操作系统第四章习题

4.2

1. 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直接支持和管理

2. 在使用一对一模型或多对多模型的系统上，用户线程由线程库调度，而内核调度内核线程。内核线程不需要与一个进程相关联，而每个用户线程都属于一个进程。内核线程通常比用户线程的维护成本更高，因为它们必须用内核数据结构表示。

4.4

因为线程比进程小，所以创建线程通常比创建进程使用更少的资源。创建一个进程需要分配一个进程控制块(PCB)，这是一个相当大的数据结构。PCB包括内存映射、打开的文件列表和环境变量。分配和管理内存映射通常是最耗时的活动。创建用户或内核线程都需要分配一个小数据结构来保存寄存器集、堆栈和优先级。

4.7

当内核线程发生页面错误时，可以切换到另一个内核线程，以一种有用的方式使用交错时间。另一方面，当发生页面错误时，单线程进程将无法执行有用的工作。因此，在程序可能出现频繁的页面错误或必须等待其他系统事件的情况下，多线程解决方案甚至在单处理器系统上也能执行得更好。

4.8

多线程进程的线程共享堆内存和全局变量。每个线程都有其独立的寄存器值集和独立的栈。

4.9

由多个用户级线程组成的多线程系统不能在多处理器系统中同时使用不同的处理器。操作系统只看到一个进程，不会将该进程的不同线程调度到不同的处理器上。因此，在多处理器系统上执行多个用户级线程不会带来性能上的好处。

4.14

线程数取决于应用程序的优先级和要求。因此，对于这种应用程序，只有线程足够了，这个线程将同时处理输入和输出操作——并发，关键点是创建与阻塞系统调用数相同的线程，因为线程将被用来阻塞。而创建额外的线程也没啥益处。因此，创建一个线程来执行输入，一个进程来执行输出。

创建四个线程来执行应用程序的CPU密集型部分。因为：线程数应该和处理核数相同。如果线程数比4小，会浪费处理资源。而大于4个线程就不能运行。

4.15

a:6个进程：P父进程，pid = fork()创建的子进程P1，if语句中fork()创建的P2，以及P，P1和P2在最后一个fork()中分别创建的P3，P4和P5——共6个进程，不算父进程的话，unique processes有5个

b：线程创建是在if中完成的。if块中执行独子进程P1。因此，进程P1将被创建一个线程。进程P2使用fork()创建。因此，进程P2也将创建一个线程。因此，将创建2个unique线程。

4.17

LINE C值＝5。线程中的子进程由父进程fork，子进程每个都有自己的内存空间。在fork之后，父进程等待子进程的完成。为子进程创建新线程，调用Runnor（）函数，将全局ValLable的值设置为5。因此，在执行此行之后，所显示的值将为5。

LINE P值＝0。在完成子进程之后，父进程中存在的全局变量的值仍然为0。因此，在执行此行之后，所显示的值将为0。

4.21

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

double average;

int minimum;

int maximum;

void \*calc\_average(void \*arg) {

int \*numbers = (int \*)arg;

int sum = 0;

for (int i = 0; i < 7; i++) {

sum += numbers[i];

}

average = sum / 7.0;

return NULL;

}

void \*calc\_minimum(void \*arg) {

int \*numbers = (int \*)arg;

minimum = numbers[0];

for (int i = 1; i < 7; i++) {

if (numbers[i] < minimum) {

minimum = numbers[i];

}

}

return NULL;

}

void \*calc\_maximum(void \*arg) {

int \*numbers = (int \*)arg;

maximum = numbers[0];

for (int i = 1; i < 7; i++) {

if (numbers[i] > maximum) {

maximum = numbers[i];

}

}

return NULL;

}

int main(int argc, char \*argv[]) {

if (argc != 8) {

printf("Usage: ./program num1 num2 num3 num4 num5 num6 num7\n");

return 1;

}

int numbers[7];

for (int i = 0; i < 7; i++) {

numbers[i] = atoi(argv[i + 1]);

}

pthread\_t threads[3];

pthread\_create(&threads[0], NULL, calc\_average, (void \*)numbers);

pthread\_create(&threads[1], NULL, calc\_minimum, (void \*)numbers);

pthread\_create(&threads[2], NULL, calc\_maximum, (void \*)numbers);

for (int i = 0; i < 3; i++) {

pthread\_join(threads[i], NULL);

}

printf("The average value is %.0f\n", average);

printf("The minimum value is %d\n", minimum);

printf("The maximum value is %d\n", maximum);

return 0;

}