Homework 4

余北辰 519030910245

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

a. 一个Web服务器接收到对网页、图片、声音等的客户端请求。若采取单线程，一次只能处理一个请求；采取多线程来处理时，可以每接收一个请求，就开辟一个新的线程进行处理。

b. 一个字处理器可以有一个线程用于显示图形界面，一个线程用于响应用户的键盘输入，还有一个线程位于后台进行拼写和语法检查。

c. RPC系统中，RPC服务器往往是多线程的。当一个服务器接收到消息时，就是用一个单独的线程来处理消息。这样可以用于处理多个并发的请求。

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

用户线程在用户层，位于内核之上，其管理无需内核的支持；
内核线程在内核层，由操作系统来直接支持和管理。

在多处理器环境下运行时，内核线程优于用户线程，因为内核线程可以使用内核的线程调度器，将不同的内核线程分配到不同的CPU上去运行；而用户线程无法在多处理器上并行运行，即使有空余的CPU可用。

在分时系统下运行时，用户线程则优于内核线程。分时系统将多任务分时进行，这需要进行调度；而用户线程上下文切换所需的开销要远低于内核线程上下文切换的开销。

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

b,c是各线程间共享的，而a,d是各线程各自独立维护的。

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?

先fork()一次后产生两个进程，两个进程中父进程后来fork()一次，子进程fork()两次，因此一共有2+4=6个进程。

b. How many unique threads are created?

第一次fork()产生的子进程会进入if中，而子进程又会在if内fork()一次，于是这两个进程会产生两个线程。

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);
}
```

CHILD: value = 5
PARENT: value = 0

对于进程而言，父子进程间的全局变量是独立的，子进程全局变量的值的改变不会影响父进程全局变量的改变。