《计算机系统工程导论》课后作业

1. 姓名： 王骏
2. 学号： 22020007104

1 线程与虚拟解释器

1. 简述如何通过编程证明两个线程共享内存?

   使用两个线程访问和修改同一个全局变量。如果线程1修改的值在线程2中可见，就说明它们共享内存。

   比如可以这么写，需要在Linux环境下进行：

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

int shared_var = 0;

void* thread_func(void* arg) {
    shared_var += 1;
    printf("Thread sees shared_var = %d\n", shared_var);
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread_func, NULL);
    pthread_create(&t2, NULL, thread_func, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("Main sees shared_var = %d\n", shared_var);
    return 0;
}
```

2. 简述如何通过编程证明线程具有独立堆栈，以及确定堆栈的大小?

   在线程中定义局部变量并打印其地址，观察每个线程中局部变量地址是否不同；可结合调试器或地址间距估算堆栈大小。

   比如可以这么写，需要在Linux环境下进行：

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

void* thread_func(void* arg) {
    int local_var;
    printf("Thread %ld: address of local_var: %p\n", (long)arg, (void*)&local_var);
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread_func, (void*)1);
    pthread_create(&t2, NULL, thread_func, (void*)2);
```

```
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```

3. 小明正在使用条件变量编写有界缓冲区发送代码。该代码的wait()函数不会将锁作为调用的参数；相反，程序员需要自行释放/获取相应的锁。小明当前的的伪代码如下（为方便起见，代码中包含了行号）。

```
1: send(bb, message):

2:   acquire(bb.lock)

3:   while (bb.in - bb.out == N):

4:     release(bb.lock)

5:     wait(bb.not_full)

6:     acquire(bb.lock)

7:   bb.buf[bb.in % N] = message

8:   bb.in = bb.in + 1

9:   release(bb.lock)

10:   notify(bb.not_empty)
```

小明知道上述代码容易出现 "通知丢失 "的问题。为了解决这个问题，她建议调换第 4 行和第 5 行的顺序。请回答下列各项的真假：

（1） 对/错：交换会导致新的条件竞争

对

原因： 如果不先释放锁就等待（wait），可能会在释放锁前错过其他线程发送的 `notify`，从而永远等待。

（2） 对/错：交换会导致死锁

错

原因： 虽然存在条件竞争，但由于wait本身会阻塞线程，不会导致两个线程互相等待资源造成死锁。

4. 以下是一段支付操作的简化代码：

```
1:  unsigned int balance = 100;

2:

3:  int Alipay_withdraw(int amt) {

4:    if (balance >= amt) {

```

```
5:      balance -= amt;

6:      return SUCCESS;

7:    } else {

8:      return FAIL;

9:    }

10: }
```

当两个线程并发支付100，简述balance会产生什么结果并说明原因。

```
unsigned int balance = 100;

if (balance >= amt) {
    balance -= amt;
}
```

问题： 两个线程都可能同时通过 `balance >= amt` 的判断，进入 `balance -= amt`，导致最终余额变成 `0` 或 **负数**。

原因： 存在竞态条件，需加锁保护临界区。

5. 小明决定设计一个计算机系统，这个系统基于他称之为picokernels的新内核架构和叫做simplePC的硬件平台。simplePC平台包含一个简单的处理器、一个基于页面的虚拟内存管理器（用于转换处理器发出的虚拟地址）、一个内存模块和一个输入输出硬件。处理器有两个特殊寄存器，即程序计数器（PC）和堆栈指针（SP）。SP 指向堆栈顶端的值。

simplePC 处理器的调用约定使用简单的堆栈模型：

l 对过程的调用会将调用指令的下一条指令的地址压入栈并且跳到过程的第一条指令开始执行。

l 从过程调用返回时弹出栈顶的地址并跳转过去。

simplePC上的程序不使用局部变量。程序的参数在寄存器中传递，寄存器不会自动保存和恢复。因此，堆栈上唯一的值就是返回地址。

小明开发了一个简单的股票跟踪系统来跟踪他加入的新公司的股票。程序从输入设备读取一个整数，并打印到输出设备上：

```
    101. boolean input_available

    1. procedure READ_INPUT()
    2. do forever
    3. while input_available = FALSE do nothing // idle loop
    4. PRINT_MSG(quote)
    5. input_available ← FALSE

    200. boolean output_done
    201. structure output_buffer at 71FFF2_hex // hardware address of output buffer
    202. integer quote

    12. procedure PRINT_MSG(m)
```

```
    13. output_buffer.quote ← m
    14. while output_done = FALSE do nothing // idle loop
    15. output_done ← false

    17. procedure MAIN()
    18. READ_INPUT()
    19. halt                  // shutdown computer
```

除了主程序外，程序还包含两个存储过程：READ_INPUT和PRINT_MSG。程序READ_INPUT自旋等待，直到输入设备（股票阅读器）将input_available设为TRUE。输入设备收到股票报价后，会将报价值输入 msg，并将 input_available 设为TRUE。

过程PRINT_MSG会在输出设备（本例中为终端）上打印消息；它会将消息中存储的值写入设备，并等待打印完成；打印完成后，输出设备会将 output_done 设为TRUE。

每行的数字对应处理器发出的地址，用于读写指令和数据。假设每行伪代码都能编译成一条机器指令，并且每个过程的末尾都有一个隐式返回。

问题① 程序中每一行开头的的这些数字代表什么?

A. 虚拟地址

B. 物理地址

C. 页面数

D. 虚拟页面中的偏移量

程序中每一行开头的的这些数字代表：虚拟地址

小明直接在 simplePC 上运行程序，从主程序开始，在某一时刻，他观察到堆栈上有以下值（只有股票跟踪程序在运行）：

stack

19

5 ← stack pointer

问题② 栈上的值5是什么意思?

A． 下一条返回指令的返回地址

B． 前一条返回指令的返回地址

C． PC当前的值

D． SP当前的值

栈上的值5是：下一条返回指令的返回地址

问题③ 处理器正在执行哪个过程?

A． READ_INPUT

B． PRINT_MSG

C． MAIN

因为返回地址为5，意味着正在执行READ_INPUT并即将返回到5（主程序中的下一条）。

问题④ PRINT_MSG将一个值写入quote**，并将其存储在 71FFF2hex 地址，并期望该值最终会出现在终端上。使用什么技术来实现这一功能?**

A. 内存映射I/O

B. 顺序I/O

C. 流(stream)

D. 远程过程调用(RPC)

内存映射I/O，输出通过写入特定地址（如71FFF2h）传递到设备。

小明希望在simplePC平台上运行他的股票追踪程序的多个实例，以便获得更频繁的更新，更准确地追踪他当前的资产。小明为系统购买了另一个输入和输出设备，将它们连接起来，并实现了一个简单的线程管理器：

```
300. integer threadtable[2];   // stores stack pointers of threads.

// first slot is threadtable[0]

302. integer current_thread initially 0;



21. procedure YIELD()
22. threadtable [current_thread ] ← sp // move value of sp into table
23. current_thread ← (current_thread + 1) modulo 2
24. sp ← threadtable [current_thread] // load value from table into sp
25. return
```

每个线程读取和写入自己的设备，并拥有自己的堆栈。小明还修改了 READ_INPUT：

```


100. integer msg[2]        // CHANGED to use array
101. boolean input_available[2]  // CHANGED to use array


30. procedure READ_INPUT()
```

```
31. do forever
32. while input_available[current_thread] = FALSE do // CHANGED
33. YIELD()                          // CHANGED
34. continue                         // CHANGED
35. PRINT_MSG(msg[current_thread])       // CHANGED to use array
36. input_available[current_thread] ← FALSE  // CHANGED to use array
```

小明启动simplePC平台，在主线程中启动每个线程。两个线程正确地来回切换。小明暂时停止程序，观察以下堆栈：

```
stack of thread 0       stack of thread 1

19                19

36 ← stack pointer       34 ← stack pointe
```

问题⑤ 线程 0 正在运行（即current_thread** = 0）。下一次线程 0 在 YIELD中执行返回指令后，处理器将运行哪条指令?**

A． 34. continue

B． 19. halt

C． 35. PRINT_MSG(**msg**[**current_thread**]);

D． 36. input_available[current_thread] ← FALSE;

36. input_available[current_thread] ← FALSE;

因为36是当前函数（READ_INPUT）中调用 PRINT_MSG 之后的下一条语句。

问题⑥ 每个线程的堆栈上可以有哪些地址值?

A． 任何指令的地址

B． 被调用过程返回的地址

C． 任意数据位置的地址

D． 指令和数据位置的地址

被调用过程返回的地址

栈上只存返回地址，不存任意地址或数据。

小明观察到，股票滴答程序中的每个线程大部分时间都在轮询其输入变量。他引入了一个程序，让设备可以用来通知线程。他还重新编排了代码，以实现模块化：

```
400. integer state[2];


40. procedure SCHEDULE_AND_DISPATCH()
41. threadtable[current_thread] ← sp
42. while (what should go here?) do // See question 7.
43. current_thread ← (current_thread + 1) modulo 2
44. sp ← threadtable[current_thread];
45. return


50. procedure YIELD()
51. state[current_thread] ← WAITING
52. SCHEDULE_AND_DISPATCH()
53. return


60. procedure NOTIFY(n)
61. state[n] ← RUNABLE
62. return
```

当输入设备收到新的股票报价时，设备会中断处理器，并将当前运行线程的 pc 保存在当前运行线程的堆栈中。然后处理器运行中断处理程序。中断处理程序返回时，会从当前堆栈中弹出返回地址，将控制权返回给线程，中断处理程序的伪代码是：

```
procedure DEVICE(n)              // interrupt for input device n

push current thread's pc on stack pointed to by sp;

while input_available[n] = TRUE do nothing; // wait until read_input is done

// with the last input

msg[n] ← stock quote

input_available[n] ← TRUE

NOTIFY(n)                  // notify thread n

return                   // i.e., pop pc
```

在执行中断处理程序期间，中断将被禁用。因此，中断处理程序及其调用的程序（如NOTIFY）不能被中断。当DEVICE返回时，中断将重新启用。

使用新线程管理器，回答下列问题：

问题⑦ 为确保线程包的正确运行，应在地址 42 处的 while 中求值什么表达式?

A． state[current_thread] = WAITING

B． state[current_thread] = RUNABLE

C． threadtable[current_thread] = SP

D． FALSE

state[current_thread] = WAITING

跳过等待线程，直到找到可运行的线程。

问题⑧ 假设线程 0 正在运行，而线程 1 没有运行（即调用了 YIELD）。为了使线程 1 能够运行，需要发生什么事件

A． 线程 0 调用YIELD

B． 输入设备 1 的中断处理程序调用NOTIFY

C． 输入设备 0 的中断处理程序调用NOTIFY

D． 没有必须发生的事件

输入设备1的中断处理程序调用NOTIFY

NOTIFY 会将 state[1] 设为 RUNABLE，使线程1在调度器中被选中。

问题⑨ 每个线程的堆栈上可以有哪些值?

A． 任何指令的地址，设备驱动程序中断程序中的指令除外

B． 所有指令的地址，包括设备驱动程序中断程序中的指令地址

C． 程序返回的地址

D． 指令和数据位置的地址

程序返回的地址

栈只用于存储调用返回地址。

问题⑩ 什么情况下线程0会死锁?

A． 在执行YIELD第一条指令之前，设备 0中断线程 0 时

B． 在线程0执行第一条YIELD指令后，设备 0 中断

C． 当设备 0中断时，线程0执行到第5题之前给出的那段代码的READ _ INPUT程序中指令 35 和 36 之间

D． 当处理器执行SCHEDULE_AND_DISPATCH且线程 0 处于WAITING状态时，设备 0 中断

当处理器执行SCHEDULE_AND_DISPATCH且线程0处于WAITING状态时，设备 0 中断

此时中断不会被处理，线程0也不会运行，系统进入无响应状态。