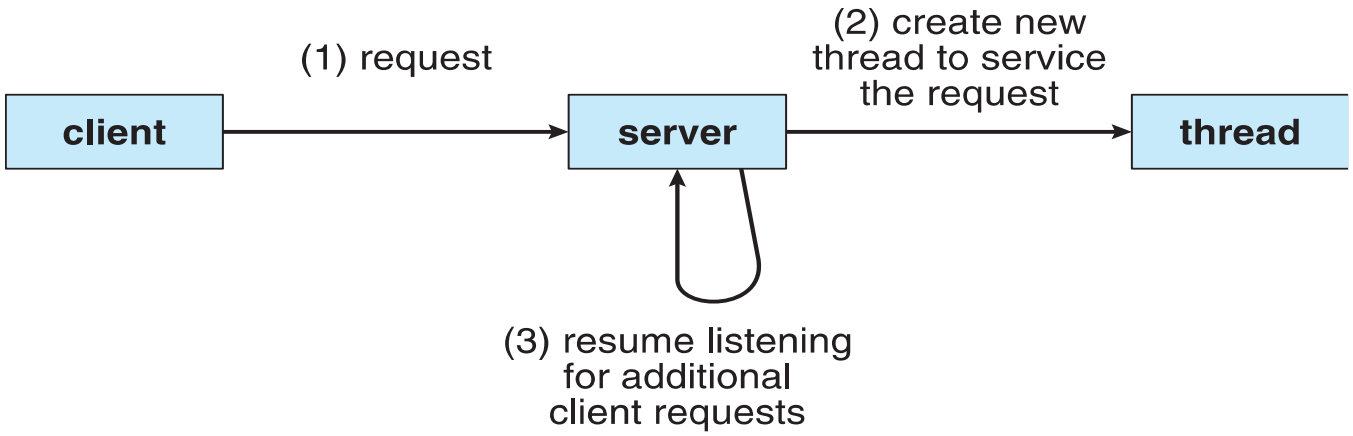
فصل ۴: ریسه (Thread)

انگیزه

- برنامه های کاربردی مدرن چند ریسه ای هستند و ریسه ها در یک کاربرد / پردازه اجرا می شوند.
- بسیاری از وظایف در برنامه های کاربردی را می توان با ریسه ها پیاده سازی نمود. برای نمونه
  - به روز رسانی صفحه نمایش
  - بازیابی داده
  - بررسی ایراد های نگارشی در ویرایشگرها
- ایجاد پردازه ها وقت گیر و پر هزینه است اما ایجاد ریسه ها به زمان و هزینه کمتری نیاز دارد
- ریسه ها سبب ساده شدن کد و افزایش کارایی می شوند.
- هسته های سیستم عامل ها معمولا چند ریسه ای هستند.
- برای نمونه : یک سرویس دهنده چند ریسه ای



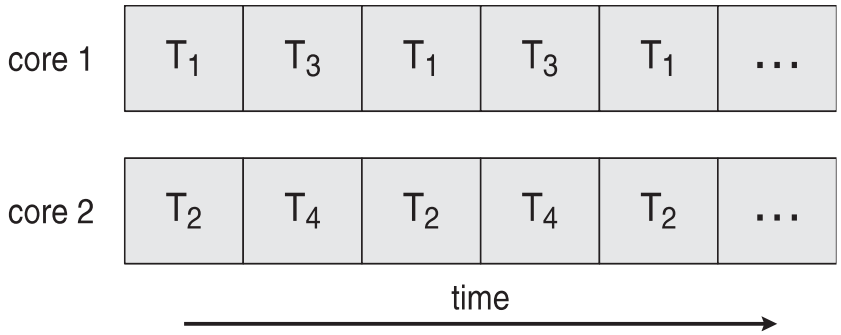
سودمندی بکارگیری ریسه ها

- **بهبود پاسخگویی:** به کارگیری ریسه سبب افزایش پاسخ گویی کاربرد می شود. اگر بخشی از یک کاربرد یا پردازه مسدود شود بخش های دیگر امکان پاسخ گویی دارند.
- **اشتراک منابع:** اشتراک منابع در ریسه ها بسیار ساده تر از اشتراک حافظه و ارتباط با پیام در پردازه ها است.
- **صرفه اقتصادی:** هزینه ایجاد ریسه، تعویض زمینه برای ریسه و سوییچ کردن پردازنده بین ریسه های مختلف بسیار کمتر از فعالیت مشابه برای پردازه است.
- **مقیاس پذیری:** اگر کامپیوتر دارای پردازنده های مختلف باشد پردازه های چند ریسه ای می توانند از مزایای آن برای بهبود سرعت پردازش و مقیاس پذیری استفاده نمایند.

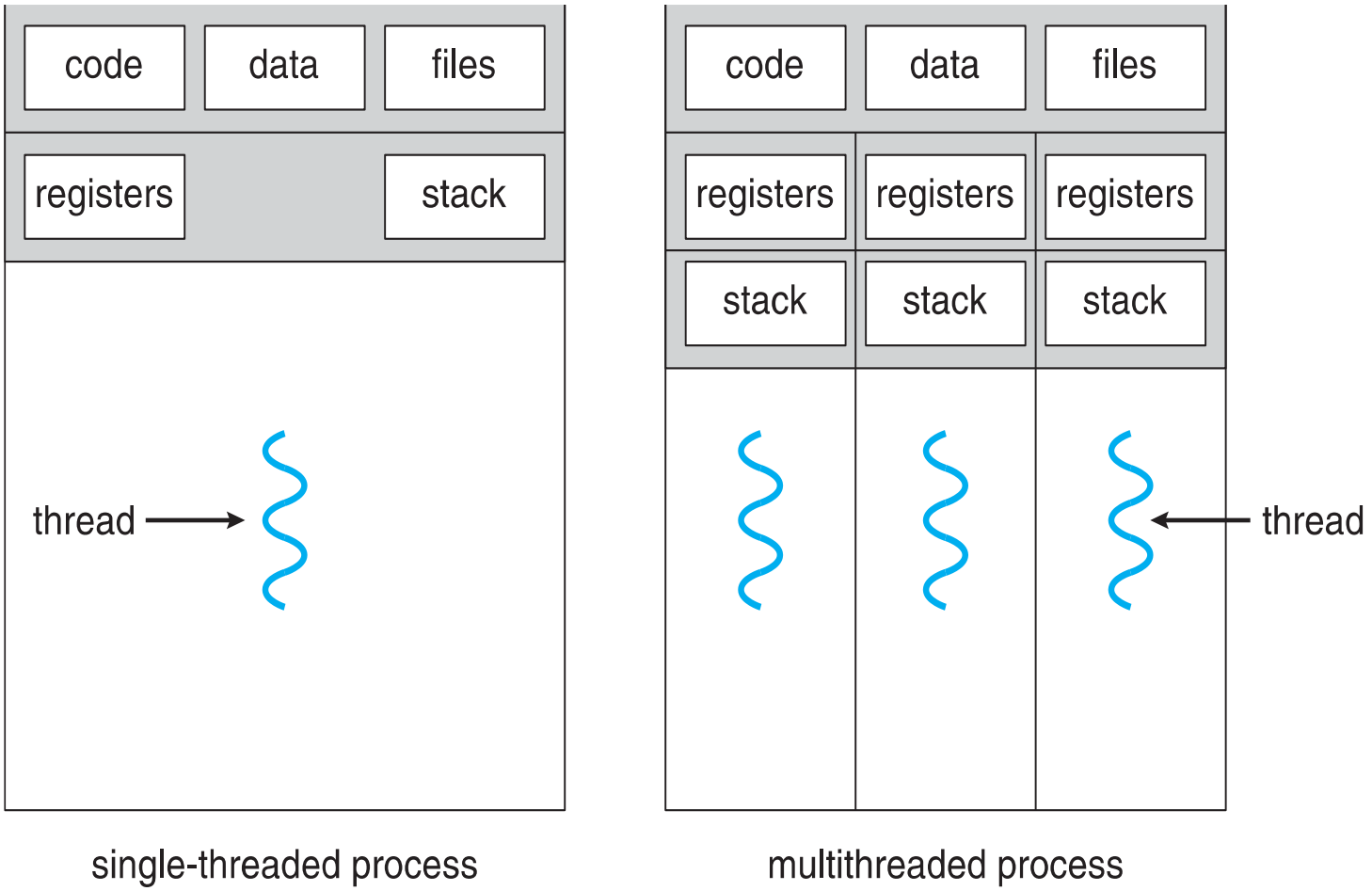
همروندی در مقابل توازی

- اجرای همروند روی یک پردازنده

- اجرای موازی روی دو پردازنده



پردازه های تک و چند ریسه ای



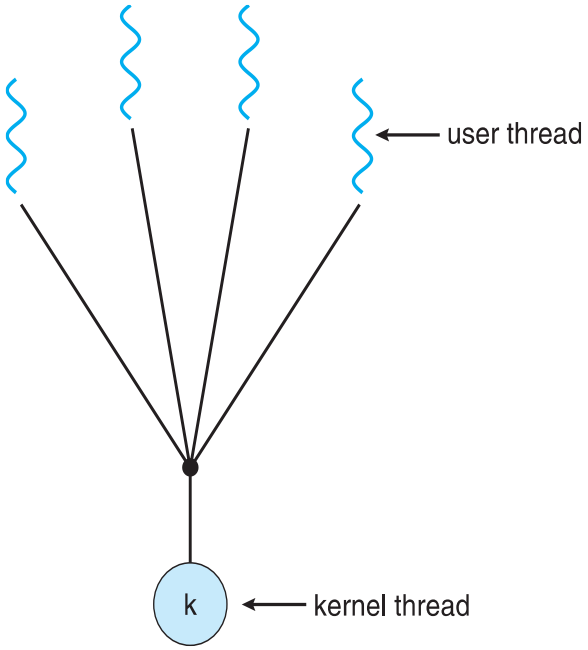
نحوه زمان بندی ریسه ها: ریسه های سطح کاربر و هسته

- **ریسه های سطح کاربر:** ریسه هایی که در مد کاربر اجرا می شوند و مدیریت آنها به وسیله کتابخانه سطح کاربر انجام می شود. سه کتابخانه مهم و معروف در این زمینه وجود دارد.
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **ریسه های سطح هسته:** ریسه هایی که مدیریت آنها بو سیله هسته انجام می شود. تقریبا همه سیستم عامل های عمومی مانند سیستم عامل های زیر این نوع ریسه را پشتیبانی می نمایند.
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

- **مدل های چند ریسه ای**
  - چند به یک
  - یک به یک
  - چند به چند

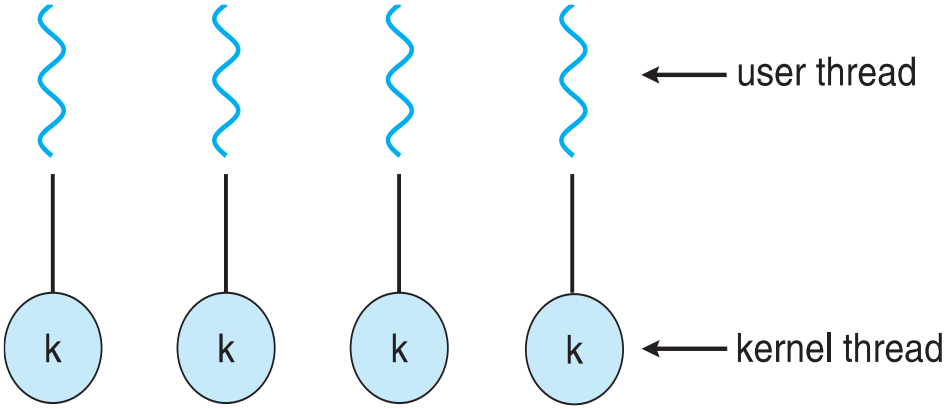
مدل چند به یک

- در این مدل چند ریسه سطح کاربر به یک ریسه سطح هسته نگاشت داده می شود.
- کاستی های این روش:
  - اگر یکی از ریسه ها مسدود شود همه ریسه ها مسدود می شوند.
  - نمی توان از امکانات چند پردازنده ای سیستم استفاده نمود زیرا در هر لحظه تنها یک ریسه در حال اجرا خواهد بود.
- در حال حاضر سیستم های کمی مانند نمونه های زیر از این روش استفاده می کنند.
  - **Solaris Green Threads**
  - **GNU Portable Threads**



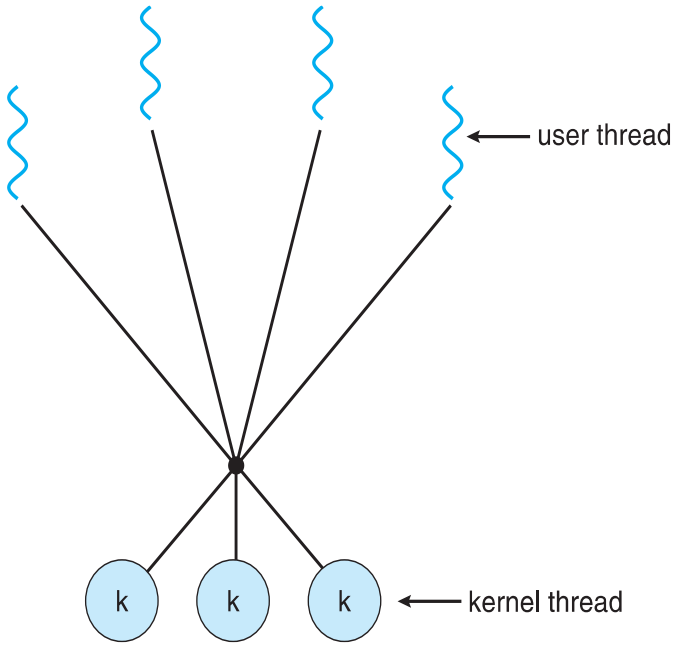
مدل یک به یک

- در این مدل یک ریسه سطح کاربر به یک ریسه سطح هسته نگاشت داده می شود.
- ایجاد هر ریسه سطح کاربر سبب ایجاد یک ریسه در سطح هسته می گردد.
- در این حالت همروندی بهتر از مدل چند به یک می باشد.
- به دلیل محدود سازی سربار، معمولا تعداد ریسه ها را برای هر پردازه محدود می کنند.
- نمونه سیستم عامل هایی که از این مدل استفاده می کنند.
  - Windows
  - Linux
  - Solaris 9 and later



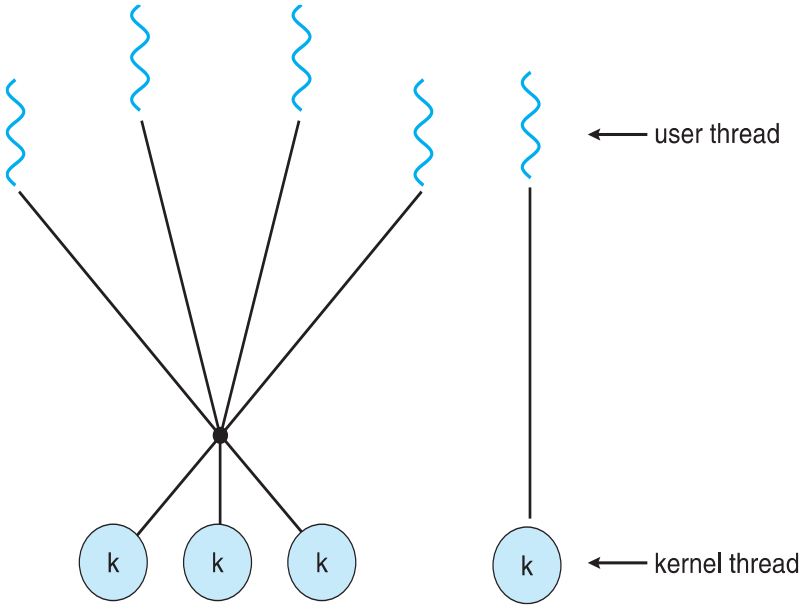
مدل چند به چند

- در این مدل چند ریسه سطح کاربر به چند ریسه سطح هسته نگاشت داده می شود.
- این روش به سیستم عامل اجازه می دهد که تعداد مناسبی ریسه هسته ایجاد نماید.
- نمونه سیستم عامل ها
  - Solaris prior to version 9
  - Windows with the *ThreadFiber* package



مدل دو سطحی

- این مدل مشابه مدل چند به چند است اما اجازه می دهد یک ریسه سطح کاربر به یک ریسه سطح هسته نگاشت داده شود.
- نمونه سیستم عامل ها
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



کتابخانه ریسه

- **کتابخانه ریسه یک API سطح برنامه نویسی برای ایجاد و مدیریت ریسه به برنامه نویس می دهد.**
- دو روش برای پیاده سازی این کتابخانه وجود دارد
  - کتابخانه در فضای حافظه کاربر
  - کتابخانه ای که توسط سیستم عامل پشتیبانی می شود.

- یکی از این کتابخانه ها Pthreads است
  - به هر دو صورت کتابخانه سطح کاربر و سطح هسته وجود دارد.
  - API استاندارد POSIX برای ایجاد و همگام سازی ریسه ها را پشتیبانی می نماید.
  - در سیستم عامل های Solaris, Linux, Mac OS استفاده شده است.

مثال از Pthreads

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }
```

مثال از Pthreads

```
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

مثال از Pthreads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

مثال از Windows

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;

   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }
```

ریسه در جاوا

- ریسه های جاوا به وسیله JVM مدیریت می شوند.
- معمولا بوسیله مدل ریسه پشتیبانی شده توسط سیستم عامل ها پیاده سازی می شود.
- یک ریسه در جاوا بوسیله یکی از دو روش زیر ایجاد می شود
  - مشتق نمودن کلاس Thread
  - پیاده سازی واسطه Runnable

```
public interface Runnable
{
    public abstract void run();
}
```

یک نمونه از ریسه جاوا

```
class Sum
{
   private int sum;

   public int getSum() {
     return sum;
   }

   public void setSum(int sum) {
     this.sum = sum;
   }
}

class Summation implements Runnable
{
   private int upper;
   private Sum sumValue;

   public Summation(int upper, Sum sumValue) {
     this.upper = upper;
     this.sumValue = sumValue;
   }

   public void run() {
     int sum = 0;
     for (int i = 0; i <= upper; i++)
        sum += i;
     sumValue.setSum(sum);
   }
}
```

یک نمونه از ریسه جاوا

```
public class Driver
{
   public static void main(String[] args) {
    if (args.length > 0) {
     if (Integer.parseInt(args[0]) < 0)
      System.err.println(args[0] + " must be >= 0.");
     else {
      Sum sumObject = new Sum();
      int upper = Integer.parseInt(args[0]);
      Thread thrd = new Thread(new Summation(upper, sumObject));
      thrd.start();
      try {
         thrd.join();
         System.out.println
                 ("The sum of "+upper+" is "+sumObject.getSum());
      } catch (InterruptedException ie) { }
     }
    }
    else
     System.err.println("Usage: Summation <integer value>"); }
}
```

پایان جلسه ۶

مسایل ریسه ها

- معنای فراخوان های سیستمی fork() و exec()
- پردازش سیگنا ل (Signal handling)
  - همزمان و نا همزمان
- پایان دادن به کار ریسه پیش از پایان آن (Thread cancellation)
  - نا همزمان یا با تاخیر
- حافظه محلی ریسه
- فعال سازی های برنامه ریز (ارتباط بین هسته و کتابخانه ریسه)
- پردازش رخداد ها (Event handling)

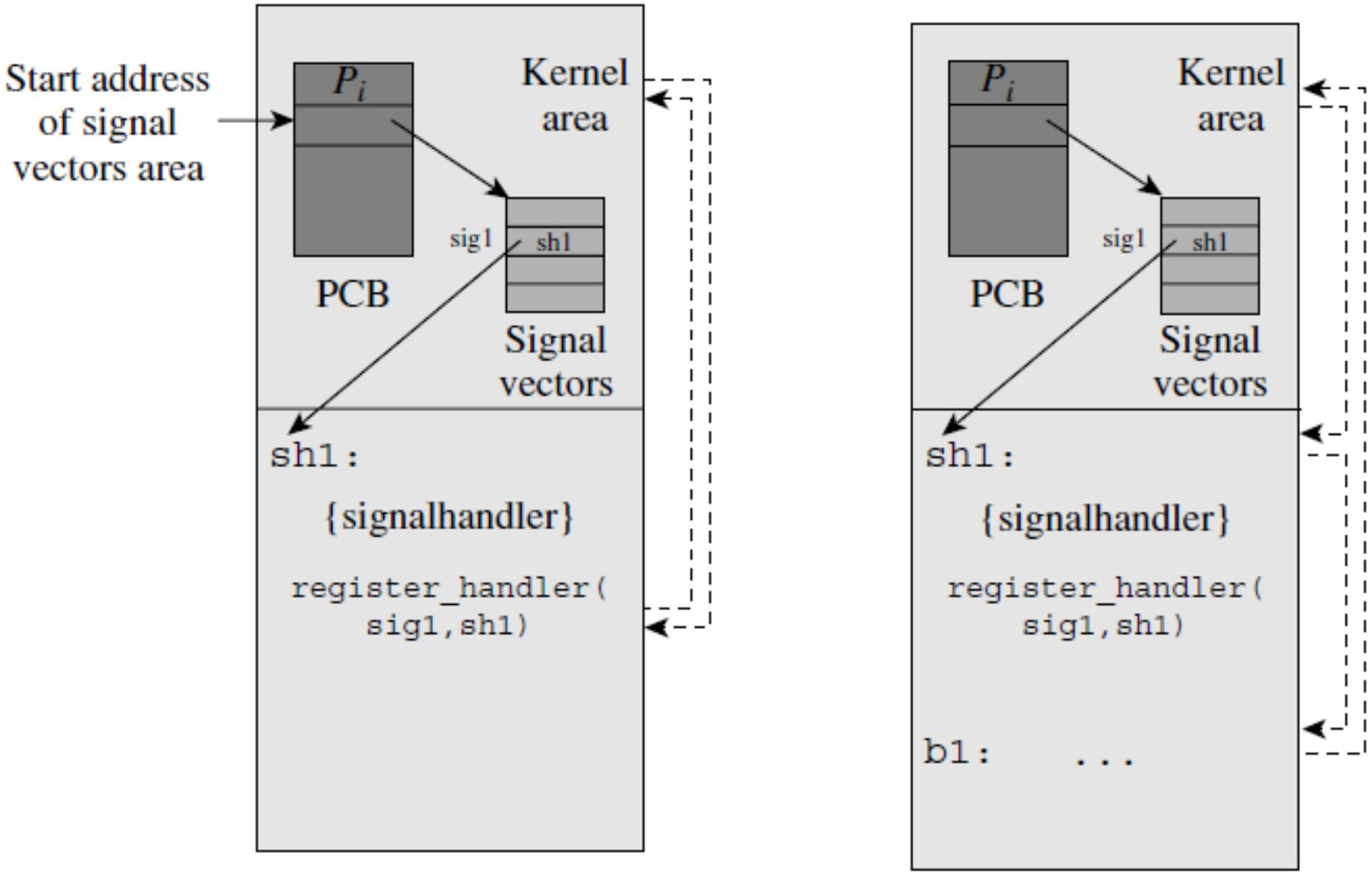
معنای فراخوان های سیستمی

- تفاوت fork و exec
- هنگام فراخوانی فراخوان سیستمی ()fork، یک نسخه از پردازه ساخته می شود. اگر یک پردازه دارای چند ریسه باشد آیا باید همه ریسه ها در پردازه فرزند ساخته شوند یا تنها یکی از آنها؟

پردازش سیگنال

- از سیگنال ها در سیستم عامل های مبتنی بر یونیکس برای اطلاع رسانی از پیش آمدن یک رخداد استفاده می شود.
- از یک پردازنده سیگنال برای پردازش سیگنال استفاده می شود.
- مراحل پردازش سیگنال
  - تولید سیگنال براساس یک رخداد مانند تقسیم بر صفر
  - تحویل سیگنال به پردازه و ریسه هدف
  - پردازش سیگنال توسط پردازنده سیگنال
    - پردازنده پیش فرض
    - پردازنده تعریف شده توسط کاربر
- هر سیگنال یک پردازنده پیش فرض در هسته دارد
- اگر یک پردازه بیش از یک ریسه داشته باشد
  - تحویل سیگنال به ریسه هدف
  - تحویل سیگنال به پردازه هدف
  - تحویل سیگنال به همه ریسه ها
  - تعبیه نمودن یک ریسه برای پردازنده سیگنال و تحویل رخداد به آن ریسه

پردازش سیگنال



پایان دادن به کار ریسه پیش از پایان آن

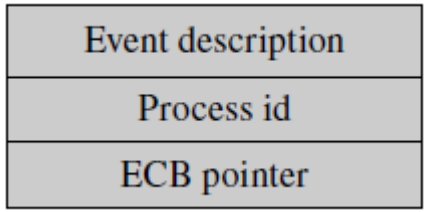
- دو حالت می توان به کار یک ریسه پایان داد
- ناهمزمان: پس از دریافت این رخداد، ریسه به کار خود پایان می دهد.
- با تاخیر: ریسه در فواصل منظمی یک flag را بررسی نموده و مشخص می کند که آیا این ریسه باید به کار خود پایان دهد یا نه. در pthread به این نقاط Cancelation point می گویند.
- مشکل پایان دادن به کار ریسه هنگامی حاد می شود که یک ریسه در حال به روز رسانی یک منبع مانند فایل باشد. این مشکل در حالت ناهمزمان ممکن است مشکل ایجاد نماید. در این حالت ممکن است منبع آزاد نشود.
- نمونه کد

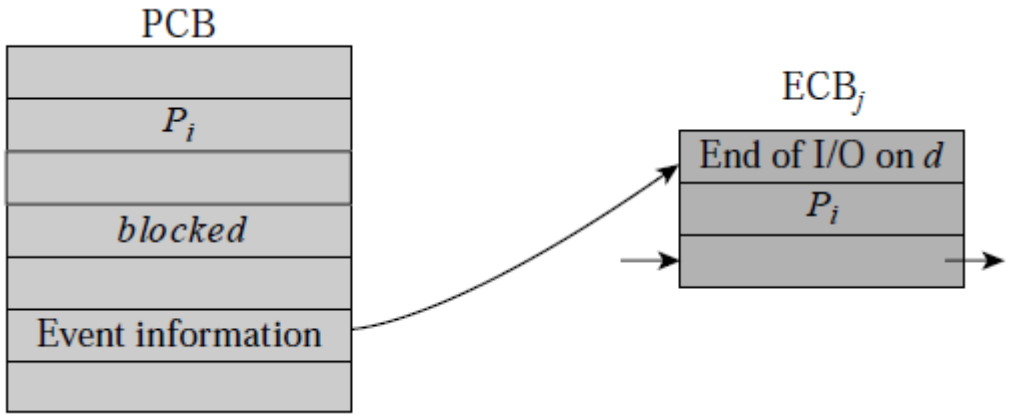
```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

. . .

/* cancel the thread */
pthread_cancel(tid);
```

حافظه محلی برای یک ریسه

- هر ریسه می تواند حافظه محلی مربوط به خود را داشته باشد
- حافظه محلی با متغیر های محلی تفاوت دارد و مانند داده های ایستا است.
- حافظه محلی برای هر ریسه یکتا است.
- نمونه ThreadLocal در جاوا

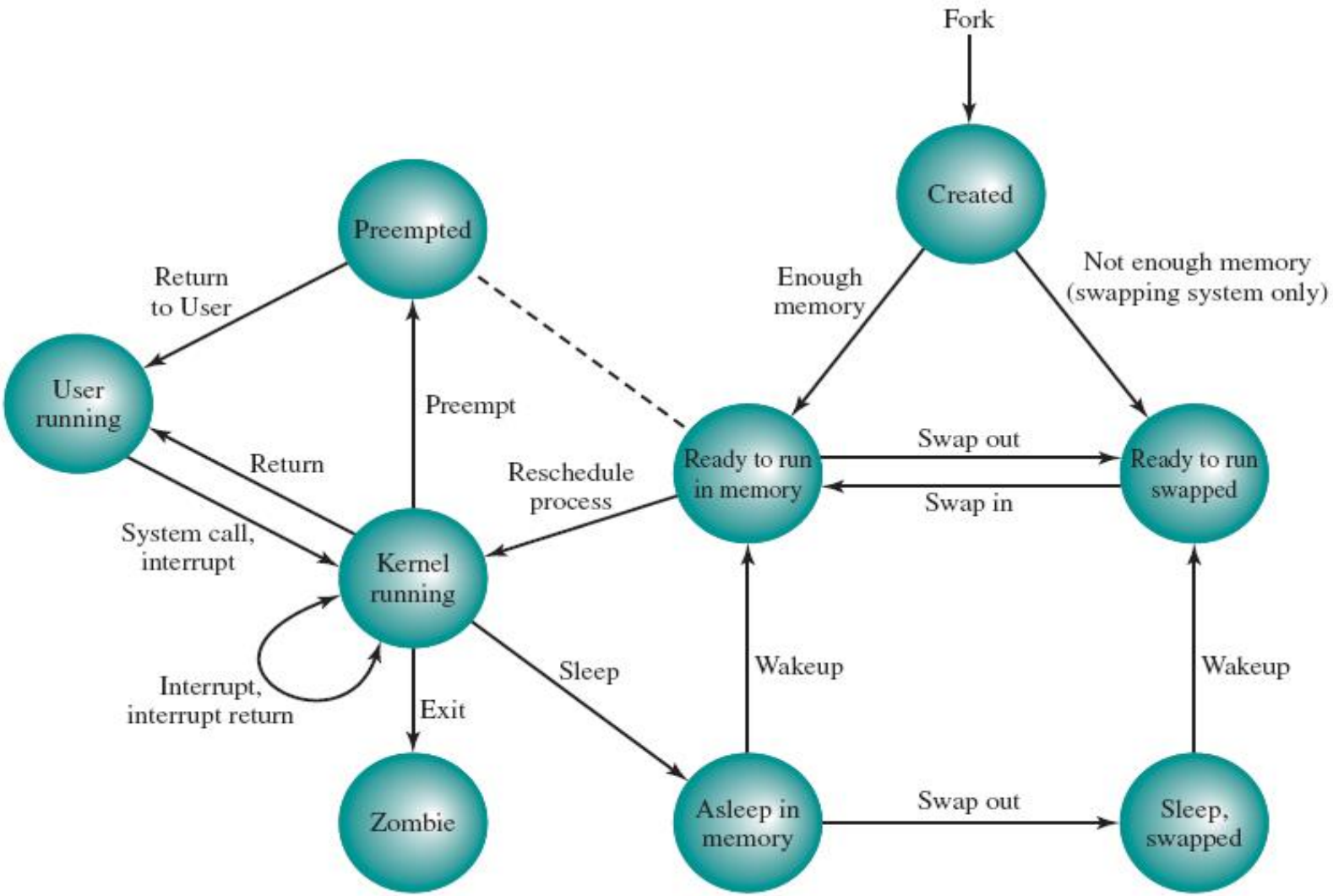
پردازش رخداد ها





پایان جلسه ۷

نمودار تغییر حالت پردازه ها در یونیکس



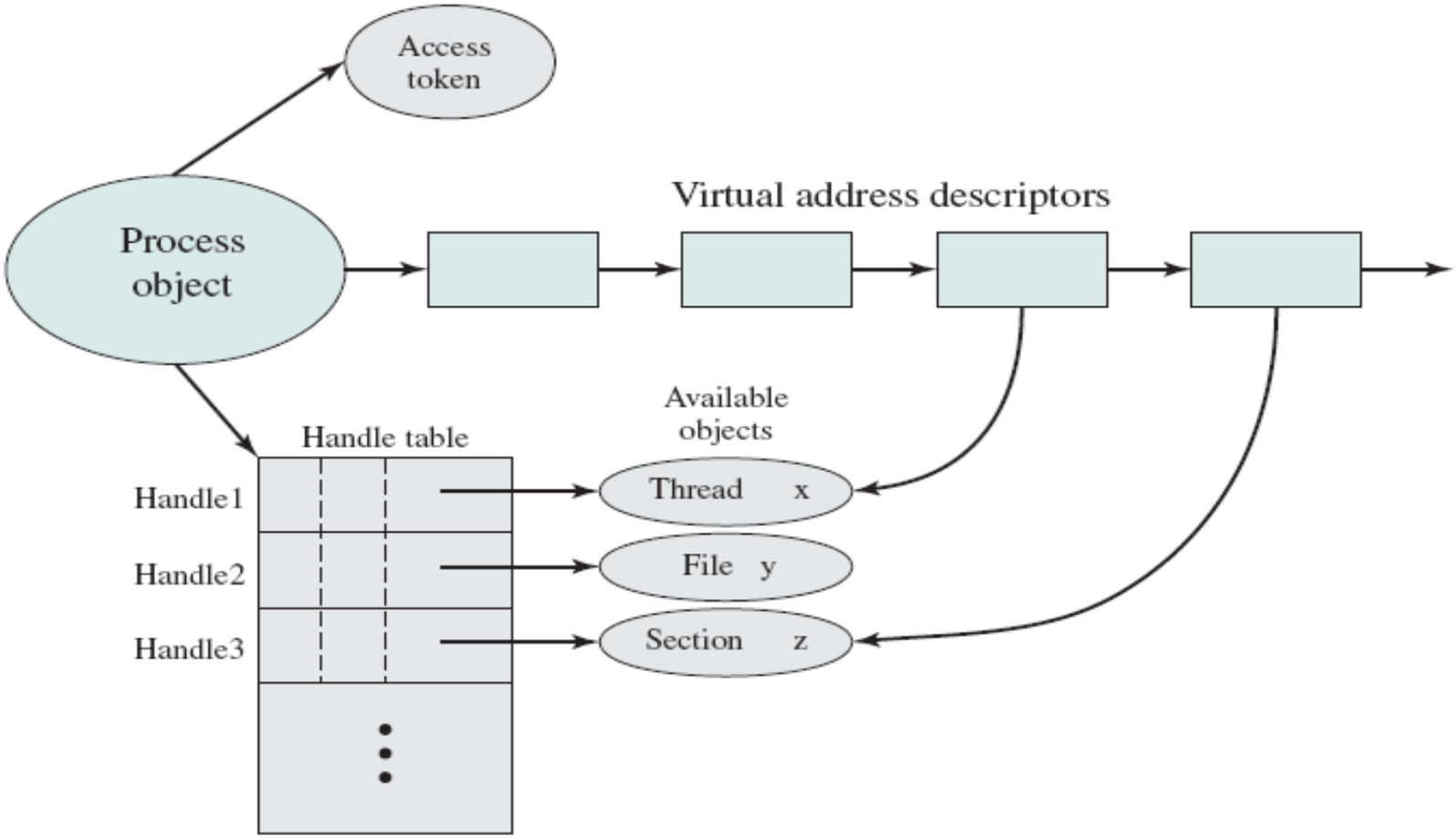
ساختمان داده PCB در یونیکس در سطح هسته

Process status	Current state of process.
Pointers	To U area and process memory area (text, data, stack).
Process size	Enables the operating system to know how much space to allocate the process.
User identifiers	The **real user ID** identifies the user who is responsible for the running process. The **effective user ID** may be used by a process to gain temporary privileges associated with a particular program; while that program is being executed as part of the process, the process operates with the effective user ID.
Process identifiers	ID of this process; ID of parent process. These are set up when the process enters the Created state during the fork system call.
Event descriptor	Valid when a process is in a sleeping state; when the event occurs, the process is transferred to a ready-to-run state.
Priority	Used for process scheduling.
Signal	Enumerates signals sent to a process but not yet handled.
Timers	Include process execution time, kernel resource utilization, and user-set timer used to send alarm signal to a process.
P_link	Pointer to the next link in the ready queue (valid if process is ready to execute).
Memory status	Indicates whether process image is in main memory or swapped out. If it is in memory, this field also indicates whether it may be swapped out or is temporarily locked into main memory.

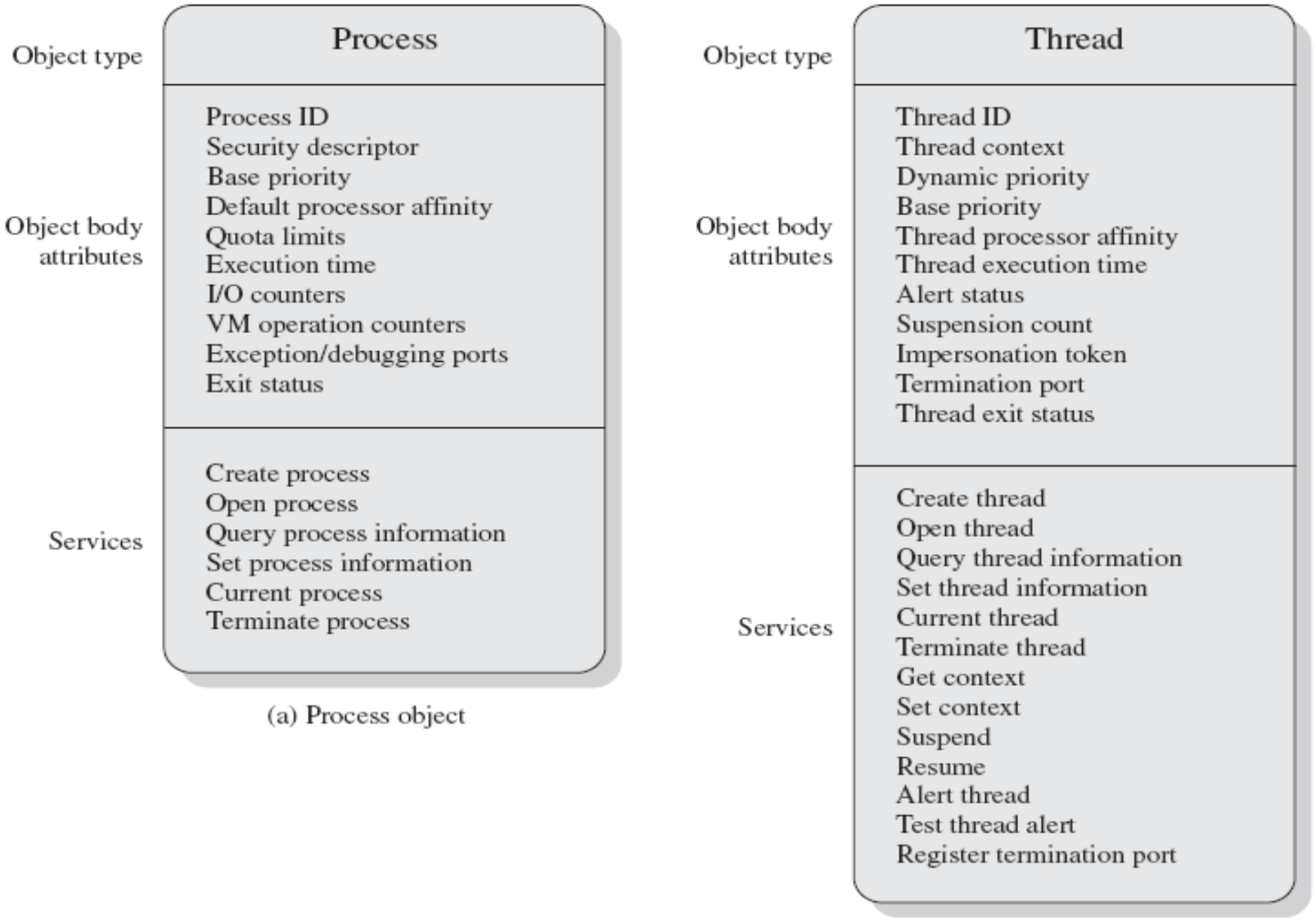
ساختمان داده PCB در یونیکس در سطح کاربر

Process table pointer	Indicates entry that corresponds to the U area.
User identifiers	Real and effective user IDs. Used to determine user privileges.
Timers	Record time that the process (and its descendants) spent executing in user mode and in kernel mode.
Signal-handler array	For each type of signal defined in the system, indicates how the process will react to receipt of that signal (exit, ignore, execute specified user function).
Control terminal	Indicates login terminal for this process, if one exists.
Error field	Records errors encountered during a system call.
Return value	Contains the result of system calls.
I/O parameters	Describe the amount of data to transfer, the address of the source (or target) data array in user space, and file offsets for I/O.
File parameters	Current directory and current root describe the file system environment of the process.
User file descriptor table	Records the files the process has opened.
Limit fields	Restrict the size of the process and the size of a file it can write.
Permission modes fields	Mask mode settings on files the process creates.

پردازه و منابع آن در Windows

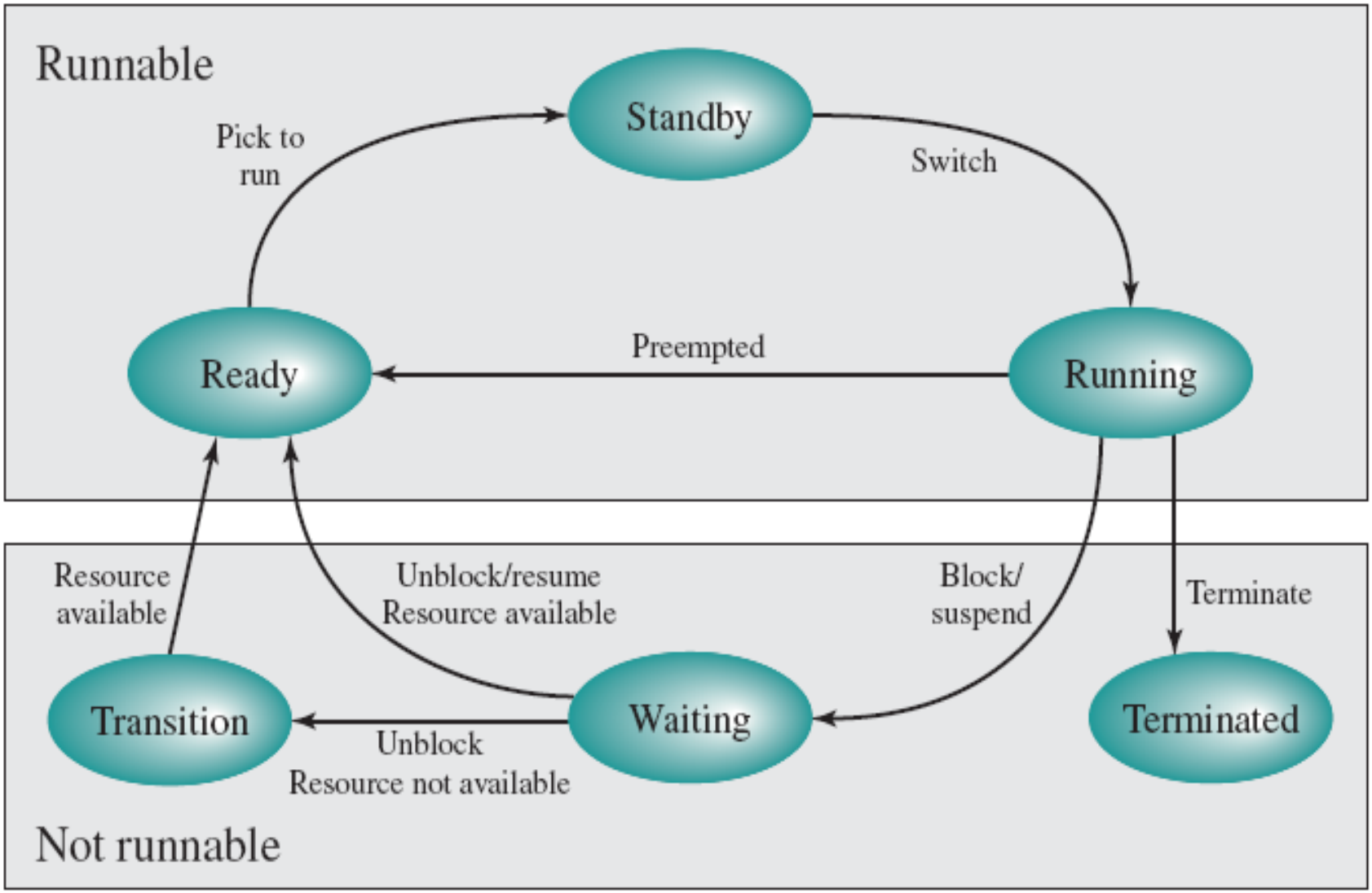


پردازه و ریسه در Windows



(a) Process object

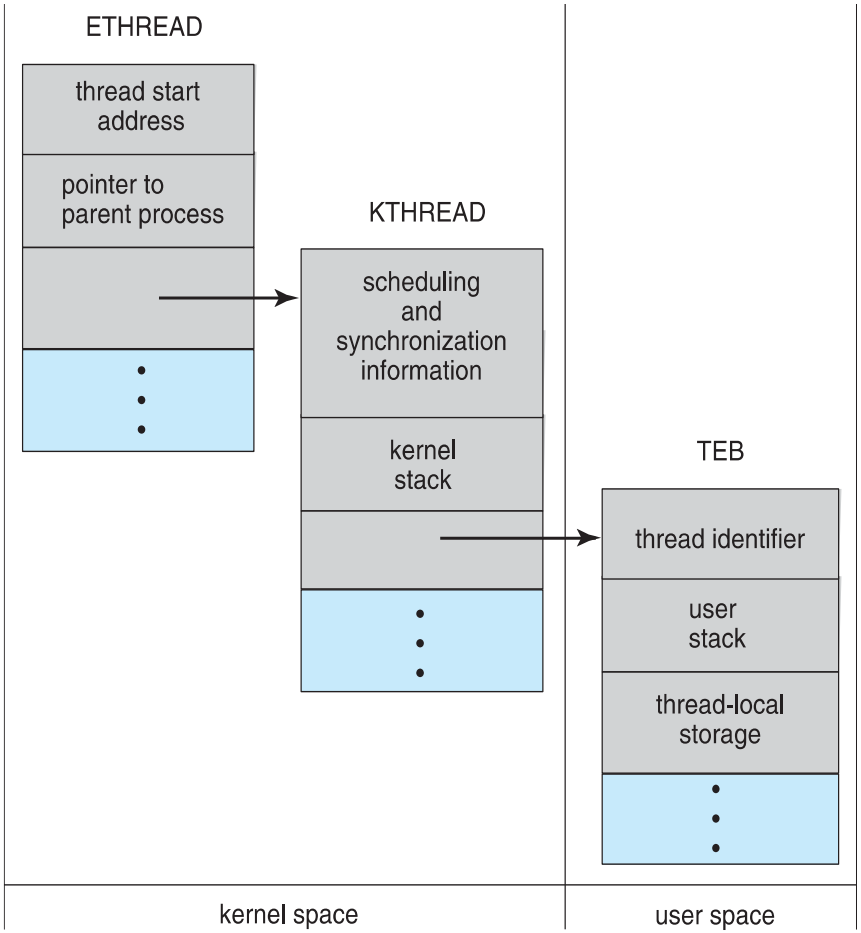
نمودار تغییر حالت ریسه ها در Windows



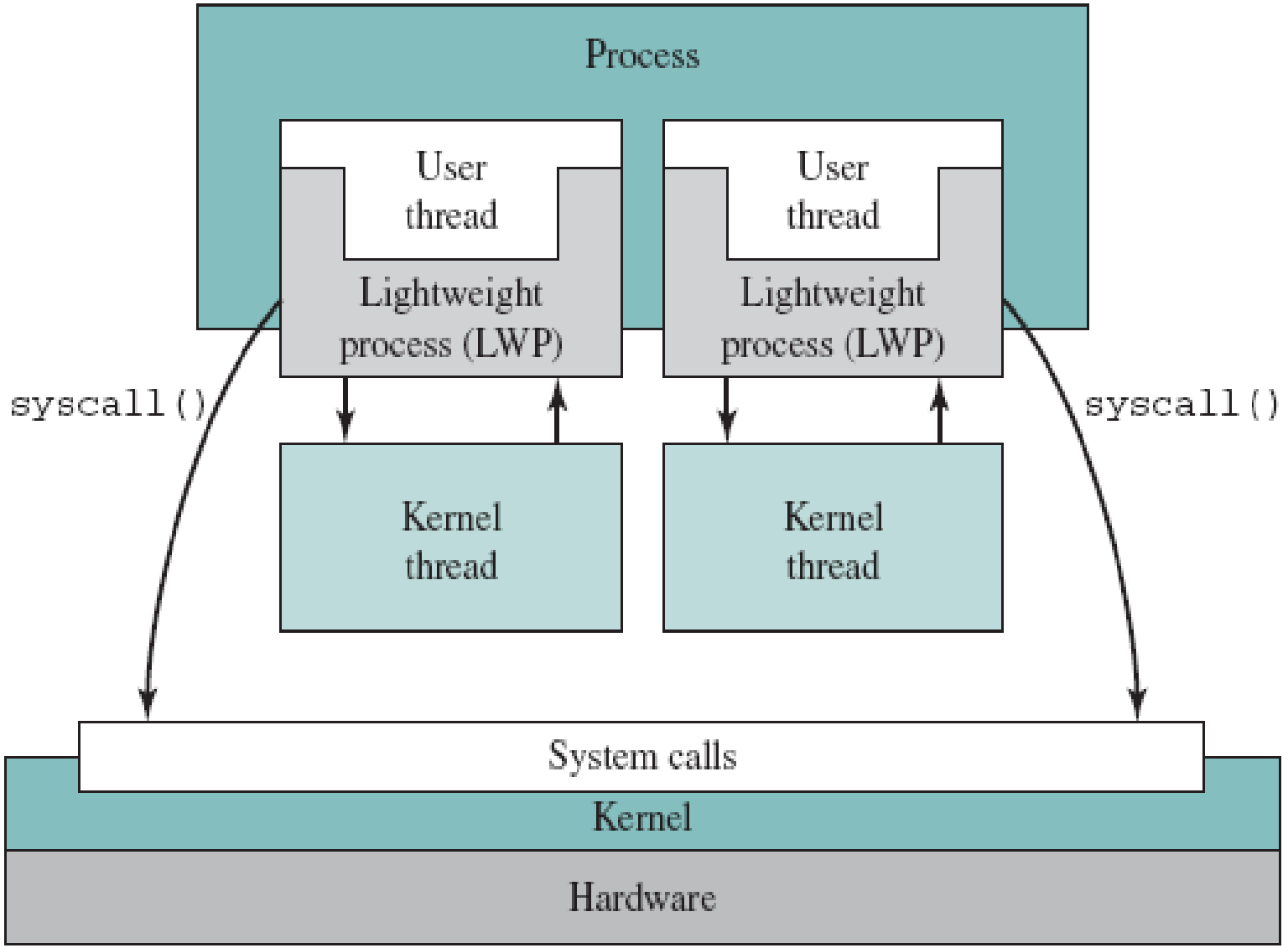
ریسه در Windows

- ویندوز API ویندوز (Win 98, Win NT, Win 2000, Win XP, and Win 7) را پیاده سازی می کند.
- مدل یک به یک را پیاده سازی می کند
- هر ریسه حاوی اطلاعات زیر است.
  - شناسه ریسه
  - مجموعه ثبات های پردازنده
  - پشته های مجزا برای مد کاربر و مد هسته
  - حافظه اختصاصی
- به مجموعه اطلاعات ثبات ها، پشته و حافظه یک ریسه Context آن ریسه می گویند.
- ویندوز از سه ساختمان داده برای مدیریت ریسه ها استفاده می کند.
  - ETHREAD (executive thread block)
    - اشاره گر به پردازه ایجاد کننده این ریسه و اشاره گر به KTHREAD
  - KTHREAD (kernel thread block)
    - اطلاعات برنامه ریزی و همگام سازی، پشته مد هسته و اشاره گر به TEB
  - TEB (thread environment block)
    - شناسه ریسه، پشته مدکاربر،حافظه اختصاصی

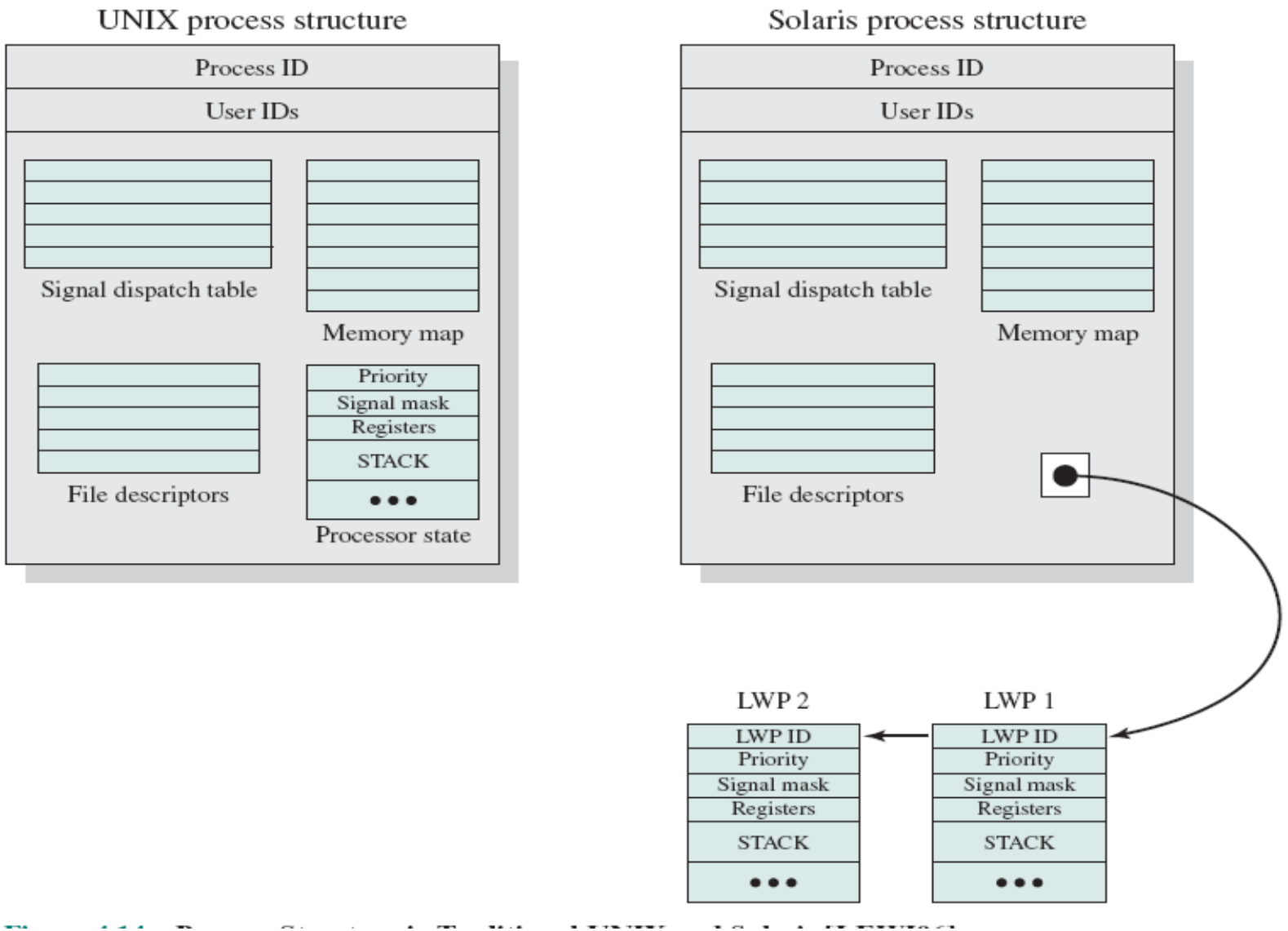
ساختمان داده های Windows برای ریسه



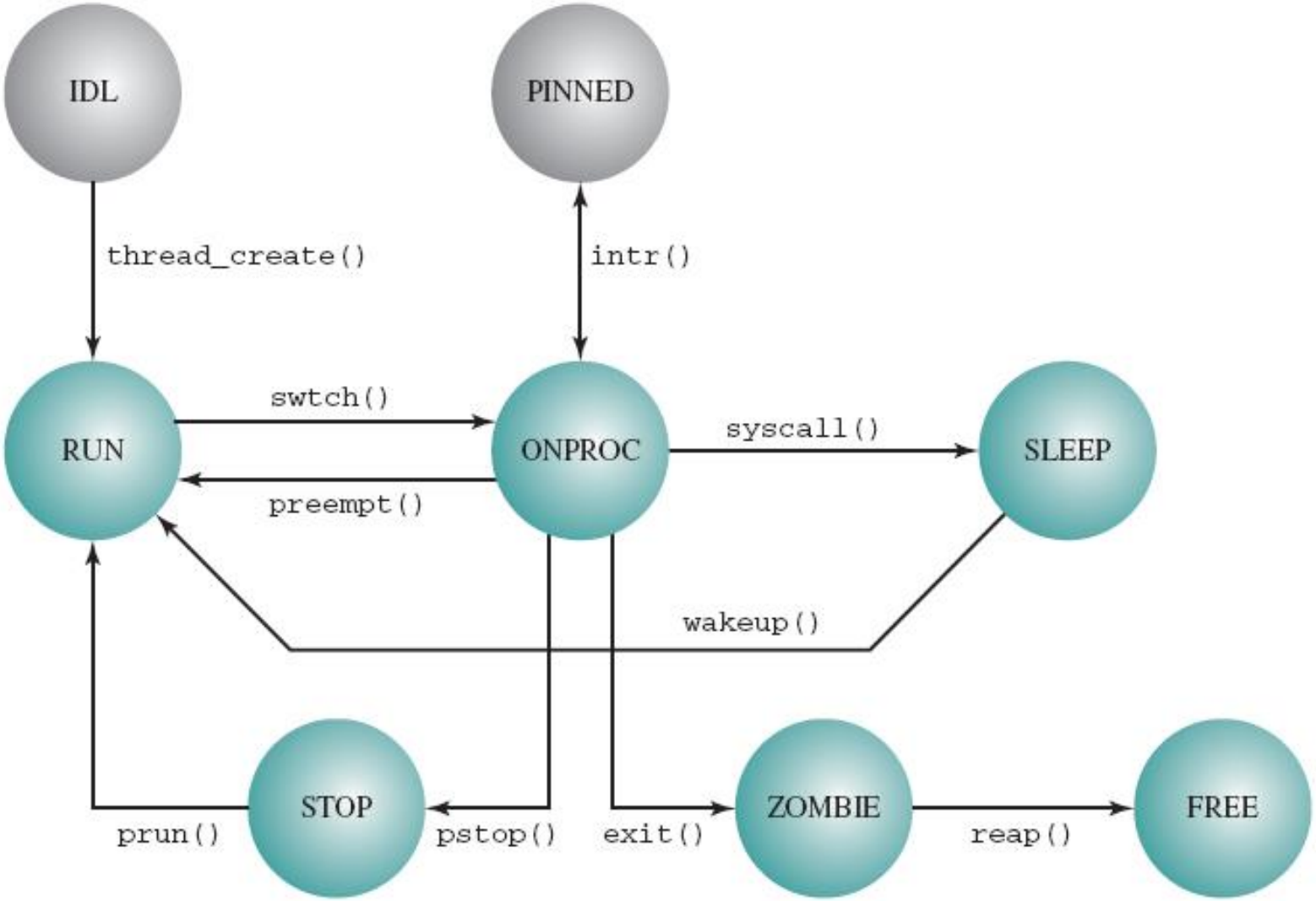
پردازه و ریسه در Solaris



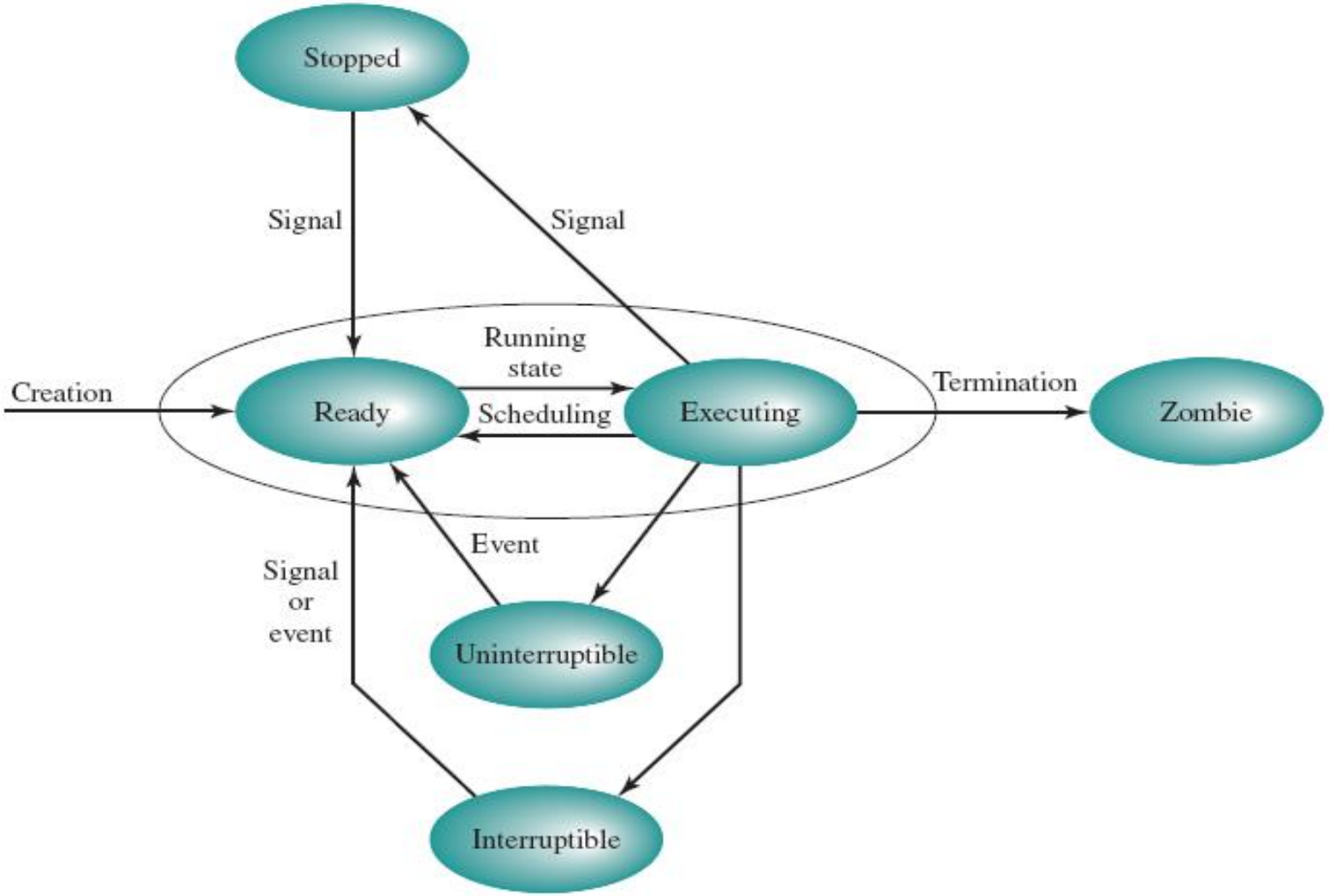
ساختارپردازه در Solaris



نمودار تغییر وضعیت ریسه در Solaris



نمودار تغییر وضعیت پردازه و ریسه در Linux



ریسه در لینوکس

- لینوکس از واژه task استفاده می کند
- ایجاد یک نخ با کمک فراخوان سیستمی **clone()**
- فراخوان سیستمی **clone** () به فرزند اجازه می دهد که فضای آدرس پدر بصورت اشتراکی استفاده نماید. برای اشتراک گذاری از `flag` های زیر استفاده می شود.

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.