1. Web服务器的发展迫使UNIX系列**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "_blank)**开始重视线程，Web服务器进程需要向多个客户端提供服务，因此逐渐舍弃进程，而用效率更高的线程。

2. 多进程模型的缺点：

a. 创建进程的过程需要大的开销；

b. 进程间通信需要用到IPC技术；

c. 经常发生“上下文切换Context Switching”，时间很长，很致命：进程A切换到进程B时，要将进程A信息移出内存，并读入进程B信息。

多线程（轻量级进程）的优点：

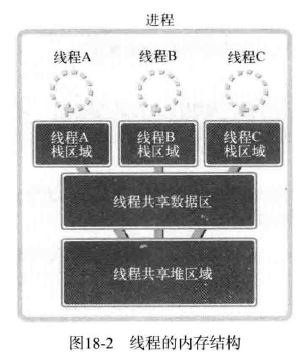
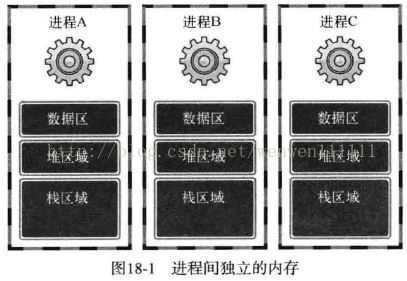
a. 线程的创建和上下文切换比进程更快；

b. 线程间通信无需特殊技术。

3. 进程VS线程

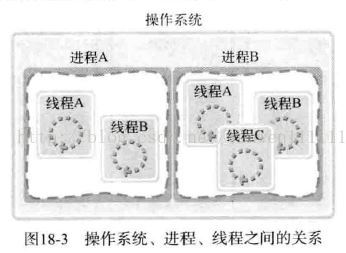
a. 每个进程都有自己的数据区、heap、stack，进程间相互独立。

线程间共享数据区和heap（线程拥有自己的栈）。



b. 进程：在操作系统构成单独执行流的单位；

线程：在进程构成单独执行流的单位；



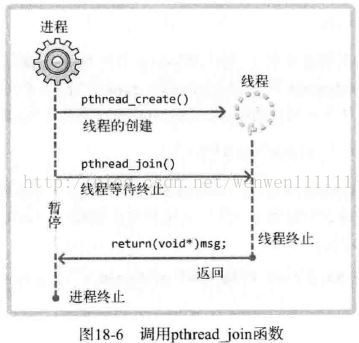
4. 线程创建pthread\_create

POSIX标准：Portable Operating System Interface for Computer Environment适用于计算机环境的可移植操作系统接口

1. #include <pthread.h>
3. **int** pthread\_create(pthread\_t \* restrict **thread**, **const** pthread\_attr\_t restrict attr, **void** \*(\*start\_routine)(**void** \*), **void** \* restrict arg)
4. //restrict用来修饰指针不能被别的指针引用
5. 成功返回0，失败其他值
6. **thread**：保存新创建进程ID的变量地址值
7. attr：用于传递线程属性的参数，NULL表示默认
8. start\_routine：线程的执行流函数指针
9. arg：第三个参数函数指针的参数信息

5. 进程/线程等待pthread\_join

1. **int** pthread\_join(pthread\_t **thread**, **void** \*\*status)
3. 成功返回0，失败其他值
5. **thread**：等待的线程ID
6. status：保存线程的main函数返回值的指针变量地址值



6. 可在临界区critical section内调用的函数

临界区：多个线程同时调用函数时可能产生问题，这类函数内部存在临界区（共同访问的那块代码）。临界区中至少存在一条这类代码。

线程安全函数：被多个线程同时调用时不会引发问题；线程安全函数的名称以\_r后缀 //与临界区无关，安全函数也可能有临界区

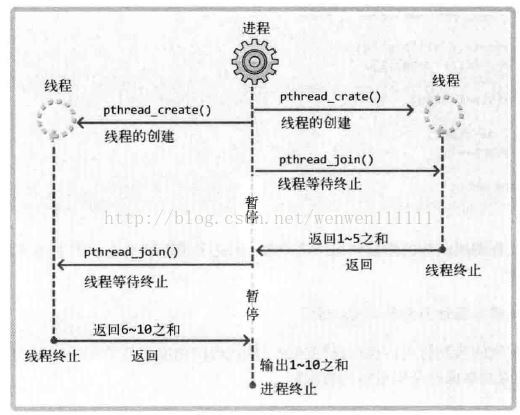
非线程安全函数：被同时调用时会引发问题。

编译使用\_r的函数方法：声明头文件前定义\_REENTRANT，也可以

# gcc -D\_REENTRANT mythread.c -o mthread -lpthread

7. 示例：工作worker线程模型

代码只说模型应用，里面存在临界区问题。



1. #include <stdio.h>
2. #include <pthread.h>
3. **void** \* thread\_summation(**void** \*arg);
5. **int** sum = 0;
7. **int** main(**int** argc, **char** \*argv[]){
8. pthread\_t id\_t1,id\_t2;
9. **int** range1[] = {1,5};
10. **int** range2[] = {6,10};
12. pthread\_create(&id\_t1,NULL,thread\_summation,(**void** \*)range1);
13. pthread\_create(&id\_t2,NULL,thread\_summation,(**void** \*)range2);
15. pthread\_join(id\_t1,NULL);
16. pthread\_join(id\_t2,NULL);
18. printf("sum = %d\n",sum);
19. **return** 0;
20. }
22. **void** \* thread\_summation(**void** \*arg){
23. **int** start = ((**int** \*)arg)[0];
24. **int** end = ((**int** \*)arg)[1];
26. **while**(start <= end){
27. sum += start;
28. start++;
29. }
31. **return** NULL;
32. }

8. 临界区问题示例

1. #include <stdio.h>
2. #include <pthread.h>
3. #include <unistd.h>
4. #include <stdlib.h>
6. #define NUM\_THREAD  100

9. **void** \* thread\_inc(**void** \*arg);
10. **void** \* thread\_des(**void** \*arg);
12. **long** **long** num = 0;  //bit64
14. **int** main(**int** argc, **char** \*argv[]){
15. pthread\_t thread\_id[NUM\_THREAD];
16. **int** i;
18. **for**(i=0;i<NUM\_THREAD;i++){
19. **if**(i%2)
20. pthread\_create(&thread\_id[i],NULL,thread\_inc,NULL);
21. **else**
22. pthread\_create(&thread\_id[i],NULL,thread\_des,NULL);
23. }
24. **for**(i=0;i<NUM\_THREAD;i++)
25. pthread\_join(thread\_id[i],NULL);
27. printf("result = %lld\n",num);
28. **return** 0;
29. }
31. **void** \* thread\_inc(**void** \*arg){
32. **int** i;
34. **for**(i=0;i<50000000;i++){
35. num += 1;
36. }
38. **return** NULL;
39. }
41. **void** \* thread\_des(**void** \*arg){
42. **int** i;
44. **for**(i=0;i<50000000;i++){
45. num -= 1;
46. }
48. **return** NULL;
49. }