操作系统第三次作业（201700130011 菁英班 刘建东）

4.1

> 题目：举两个多线程程序设计的例子，其中多线程的性能比单线程的性能差。

1. 在单核单处理器的环境下，多线程只是加快单个线程的响应速度，对于总执行时间没有提高，甚至会因为线程调度而降低效率。
2. 当线程处理的是比较小，或者各线程间有复杂连接的任务时，单线程性能更好。例如一个线程输出10万条打印信息，和1000个线程各输出100条打印信息，由于输出端是临界资源，后者多线程抢占时间开销以及同步的时间开销非常大，因此前者性能更好。

4.4

> 题目：在多线程进程中，下列哪些程序状态组成被共享?
>
> a. 寄存器值
>
> b. 堆内存
>
> c. 全局变量
>
> d. 栈内存

在一个多线程进程中，全局变量、堆内存属于共享内存，而寄存器值、栈内存属于私有的、独立的内存。

4.7

> 题目：LINE C 和 LINE P 的输出值。

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

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

int main(int argc, char *argv[])
{
int 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 5，LINE P 输出 0。$fork()$ 函数先继承后分离，子进程继承父进程的所有信息，但是运行在不同的地址空间中，互不影响。父进程中 pid 返回子进程的进程号，子进程 pid 返回 0，子进程中对 $value$ 的修改对父进程中的 $value$ 没有影响，因此父进程最终输出仍为 0，而子进程输出为 5。

6.1

> 题目：下述程序是否满足了临界区问题的三个要求。

```
do {
    flag[i] = TRUE;
    while (flag[j]) {
        if (turn == j) {
            flag[i] = false;
            while (turn == j)
                ; // do nothing
            flag[i] = TRUE;
        }
    }
        // critical section
    turn = j;
    flag[i] = FALSE;
        // remainder section
}while (TRUE);
```

> 要求 1：互斥（Mutual Exclusion）：P_i 在临界区中，P_j 不能进去。

上述程序中，只有当 $turn == i, flag[i] == true$ 且 $flag[j] == false$ 时，P_i 才能进入临界区，同理可得 P_j 所需条件。因此 P_i 在临界区时，P_j 一定进不去。

> 要求 2：前进（Progress）：临界区无进程时，P_i 可以进入。

当临界区无进程时，可能是程序刚开始，也可能是 P_j 刚退出临界区。如果是程序刚开始，$flag[i] == true$ 且 $flag[j] == false$，P_i 进入临界区，且 P_j 无法进入。

如果是 P_j 刚退出临界区，则 $turn == i, flag[j] == false, flag[i] == true$，则 P_i 成功进入临界区。

> 要求 3：等待时间有限（Bounded Waiting）：进程 P_i 不会无限制地等待。

由于 $turn$ 可能取值仅为 i 与 j，且 $turn$ 不会重复为 j，因为一旦 P_j 退出临界区之后，$turn == i$，因此 P_i 不会无限制地等待。

因此上述程序满足了临界区的三个要求。

6.4

> 题目：试解释为什么自旋锁对单处理器系统不合适而对多处理器系统合适。

自旋锁，即忙式等待，即进程反复检查锁变量是否可用，仅适用于锁使用者保持锁时间比较短的情况下。而在单处理器系统中，终究只有一个处理器，反复检查锁变量是否可用也不会改变结果，反而在反复检查的过程中非常降低效率，因此不适合。

而在多处理器系统中，线程可能会在另一个处理器上获得锁，因此对比普通锁，减少了上下文调度等开销，效率更高，也更合适。

6.5

> 题目：试解释为什么在单处理器系统上通过禁止中断实现同步原语方法不适用于用户级程序。

如果用户级程序具有禁止中断的能力，那么它就有可能禁止所有中断，从而避免上下文切换，进而独占处理器不让其他程序使用，同时使得整个系统处于瘫痪状态，因此不可行。

6.6

> 题目：试解释为什么通过禁止中断实现同步原语不适合于多处理器系统。

在多处理器系统上，禁止中断只是对于单个处理器来说的，因此即使你在当前处理器上禁止了中断，其它程序还是有可能在其它处理器上运行，无法保证互斥，仍然可能出错，因此不适合于多处理器系统。

6.16

> 题目：试解释管程的signal()操作与信号量signal()操作的区别。

管程中的 $signal()$ 操作与信号量中的 $signal()$ 操作适用范围不同。

在管程中，当执行 $signal()$ 操作且无等待线程时，OS会自动忽略 $signal()$ 操作，不予执行。而随后执行的 $wait()$ 操作，相关线程也会被阻塞。

但在信号量中，就算没有等待的线程，每个 $signal()$ 也会被执行，相应的信号量值也会增加，之后的 $wait()$ 操作也会因为之前信号量的增加而成功执行。

6.x

> 题目：系统中有多个生产者和消费者进程，共享一个可以存放1000个产品的缓冲区，当缓冲区未满时，生产者可以放入一件产品，缓冲区不空时，消费者可以从缓冲区中取产品，否则等待。要求消费者从缓冲区连续取走10件产品后，其它消费者才可以再取产品。试用信号量机制描述生产者和消费者之间的同步关系。

```
int in = 0, out = 0;
item buffer[1000]; //缓冲区
//mutex1 用于缓冲区，mutex2 用于消费者
semaphore mutex1 = 1, mutex2 = 1, empty = 1000, full = 0;
void producer(){
  do{
    //producer an item next_item;
    ...
    wait(empty);
    wait(mutex1);
    buffer[in] = next_item;
    in = (in+1)%1000;
    signal(mutex1);
    signal(full);
  }while(TRUE);
}

void consumer(){
  do{
    wait(mutex2);
    for(int i = 1; i <= 10; i++){
      wait(full);
      wait(mutex1);
      next_item = buffer[out];
      //consumer the next_item
      ...
      out = (out+1)%1000;
      signal(mutex1);
        signal(empty);
    }
    signal(mutex2);
  }
}

void main(){
  cobegin
    producer(); consumer();
  coend
}
```