):

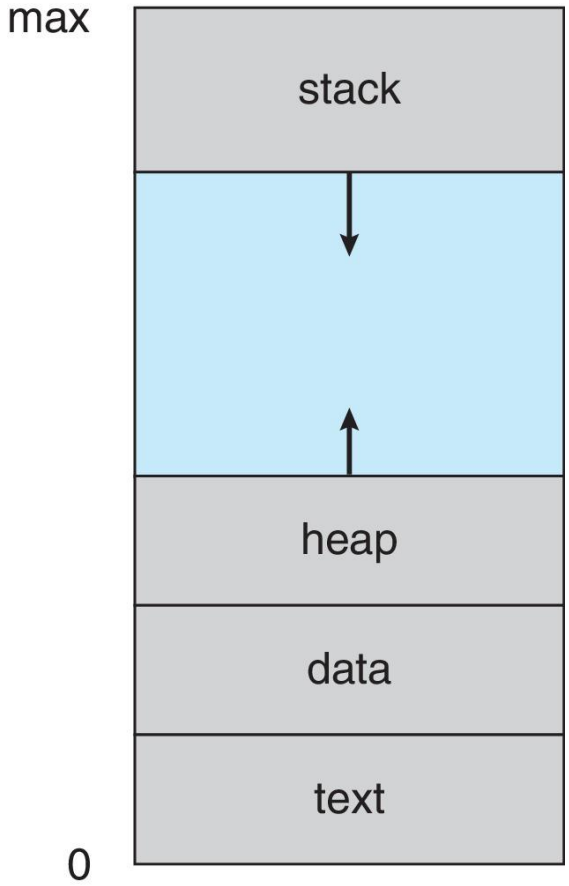
# مفاهیم فرآیند

- یک سیستم عامل انواع برنامه ها را اجرا می کند که به صورت فرایند اجرا می شوند.
- **فرایند :** برنامه ای در حال اجرا؛ اجرای فرآیند باید به صورت متوالی پیشرفت کند. اجرای موازی دستورالعمل های یک فرآیند واحد وجود ندارد.
- **شامل چندین بخش است:**
  - **کد برنامه، همچنین به عنوان بخش متن شناخته می شود:** این به دستورالعمل های واقعی که برنامه از آنها ساخته شده است اشاره دارد.
  - **فعالیت جاری شامل شمارنده برنامه، رجیسترهای پردازنده:** این شامل اطلاعاتی در مورد وضعیت فعلی برنامه است، مانند اینکه کدام دستورالعمل در حال اجرا است و هر داده موقتی که با آن کار می کند.
  - **Stack حاوی داده های موقت:** پشته بخشی از حافظه است **که داده های موقتی** را که برنامه در **طول اجرای** خود استفاده می کند، ذخیره می کند. این شامل مواردی مانند **پارامترهای تابع، آدرس‌های بازگشت** و **متغیرهای محلی** است.
  - **بخش داده حاوی متغیرهای سراسری:** این بخش داده هایی را ذخیره می کند که توسط قسمت های مختلف برنامه قابل دسترسی است.

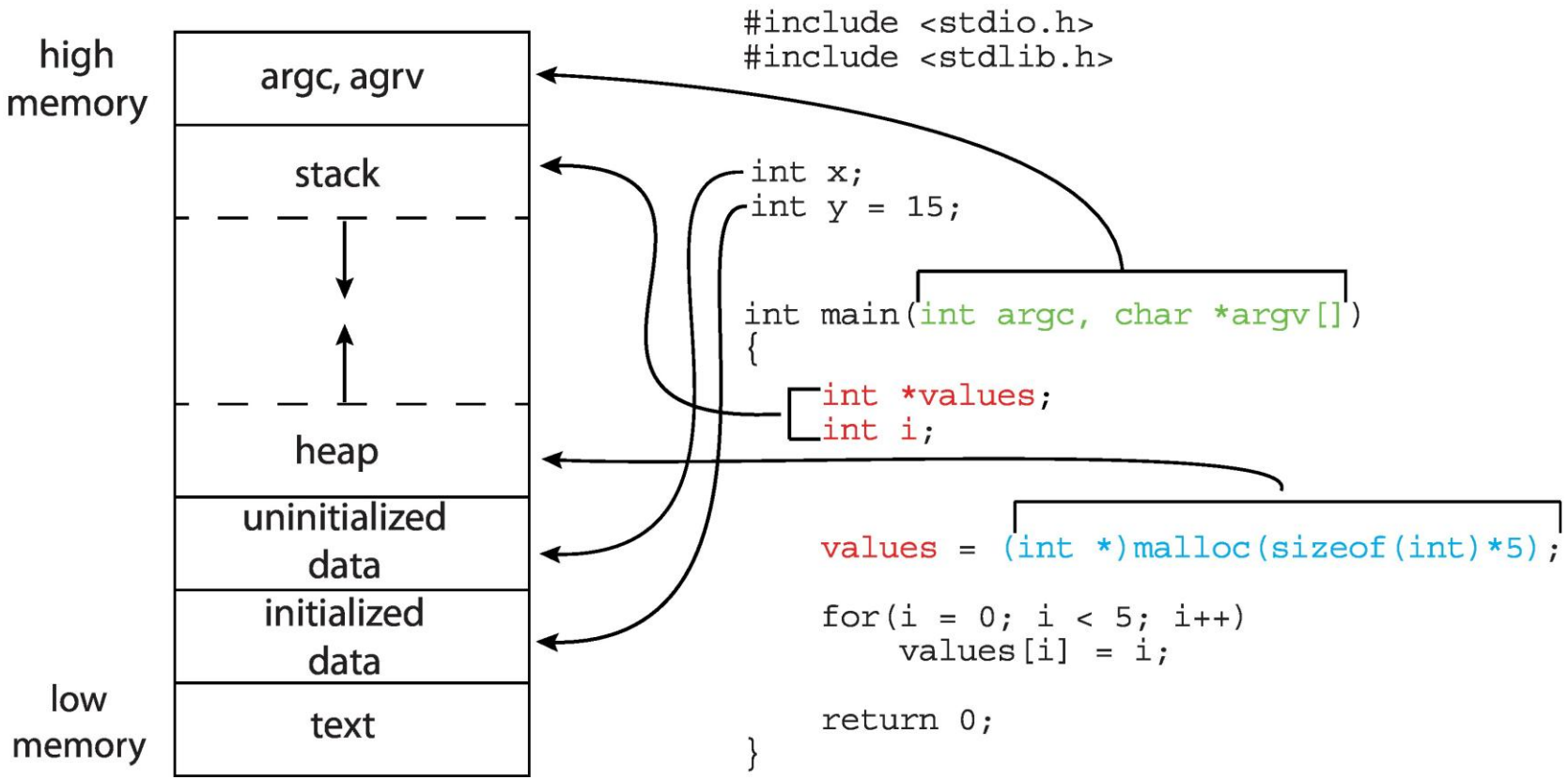
# مفاهیم فرآیند

- **Heap (پشته) حاوی حافظه اختصاص یافته به صورت پویا در زمان اجرا:** هیپ ناحیه دیگری از حافظه است که برنامه می تواند در حین اجرا برای اختصاص حافظه حسب نیاز از آن استفاده کند. برنامه نویس این قسمت را کنترل می کند و میتواند درخواست حافظه نماید و به صورت اشاره گر سیستم عامل این قسمت را در اختیار برنامه نویس قرار می دهد و به صورت خودکار پاک نمیشود و باید Garbage collector آن را حذف کند.
- برنامه یک موجودیت غیرفعال است که روی دیسک ذخیره می شود (فایل اجرایی)؛ در مقابل فرآیند فعال است.
- برنامه زمانی به فرآیند تبدیل می شود که فایل اجرایی به حافظه بارگذاری شود.
- اجرای برنامه از طریق کلیک های ماوس رابط گرافیکی **GUI**، وارد کردن نام آن در خط فرمان و غیره آغاز می شود.
- یک برنامه می تواند منجر به چندین فرآیند شود.
  - سناریویی را در نظر بگیرید که کاربران متعددی یک برنامه یکسان را اجرا می کنند.

# فرآیند در حافظه



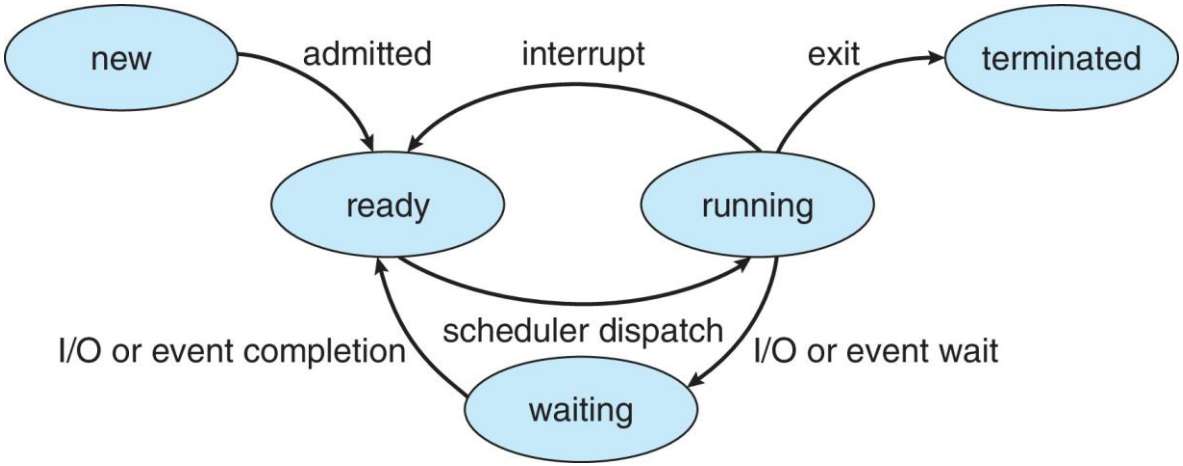

# چیدمان حافظه در یک برنامه C




# وضعیت یک فرآیند

- **با اجرای یک فرآیند، حالت آن تغییر می کند:**
- **ایجاد New** فرآیند در حال ایجاد است.
- **در حال اجرا Running** دستورالعمل ها در حال اجرا هستند.
- **در انتظار Waiting** فرآیند منتظر رخ دادن رویدادی است.
- **آماده Ready** فرآیند در انتظار اختصاص به یک پردازنده است.
- **خاتمه یافته Terminated** فرآیند اجرای خود را به پایان رسانده است.

# نمودار وضعیت یک فرآیند

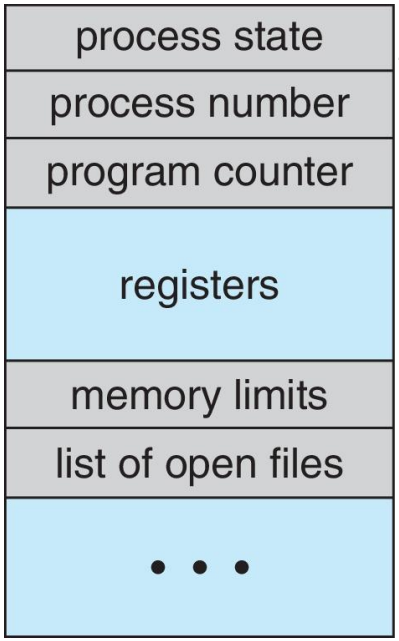



# بلاک کنترل فرآیند PCB

**اطلاعات مرتبط با هر فرآیند (همچنین به عنوان بلوک کنترل وظیفه شناخته می شود)**

**اطلاعات مربوط به فرآیند (Process Information):**

- **حالت فرآیند (Process State):** این نشان دهنده وضعیت فعلی فرآیند است،
  - مانند در حال اجرا (running)، در انتظار (waiting) و غیره.
- **شمارنده برنامه (Program Counter):** این موقعیت دستورالعمل بعدی است که توسط فرآیند اجرا خواهد شد.
- **رجیسترهای CPU:** این شامل محتوای تمام رجیسترهای اختصاصی فرآیند در CPU می شود.
- **اطلاعات زمان‌بندی CPU:** شامل اولویت‌های فرآیند و اشاره‌هایی به صف‌های زمان‌بندی است.
- **اطلاعات مدیریت حافظه:** نشان دهنده حافظه اختصاص یافته به فرآیند است.
- **اطلاعات حسابداری:** شامل زمان استفاده شده از CPU، زمان سپری شده از شروع اجرا و محدودیت‌های زمانی است.
- **اطلاعات وضعیت I/O:** این شامل دستگاه‌های I/O اختصاص یافته به فرآیند و لیستی از فایل‌های باز است.




# نخ یا رشته Threads

- تا کنون، هر فرایند دارای یک رشته اجرایی واحد بوده است.
- فرض کنید به ازای هر فرایند چندین شمارنده برنامه داشته باشیم.
  - چندین مکان می توانند به طور همزمان اجرا شوند.
- چندین رشته کنترل <- رشته ها
- سپس باید فضایی برای جزئیات رشته، چندین شمارنده برنامه در PCB داشته باشیم.
- در فصل ۴ به طور مفصل بررسی می شود.

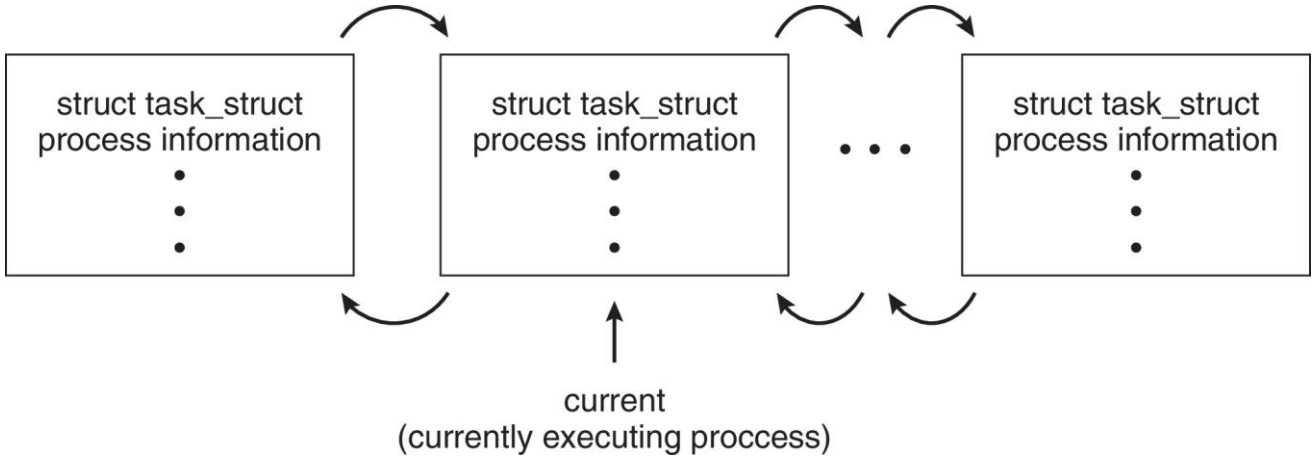
# رشته

- یک رشتهThread اجرایی **کوچکترین توالی دستورالعمل هایی** است که می تواند توسط یک **زمانبندی به طور مستقل** مدیریت شود
- آنها **اجزای کوچکی از یک فرایند** هستند و چندین رشته می توانند به طور همزمان اجرا شوند و کد، حافظه، متغیرها و غیره را به اشتراک بگذارند
- هر رشته کد، داده و بلوک های پشته (heap) یکسانی را به اشتراک می گذارد، اما استک (stack) خاص خود را خواهد داشت
- رشته ها اغلب فرآیندهای سبک وزن نامیده می شوند زیرا حافظه پشته (stack)مخصوص به خود را دارند.

# نمایش فرآیند در لینکوس

Represented by the C structure `task_struct`

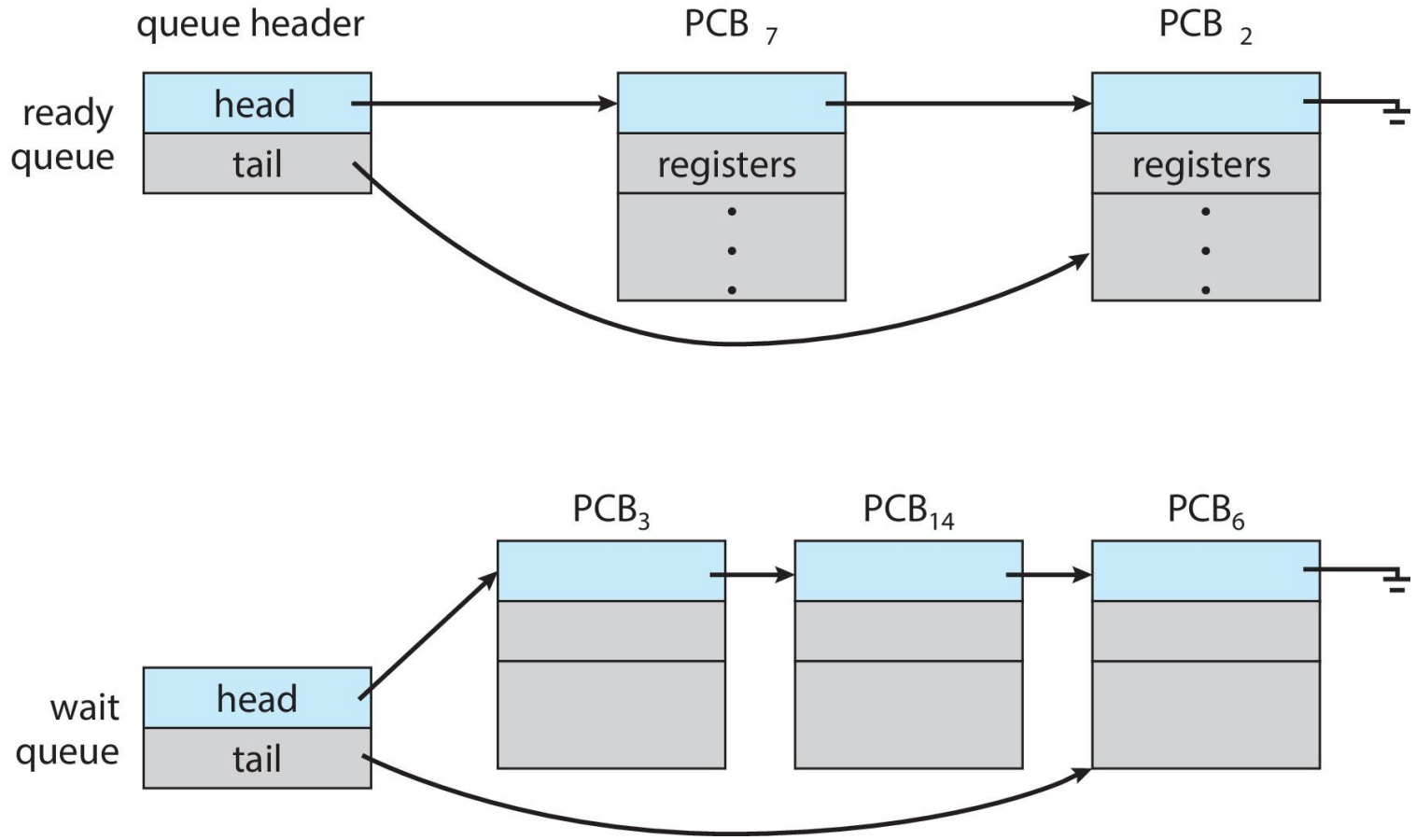
```
pid t_pid;                    /* process identifier */
long state;                   /* state of the process
*/
unsigned int time_slice /* scheduling information */
struct task_struct *parent;/* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files;/* list of open files */
struct mm_struct *mm;    /* address space of this process
*/
```




# زمان بندی فرآیند

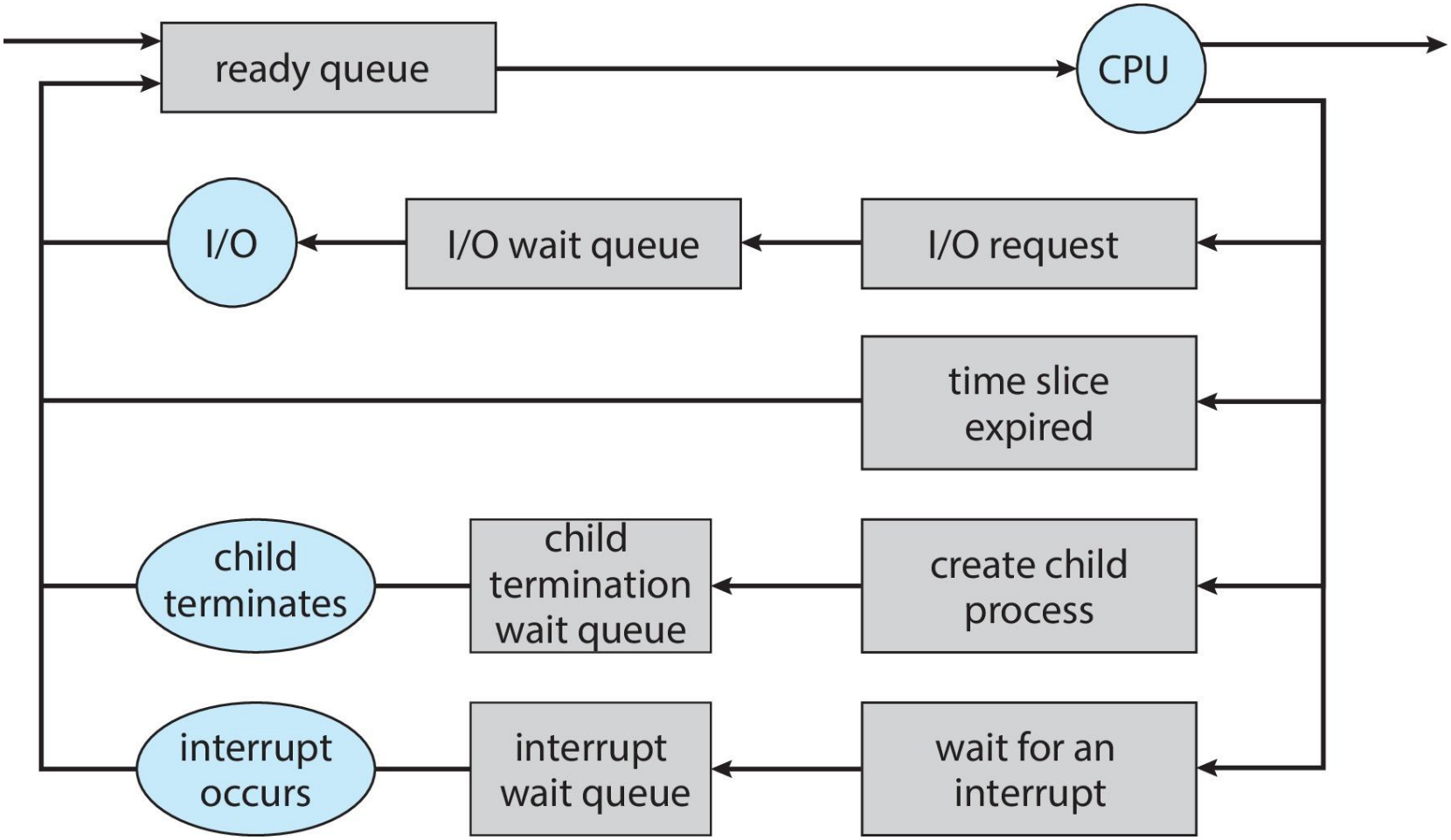
- فرایند زمانبندی کننده از بین فرایندهای موجود برای اجرای بعدی روی هسته CPU انتخاب می کند.
- **هدف** - به حداکثر رساندن استفاده ازCPU ، به سرعت فرآیندها را روی هسته CPU سوئیچ کنید.
- صف های زمانبندی فرآیندها را حفظ می کند:
  - **صف آماده** - مجموعه ای از تمام فرآیندهایی که در **حافظه اصلی** قرار دارند، آماده و منتظر اجرا هستند.
  - **صف های انتظار** - مجموعه ای از فرآیندهایی که منتظر یک رویداد (یعنی ورودی/خروجی) هستند.
  - فرایندها بین صف های مختلف مهاجرت می کنند.

# صف آماده و انتظار



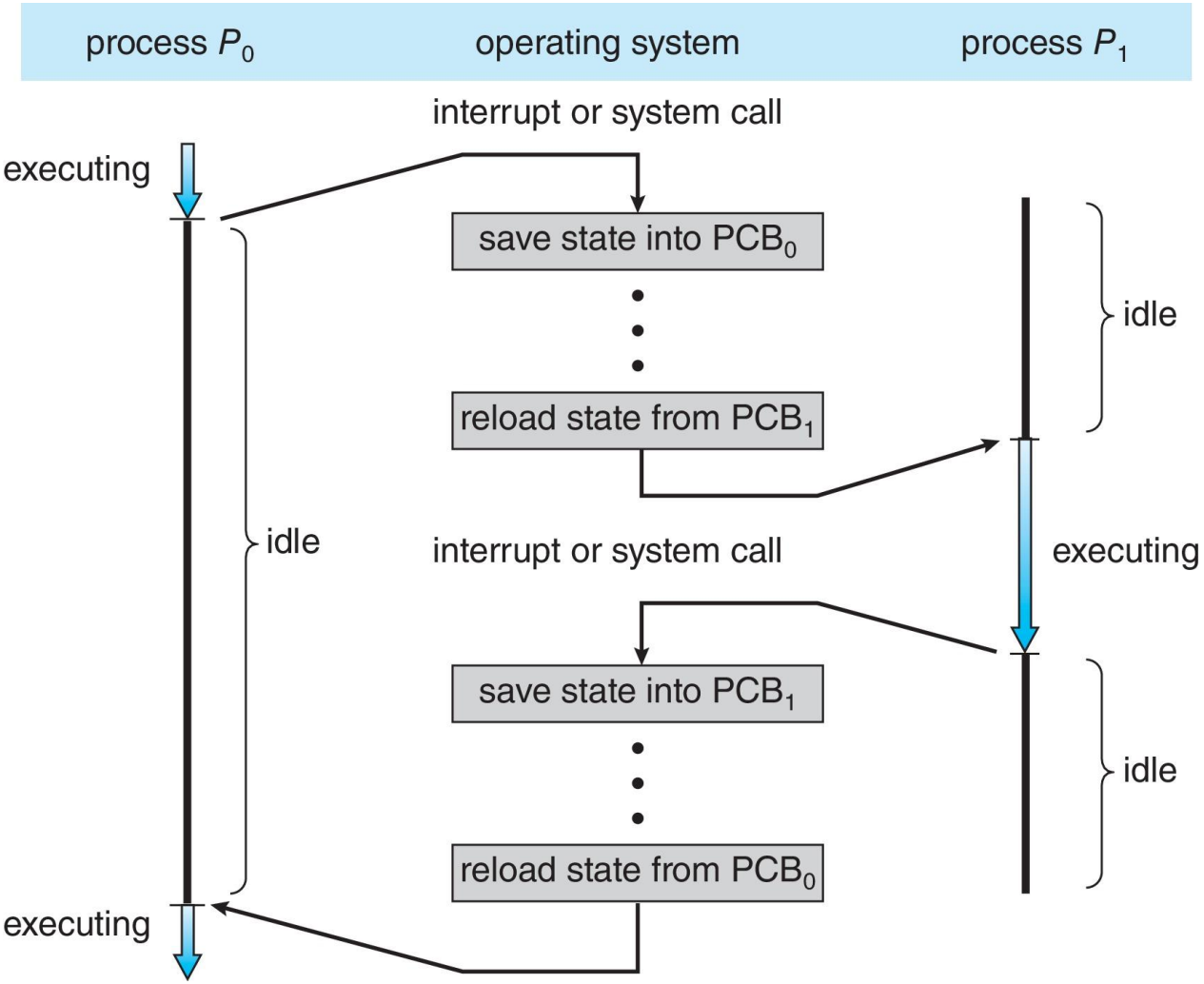

**تعداد فرآیندهایی که در حال حاضر در حافظه قرار دارند، به عنوان درجه چندبرنامگی(چندفرآیندی) شناخته می‌شود.**

# نمایش زمان بندی فرآیندها




# CPU Switch From Process to Process

- وقتی CPU از یک فرایند به فرایند دیگری تغییر می‌کند، یک تغییر زمینه (context switch) رخ می‌دهد.



# Context Switch تغییر زمینه

- زمانی که CPU از یک فرآیند به فرآیند دیگر سوئیچ می کند، سیستم باید وضعیت فرآیند قدیمی را ذخیره کند و وضعیت ذخیره شده برای فرآیند جدید را از طریق یک سوئیچ زمینه بارگذاری کند .زمینه یک فرآیند در PCB نشان داده شده است.
- زمان سوئیچ زمینه صرفا سربار است. سیستم در هنگام سوئیچ کاری مفید انجام نمی دهد.
  - هرچه سیستم عامل و **PCB پیچیده** تر باشند، **سوئیچ زمینه طولانی** تر می شود.
  - زمان وابسته به **پشتیبانی سخت افزار** است.

# عملیاتهای فرآیند

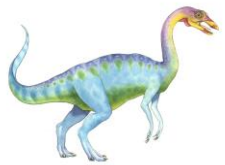
- سیستم باید مکانیزم هایی برای موارد زیر ارائه دهد:
- ایجاد فرآیند
- خاتمه فرآیند

# ایجاد فرآیند

- **فرآیند والد فرآیندهای فرزند را ایجاد می‌کند** که به نوبه خود فرآیندهای دیگری را ایجاد می‌کنند **و درختی از فرآیندها** را تشکیل می‌دهند.
- فرآیند از طریق یک شناسه فرآیند **(pid)** شناسایی و مدیریت می شود.
- گزینه های **اشتراک گذاری منابع**:
  - **والد و فرزندان همه منابع را به اشتراک** می گذارند.
  - **فرزندان زیرمجموعه ای از منابع والد** را به اشتراک می گذارند.
  - **والد و فرزند هیچ منبعی را به اشتراک** نمی گذارند.
- گزینه های اجرا:
  - **والد و فرزندان** به طور **همزمان** اجرا می شوند.
  - **والد منتظر** می ماند **تا فرزندان خاتمه** یابند.

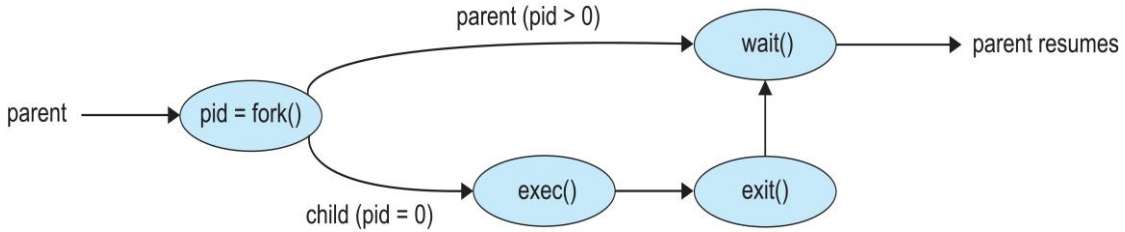
# ایجاد فرآیند (ادامه)

- فضای آدرس
  - فرزند **همانندی**duplicate از **والد** (همان فضای آدرس والد را دارد). باعث ارتباط راحتر فرزند و والد می شود
  - فرزندی با یک برنامه جدید در آن بارگذاری می شود.
- مثال‌های یونیکس:
  - فراخوان سیستمی fork() فرآیند جدیدی ایجاد می‌کند.
  - فراخوان سیستمی exec() پس از یک fork() برای جایگزینی فضای حافظه فرآیند با یک برنامه جدید استفاده می شود.
  - فرآیند والد با فراخوان wait() منتظر خاتمه فرزند می ماند.

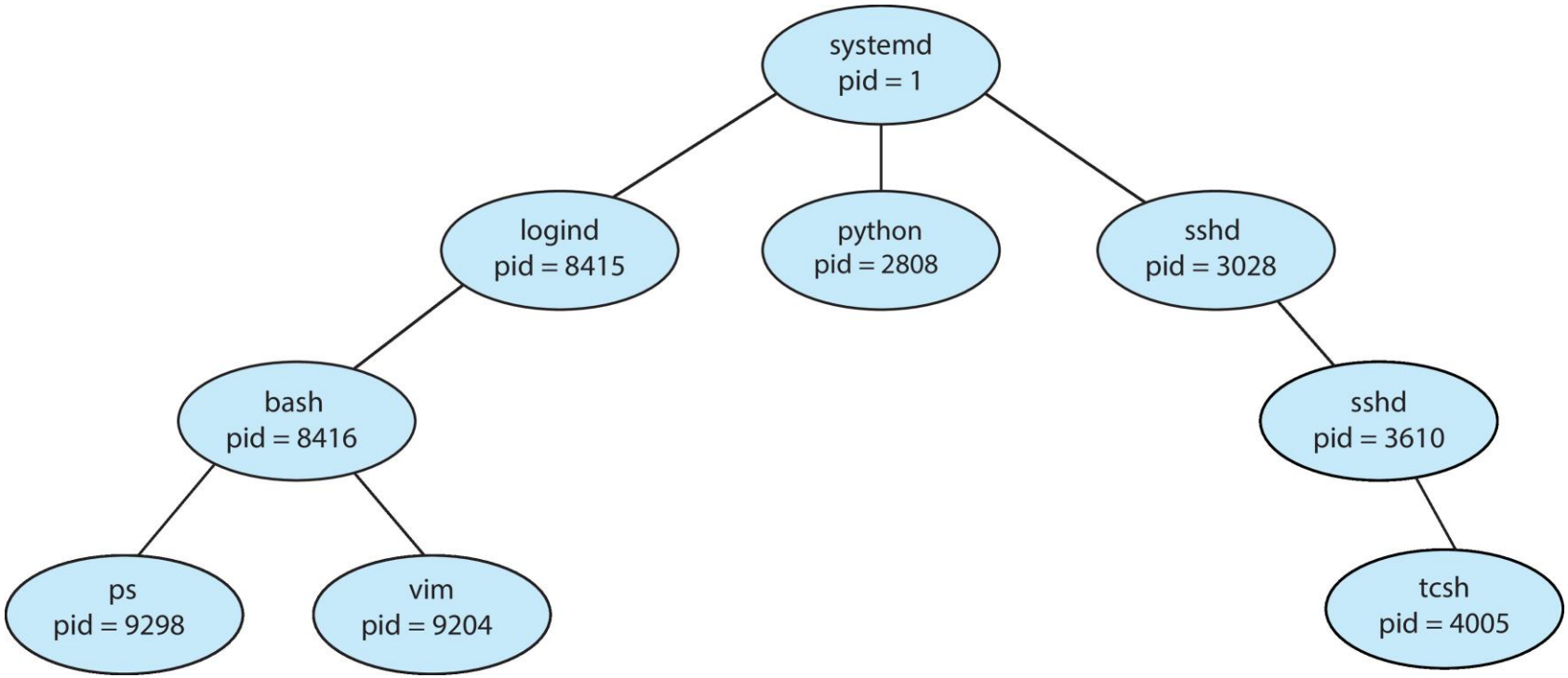


# C Program Forking Separate Process

```
#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");
       return 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");
    }

    return 0;
}
```




# درختی از فرآیندها در لینوکس




# Creating a Separate Process via Windows API

```
#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 */
     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);
}
```

# اتمام فرآیند

- اتمام فرایند: **فرآیند آخرین دستور** را اجرا می کند یا با استفاده از **فراخوان سیستمی** `exit ( )` از سیستم عامل درخواست حذف آن را می کند.
  - فرآیندخاتمه یافته :داده های **وضعیت**(عدد صحیح) را از فرزند به والد از طریق `wait ( )` برمی گرداند.
  - **منابع** فرآیند توسط سیستم عامل **آزاد** می شوند.

- فرآیند والد ممکن است اجرای فرآیندهای فرزند را با استفاده از فراخوان سیستمی `abort ( )` خاتمه دهد. از دلایل انجام این کار عبارتند از:
  - **استفاده** فرزند از منابع اختصاص داده شده **فراتر** رفته است.
  - وظیفه اختصاص داده شده به **فرزند** دیگر **مورد نیاز نیست**.
  - فرآیند والد در حال خروج است و سیستم عامل اجازه نمی دهد فرزندی در صورت خاتمه والد به کار خود ادامه دهد.

# اتمام فرآیند

- برخی از سیستم عامل ها اجازه نمی دهند فرزند در صورت خاتمه والد وجود داشته باشد. اگر یک فرآیند خاتمه یابد، تمام فرزندان آن نیز باید خاتمه یابند.
  - خاتمه آبشاری .(Cascading Termination) تمام فرزندان، نوه ها و غیره خاتمه می یابند.
  - خاتمه توسط سیستم عامل آغاز می شود.

# اتمام فرآیند

- فرآیند والد با استفاده از فراخوان سیستمی `wait()` منتظر خاتمه یک فرآیند فرزند می شود. این فراخوان اطلاعات وضعیت و شناسه فرآیند (pid) فرآیند خاتمه یافته را برمی گرداند

```
pid = wait(&status);
```

# اتمام فرآیند

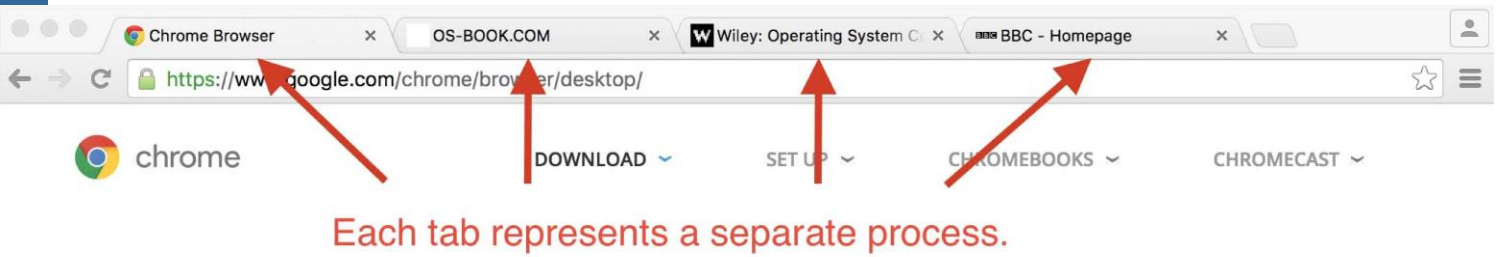
- **فرآیند یتیم**، فرآیندی است که خاتمه نیافته و فرآیند والد آن به اتمام رسیده یا خاتمه پیدا کرده است.
  - **اگر والد بدون فراخوانی `wait()`** خاتمه یابد، فرآیند به یک یتیم (Orphan) تبدیل می شود.

- **فرآیند زامبی**، فرآیندی است که خاتمه یافته اما همچنان ورودی‌ای برای آن در جدول فرآیندها وجود دارد.
  - فرآیندی **خاتمه یافته ولی اگر هیچ والدی منتظر نباشد** (والد فراخوان `wait()` را اجرا نکرده باشد) فرآیند به یک زامبی (Zombie) تبدیل می شود.

# معماری چندپردازنده ای-مرورگر کروم

- بسیاری از **مرورگرهای وب به عنوان یک فرآیند واحد** اجرا می شدند
- اگر یک وب سایت باعث ایجاد مشکل شود، کل مرورگر ممکن است گیر کند یا خراب شود.
- مرورگر گوگل کروم چند فرآیندی است و از سه نوع فرآیند مختلف تشکیل شده است:

1. **فرآیند مرورگر رابط کاربری**، دیسک و I/O شبکه را مدیریت می کند.
2. **فرآیند رندر** کننده صفحات وب را رندر می کند، با HTML و جاوا اسکریپت سروکار دارد. **برای هر وب سایتی که باز می شود یک رندر کننده جدید** ایجاد می شود.
3. برای هر نوع **افزونه یک فرآیند** پلاگین وجود دارد.
4. در یک سندباکس (Sandbox) اجرا می شود یعنی دسترسی I/O دیسک و شبکه را محدود می کند و تأثیر سوء استفاده های امنیتی را به حداقل می رساند.



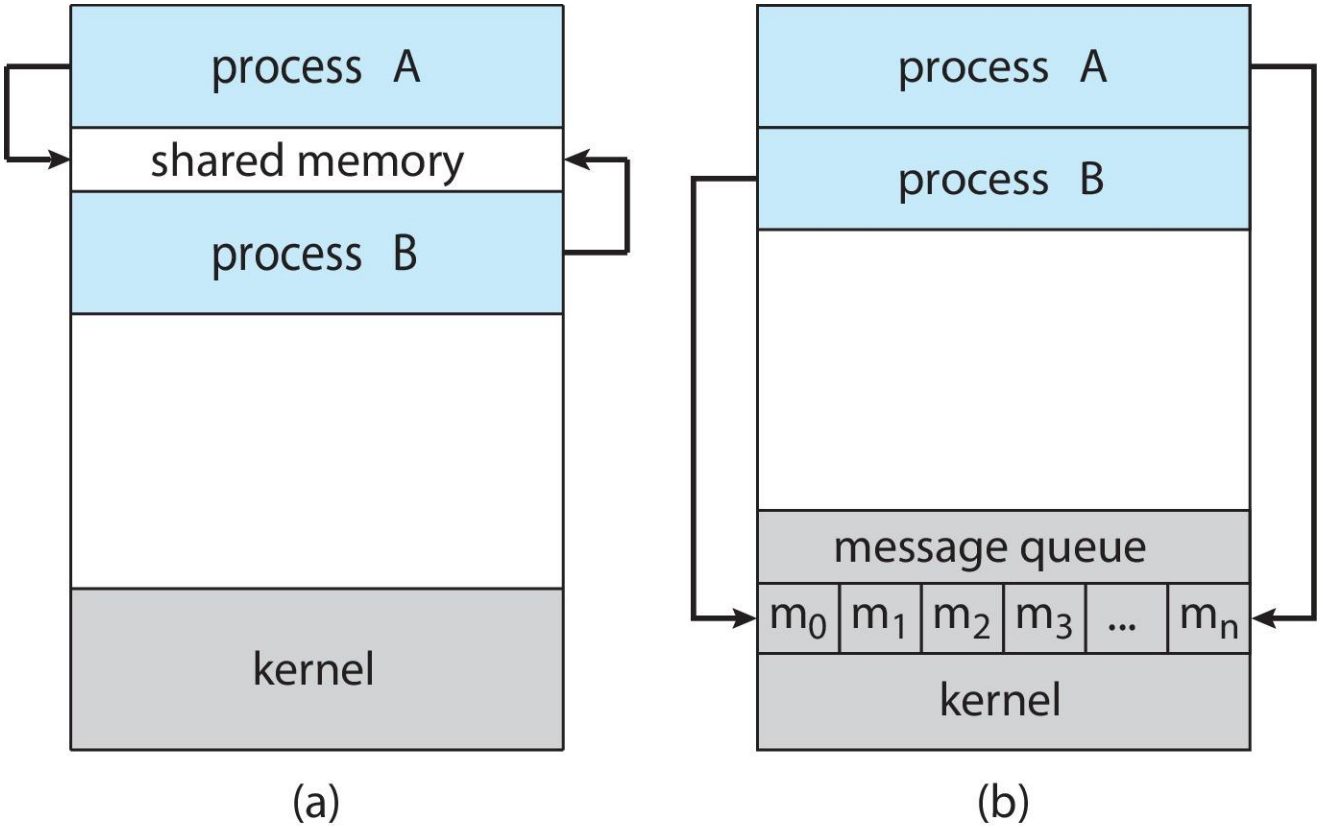

# ارتباط بین فرایندی (IPC)

- فرآیندهای درون یک سیستم ممکن است مستقل یا همکار باشند.
- فرآیندهای همکار می توانند بر فرآیندهای دیگر تأثیر بگذارند یا تحت تأثیر آنها قرار بگیرند
  - دلایل استفاده از فرآیندهای همکار:
    - **اشتراک گذاری** اطلاعات
    - **سرعت بخشیدن** به محاسبات
    - **ماژولار** بودن
    - **سهولت** استفاده
- فرآیندهای همکار به ارتباط بین فرآیند (IPC) نیاز دارند.
- دو مدل IPC وجود دارد: ۱-حافظه مشترک ۲- ارسال پیام

# مدل های ارتباطی فرایندی

(a) Shared memory.

(b) Message passing.



# مسئله تولیدکننده و مصرف کننده

- الگوی فرآیندهای همکار:
  - فرآیند تولید کننده اطلاعاتی را تولید می کند که توسط یک فرآیند مصرف کننده مصرف می شود.
- دو نوع:
  - **بافر بدون محدودیت** :(Unbounded Buffer) هیچ محدودیت عملی برای اندازه بافر قائل نمی شود:
    - **تولید کننده** هرگز **منتظر نمی ماند.**
    - مصرف کننده در صورت عدم وجود بافر برای مصرف، منتظر می ماند.
  - **بافر محدود** :(Bounded Buffer) فرض می کند که اندازه بافر ثابتی وجود دارد:
    - اگر **همه بافرها پر** باشند، تولید کننده **باید منتظر بماند.**
    - مصرف کننده در صورت عدم وجود بافر برای مصرف، منتظر می ماند.

# IPC – اشتراک حافظه

- **حافظه مشترک** ناحیه ای از حافظه است که **بین فرآیندهایی** که می خواهند ارتباط برقرار کنند به اشتراک گذاشته می شود.
- **ارتباط** تحت **کنترل فرآیندهای کاربر** است نه **سیستم عامل.**
- موضوع اصلی ارائه مکانیزمی است که به فرآیندهای کاربر اجازه دهد تا زمانی که به حافظه مشترک دسترسی پیدا می کنند، اقدامات خود را همزمان سازی کنند.
- همزمانی سازی در فصل های ۶ و ۷ مفصل پرداخته شده است.

# راهکار اشتراک حافظه- بافر محدود

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- راه حل صحیح است، اما فقط می تواند از تعداد یک منهای اندازه بافر عنصر استفاده کند.

# Producer Process – Shared Memory

```
item next_produced;

while (true) {
   /* produce an item in next produced */
   while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing-buffer is full*/
   buffer[in] = next_produced;
   in = (in + 1) % BUFFER_SIZE;
}
```

# Consumer Process – Shared Memory

```
item next_consumed;

while (true) {
        while (in == out)
                ; /* do nothing- buffer is empty */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */
}
```

# آیا می‌شود همه بافر را پرکرد؟

- **در روش قبل حداکثر BUFFER_SIZE - 1 آیتم می‌توانند به طور همزمان در بافر قرار داشته باشند.**
- راه حل:
  - فرض کنید می‌خواهیم راه‌حلی برای مشکل مصرف‌کننده-تولیدکننده ارائه کنیم که تمام بافرها را پر کند .
  - با داشتن یک **شمارنده صحیح که تعداد بافرهای پر را ردیابی** می‌کند، انجام دهیم .
  - در ابتدا، **شمارنده روی ۰ تنظیم شده است** .این شمارنده صحیح توسط تولید کننده پس از تولید یک بافر جدید افزایش می یابد .
  - شمارنده صحیح توسط مصرف کننده پس از **مصرف یک بافر کاهش** می یابد.

# تولید کننده

```
while (true) {
        /* produce an item in next produced */

        while (counter == BUFFER_SIZE)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
        counter++;
}
```

# مصرف کننده

```
while (true) {
        while (counter == 0)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
          counter--;
        /* consume the item in next consumed */
}
```

# Race Conditionشرط مسابقه

- **`counter++`** could be implemented as

```
register1 = counter
register1 = register1 + 1
counter = register1
```

- **`counter--`** could be implemented as

```
register2 = counter
register2 = register2 - 1
counter = register2
```

- Consider this execution **interleaving** with "count = 5" initially:

S0: producer execute `register1 = counter` {register1 = 5}
S1: producer execute `register1 = register1 + 1` {register1 = 6}
S2: consumer execute `register2 = counter` {register2 = 5}
S3: consumer execute `register2 = register2 - 1` {register2 = 4}
S4: producer execute `counter = register1` {counter = 6 }
S5: consumer execute `counter = register2` {counter = 4}

# IPC – ارسال پیام

- **فرایندها بدون نیاز به متغیرهای مشترک با یکدیگر ارتباط برقرار می کنند**
- سیستم ارتباط بین فرایندی (IPC) دو عملیات را ارائه می دهد:
  - ارسال(پیام) ((send(message
  - دریافت(پیام) ((receive(message
- اندازه پیام یا ثابت است یا متغیر.

# Message Passing (Cont.)

- اگر فرایندهای P و Q بخواهند با هم ارتباط برقرار کنند، باید:
  - یک **لینک ارتباطی** بین خود **ایجاد** کنند.
  - پیام ها را از طریق ارسال/دریافت رد و بدل کنند.
- مسائل پیاده سازی:
  - لینک ها چگونه ایجاد می شوند؟
  - آیا یک لینک را می توان با بیش از دو فرایند مرتبط کرد؟
  - چند لینک می تواند بین هر جفت از فرایندهای در حال ارتباط وجود داشته باشد؟
  - ظرفیت یک لینک چقدر است؟
  - اندازه پیامی که لینک می تواند تحمل کند ثابت است یا متغیر؟
  - یک لینک یک طرفه است یا دو طرفه؟

# Implementation of Communication Link

- **فیزیکی:**
  - حافظه مشترک
  - باس سخت افزار
  - شبکه
- **منطقی:**
  - مستقیم یا غیرمستقیم
  - ارتباط همزمان یا ناهمزمان
  - بافرگیری خودکار یا صریح

# ارتباط مستقیم

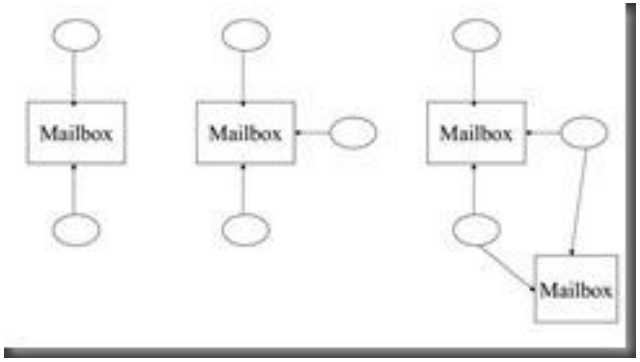
- فرایندها باید یکدیگر را به طور واضح نامگذاری کنند:
  - **send** (*P, message*) – ارسال یک پیام به فرایند P
  - **receive**(*Q, message*) – دریافت یک پیام از فرایند Q

- **ویژگی های لینک ارتباطی**
  - **لینک ها به طور خودکار** ایجاد می شوند.
  - یک **لینک** دقیقاً به **یک جفت** از فرایندهای در حال ارتباط وصل است.
  - بین **هر جفت** فقط یک **لینک** وجود دارد.
  - لینک **ممکن است یک طرفه** باشد، اما **معمولاً دو طرفه** است.

# ارتباط غیر مستقیم

- **پیام ها از صندوق‌های پستی** (همچنین به عنوان **پورت** شناخته می شوند) هدایت و دریافت می شوند.
  - هر **پورت** یک **شناسه** منحصر به فرد دارد.
  - فرایندها فقط در صورتی می توانند ارتباط برقرار کنند که یک **پورت** را به اشتراک بگذارند.
- **ویژگی های لینک ارتباطی (مدل پورت)**
  - **لینک** فقط در صورتی **برقرار** می شود که فرایندها یک **پورت مشترک** داشته باشند.
  - یک **لینک** ممکن است با **چندین فرایند** مرتبط باشد.
  - هر **جفت از فرایندها** ممکن است **چندین لینک ارتباطی** را به اشتراک بگذارند.
  - لینک ممکن است یک طرفه یا دو طرفه باشد.

# (ادامه) ارتباط غیر مستقیم

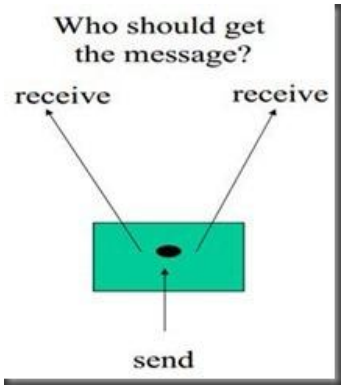



- عملیات:
  - **ایجاد** یک **پورت** یا صندوق پستی جدید
  - **ارسال** و **دریافت پیام** از طریق **پورت**
  - **حذف** یک پورت
- عملگرهای ابتدایی به صورت زیر تعریف می شوند:

- **`send`**(*A, message*) – send a message to port A
- **`receive`**(*A, message*) – receive a message from port A

# (ادامه) ارتباط غیر مستقیم

- اشتراک پورت:
  - فرض کنید سه فرایندP1 ، P2و P3 پورت Aرا به اشتراک می گذارند
  - P1 ارسال می کند؛ P2 و P3 دریافت می کنند.
  - مسئله: چه کسی پیام را دریافت می کند؟
- راه‌حل‌ها:
  - اجازه دهید یک لینک حداکثر با دو فرآیند مرتبط باشد.
  - فقط به **یک فرایند** در یک زمان اجازه دهید یک **عملیات دریافت** را اجرا کند.
  - اجازه دهید **سیستم به طور دلخواه گیرنده** را **انتخاب کند**. **فرستنده** مطلع می شود که **گیرنده** چه کسی بوده است.



# Synchronization همگامی

- ارسال پیام ممکن است مسدود کننده یا غیرمسدود کننده باشد.
- مسدود شدن به عنوان همزمان در نظر گرفته می شود.
  - **ارسال مسدود کننده** - فرستنده تا زمانی که پیام دریافت نشود مسدود می شود. تا ارسال دیگری انجام نشود.
  - **دریافت مسدود کننده** - گیرنده تا زمانی که پیامی در دسترس نباشد مسدود می شود.
- غیر مسدود شدن به عنوان ناهمزمان در نظر گرفته می شود.
  - **ارسال غیر مسدود کننده** - فرستنده پیام را ارسال می کند و ادامه می دهد.
  - **دریافت غیر مسدود کننده** - گیرنده دریافت می کند:
    - یک **پیام معتبر**، یا
    - **پیام تهی**(null)
- ترکیب‌های مختلفی امکان‌پذیر است:
  - اگر هم ارسال و هم دریافت مسدود کننده باشند، یک دیدار (rendezvous) داریم.

# Producer-Consumer: Message Passing

- **Producer**

```
message next_produced;
while (true) {
/* produce an item in next_produced */
 send(next_produced);
}
```

- **Consumer**

```
message next_consumed;
while (true) {
 receive(next_consumed)
/* consume the item in next_consumed */
}
```

# بافر کردن

- صف پیام ها به لینک ارتباطی متصل هستند و به سه روش پیاده سازی می شوند:

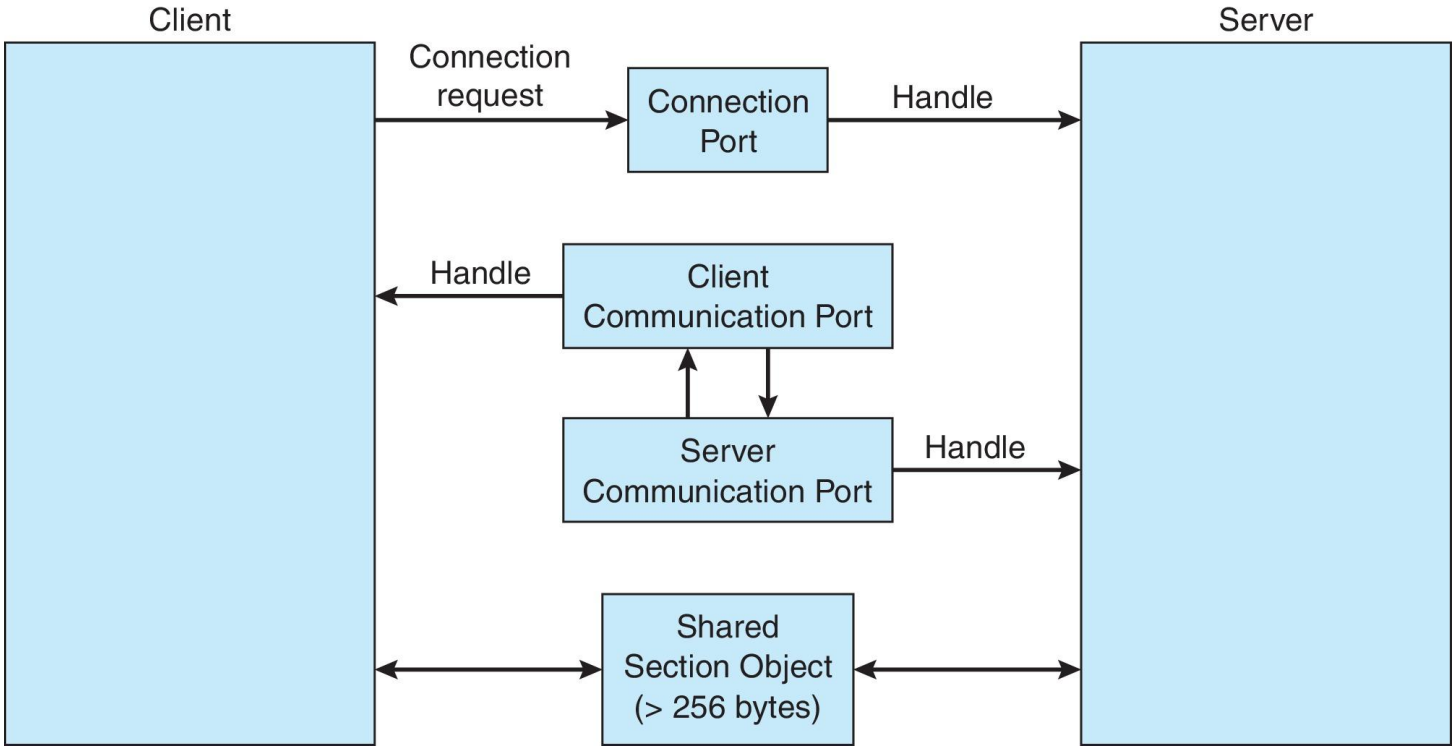
1. **ظرفیت صفر:(Zero Capacity)**
   - هیچ پیامی روی لینک صف نمی‌شود.
   - فرستنده باید منتظر گیرنده بماند :دیدار یا (rendezvous)
2. **ظرفیت محدود:(Bounded Capacity)**
   - صف دارای طول محدودی از n پیام است.
   - اگر لینک پر باشد، فرستنده باید منتظر بماند.
3. **ظرفیت نامحدود:(Unbounded Capacity)**
   - صف دارای طول نامحدود است.
   - فرستنده هرگز منتظر نمی ماند.

# Examples of IPC Systems – Windows

- **فراخوان محلی پیشرفته (Local Procedure Call - LPC)**
- این روش **بر پایه ارسال پیام** است.
- تنها بین فرآیندهای روی **یک سیستم واحد** کار می کند.
- از **پورت** برای **ایجاد و نگهداری کانال های ارتباطی** استفاده می کند.

- سیسستم عامل ویندوز امکان پشتیبانی از چندین محیط عملیاتی یا زیرسیستم را فراهم می‌آورد.
- برنامه‌های کاربردی از طریق یک مکانیزم ارسال و دریافت پیام که به آن IPC ارتباط بین فرایندها، نیز گفته می‌شود، با این زیرسیستم‌ها ارتباط برقرار می‌کنند.

# Local Procedure Calls in Windows




# Examples of IPC Systems – Windows

- ارتباط به شرح زیر عمل می کند:
  - کلاینت یک هندل به آبجکت پورت اتصال زیرسیستم باز می کند.
  - کلاینت درخواست اتصال ارسال می کند.
  - سرور دو پورت ارتباطی خصوصی ایجاد می کند و هندل یکی از آنها را به کلاینت برمی گرداند.
  - کلاینت و سرور از هندل پورت مربوطه برای ارسال پیام یا callback و گوش دادن به پاسخ ها استفاده می کنند.

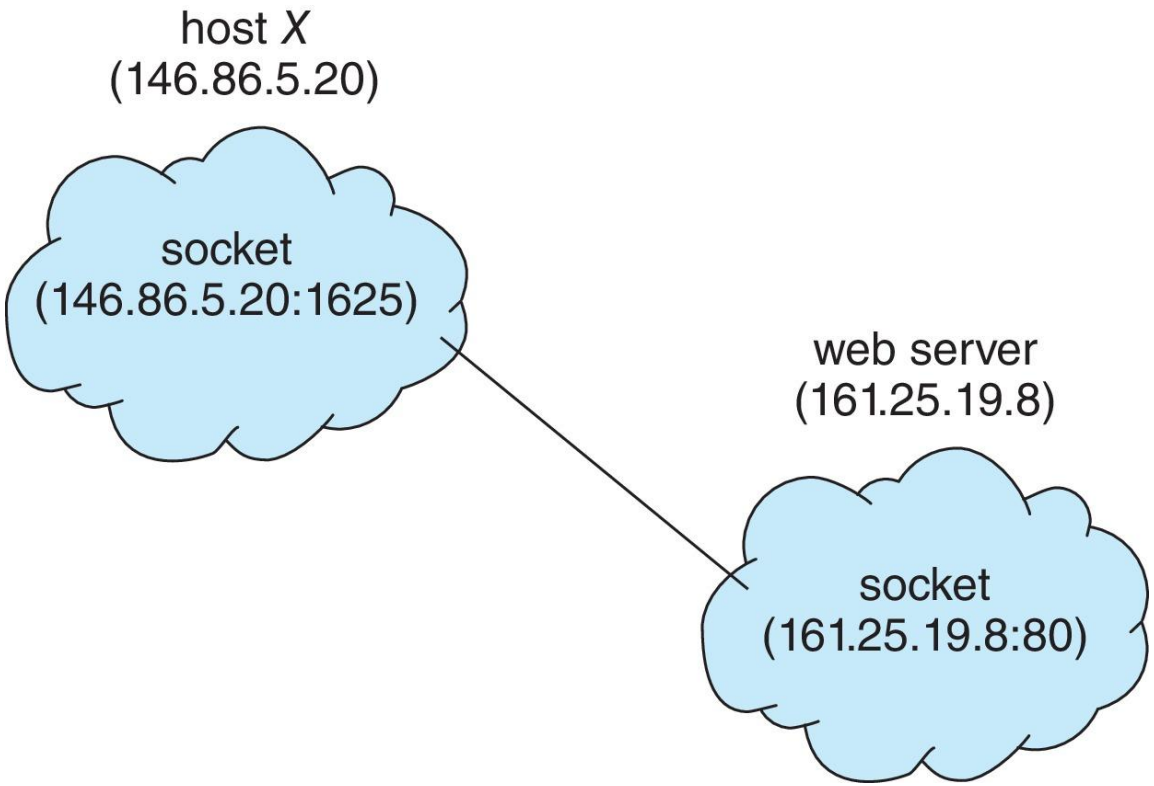
# Communications in Client-Server Systems

- سوکت ها Sockets
- فراخوان رویه از راه دور RPC - Remote Procedure Calls

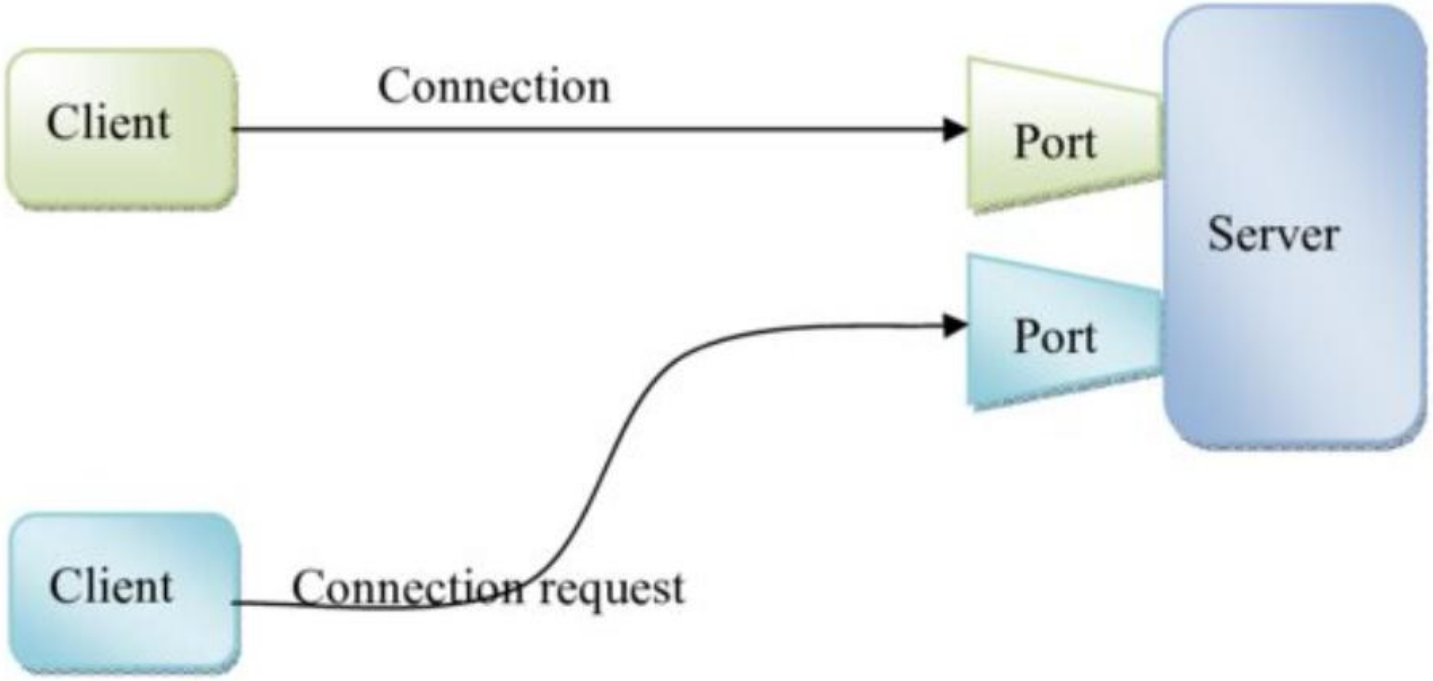
# Sockets

- سوکت به عنوان یک نقطه انتهایی (end-points) برای برقراری ارتباط تعریف می شود.
- ترکیبی از آدرس IP و پورت - عددی که در ابتدای بسته پیام برای تمایز قائل شدن بین سرویس های شبکه روی یک میزبان گنجانده شده است .سوکت **۱۶۱.۲۵.۱۹.۸:۱۶۲۵** به پورت ۱۶۲۵ روی میزبان ۱۶۱.۲۵.۱۹.۸ اشاره دارد.
- ارتباط بین یک جفت سوکت برقرار می شود.
- تمام پورت های زیر ۱۰۲۴ درگاه های شناخته شده هستند که برای سرویس های استاندارد استفاده می شوند.
- آدرس IP خاص ۱۲۷.۰.۰.۱ (loopback) برای اشاره به سیستمی که فرآیند در حال اجرا است استفاده می شود.

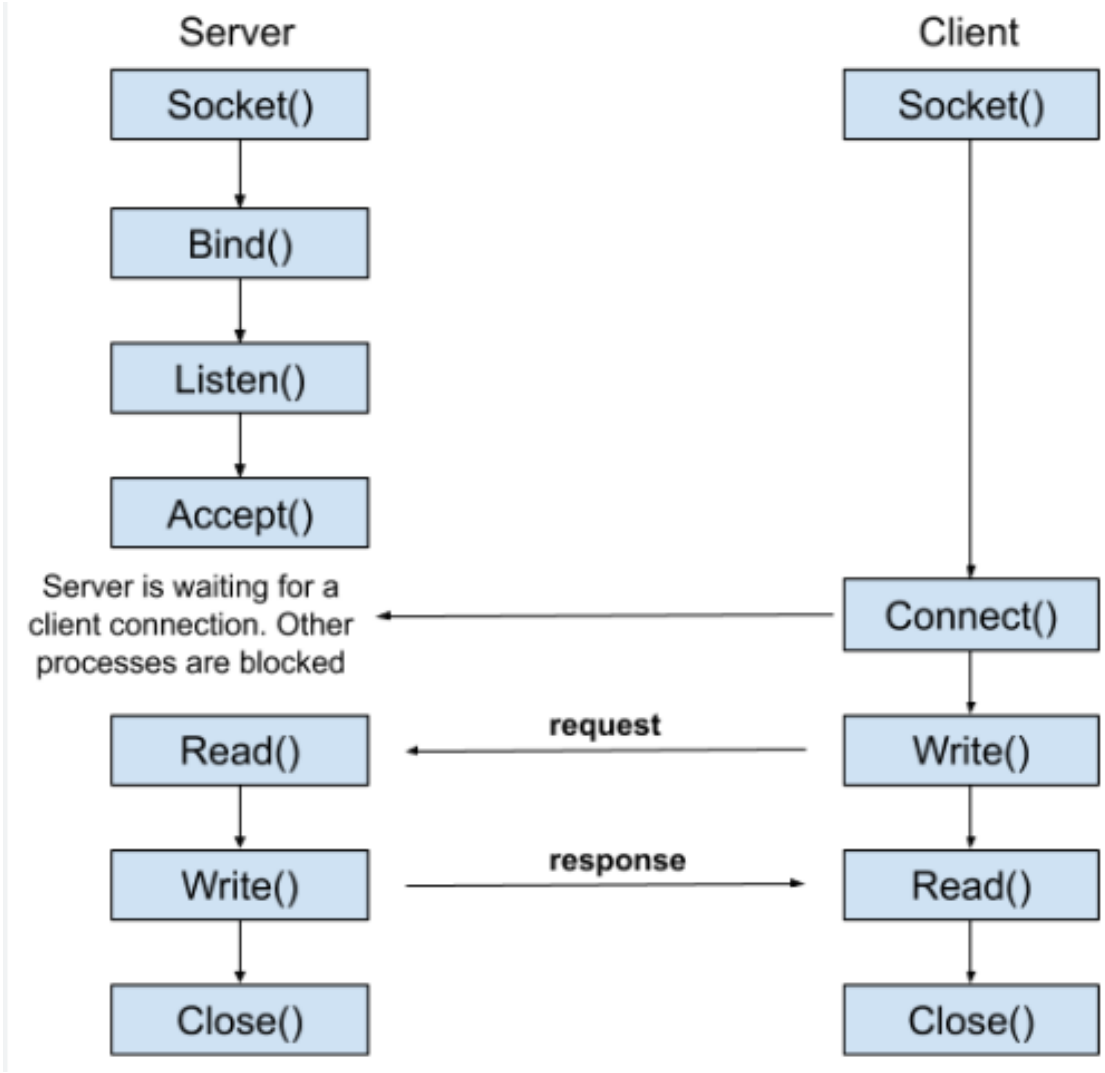
# Socket Communication



# Socket Communication



# Socket Communication




# Sockets in Java

```
import java.net.*;
import java.io.*;

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);
      }
   }
}
```

- سه نوع سوکت وجود دارد:
  - اتصال محور(TCP)
  - بدون اتصال(UDP)
  - MulticastSocket - داده ها را می توان به چندین گیرنده ارسال کرد.

# Sockets in Java

The equivalent Date client

```
import java.net.*;
import java.io.*;

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);
      }
   }
}
```

# Remote Procedure Calls

- فراخوانی رویه از راه دور (RPC) یک مکانیزم برای انتزاع فراخوانی رویه بین فرآیندها در سیستم‌های توزیع شده بر بستر شبکه است.
- به عبارت دیگر، این پروتکل به برنامه‌نویس این امکان را می‌دهد تا توابع موجود روی یک کامپیوتر دیگر در شبکه را اجرا کند، گویی آن توابع به صورت محلی در دسترس هستند.

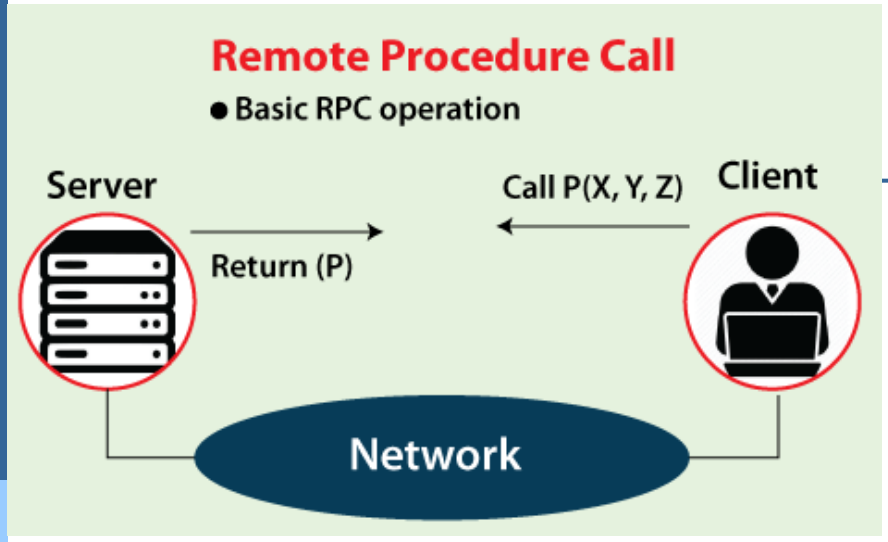
# فراخوان رویه از راه دور(RPC - Remote Procedure Calls)

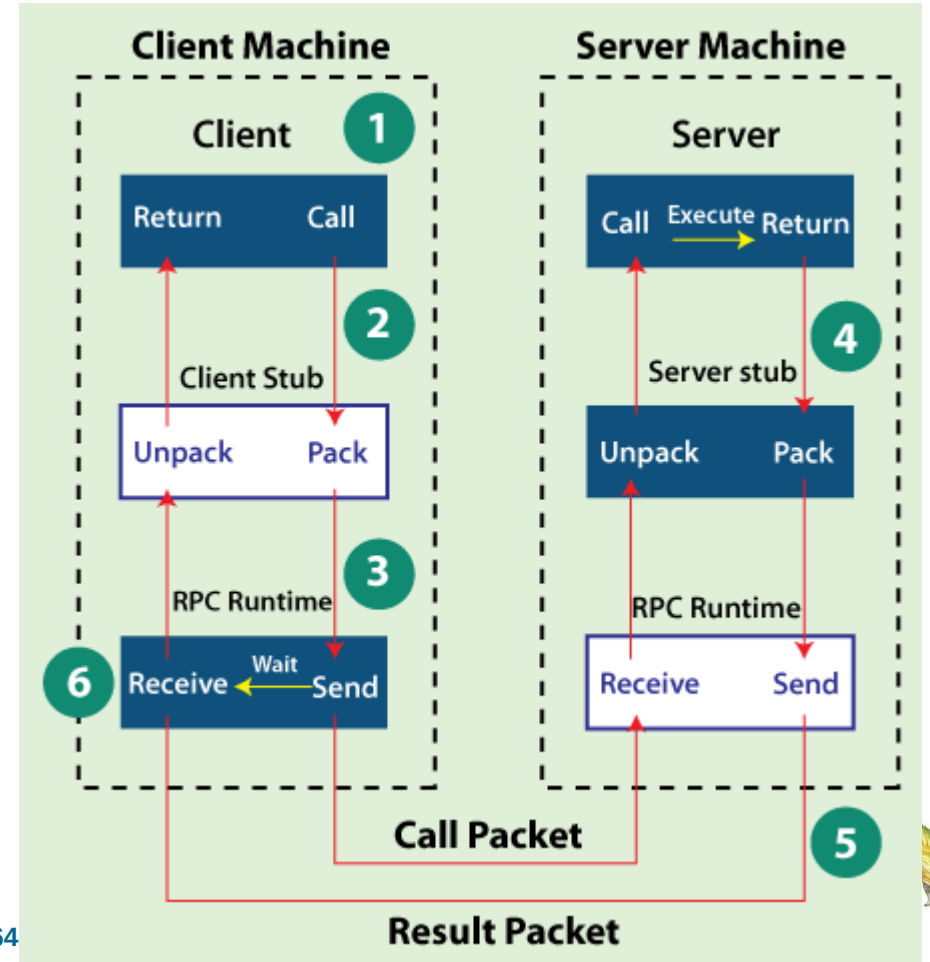
- RPC از پورت‌ها برای تمایز قائل شدن بین سرویس‌های مختلف استفاده می‌کند.
- Stub یک پروکسی در سمت کلاینت است که به عنوان نماینده **پروسیجر واقعی** روی سرور عمل می‌کند.
- Stub سمت کلاینت مسئولیت یافتن سرور مقصد و آماده‌سازی (marshaling) پارامترهای ارسالی را برعهده دارد.
- در سمت سرور، Stub دیگر این پیام را دریافت کرده و پارامترهای آماده‌سازی شده را از حالت آماده‌سازی خارج (unmarshaling) کرده و در نهایت، پروسیجر مورد نظر را روی سرور اجرا می‌کند.
- بر روی سیستم‌عامل ویندوز، کدهای Stub بر اساس مشخصات نوشته شده با زبان تعریف رابط مایکروسافت (MIDL) کامپایل می‌شوند.

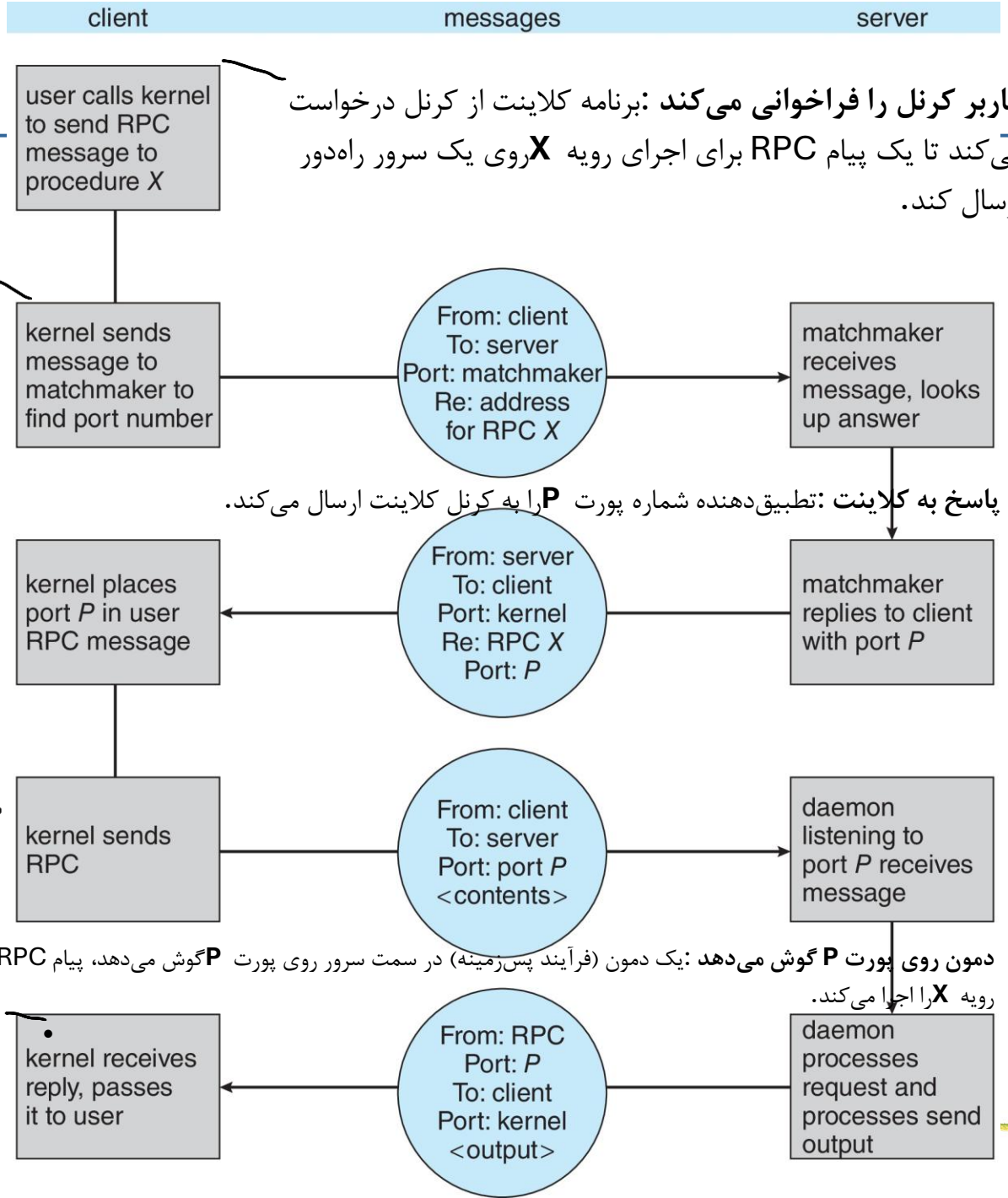
# Remote Procedure Calls (Cont.)

- نمایش داده‌ها برای سازگاری با ساختارهای مختلف از طریق فرمت نمایش داده خارجی (XDL:eXternal Data Representation) در فراخوانی RPC مدیریت می‌شود.

- ارتباطات راه دور نسبت به ارتباطات محلی، دارای سناریوهای خطای بالقوه بیشتری است.
  - با این حال، این پروتکل تضمین می‌کند که پیام‌ها حداکثر یک بار و به صورت صحیح، به گیرنده (مقصد) ارسال شوند (برخلاف سناریوی "حداکثر یک بار" در برخی پروتکل‌های دیگر).


Remote Procedure Call
Basic RPC operation
Server
Call P(X, Y, Z)
Client
Return (P)
Network


Client Machine
Server Machine
Client
1
Server
Return
Call
Call
Execute
Return
2
4
Client Stub
Server stub
Unpack
Pack
Unpack
Pack
3
RPC Runtime
RPC Runtime
6
Receive
Wait
Send
Receive
Send
Call Packet
5
Result Packet




- **کاربر کرنل را فراخوانی می‌کند** :برنامه کلاینت از کرنل درخواست می‌کند تا یک پیام RPC برای اجرای رویه **X** روی یک سرور راه‌دور ارسال کند.

- **کرنل با تطبیق‌دهنده تماس می‌گیرد** :از آنجا که کلاینت پورت سرور را نمی‌داند، کرنل یک درخواست جست‌وجو به **سرویس تطبیق‌دهنده** یک رجیستری متمرکز که نام‌های RPC را به پورت‌های سرور نگاشت می‌کندارسال می‌کند.

- **پاسخ به کلاینت** :تطبیق‌دهنده شماره پورت **P** را به کرنل کلاینت ارسال می‌کند.

- **کرنل RPC را ارسال می‌کند** :با استفاده از پورت **P**، کرنل کلاینت درخواست RPC واقعی را به سرور ارسال می‌کند

- **دمون روی پورت P گوش می‌دهد** :یک دمون (فرآیند پس‌زمینه) در سمت سرور روی پورت **P** گوش می‌دهد، پیام RPC را دریافت می‌کند و رویه **X** را اجرا می‌کند.

- کرنل کلاینت پاسخ را دریافت کرده و نتیجه را به برنامه کاربر منتقل می‌کند و چرخه RPC را تکمیل می‌کند.

# End of Chapter 3