

Operating Systems

سیستمهای عامل

مجموعه اسلایدهای شماره ۵

دکتر خانمیرزا

h.khanmirza@kntu.ac.ir

دانشکده کامپیوتر

دانشگاه صنعتی خواجه نصیرالدین طوسی



رسمان‌ها

Threads

■ یک زمینه اجرایی پشت سرهم (Sequential) در داخل فرآیند است

■ چند ریسمانی (multithreading)

■ یک برنامه که از چندین فعالیت همزمان تشکیل شده باشد

■ گاهی با نام چند وظیفگی (multitasking) هم شناخته می‌شود

■ در گذشته فرآیندها عموماً دارای یک زمینه اجرایی بودند. با پیدایش ریسمانها اصطلاح فرآیندهای سبک‌وزن (light-weight process) در برابر فرآیندهای سنگین‌وزن (heavy-weight process) رایج شد.

■ این اصطلاح گمراه کننده است چرا که ریسمانها دارای مفهوم محافظت و فضای آدرس جداگانه نیستند

■ فرآیند سنگین وزن یعنی یک فرآیند با یک ریسمان

■ چرا با وجود فرآیند مفهوم ریسمان هنوز وجود دارد؟

■ یا بعبارت دیگر چرا با وجود فرآیند، علاقمند هستیم در یک فرآیند چندین ریسمان داشته باشیم؟

■ در برنامه مقابل چه اتفاقی میفتد؟ آیا عبارت «Finished!» چاپ خواهد شد؟

```
1 int main(void){  
2     while(1);  
3     printf("Finished!");  
4 }
```

■ در فرآیند تک ریسمانی روال اجرا پشت سرهم است، بنابراین اجرا هیچگاه به خط دوم کد نمی‌رسد

```

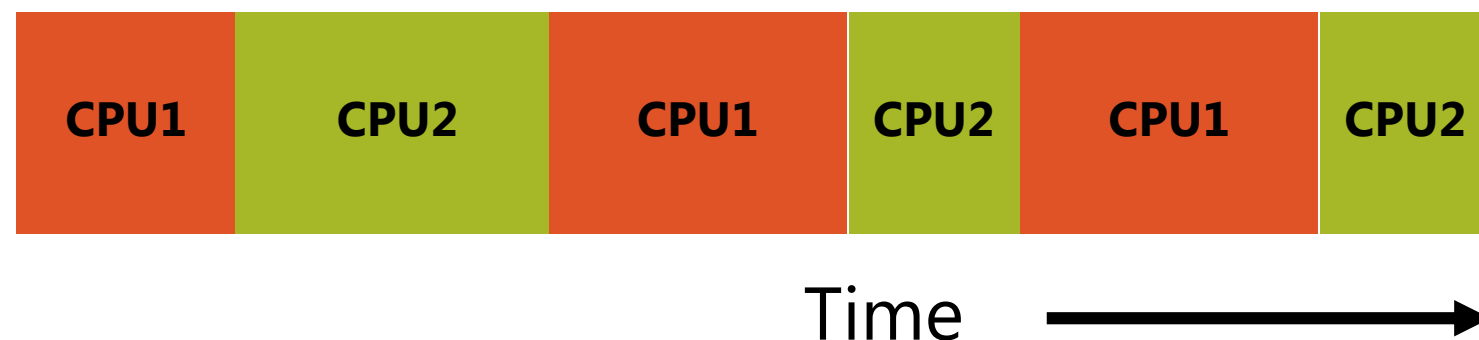
1  int main(void){
2      thread_fork(while(1));
3      thread_fork(sprintf("Finished!"));
4  }

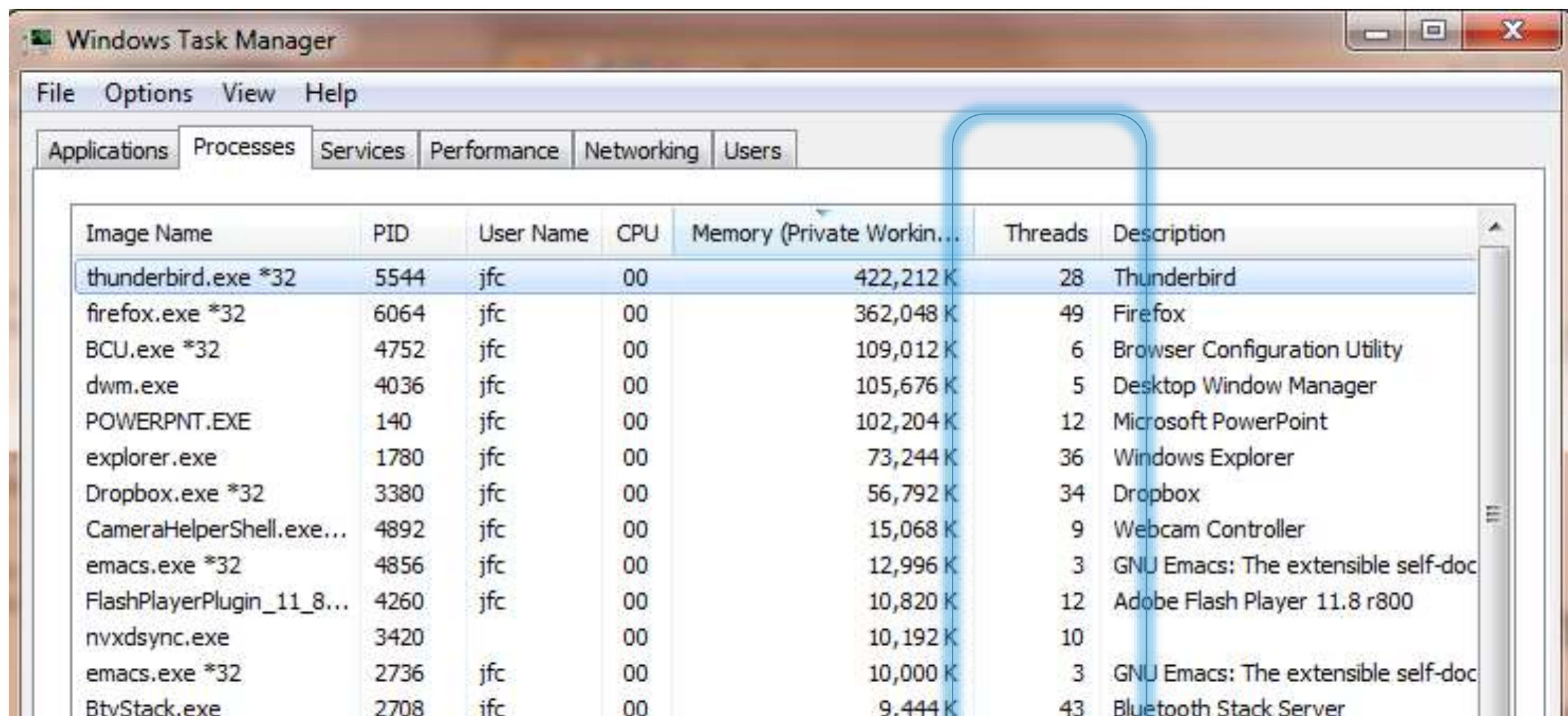
```

■ در کد بالا برای هر خط کد یک ریسمان ایجاد می‌شود و این دو ریسمان به شکل همزمان اجرا می‌شوند

■ با یک نگاه دیگر انگار در سیستم عامل دو پردازنده مجازی وجود دارد که هر ریسمان در آن پردازنده به شکل اختصاصی اجرا می‌شود

■ به یاد دارید که ریسمان را یک زمینه اجرایی تعریف کردیم که شامل اجزای یک پردازنده بود که برای اجرای یک ریسمان کافی باشد



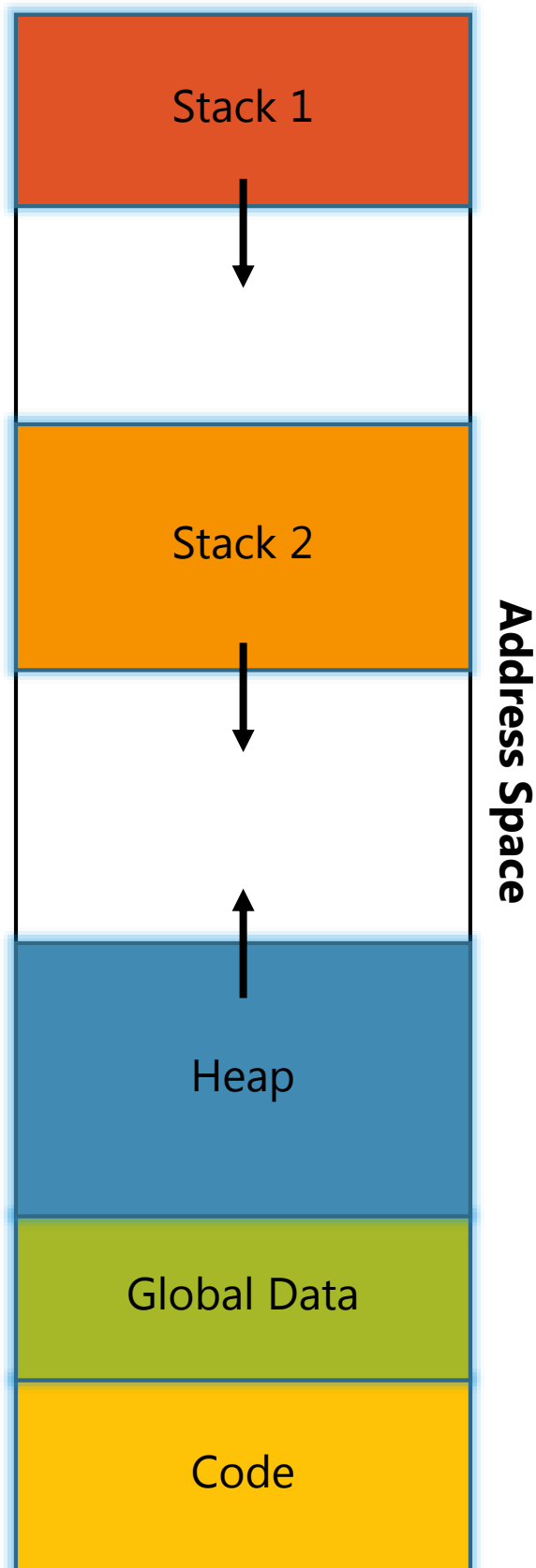


Windows Task Manager

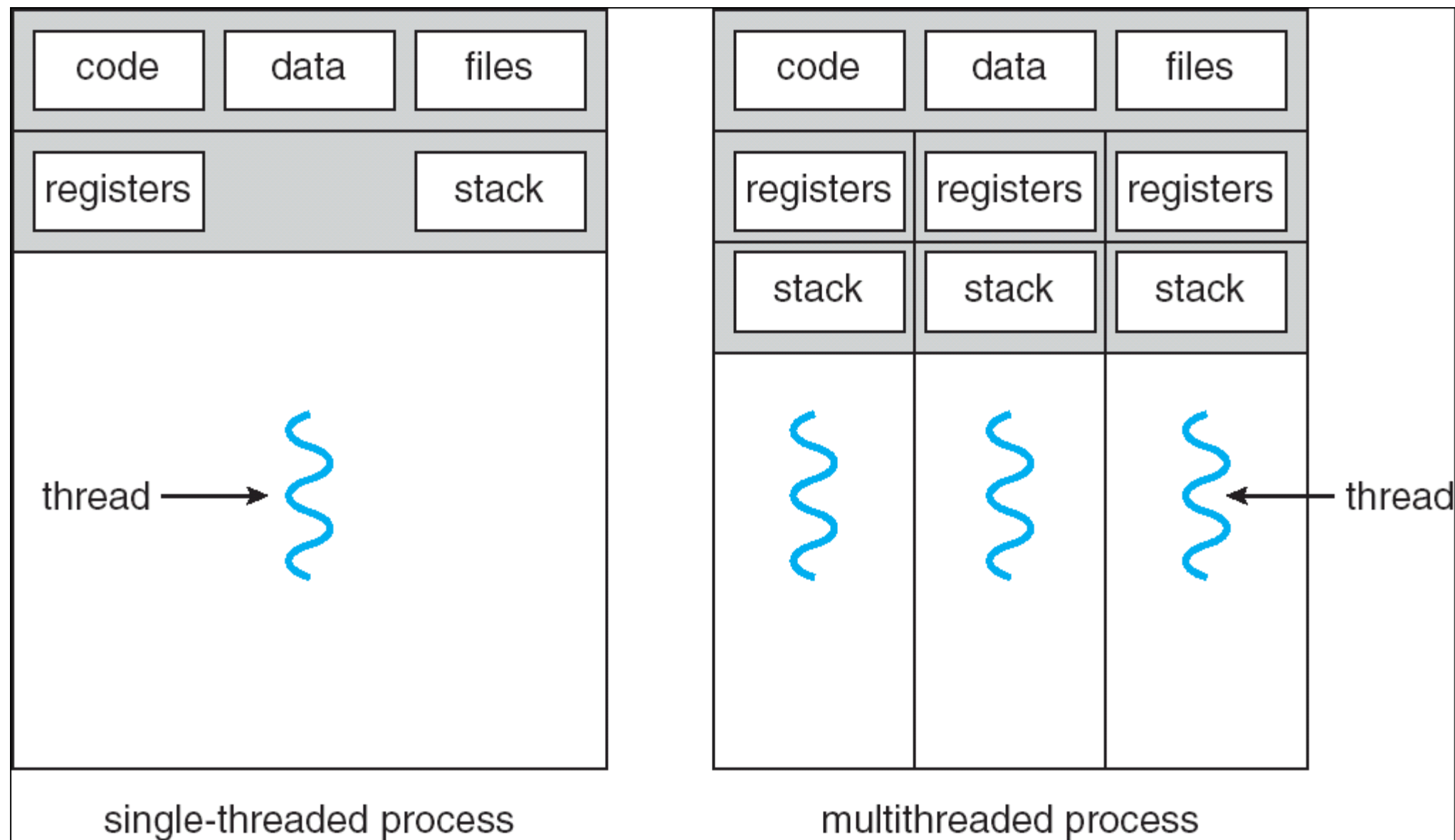
File Options View Help

Applications Processes Services Performance Networking Users

Image Name	PID	User Name	CPU	Memory (Private Workin...	Threads	Description
thunderbird.exe *32	5544	jfc	00	422,212 K	28	Thunderbird
firefox.exe *32	6064	jfc	00	362,048 K	49	Firefox
BCU.exe *32	4752	jfc	00	109,012 K	6	Browser Configuration Utility
dwm.exe	4036	jfc	00	105,676 K	5	Desktop Window Manager
POWERPNT.EXE	140	jfc	00	102,204 K	12	Microsoft PowerPoint
explorer.exe	1780	jfc	00	73,244 K	36	Windows Explorer
Dropbox.exe *32	3380	jfc	00	56,792 K	34	Dropbox
CameraHelperShell.exe...	4892	jfc	00	15,068 K	9	Webcam Controller
emacs.exe *32	4856	jfc	00	12,996 K	3	GNU Emacs: The extensible self-doc
FlashPlayerPlugin_11_8...	4260	jfc	00	10,820 K	12	Adobe Flash Player 11.8 r800
nvxdsync.exe	3420		00	10,192 K	10	
emacs.exe *32	2736	jfc	00	10,000 K	3	GNU Emacs: The extensible self-doc
BtvStack.exe	2708	ifc	00	9,444 K	43	Bluetooth Stack Server



- در این فرآیند دو ریسمانی
- دو پشته برای هر ریسمان وجود دارد (چرا؟)
 - طبیعتاً هر دو پشته به سمت آدرسهای پایین رشد می کنند
 - ممکن است پشته یک ریسمان رشد کرده و پشته دیگری را خراب کند
- دو مجموعه مقادیر برای ثباتهای پردازنده وجود دارد
 - مفهوم پردازنده مجازی
- با توجه به این شکل هر ریسمان یکسری اطلاعات اختصاصی خود را داشته و برخی اطلاعات مشترک با سایر ریسمانها دارد



■ چند رسمان یک فرآیند چند مجموعه وضعیت (state) دارند

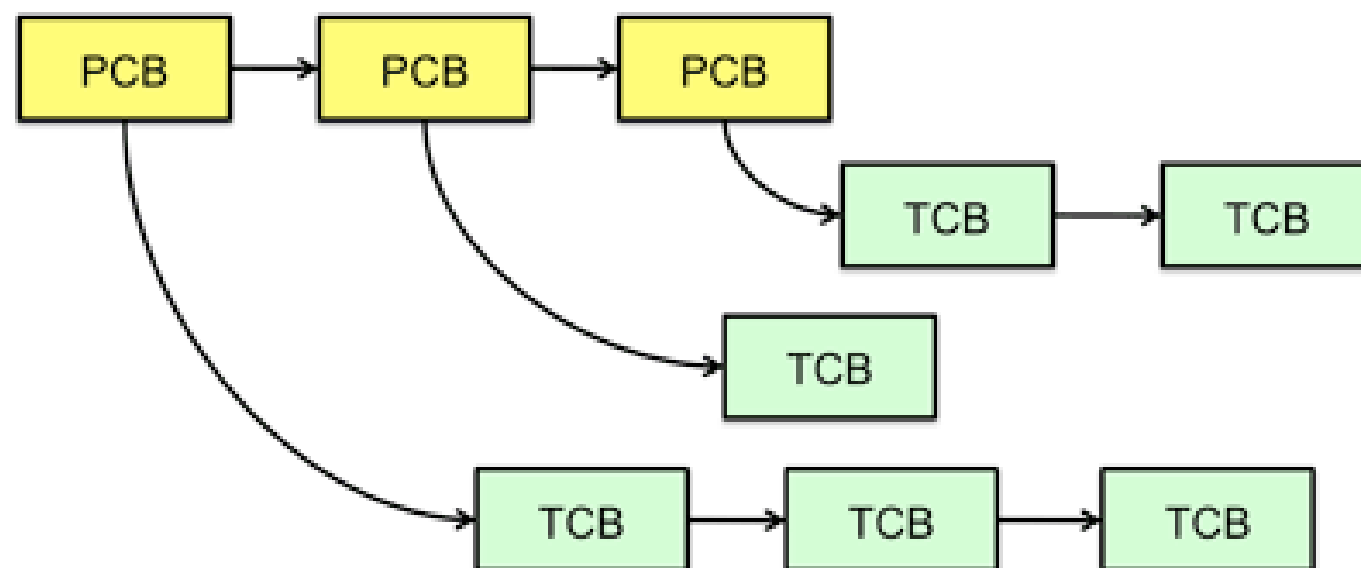
■ وضعیت مشترک (shared state)

- برای تمامی رسمانهای یک فرآیند مشترک است و برای همه آنها قابل دسترسی (خواندن و نوشتن) است
- محتوای حافظه در بخش کد، داده و heap و فایلها و ارتباطهای شبکه، فضای آدرس
- این اطلاعات در بلاک PCB فرآیند ذخیره می‌شوند

■ وضعیت خصوصی (private state)

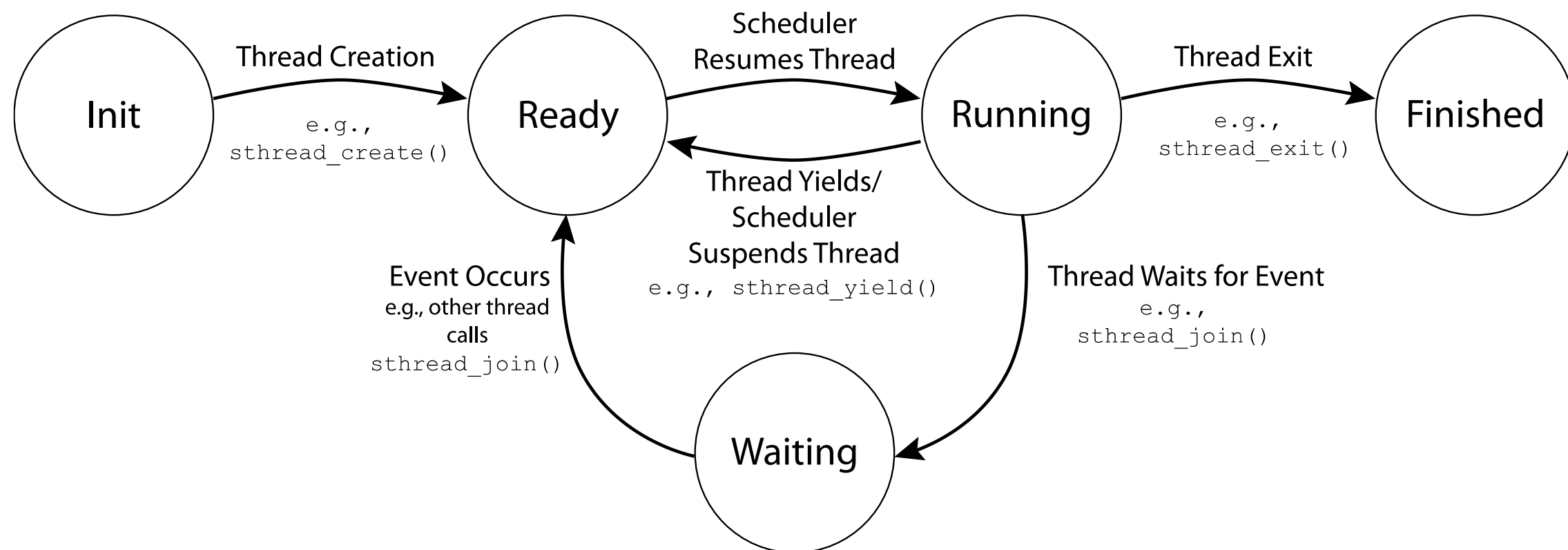
- مقادیر ثباتهای پردازنده و پشته
- اطلاعات زمان‌بندی مانند تقدم، مدت زمان اجرا، وضعیت اجرا
- اشاره‌گر به بلاک فرآیند
- این اطلاعات برای هر رسمان در TCB (Thread Control Block) ذخیره می‌شود

■ هر بلاک فرآیند به چندین بلاک رسمان اشاره می کند



- تعویض زمینه بین رسمانهای یک فرآیند یک تعویض زمینه سریع و ساده تر است
- تعویض زمینه بین رسمانهای دو فرآیند مختلف همان تعویض زمینه بین دو فرآیند است

■ دوره زندگی ريسمانها همانند فرآیندهاست



■ برنامه چند رسمانی در مقایسه با چند فرآیندی دارای مزایای زیر است:

■ مصرف حافظه در چند رسمانی کمتر است

■ بخشهای کد و داده و heap مشترک است

■ اشتراک منابع بین رسمانها تقریبا (?) بدون هزینه و یا کم هزینه تر است

■ تعویض زمینه بین رسمانها سریعتر و سبک تر است

■ در تعویض زمینه بین رسمانها کارهای کمتری انجام می گیرد

■ مقادیر برخی ثباتها ثابت بوده و نیاز به بازنشانی (reset) کامل مکانیزم محافظت نیست.

■ جداول آدرس IO نیاز به تغییر ندارد (بعدا توضیح داده می شود)



■ مقایسه دو رهیافت طراحی برنامه

- مرورگر فایرفاکس (Firefox Browser) برای هر برگه (tab) یک رسمان اختصاص می‌دهد
- مرورگر کروم (Chrome Browser) برای هر برگه یک فرآیند ایجاد می‌کند
- کروم برای دسترسی به دیسک و ترافیک شبکه از یک فرآیند Browser استفاده می‌کند
- برای نمایش و تعامل با کاربر برای هر برگه یک فرآیند Renderer ایجاد می‌شود
- هر add-on (افزونه) هم یک فرآیند جداست

■ دقت کنید که این مقایسه در تئوری است و در عمل عوامل زیادی در عملکرد مرورگرها دخالت دارند.

▼ Firefox	14.3	22:09.41	59	180	0.2	4:13.00	696
FirefoxCP Web Content	0.0	0.67	21	0	0.0	0.00	1371
FirefoxCP Web Content	0.0	5:03.37	28	0	0.0	0.00	699
FirefoxCP Web Content	0.0	7.15	25	0	0.0	0.00	702
FirefoxCP Web Content	0.0	15.91	29	1	0.0	0.00	701
FirefoxCP Web Content	0.2	1:01.15	31	137	0.0	0.00	1358
FirefoxCP WebExtensions	0.0	4.43	25	1	0.0	0.00	703

▼ Google Chrome	1.6	54.12	33	4	0.0	0.00	1833
AlertNotificationService	0.0	0.03	2	0	0.0	0.00	1846
chrome_crashpad_handler	0.0	0.02	4	0	0.0	0.00	1839
Google Chrome Helper	0.1	16.13	9	2	0.0	0.00	1844
Google Chrome Helper (GPU)	8.8	15.72	9	92	0.2	2.46	1842
Google Chrome Helper (Renderer)	0.0	0.15	12	0	0.0	0.00	1927
Google Chrome Helper (Renderer)	0.0	0.27	12	0	0.0	0.00	1852
Google Chrome Helper (Renderer)	0.0	0.30	12	0	0.0	0.00	1858
Google Chrome Helper (Renderer)	0.0	0.84	13	0	0.0	0.00	1850
Google Chrome Helper (Renderer)	0.0	0.49	12	0	0.0	0.00	1857
Google Chrome Helper (Renderer)	0.0	0.47	12	0	0.0	0.00	1859
Google Chrome Helper (Renderer)	0.0	0.92	13	0	0.0	0.00	1855
Google Chrome Helper (Renderer)	0.0	1.90	15	0	0.0	0.00	1861
Google Chrome Helper (Renderer)	0.0	0.86	13	0	0.0	0.00	1908
Google Chrome Helper (Renderer)	0.0	1.16	14	0	0.0	0.00	1924
Google Chrome Helper (Renderer)	0.2	1.15	12	4	0.0	0.00	1926
Google Chrome Helper (Renderer)	0.0	1.39	13	0	0.0	0.00	1903
Google Chrome Helper (Renderer)	7.1	5.69	14	11	0.0	0.00	1922
Google Chrome Helper (Renderer)	0.0	18.26	17	0	0.0	0.00	1866
Google Chrome Helper (Renderer)	0.0	14.25	12	0	0.0	0.00	1856
MTLCompilerService	0.0	0.05	2	0	0.0	0.00	1900
MTLCompilerService	0.0	0.20	2	0	0.0	0.00	1899
VTDecoderXPCService	0.0	0.12	2	0	0.0	0.00	1848

چند رسمانی

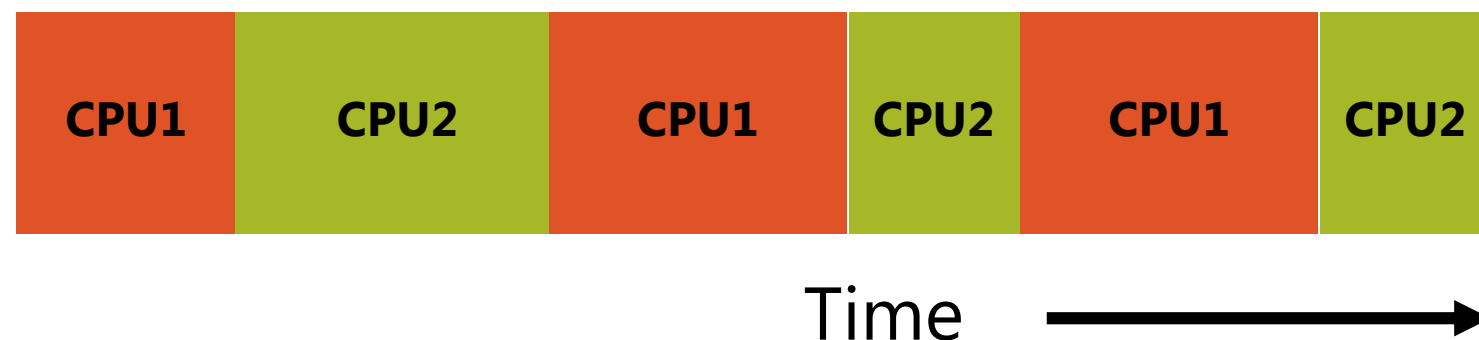
- مصرف حافظه کمتر
- فایرفاکس بهتر است چرا که با اشتراک برخی بخشها در حافظه بین برگهها مصرف کمتری دارد
- شروع به کار سریعتر
- فایرفاکس سریعتر عمل می کند چرا که فقط یک فرآیند را در حافظه لود می کند
- جابجایی بین برگهها
- فایرفاکس بهتر عمل می کند چرا که این کار به نوعی تعویض زمینه بین رسمانهاست
- استفاده از منابع مشترک (cache)
- در فایرفاکس بهتر و سریعتر است چرا که این heap بوده و بین رسمانها مشترک و بدون تشریفات خاصی قابل دسترسی است
- در بازیابی و نمایش صفحات تکراری چون از cache استفاده می شود فایرفاکس سریعتر عمل می کند
- امنیت
- در کروم بهتر است چرا که فضای آدرس برگهها از هم جداست. در فایرفاکس ممکن است برگهها بتوانند به حافظه هم دسترسی داشته باشند
- اطمینان اجرایی بالا
- در کروم بهتر است چرا که اگر یک برگه دچار اشکال شود در اجرای بقیه برگهها اشکالی پیش نخواهد آمد

پشتیبانی سخت افزاری از اجرای موازی

سیستم تک پردازنده‌ی تک‌هسته‌ای

```
1 int main(void){
2     thread_fork(while(1));
3     thread_fork(sprintf("Finished!"));
4 }
```

- در یک سیستم تک‌پردازنده ریسمان‌ها با روش همزمانی (concurrency) اجرا می‌شوند
- به هر ریسمان یک مقدار مشخص زمان اختصاص داده می‌شود و اگر در آن مدت تمام نشد باید منتظر بماند تا بار بعدی پردازنده به آن اختصاص یابد.



سیستم چند پردازنده‌ی تک‌هسته‌ای

▶ در یک سیستم چند پردازنده، چندین پردازنده روی یک برد قرار دارند.

▶ پردازنده‌ها از طریق یک گذرگاه (bus) اختصاصی به هم متصل هستند

▶ در مدل اجرایی متقارن بر روی تمامی پردازنده‌ها یک سیستم عامل نصب شده و همان یک سیستم عامل همه پردازنده‌ها را مدیریت می‌کند

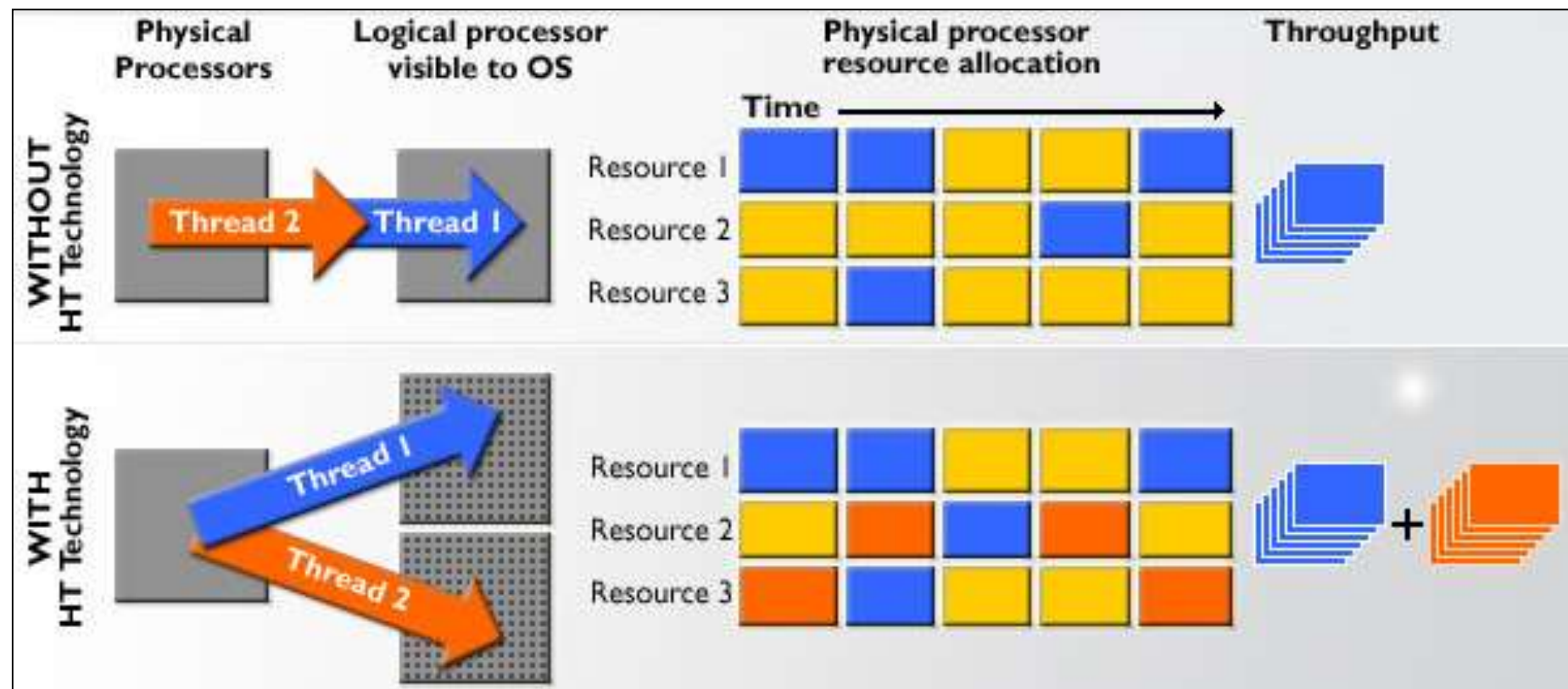
▶ در این حالت می‌توان ریسمانها را به شکل موازی (Parallel) اجرا کرد.

▶ یعنی هر پردازنده مجازی بر روی یک پردازنده فیزیکی در همان لحظه اجرا می‌شود.



پردازنده‌های چند ریسمانی

Wasted execution
Unit slots



- هر ریسمان برای اجرا بر روی یک پردازنده در کنار محتوای حافظه از پردازنده چه اجزایی را به شکل اختصاصی لازم دارد: یک مجموعه ثبات
- اگر در یک پردازنده به شکل سخت‌افزاری چندین مجموعه ثبات داشته باشیم می‌توانیم زمانی که یک ریسمان در حالت انتظار است سریعاً اجرا را از ریسمان دیگر ادامه دهیم
- در برخی منابع ریسمانهای سخت‌افزاری را گاهی هسته‌های منطقی نیز می‌نامند
- ریسمانها می‌توانند از یک فرآیند و یا چند فرآیند مختلف باشند

پردازنده‌های چند ریسمانی

■ مزایای ریسمانهای سخت‌افزاری:

■ تعویض زمینه بسیار سریعتر است

■ نیازی به ذخیره وضعیت ریسمان قبلی و لود ریسمان جدید نیست همه وضعیتها در سخت‌افزار آماده است

■ بهره‌وری (utilization) منابع افزایش می‌یابد، چون زمان سربار تعویض زمینه کاهش می‌یابد.

■ چندریسمانی سخت‌افزاری را می‌توان در تنظیمات BIOS غیرفعال کرد

پردازنده‌های چند ریزمانی

■ روشهای زمانی‌بندی ریزمانهای سخت‌افزاری

■ روش چند ریزمانی دانه درشت (Coarse-grained)

- یک ریزمان اجرا می‌شود تا به حالت انتظار رفته و یا به دستوری که اجرای آن خیلی طول میکشد برسد. در این حالت ریزمان بعدی اجرا می‌شود.

■ روش چند ریزمانی دانه ریز (fine-grained)

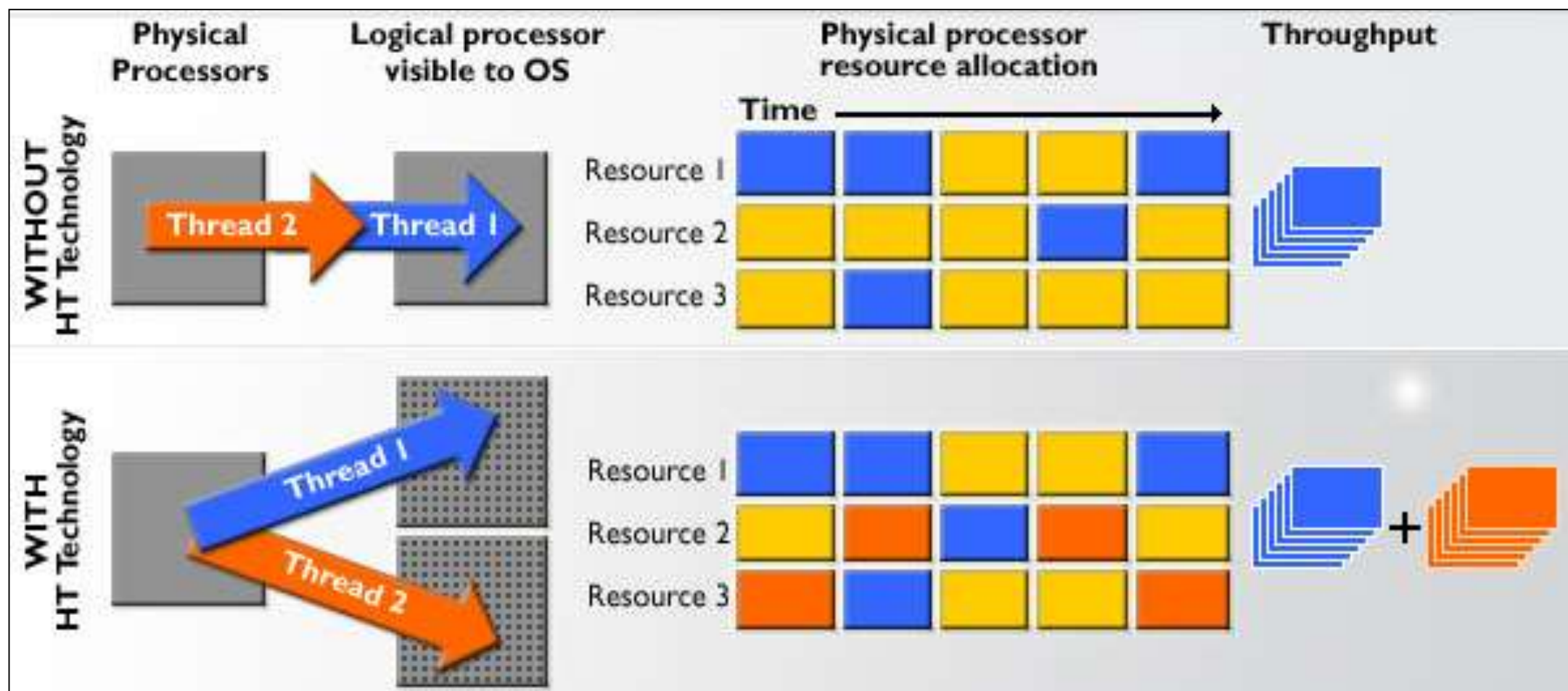
- در این روش از هر ریزمان یک مدت زمان مشخص اجرا می‌شود که بسیار به روش همزمانی در سیستم تک‌پردازنده شبیه است

پردازنده‌های چند ریسمانی

روشهای زمانی بندی ریسمانهای سخت‌افزاری

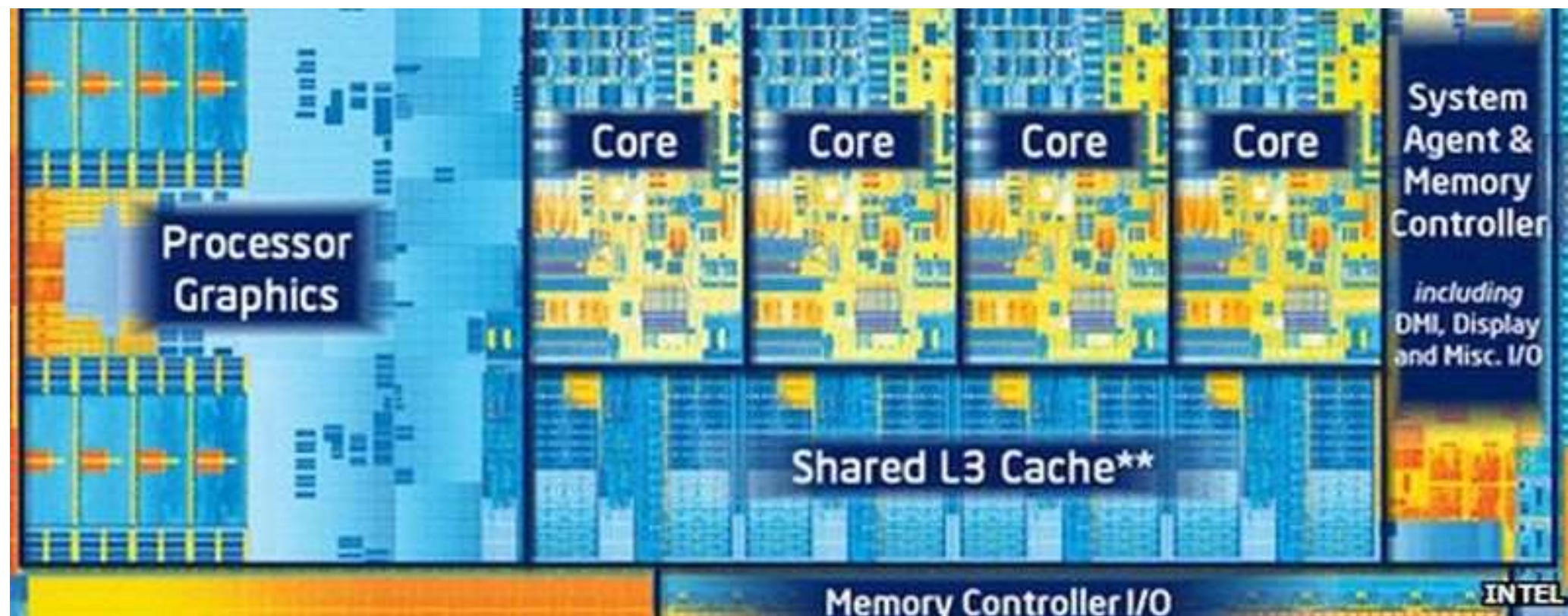
روش چند ریسمانی همزمان (Simultaneous MultiThreading-SMT)

- در این روش در هر سیکل از چندین ریسمان دستورات اجرا می‌شوند
- برای هر ریسمان پنجره اجرا در نظر گرفته می‌شود و بسته به منابع محاسباتی هر میزان که بتوان از ریسمانهای مختلف دستور اجرا می‌شود
- اینترنت این تکنولوژی را Hyper Threading می‌نامد



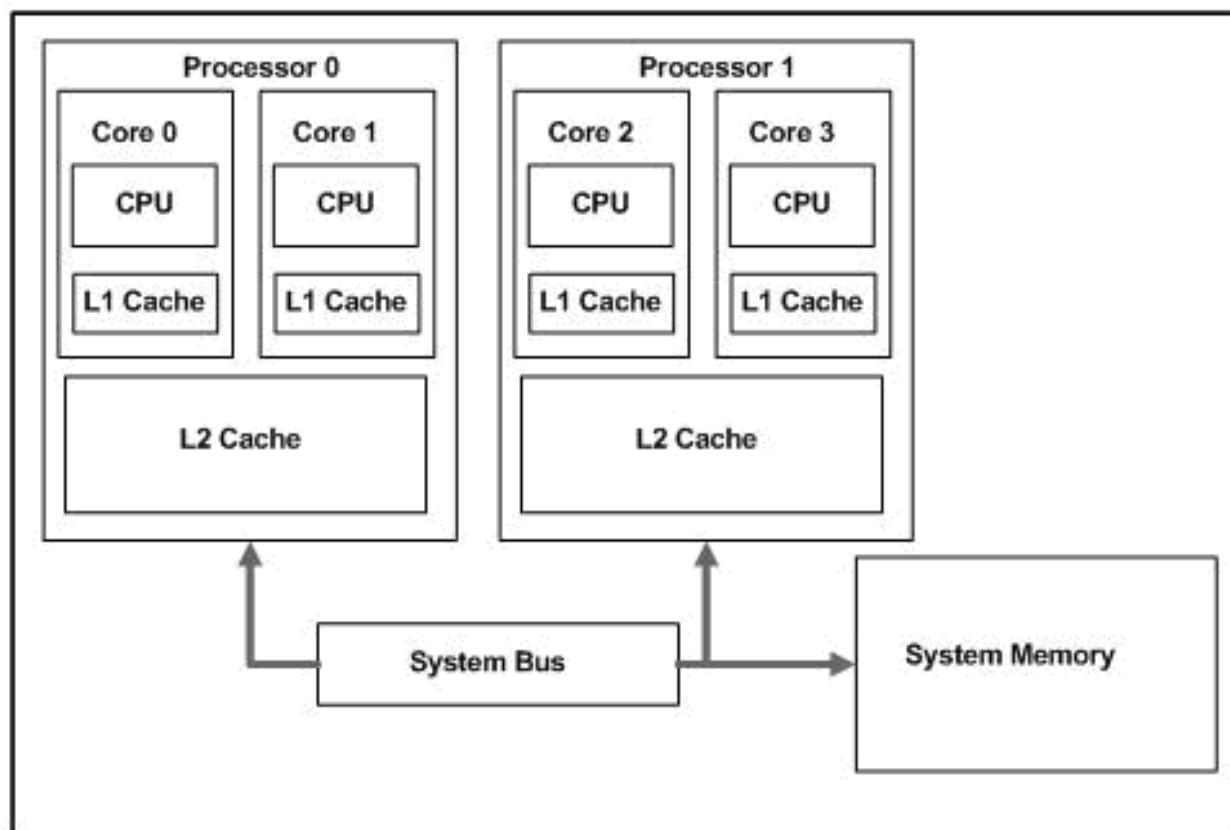
پردازنده‌های چند هسته‌ای (multicore)

- در پردازنده‌های چند هسته بطور واقعی و فیزیکی چندین هسته پردازنده در کنار هم قرار دارند.
- عموماً این هسته‌ها در cache لایه سه (L3) با هم ارتباط دارند.
- در برخی معماریها L2 هم مختص هر پردازنده است و در برخی بین دو هسته مشترک است
- هر هسته می‌تواند خودش چند ریسمانی باشد



پردازنده‌های چند هسته‌ای (multicore)

- تفاوت با سیستم چند پردازنده
 - هسته‌ها کنار هم و در یک تراشه هستند و هزینه و قیمت کاهش می‌یابد
 - بدلیل نزدیکی گذرگاه ارتباطی بین هسته‌ها سریعتر است
 - در صورتی که داده‌های مشترک بین هسته‌ها استفاده شود به دلیل سرعت بالای حافظه cache این اشتراک بسیار سریع است. به خصوص در داده‌های با حجم بالا این مکانیزم بسیار موثر است



توابع ریسمانها

thread_fork(func_name, args)

این تابع یک رسمان ایجاد می‌کند و این رسمان تابع func(args) را اجرا می‌کند
با این تابع رسمان به وضعیت New میرود و پس از مدتی به وضعیت ready خواهد رفت

thread_yield()

رسمان با این تابع پردازنده را به صورت داوطلبانه و قبل از گرفته شدن به وسیله وقفه تایمر رها می‌کند.
با این تابع رسمان به وضعیت ready میرود

thread_join(thread_handle)

یک رسمان منتظر اتمام یک رسمان دیگر می‌شود. رسمانی که فراخوانی انجام داده تا تمام شدن رسمان دیگر در این تابع منتظر می‌ماند
با این تابع رسمان به وضعیت waiting می‌رود و پس از اتمام رسمان دیگر رسمان به حالت ready برمیگردد

thread_exit()

برای اعلام اتمام کار یک رسمان به سیستم عامل
با این تابع رسمان به وضعیت terminated می‌رود.

- در لینوکس کتابخانه‌های متعددی برای برنامه‌نویسی با رسمانهها وجود دارد
- کتابخانه pthread یکی از معروفترین‌هاست
- کتابخانه رسمان زبان جاوا یکی دیگر از کتابخانه‌هاست که فقط با زبان جاوا قابل استفاده است

```

1  #include <pthread.h>
2  #include <stdio.h>
3  /* this data is shared by the thread(s) */
4  int sum;
5
6  /* threads call this function */
7  void *runner(void *param);
8
9  int main(int argc, char *argv[]){
10     pthread_t tid;
11     /* set of thread attributes */
12     pthread_attr_t attr;
13     /* get the default attributes */
14     pthread_attr_init(&attr);
15     /* create the thread */
16     pthread_create(&tid, &attr, runner, argv[1]);
17     /* wait for the thread to exit */
18     pthread_join(tid, NULL);
19     printf("sum = %d\n", sum);
20 }

```

```

21
22  /* The thread will begin control in
23  void *runner(void *param){
24      int i, upper = atoi(param);
25      sum = 0;
26      for (i = 1; i <= upper; i++)
27          sum += i;
28      pthread_exit(0);
29  }
30

```

- مثال از زبان جاوا
- جمع چند رئیس‌مانی اعضای یک آرایه

```
1 public class ParallelSum {
2     static final int THREAD_COUNT = 5;
3
4     static int[] Data = new int[5000];
5     static int[] Sums = new int[THREAD_COUNT];
6
7     static int threadShare = Data.length / THREAD_COUNT;
8
9     public static void main(String[] args) {
10         Random random = new Random(new Date().getTime());
11         Arrays.setAll(Data, i -> random.nextInt());
12
13         try {
14             Thread[] threads = new Thread[THREAD_COUNT];
15
16             for (int i = 0; i < THREAD_COUNT; i++) {
17                 threads[i] = new Thread(new PartialSum(i, (i==THREAD_COUNT-1)));
18                 threads[i].start();
19             }
20
21             for (int i = 0; i < THREAD_COUNT; i++)
22                 threads[i].join();
23
24             int totalSum = Arrays.stream(Sums).sum();
25
26             System.out.println("totalSum = " + totalSum);
27         } catch (Exception ex) {
28             ex.printStackTrace();
29         }
30     }
```

```

32     public static class PartialSum implements Runnable {
33         private final int index;
34         private final boolean lastThread;
35
36         PartialSum(int index, boolean lastThread) {
37             this.index = index;
38             this.lastThread = lastThread;
39         }
40
41         public void run() {
42             int startIndex = index * threadShare;
43             int stopIndex = (lastThread) ? Data.length : (index + 1) * threadShare;
44             Sums[index] = Arrays.stream(Data, startIndex, stopIndex).sum();
45
46             System.out.println("thread " + index + " finished");
47         }
48     }
49 }
50 |

```


استخر رءسمان (Thread Pool)

- يك سرور وب را در نظر بگيريد
- براي پاسخ به هر درخواست يك رءسمان اختصاص داده مي شود
- تعداد کاربران همزمان اين سرور عموما بسيار زياد است
- مشكلات رهيافت اتخاذ شده
- ايجاد رءسمان اگر چه نسبت به ايجاد فرآيند سبك است اما بهر حال در تعداد زياد هزينه بر است
- هر رءسمان بهر حال بخشي از حافظه را اشغال مي كند
- بدليل تعداد بالاي رءسمانهاي آماده به كار زمان بعدي اختصاص پردازنده به رءسمانها افزايش مي يابد. در نتيجه زمان پاسخ دهی سايت (تاخير پاسخ به فعاليت کاربر) افزايش مي يابد
- در استخر رءسمان يك تعداد مشخصي رءسمان آماده به كار ايجاد مي شود
- وظائف به ترتيب در يك صف گذاشته مي شوند
- رءسمانها كارها را يكي يكي از سر صف برداشته و پس از اتمام يك كار، كار بعدي را از سر صف برميدارند
- مثال استخر فرآيندها در جاوا


```

1 public class ParallelSumThreadPool {
2     private static final ExecutorService executorService =
Executors.newFixedThreadPool(THREAD_COUNT);
3
4     public static void main(String[] args) {
5         Random random = new Random(new Date().getTime());
6         Arrays.setAll(Data, i -> random.nextInt());
7
8         try {
9             Future[] futures = new Future[THREAD_COUNT];
10            for (int i = 0; i < futures.length; i++)
11                futures[i] = executorService.submit(
12                    new ParallelSum.PartialSum(i, (i == THREAD_COUNT - 1)));
13
14            for (Future future : futures) future.get();
15
16            int totalSum = Arrays.stream(Sums).sum();
17            System.out.println("totalSum = " + totalSum);
18        } catch (Exception e) {
19            e.printStackTrace();
20        } finally {
21            executorService.shutdownNow();
22        }
23    }
24
25    @Override
26    protected void finalize() throws Throwable {
27        super.finalize();
28        executorService.shutdownNow();
29    }
30 }

```

اجرای رسمانهها

- شبه کد مقابل کد توزیع (dispatching) سیستم عامل را نشان می‌دهد.
- شبه کد مهمترین کار سیستم عامل است

```

1 while(true){
2   new_tcb = select_next_tcb();
3   save_cpu_state(cur_tcb);
4   load_cpu_state(new_tcb);
5   cur_tcb = new_tcb;
6   run_thread();
7   thread_house_keeping();
8 }

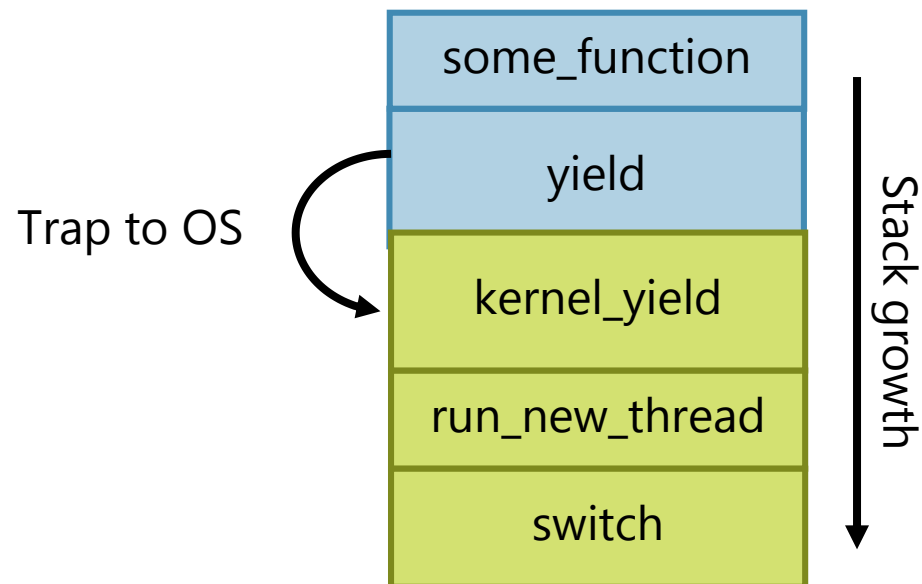
```

- این حلقه چه زمانی تمام می‌شود؟
- اگر اتفاق بدی بیفتد و سیستم عامل crash کند (اصطلاحاً panic گفته می‌شود)
- کاربر دستور خاموش (halt) بدهد (که در این شبه کد نیست!)
- اگر پردازنده دست یک ریسمان باشد چه زمانی تابع run_thread تمام می‌شود؟
- رویدادهای داخلی
- رویدادهای خارجی

اجرای رسمانها

- اگر پردازنده دست یک رسمان باشد چه زمانی تابع `run_thread` تمام می شود؟
- رویدادهای داخلی
- درخواست IO که بطور ضمنی پردازنده از دست رسمان گرفته شده و به صف `waiting` برده می شود
- انتظار برای دریافت سیگنال (بعداً بحث خواهد شد)
- فراخوانی تابع `yield` که رسمان داوطلبانه پردازنده را واگذار می کند
- رویدادهای خارجی
 - وقفه های سخت افزاری و یا نرم افزاری
 - وقفه تایمر

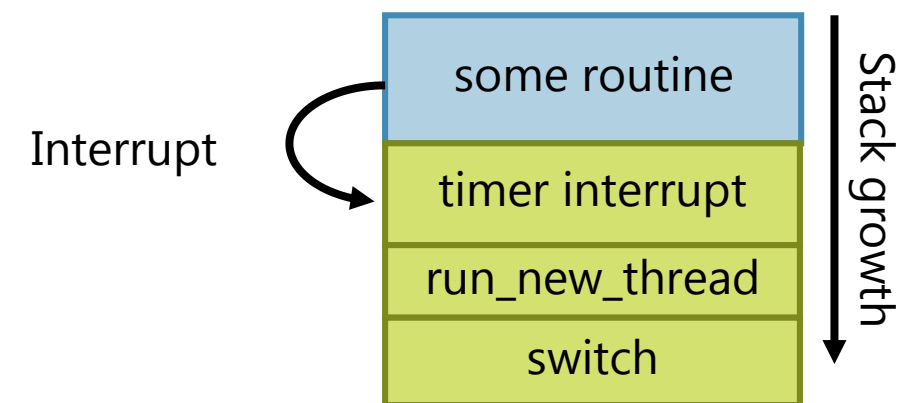
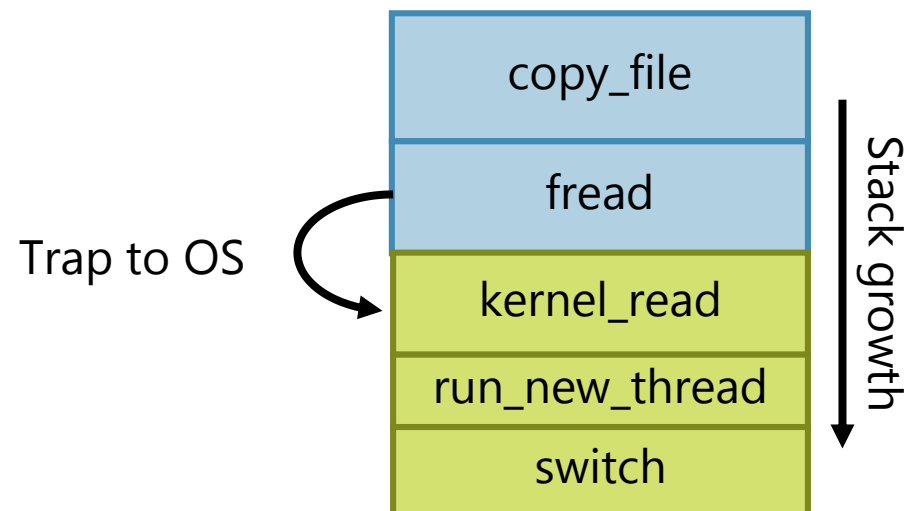
- در واقع در داخل فراخوانی تابع yield تابع مربوط به تعویض زمینه فراخوانی می‌شود
- عملکرد این تابع جالب است چرا که از یک رئسمان فراخوانی می‌شود ولی در موقع return از یک رئسمان دیگر سر در می‌آورد
- در تابع switch عملاً مقدار ثبات PC عوض شده و به بخش کد رئسمان جدید اشاره می‌کند
- تابع thread_house_keeping چه زمانی اجرا می‌شود؟
- در سیستم عامل‌های قدیمی این تنها راه پیاده‌سازی برنامه‌های چند رئسمانی بوده است



```
run_new_thread() {
    newThread = select_next_thread();
    switch(curThread, newThread);
    thread_house_keeping(); /* do any cleanup */
}
```

تعویض زمینه با IO و وقفه

- در اجرای IO هم در تابع kernel_read روال خواندن از مثلا دیسک آغاز شده و در انتها تابع run_new_thread فراخوانی می شود.
- در وقفه ها نیز اجرای رءسمان فعلی متوقف و روتیم وقفه اجرا می شود که در مورد وقفه تایمر همان تابع run_new_thread اجرا خواهد شد.



- آیا کرنل هم چند ریسمانی است
- شاید بیش از ۱۰ سال پیش کرنل فقط به صورت یک ریسمانی قابل اجرا بود
- از زمانی که تعداد هسته‌ها افزایش یافت این رویه تغییر پیدا کرد
- کرنل ممکن است برای موارد زیر یک ریسمان جدا اجرا کند
 - برای هر فرآیند کاربری
 - برای گامهای مختلف اجرای یک IO
 - برای ارتباط با تجهیزات از طریق device driver مربوط

- دو نوع پیاده‌سازی ریسمان وجود دارد
 - حالت هسته (kernel-mode)
 - حالت کاربری (user-mode)
- ریسمانهای حالت هسته
 - این ریسمانها توسط هسته سیستم عامل به شکل بومی پشتیبانی می‌شوند
 - بازاء هر ریسمان حالت هسته یک TCB ایجاد می‌شود
 - در واقع سیستم عامل از تمامی ریسمانهای حالت هسته اطلاع داشته و تک به تک توسط سیستم عامل زمان‌بندی می‌شوند
 - هر ریسمان به طور مستقل اجرا بخشی از کد را اجرا کرده و مستقل از هم می‌توانند به حالت انتظار بروند.
- مشکل اصلی در ریسمانهای حالت هسته هزینه بالاست
 - این هزینه همچنان کمتر از مدیریت فرآیندهاست
 - برای زمان‌بندی و نیز اجرای توابع مربوط به ریسمانها نیاز به فراخوانی سیستمی وجود دارد

■ ریسمانهای حالت کاربری (user-mode)

■ ریسمانها در حالت کاربری ایجاد می‌شوند و تمامی توابع مربوط به ریسمانها تمام در حالت کاربری اجرا می‌شود

■ زمان‌بندی بین ریسمانها و توزیع زمان اجرا بین آنها توسط کتابخانه‌های سطح کاربری انجام می‌گیرد

■ در اجرای توابع ریسمانها نیازی به فراخوانی سیستمی نیست و سرعت اجرا باز هم بالاتر می‌رود

■ مشکلات

■ هسته سیستم عامل از وجود چندین ریسمان اصلاً مطلع نیست و در نتیجه برای یک ریسمان زمان‌بندی انجام می‌دهد.

■ زمانی که یک ریسمان به حالت انتظار برود تمامی ریسمانها در حالت انتظار قرار می‌گیرند و سایر ریسمانها قابل اجرا نیستند

■ چرا که سیستم عامل از وجود چند ریسمان مطلع نیست و زمانی که یک ریسمان فراخوانی سیستمی انجام می‌دهد در هسته سیستم عامل است و همانجا به حالت انتظار رفته و پردازنده از آن گرفته می‌شود.

■ بنابراین کتابخانه سطح کاربری که زمان‌بندی را انجام می‌دهد نمی‌تواند ریسمان دیگری را جایگزین کند.

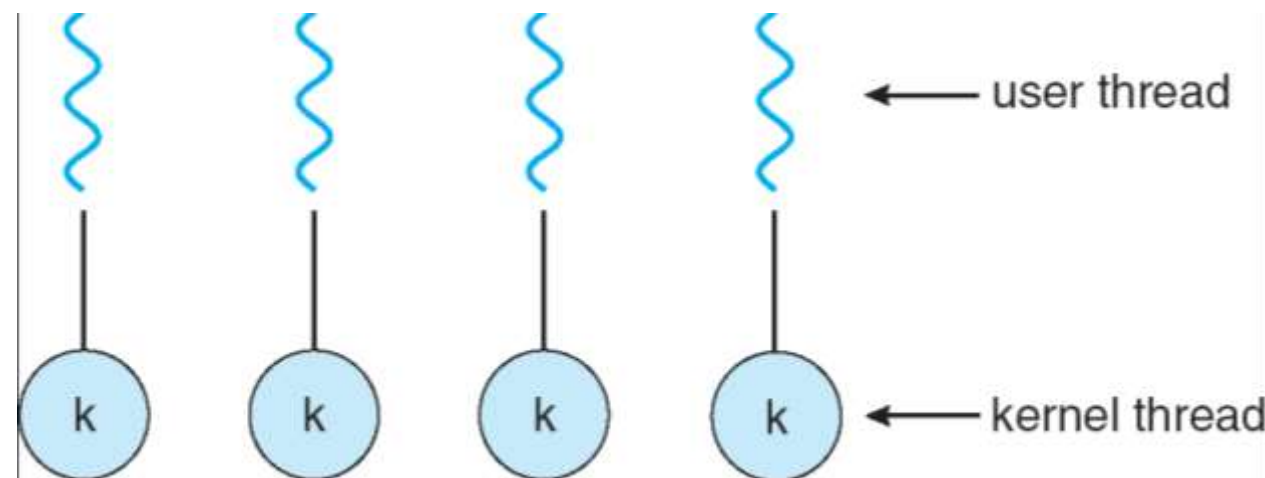
■ ریسمانهای فعال سازی زمان بندی (scheduler activations)

■ برای رفع مشکل بلاک شدن کل ریسمانهای حالت کاربری در اثر بلاک شدن یک ریسمان، روش فعال سازی زمان بندی مطرح شده است
■ این روش در برخی سیستم عاملها نظیر ویندوز پیاده سازی شده است

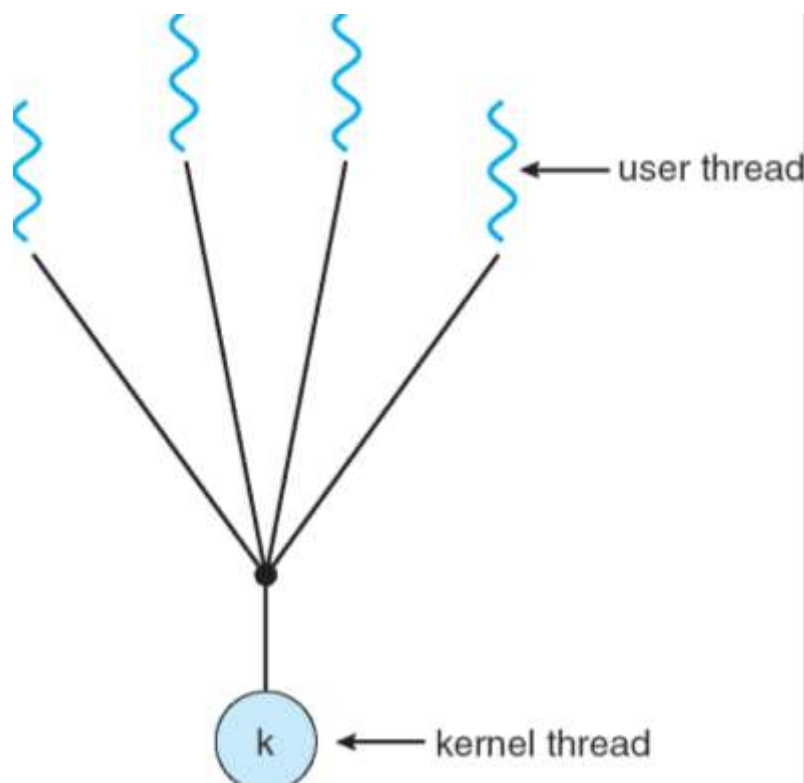
■ زمانی که یک ریسمان به حالت انتظار وارد شد هسته سیستم عامل به کتابخانه سطح کاربری اطلاع می دهد
■ به نوعی یک فراخوانی برعکس فراخوانی سیستمی اتفاق می افتد که با اصطلاح upcall معرفی شده است

■ در این حالت کتابخانه زمان بندی سطح کاربری می تواند یک ریسمان سطح کاربری دیگر را قبل از اتمام برش زمانی بر روی پردازنده اجرا کند

■ در پیاده‌سازی ریسمان‌های سطح هسته در واقع یک نگاشت یک‌به‌یک بین ریسمانی که در سطح کاربری ایجاد شده و ریسمانی که در سطح هسته شناخته می‌شود وجود دارد



■ در پیاده‌سازی صرفاً کاربری نیز یک نگاشت یک به چند (one-to-many) وجود دارد



- در برخی سیستم عامل‌ها نگاشت چند به چند پیاده‌سازی شده است بدین نحو که چند ریسمان سطح کاربری به چند ریسمان سطح هسته نگاشته می‌شوند.
- بنابراین سیستم عامل به جای شناختن یک ریسمان در سطح هسته حالا چندین ریسمان می‌شناسد و بین آنها زمان‌بندی می‌کند

