程的操作系统上实现。过去所有的操作系统都属于这个范围，即使现在也有一些操作系统还是不支持线程。通过这一方法，可以用函数库实现线程。

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

#define NUMBER_OF_THREADS    10

void *print_hello_world(void *tid)
{
      /* 本函数输出线程的标识符，然后退出。 */
      printf("Hello World. Greetings from thread %d0, tid);
      pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
      /* 主程序创建10个线程，然后退出。 */
      pthread_t threads[NUMBER_OF_THREADS];
      int status, i;

      for(i=0; i < NUMBER_OF_THREADS; i++) {
            printf("Main here. Creating thread %d0, i);
            status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

            if (status != 0) {
                  printf("Oops. pthread_create returned error code %d0, status);
                  exit(-1);
            }
      }
      exit(NULL);
}
```

图2-15 使用线程的一个例子程序

所有的这类实现都有同样的通用结构，如图2-16a所示。线程在一个运行时系统的顶部运行，这个运行时系统是一个管理线程的过程的集合。我们已经见过其中的四个过程：pthread_create，pthread_exit，pthread_join和pthread_yield。不过，一般还会有更多的过程。

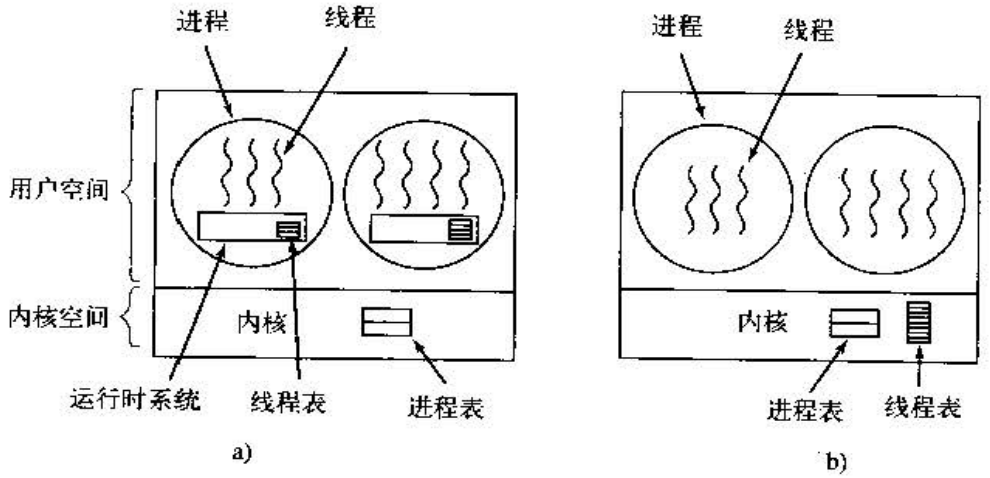

图2-16 a) 用户级线程包；b) 由内核管理的线程包

在用户空间管理线程时，每个进程需要有其专用的线程表（thread table），用来跟踪该进程中的线程。这些表和内核中的进程表类似，不过它仅仅记录各个线程的属性，如每个线程的程序计数器、堆栈