第 4 章作业

4.1 Provide three programming examples in which multithreading provides better performance than a single-threaded solution.

答：

（1）一个Web服务器接收有关网页、图像、声音等的客户请求。一个繁忙的Web服务器可能有多个客户同时访问它。如果一个Web服务器作为单个线程的传统进程来执行，那么只能一次处理一个请求；这样，客户可能需要等待很长时间，其请求才能被处理。如果Web服务器进程是多线程的，那么就能在用一个线程处理一个客户的请求时，用其他线程继续监听其他客户可能的请求，于是可以同时服务多个客户而不需要客户等待很长时间。

（2）一个字处理器可能有一个线程用于显示图形，另一个进程用于响应用户的键盘输入，还有一个线程在后台进行拼写和语法的检查。如果这些都只在一个线程实现，那么每一步的响应时间都会有延迟，从而影响使用。

（3）RPC服务器通常是多线程的。当一个服务器收到消息时，它使用一个单独线程来处理消息。这允许服务器处理多个并发请求。如果使用单线程则无法同时处理多个请求。

4.4 What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

答：用户线程位于内核之上，它的管理无需内核支持；而内核线程由操作系统来直接支持与管理。用户线程的管理由用户空间的线程库来完成，效率更高。内核不知道用户线程的存在，知道内核线程的存在。创建用户线程所需的资源比创建内核线程所需的资源少。

用户线程优于内核线程的情况：

如果内核是时间共享的，那么用户线程比内核线程更好。因为在时间共享系统中，上下文切换频繁发生，内核线程间的上下文切换开销很高，几乎与进程相同，而用户线程间的上下文切换的开销远小于内核线程。

内核线程优于用户线程的情况：

如果内核是单线程的，那么内核线程优于用户线程。因为执行阻塞系统调用的任何用户线程都将导致整个进程阻塞，即使其他线程在应用程序中可以运行。

如果是在多处理器环境中，那么内核线程优于用户线程。因为内核线程可以同时在不同的处理器上运行，而用户线程只能在一个处理器上运行，即使多个处理器可用。

4.10 Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values

b. Heap memory

c. Global variables

d. Stack memory

答：b c

堆内存和全局变量在多线程进程中共享。

在多线程进程中，每个线程都有自己的堆栈和寄存器值。

4.17 Consider the following code segment:

```
pid_t pid;

pid = fork();
if (pid == 0) { /* child process */
   fork();
   thread_create( . . .);
}
fork();
```

a. How many unique processes are created?

答：6个。父进程P1，父进程通过第一个fork()创建P2，P2通过第二个fork()创建P3，P1、P2、P3通过第三个fork()创建P4、P5、P6。

b. How many unique threads are created?

答：2个。P2、P3通过thread_create()创建T1、T2。

4.19 The program shown in Figure 4.23 uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

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

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid;
pthread_attr_t attr;

   pid = fork();

   if (pid == 0) { /* child process */
      pthread_attr_init(&attr);
      pthread_create(&tid,&attr,runner,NULL);
      pthread_join(tid,NULL);
      printf("CHILD: value = %d",value); /* LINE C */
   }
   else if (pid > 0) { /* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE P */
   }
}

void *runner(void *param) {
   value = 5;
   pthread_exit(0);
}
```

Figure 4.22 C program for Exercise 4.19.

答：

LINE C 输出：

```
CHILD: value = 5
```

LINE P 输出：

```
PARENT: value = 0
```

父进程与子进程各有各的内存空间，所以父进程与子进程的 value 的值互不影响。

对于子进程，子进程创建的新线程共享子进程的内存空间，所以在新线程中修改 value 的值为 5，子进程输出 value 的值为 5。

对于父进程，value 的值没有变化，输出为 0。