Chapter 4 Homework

陈文迪 519021910071

作业中的引用内容均已标出

4.1 给出三个多线程编程比单线程编程性能更优的编程例子。

1. 在设计GUI时，多线程更优。若是采用单线程编程，则一处阻塞或冗长操作就可能导致整个界面反应迟钝；而多线程编程可以单独分配一个线程给耗时操作，应用程序的其他部分仍然可以及时响应用户的操作。
2. 对于Web服务器，多线程更优。Web服务器需要同时处理多个并发请求。若是采取单线程编程，则服务器一次只能处理一位客户的请求，而无法相应其他客户；而若是采用多线程编程，则可以分配一个新线程来监听其他客户的请求，减少客户的等待时间。
3. 在多核或多线程系统上开发并行软件时，多线程更优。例如将一个数组中每个元素加一，这样的操作可以并行化，更适合采用多线程编程。我们可以为每个核分配一个线程，同时利用多个计算核进行计算，提高系统的处理效率。

4.4 用户线程和内核线程之间的两个区别是什么？在哪种情况下一种类型比另一种更好?

第一个区别是二者的管理方式不同。程序员一般通过线程库来创建和管理用户线程，这是因为

> 用户线程对程序员来说是可见的，而对内核来说是未知的。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。

第二个区别是硬件与二者的关系。无论我们采用哪种多线程模型，若要想实现硬件上的并行处理，都必须通过多个内核线程进行。这是因为只有内核线程可以被分配到某一处理器/计算核上，否则即使多个用户线程也仅仅支持了并发而非并行。

二者发挥着不同的功能，并不存在哪一种绝对比另一种好。但是，在管理效率上，二者确实有差异，

> 用户线程与内核线程相比，创建和管理要更快，因为它不需要内核干预。

程序员可以在用户空间下利用线程库完成线程管理，效率更高。

4.10 多线程进程中的线程之间共享以下哪些程序状态?

1. 寄存器值
2. 堆内存
3. 全局变量
4. 栈内存

回答:

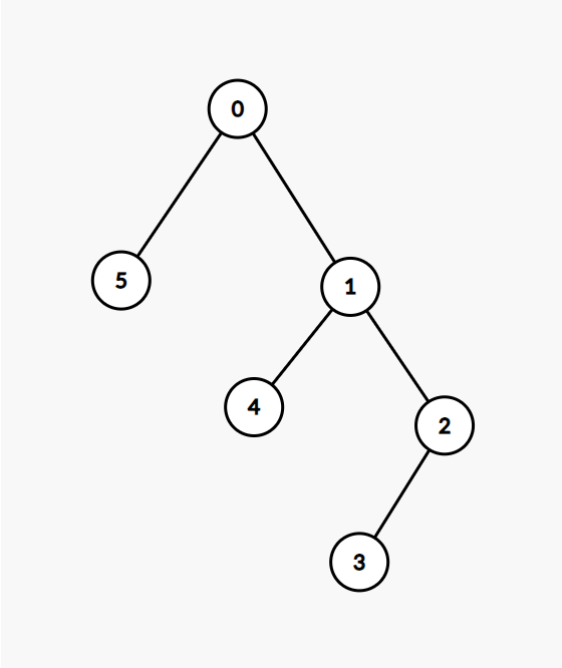
> 对于POSIX和Windows线程，全局声明（即在函数之外声明的）的任何数据，可为同一进程的所有线程共享。对于Java，线程对共享数据的访问必须加以显式安排。由于每个线程都有自己的堆栈，每个线程都有自己的本地数据。

而一般来说，每个线程也具有不同的寄存器值。而多线程之间可以通过堆中动态分配的数据进行数据共享。因此，全局变量和堆内存是共享的。

4.17 考虑下列代码片段，问：(a) 有多少不同的进程被创建了 (b) 有多少不同线程被创建了?

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

(a) 整段代码执行完后产生的进程树如下图所示，可以看到一共有6个进程。



(b) 对于每一个进程，至少有一个线程，对于进程1、2，他们都调用了 `thread create(...)` ，所以有两个线程。由于最后一次 `fork()` 之后没有调用 `exec()` ，所以所有的线程都会被复制。因此总线程数是

$$（1 + 2 \times 2） \times 2 = 10$$

4.19 下列程序使用了 Pthreads API，请分析在LINE C处和LINE P处的输出分别是什么?

```
#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);
}
```

LINE C 输出： CHILD: value = 5

LINE E 输出： PARENT: value = 0

对于Pthreads来说，全局数据确实可被同一进程的所有线程共享，但在本题中，程序先通过 `fork()` 创建了一个新进程再在该进程中创建一个新线程，而这两个进程之间全局数据是不共享的。