操作系统

Operating Systems

刘川意　教授

哈尔滨工业大学（深圳）

2021年9月

Module 2: 线程（Thread）

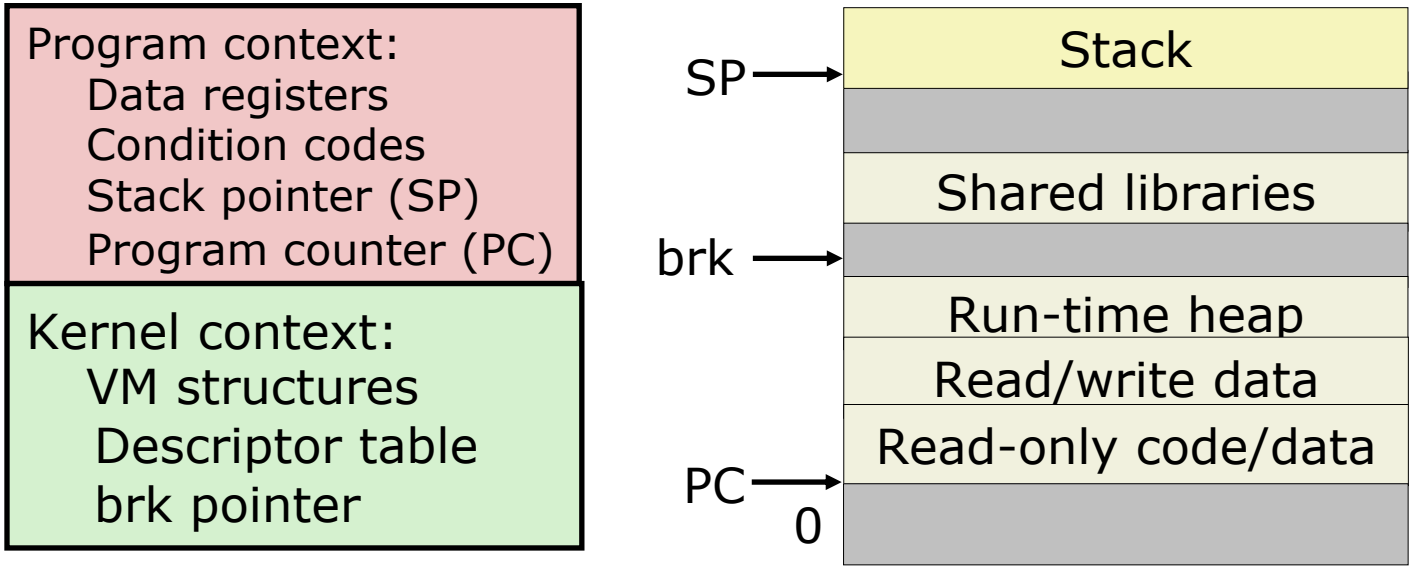
1. 线程的引入与介绍
2. Posix 线程

Traditional View of a Process

- Process = process context(进程上下文) + code, data, and stack

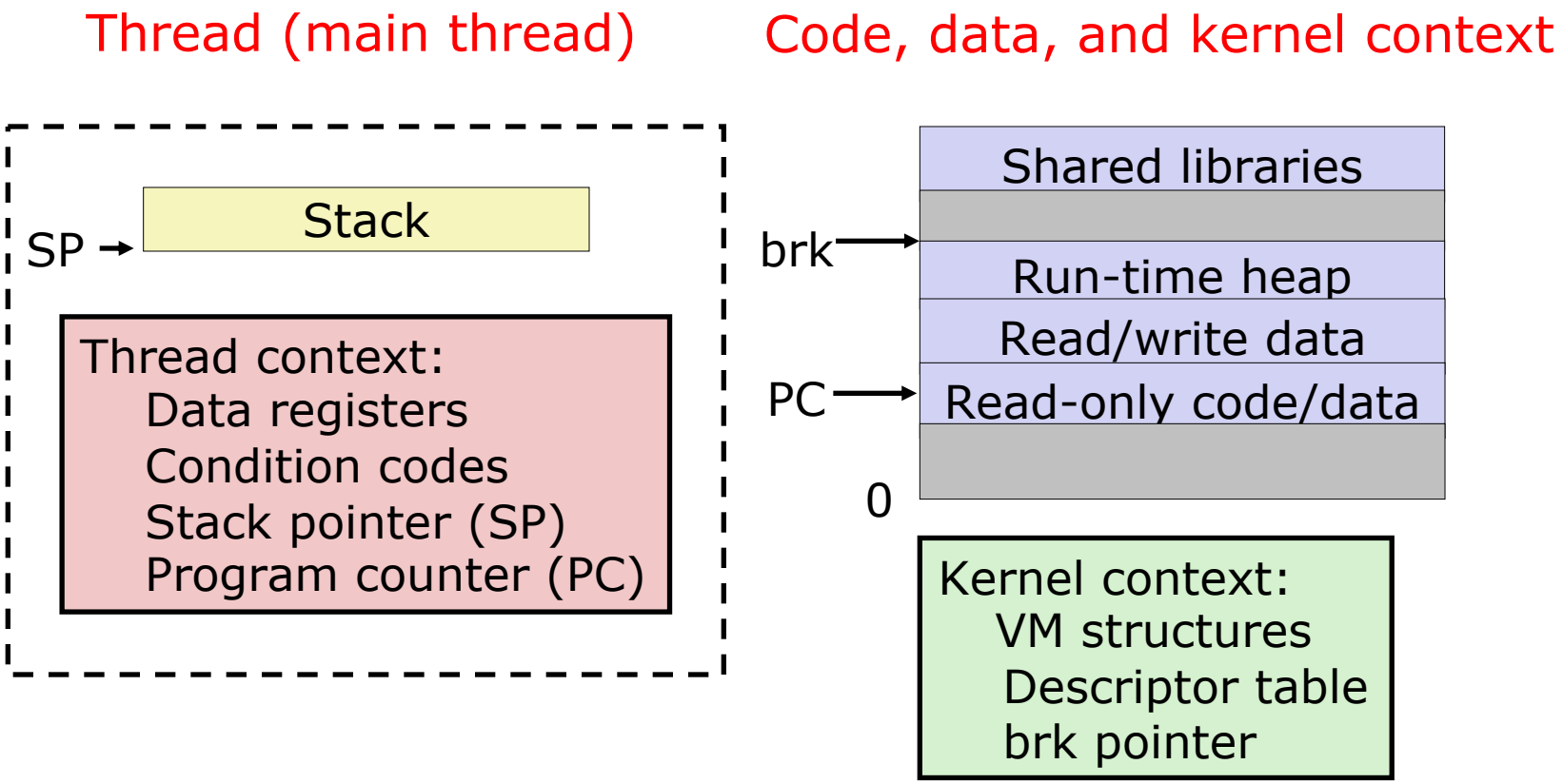
Process context

Code, data, and stack

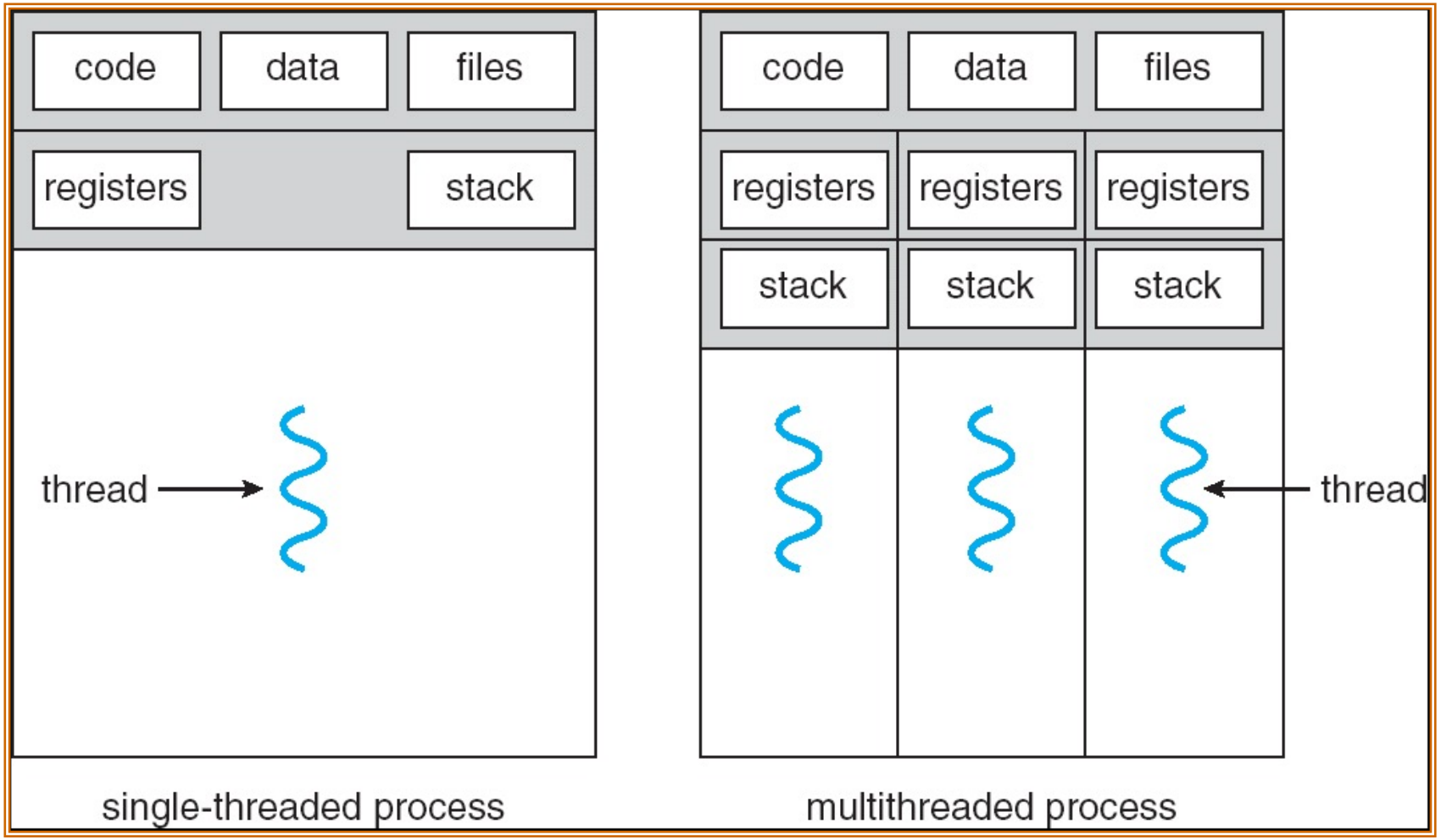


Alternate View of a Process

- Process = thread（线程）+ code, data, and kernel context



Single vs. Multithreaded Processes



（单线程进程）　　（多线程进程）

Modern Processes: Multiple Threads(多线程进程)

- Multiple threads can be associated with a process
  - Each thread has its own logical control flow
  - **Shares address space**（共享所属进程的地址空间）with other threads belonging to the same process
  - Each thread has its own stack for local variables（局部变量）
    - but not protected from other threads
  - Each thread has its own thread id (TID)
- Why separate concepts of threads and processes?
  - Threads: Concurrency（并发）
  - Processes: Protection

A Process With Multiple Threads（多线程进程）

Thread 1 (main thread主线程)

stack 1

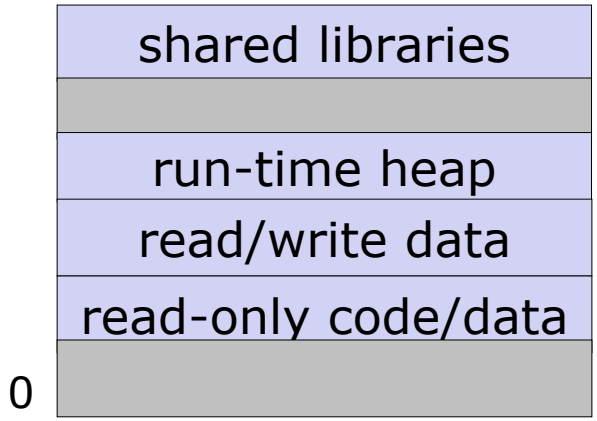
Thread 1 context:
Data registers
Condition codes
SP1
PC1

Thread 2 (peer thread对等线程)

stack 2

Thread 2 context:
Data registers
Condition codes
SP2
PC2

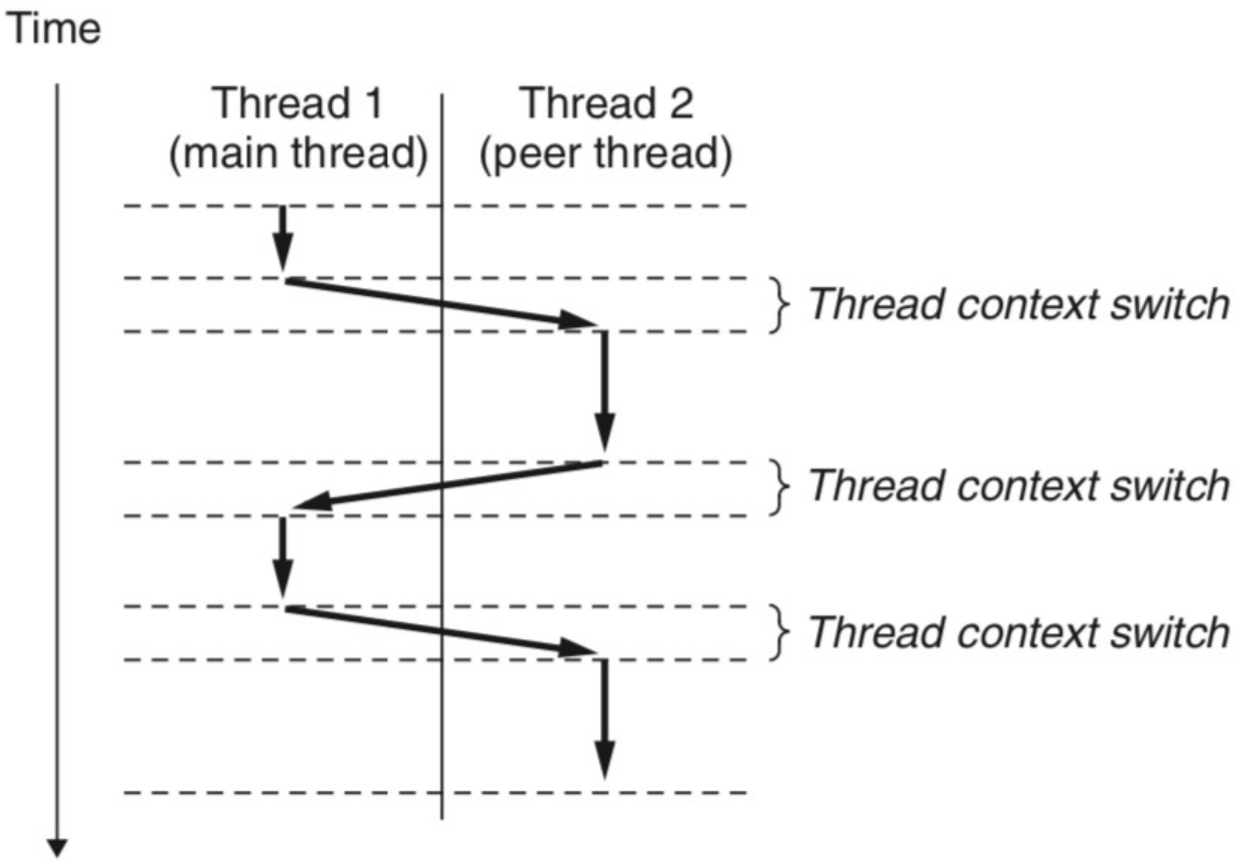
Shared code and data



Kernel context:
VM structures
Descriptor table
brk pointer

Main thread(主线程)

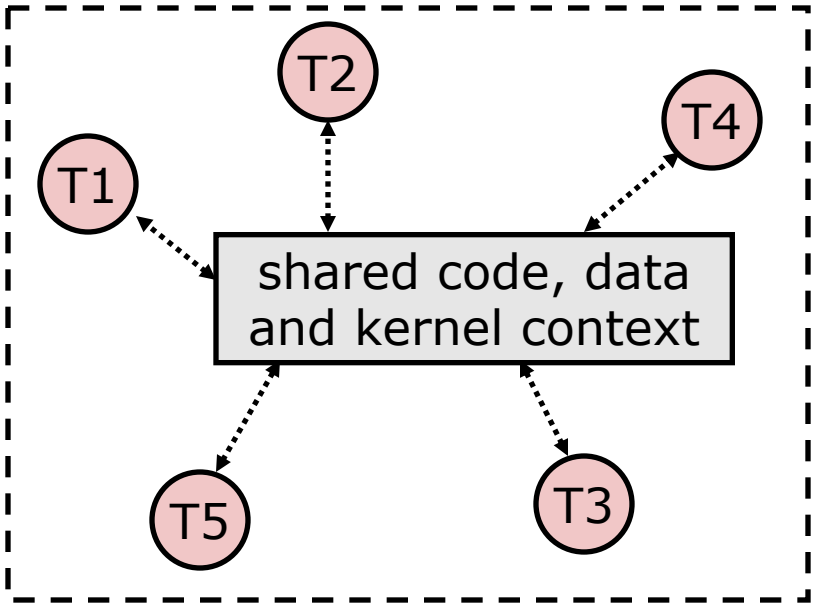
- Each process begins life as a single thread called the *main thread*(*主线程*).
- At some point, the main thread creates a *peer thread*(*对等线程*), and from this point in time the two threads run concurrently(并发).
- 并发线程执行



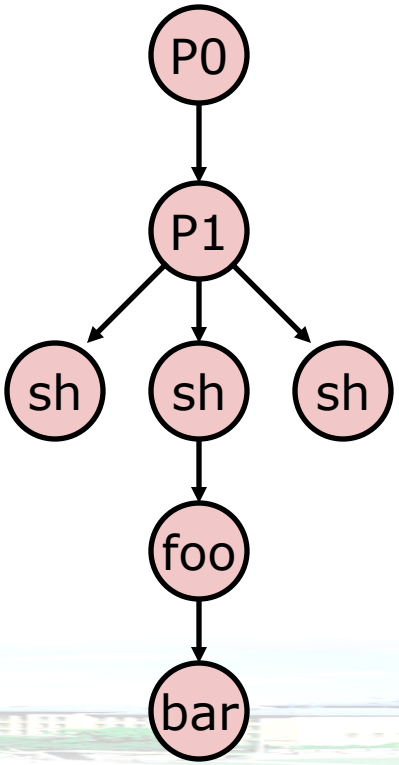
pool of peers（线程池）

- Threads associated with process form a pool of peers（线程池）
  - Unlike processes which form a tree hierarchy

Threads associated with process



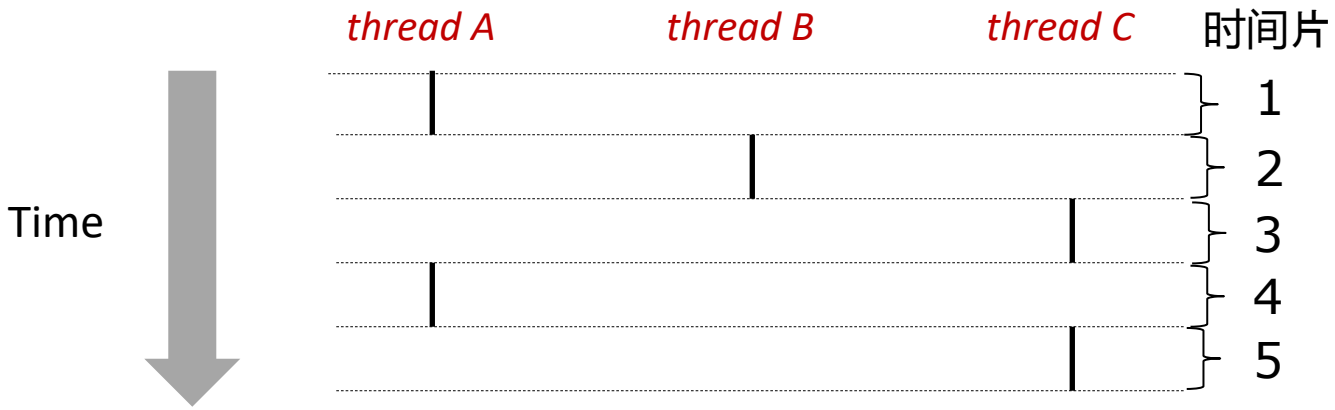
Process hierarchy



Concurrent Threads（并发线程）

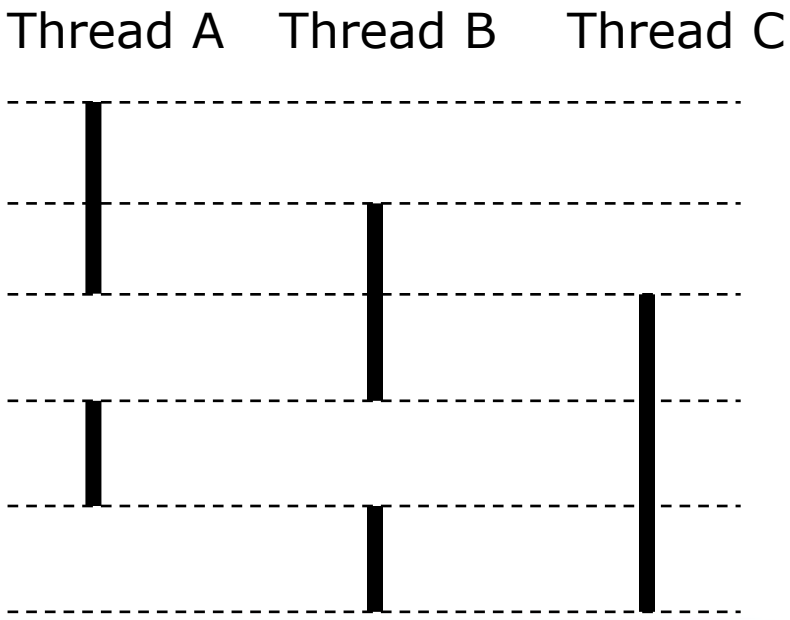
- Two threads are *concurrent* if their flows overlap in time
- Otherwise, they are sequential

- Examples:
  - Concurrent: A & B, A&C
  - Sequential: B & C



Concurrent Thread Execution（并发线程的执行）

- Single Core Processor（单核处理器）
  - Simulate parallelism by time slicing（通过时间片模拟并行）
- Multi-Core Processor（多核处理器）
  - Can have true parallelism（能够实现真正的并行）



Run 3 threads on 2 cores

Threads（线程）vs. Processes（进程）

- How threads and processes are similar
  - Each has its own logical control flow
  - Each can run concurrently with others (possibly on different cores)
  - Each is context switched（上下文切换）
- How threads and processes are different
  - Threads share all code and data (except local stacks)
    - Processes (typically) do not
  - Threads are somewhat less expensive than processes
    - Process control (creating and reaping) twice as expensive as thread control
    - Linux numbers:
      - ~20K cycles to create and reap a process
      - ~10K cycles (or less) to create and reap a thread

Threads（线程）vs. Processes（进程）

【例题】　在下面的叙述中正确的是()。

A. 线程是比进程更小的能独立运行的基本单位

B. 引入线程可提高程序并发执行的程度，可进一步提高系统效率

C. 系统级线程和用户级线程的切换都需要内核的支持

D. 一个进程一定包含多个线程

答案：B

解析：A，线程不能独立运行，线程需要进程所获得的资源

C，在用户级线程中，有关线程管理的所有工作都由应用程序完成，无需内核的干预，内核意识不到线程的存在。

D，一个进程至少包含一个主线程（线程数量大于等于1）

Module 2: 线程（Thread）

1. 线程的引入与介绍
2. Posix 线程

Posix Threads Interface（Pthreads 接口）

- *Pthreads:* Standard interface for ~60 functions that manipulate threads from C programs（C程序中处理进程的标准接口）
  - Creating threads（创建线程）
    - `pthread_create()`
  - Terminating threads（终止线程）
    - `pthread_cancel()`
    - `pthread_exit()`
    - `exit()` [terminates all threads] , `RET` [terminates current thread]
  - Joining and Detaching threads（回收和分离线程）
    - `pthread_join()`
    - `pthread_detach()`

Creating Threads(创建线程)

- `int pthread_create(pthread_t *tid, pthread_attr_t *attr, func *f, void *arg)`
  - `tid` : TID for the new thread returned by the subroutine.
  - `attr`(属性): set thread attributes. You can specify a thread attributes object, or NULL for the default values.
  - `f` : the C routine that the thread will execute once it is created.
  - `arg`(参数): A single argumet that may be passed to *start_routine*. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

- **Thread Attributes(线程属性):**
  - By default, a thread is created with certain attributes
  - Some of these attributes can be changed by the programmer via the thread attribute object.
  - `pthread_attr_init` and `pthread_attr_destroy` are used to initialize/destroy the thread attribute object.

Example 1：Creating Threads(创建线程)

- 当一个程序创建了多个线程以后，其在多核机和单核机上面运行会有什么差别呢？多线程的执行速度一定比顺序执行快吗？
- 顺序执行多个task VS 并行执行多个task
- 单核 VS 多核

```
1. /* A task that takes some time to complete. The ID identifies
2.  distinct tasks for printed messages. */
3.
4. void *task(void *ID) {
5.   long id = (long)ID;
6.   printf("Task %d started\n", id);
7.   int i;
8.   double result = 0.0;
9.   for (i = 0; i < 10000000; i++) {
10.    result = result + sin(i) * tan(i);
11.  }
12.  printf("Task %d completed with result %e\n", id, result);
13.}
```

pthread_1.c

Example 1：Creating Threads(创建线程)

```
1. void *print_usage(int argc, char *argv[]) {
2.   printf("Usage: %s serial|parallel num_tasks\n", argv[0]);
3.   exit(1);
4. }
5.
6. int main(int argc, char *argv[]) {
7.   if (argc != 3) {print_usage(argc, argv);}
8.   int num_tasks = atoi(argv[2]);
9.
10.  if (!strcmp(argv[1], "serial")) {
11.    serial(num_tasks);
12.  } else if (!strcmp(argv[1], "parallel")) {
13.    parallel(num_tasks);
14.  }
15.  else {
16.    print_usage(argc, argv);
17.  }
18.
19.  printf("Main completed\n");
20.  pthread_exit(NULL);
21.}
```

pthread_1.c

Example 1：Creating Threads(创建线程)

```
root@ubuntu:/home/liu/Desktop/OS# ./pthreads_1
Usage: ./pthreads_1 serial|parallel num_tasks
root@ubuntu:/home/liu/Desktop/OS# ./pthreads_1 serial 2
Task 0 started
Task 0 completed with result 3.135632e+06
Task 1 started
Task 1 completed with result 3.135632e+06
Main completed
root@ubuntu:/home/liu/Desktop/OS# ./pthreads_1 parallel 2
Creating thread 0
Creating thread 1
Main completed
Task 1 started
Task 0 started
Task 1 completed with result 3.135632e+06
Task 0 completed with result 3.135632e+06
```

Example 1：Creating Threads(创建线程)

```
/* Parallel(并行):Run 'task' num_tasks times, creating a separate thread
for each call to 'task'. */
1.void *parallel(int num_tasks)
2.{
3.   int num_threads = num_tasks;
4.   pthread_t thread[num_threads
5.   int rc;
6.   long t;
7.   for (t = 0; t < num_threads; t++) {
8.     printf("Creating thread %ld\n", t);
9.     rc = pthread_create(&thread[t], NULL, task, (void *)t);
10.     if (rc) {
11.       printf("ERROR: return code from pthread_create() is %d\n", rc);
12.       exit(-1);
```

Thread ID

Thread attributes (usually NULL)

*Thread arguments (void *p)*

Thread routine

pthread_1.c

```
1. void *serial(int num_tasks) {
2.    long i;
3.    for (i = 0; i < num_tasks; i++)
4.    {
5.      task((void *)i);
6.    }
7. } /*Serial（顺序）: Run ‘task’ num_tasks times serially.*/
```

pthread_1.c

Example 1：Creating Threads(创建线程)

VMware中设置处理器为单核：

Example 1：Creating Threads(创建线程)

VMware中单核处理器：

顺序执行：

```
root@ubuntu:/home/liu/Desktop/OS# time ./pthreads_1 serial 4
Task 0 started
Task 0 completed with result 3.135632e+06
Task 1 started
Task 1 completed with result 3.135632e+06
Task 2 started
Task 2 completed with result 3.135632e+06
Task 3 started
Task 3 completed with result 3.135632e+06
Main completed

real    0m2.857s
user    0m2.813s
sys     0m0.004s
```

Example 1：Creating Threads(创建线程)

VMware中单核处理器：

并行执行：

```
root@ubuntu:/home/liu/Desktop/OS# time ./pthreads_1 parallel 4
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Main completed
Task 3 started
Task 2 started
Task 1 started
Task 0 started
Task 3 completed with result 3.135632e+06
Task 1 completed with result 3.135632e+06
Task 2 completed with result 3.135632e+06
Task 0 completed with result 3.135632e+06

real    0m2.942s
user    0m2.832s
sys     0m0.004s
```

Example 1：Creating Threads(创建线程)

VMware中设置为4核处理器：

Example 1：Creating Threads(创建线程)

VMware中设置4核处理器：

顺序执行：

```
root@ubuntu:/home/liu/Desktop/OS# time ./pthreads_1 serial 4
Task 0 started
Task 0 completed with result 3.135632e+06
Task 1 started
Task 1 completed with result 3.135632e+06
Task 2 started
Task 2 completed with result 3.135632e+06
Task 3 started
Task 3 completed with result 3.135632e+06
Main completed

real    0m3.001s
user    0m2.971s
sys     0m0.000s
```

Example 1：Creating Threads(创建线程)

VMware中设置4核处理器：

并行执行：

```
root@ubuntu:/home/liu/Desktop/OS# time ./pthreads_1 parallel 4
Creating thread 0
Creating thread 1
Task 0 started
Creating thread 2
Task 1 started
Creating thread 3
Main completed
Task 2 started
Task 3 started
Task 2 completed with result 3.135632e+06
Task 3 completed with result 3.135632e+06
Task 0 completed with result 3.135632e+06
Task 1 completed with result 3.135632e+06

real    0m0.802s
user    0m3.148s
sys     0m0.004s
```

Passing Arguments to Threads（向线程传递参数）

- pthread_create()允许程序员将一个参数传递给线程并启动例程。对于必须传递多个参数的情况，通过创建一个包含所有参数的结构体，然后在pthread_create()中传递一个指向该结构体的指针，可以轻松克服此限制

- 所有参数必须通过引用传递并强制转换为(void *)

- 考虑到不确定的启动和调度，如何将数据安全地传递给新创建的线程？

```
1. void *PrintHello(void *threadid)
2. {
3.     long taskid;
4.     sleep(1);
5.     taskid = *(long *)threadid;
6.     printf("Hello from thread %ld\n", taskid);
7.     pthread_exit(NULL);
8. }
```

Passing Arguments to Threads（向线程传递参数）

pass_arg2.c

```
1. int main(int argc, char *argv[])
2. {
3.   pthread_t threads[NUM_THREADS]; /* 设置NUM_THREADS为4 */
4.   int rc;
5.   long t;
6.
7.   for(t=0;t<NUM_THREADS;t++) {
8.     printf("Creating thread %ld\n", t);
9.     rc = pthread_create(&threads[t], NULL, PrintHello, (void *) &t);
10.    if (rc) {
11.      printf("ERROR; return code from pthread_create() is %d\n", rc);
12.      exit(-1);
13.    }
14.   }
15.  pthread_exit(NULL);
16.}
```

所有线程共用变量t

```
liu@ubuntu:~/Desktop/OS$ ./pass_arg2
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Hello from thread 4
Hello from thread 4
Hello from thread 4
Hello from thread 4
```

Passing Arguments to Threads（向线程传递参数）

pass_arg1.c

```
1. int main(int argc, char *argv[])
2. {
3.   pthread_t threads[NUM_THREADS];
4.   long taskids[NUM_THREADS];
5.   int rc, t;
6.
7.   for(t=0;t<NUM_THREADS;t++) {
8.     taskids[t] = t;
9.     printf("Creating thread %d\n", t);
10.    rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
11.    if (rc) {
12.      printf("ERROR; return code from pthread_create() is %d\n", rc);
13.      exit(-1);
14.    }
15.  }
16.  pthread_exit(NULL);
17.}
```

向每个线程单独传递一个变量

```
liu@ubuntu:~/Desktop/OS$ ./pass_arg1
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Hello from thread 0
Hello from thread 1
Hello from thread 2
Hello from thread 3
```

Terminating Threads（终止线程）

- `void pthread_exit(void *thread_return)`
- There are several ways in which a thread may be terminated:
  - The thread terminates *implicitly* when its top-level thread routine returns.
  - The thread terminates *explicitly* by calling the `pthread_exit` function.
  - If the main thread calls `pthread_exit`, it waits for all other peer threads to terminate and then terminates the main thread and the entire process with a return value of `thread_return`.
  - The entire process is terminated due to making a call to the `exit()`

- `int pthread_cancel(pthread_t tid)`
  - calling the `pthread_ cancel` function with the ID of the current thread.

Pthread Creation and Termination

```
1. void *PrintHello(void *threadid)
2. {
3.     long tid;
4.     tid = (long)threadid;
5.     printf("Hello World! It's me, thread #%ld!\n", tid);
6.     pthread_exit(NULL);
7. }
8. int main(int argc, char *argv[])
9. {
10.    pthread_t threads[NUM_THREADS]; /* NUM_THREADS = 5 */
11.    int rc;
12.    long t;
13.    for(t=0;t<NUM_THREADS;t++){
14.      printf("In main: creating thread %ld\n", t);
15.      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
16.      if (rc){
17.        printf("ERROR; return code from pthread_create() is %d\n", rc);
18.        exit(-1);
19.        }
20.      }
21.    /* Last thing that main() should do */
22.    pthread_exit(NULL);
23.}
```

回收当前线程

回收主线程

terminate_pthread.c

Pthread Creation and Termination

第一次运行：

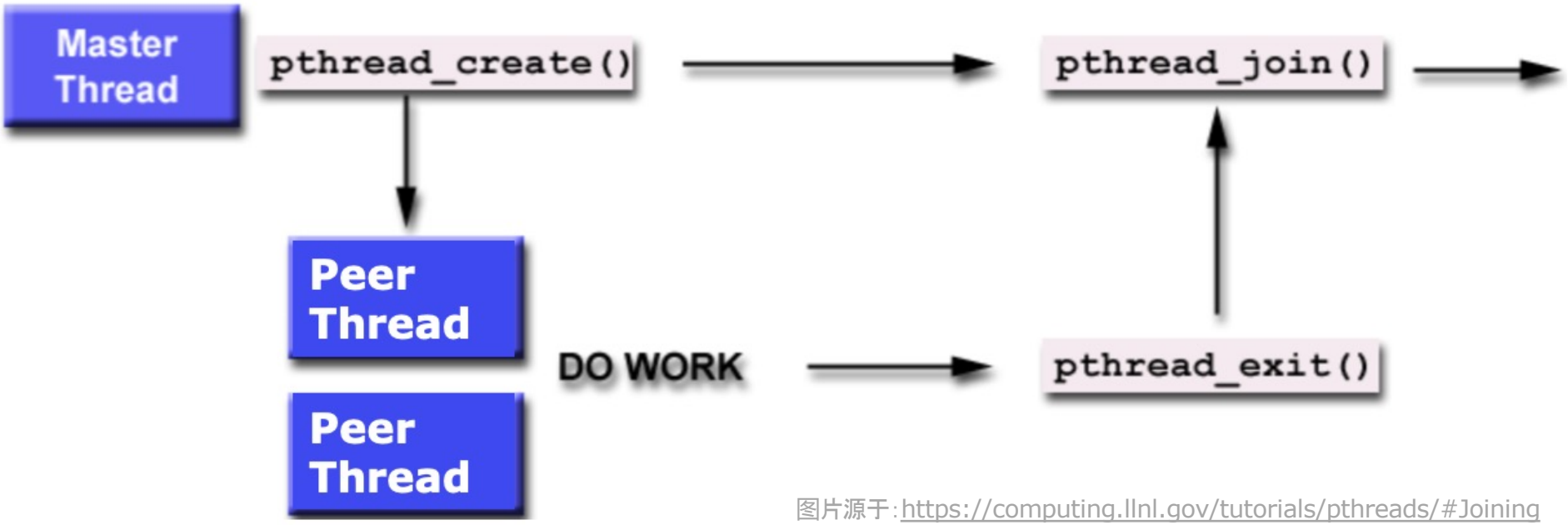
```
liu@ubuntu:~/Desktop/OS$ ./terminate_pthread
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
In main: creating thread 3
Hello World! It's me, thread #2!
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

第二次运行：

```
liu@ubuntu:~/Desktop/OS$ ./terminate_pthread
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
Hello World! It's me, thread #0!
Hello World! It's me, thread #2!
Hello World! It's me, thread #1!
In main: creating thread 3
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

Joining（回收）and Detaching（分离）Threads

- `int pthread_join(pthread_t tid, void **thread_return)`
  - The `pthread_join` function blocks until thread `tid` terminates
  - unlike the Linux `wait` function, the `pthread_join` function can only wait for a specific thread to terminate
  - A joining thread can match one `pthread_join()` call. It is a logical error to attempt multiple joins on the same thread.



图片源于：https://computing.llnl.gov/tutorials/pthreads/#Joining

Joining（可结合的）and Detaching（分离的）Threads

- `int pthread_detach(pthread_t tid)`
  - The `pthread_detach` function detaches the joinable thread `tid`.
  - By default, threads are created joinable.
  - each joinable thread should be either explicitly reaped by another thread or detached by a call to the `pthread_detach` function.

- To explicitly create a thread as joinable or detached, the attr argument in the `pthread_create()` routine is used. The typical 4 step procedure is:
  - Declare a pthread attribute variable of the `pthread_attr_t` data type
  - Initialize the attribute variable with `pthread_attr_init()`
  - Set the attribute detached status with `pthread_attr_setdetachstate()`
  - When done, free library resources used by the attribute with `pthread_attr_destroy()`

Pthreads Creation and Joining

```
1. void *BusyWork(void *t)
2. {
3.     int i;
4.     long tid;
5.     double result=0.0;
6.     tid = (long)t;
7.     printf("Thread %ld starting...\n",tid);
8.     for (i=0; i<1000000; i++)
9.     {
10.        result = result + sin(i) * tan(i);
11.    }
12.    printf("Thread %ld done. Result = %e\n",tid, result);
13.    pthread_exit((void*) t);
14.}
```

join_pthread.c

Pthreads Creation and Joining

```
1.  /* Initialize and set thread detached attribute */
2.     pthread_attr_init(&attr);
3.     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
4.
5.     for(t=0; t<NUM_THREADS; t++) {
6.        printf("Main: creating thread %ld\n", t);
7.        rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
8.        if (rc) {
9.           printf("ERROR; return code from pthread_create() is %d\n", rc);
10.          exit(-1);
11.          }
12.       }
13.  /* Free attribute and wait for the other threads */
14.    pthread_attr_destroy(&attr);
15.    for(t=0; t<NUM_THREADS; t++) {
16.       rc = pthread_join(thread[t], &status);
17.       if (rc) {
18.          printf("ERROR; return code from pthread_join() is %d\n", rc);
19.          exit(-1);
20.          }
21.       printf("Main: completed join with thread %ld having a status of %ld\n",t,
    (long)status);
22.       }
23.
24. printf("Main: program completed. Exiting.\n");
25. pthread_exit(NULL);
```

join_pthread.c

向线程传递参数

```
1. void *thread(void *vargp);
2. char **ptr;  /* Global variable */ //line:conc:sharing:ptrdec
3. int main()
4. {
5.     int i;
6.     pthread_t tid;
7.     char *msgs[N] = {
8.     "Hello from foo",
9.     "Hello from bar"
10.     };
11.     ptr = msgs;
12.     for (i = 0; i < N; i++)
13.         Pthread_create(&tid, NULL, thread, (void *)i);
14.     Pthread_exit(NULL);
15.}
16.void *thread(void *vargp)
17.{
18.     int myid = (int)vargp;
19.     static int cnt = 0; //line:conc:sharing:cntdec
20.     //line:conc:sharing:stack
21.     printf("[%d]: %s (cnt=%d)\n", myid, ptr[myid], ++cnt);
22.     return NULL;
23.}  /* $end sharing */
```

sharing.c

向线程传递参数

变量实例	被主线程引用？	被对等线程P0引用？	被对等线程P1引用？	说明
ptr	✔	✔	✔	A **global variable** that is written by the main thread and read by the peer threads.
cnt	✘	✔	✔	A **static variable** with only one instance in memory that is read and written by the two peer threads.
i.m	✔	✘	✘	A **local automatic variable** stored on the stack of the main thread
msgs.m	✔	✔	✔	A **local automatic variable** stored on the main thread's stack and referenced indirectly through **ptr** by both peer threads.
myid.p0	✘	✔	✘	Instance of a **local automatic variable** residing on the stacks of peer thread 0
myid.p1	✘	✘	✔	Instance of a **local automatic variable** residing on the stacks of peer thread 1