操作系统第4次书面作业

- 数据科学与计算机学院
- 软工三班
- 米家龙
- 18342075

米客华 18362075 软工3班

6.1 算法满足条件

① 该算法中，如果两进程标识为真，则只有一个进程成功运行，当其更新返回变量时，等待的进程进入临界区域，因此互斥条件能够保证。

② 就绪的进程能够通过标志返回变量。算法没有提供严格的交替。如果进程想进入自己的临界区域，可将它标识为真，然后进入临界区域。如果要退出临界区域，可以通过设置转向其它进程的值。如果这个进程想要在其它在其它进程之前再次进入它的临界，会重复这个进程：进入临界区域，在退出时转向另一个进程

③ 在返回变量的过程中，等待表现。如果两进程想在其它进程前它们的临界区域，会将其标志的值设为真，只有轮到的线程可执行，其它线程都处于等待状态。如果等待表现，则重复进入-退出过程。Dekker算法在一个进程中设置一个转向其它进程的值，从而保证另一个进程能进入它的临界区域

6.2

① 当无其它进程在CS中有设置的标识量匹时，进程才能进入临界区域。因为进程在设置自身标识变量前要先检查其它进程状态从而保证互斥

② 有空给进入进程，如果同时都设置自己的标识变量，然后检查是否有其它进程也在设置，进程们会发现存在竞争，从而在外层while循环中进行下一次迭代——重置它们的标识量，现在将只有1个进程能设置轮次变量到CS，而该进程检查的输出信号会最接近轮次。因此，只有一个进程能在CS中设置标识，而其它信号在轮次与该进程同时不能，这个进程只能进临界区域

③ 有限等待：当一个进程打算进入临界区域时，标识不再为空闲，而信号在轮次和该变量之间的进程不能进入临界区域，而信号在上述两者之间的则可以，但轮次值会接近n，最终要么轮次变为k，要两者同时进程，使得k进入临界区域

6.3 ① 含义：进程等待满足一个没有闲置处理器的严格循环的条件

② 可避免，但需要将进程处于沉睡，达到某特定条件下唤醒量设进程

6.4 自旋锁若要打破进程，条件只能在执行其它进程时获得，单处理器系统无法做到。

6.5 如果该进程能被停止中断，则能停止计时器中断，防止上下文切换，从而一直占用处理器。

6.7 见下文

6.8 见下文

6.9 见下文

6.10 见下文

6.11 见下文

6.13 见下文

6.14 ① 将生产的值复制到监视器的本地缓冲区中，并将其从监视器的本地缓冲区复制回消费者。如果每个缓冲区都使用大量内存，这些操作可能花费很大。花费增大意味着在进程处于生产/消费操作中时，监视器将会保留更长时间，因此降低系统的整体吞吐量。

② 可通过将指针存储到监视器内的缓冲区而不是存储缓冲区本身，可以缓解该问题，可修改代码以完成上述操作，从而增加监视器的吞吐量。

6.16 ① 当执行signal操作并且无等待线程时，那么系统会忽略操作，认为signal操作没有发生过。如果随后执行wait操作，相关线程会阻塞

② 在信号量中，即使等待线程，每个signal操作都会是相应的信号量的值增加，接下来的等待操作因之前的信号量值的增加而马上成功进行。

7.1

a. ① 互斥：只有一辆车占据道路上的位置

② 占有并等待：一辆车占据道路上的位置并且等待前进

③ 非抢占：一辆车不能从道路上当前位置移开

④ 循环等待：每辆车正等待后续汽车向前发展

b. 汽车不得进入十字路口

7.2 a. 4个必要条件

① 互斥：需要相同的筷子

② 抢占并等待：筷子在手，等待其它筷子

③ 非抢占：筷子被分配后不能强行拿走

④ 循环等待可能出现

b. 避免方式：

① 允许同时分享筷子

② 如果无法获得筷子，那么允许其它人放弃自己的筷子

③ 如果允许筷子被强行拿走

④ 对筷子进行编号，按照优先级获取筷子

7.5 a. 可以安全解决

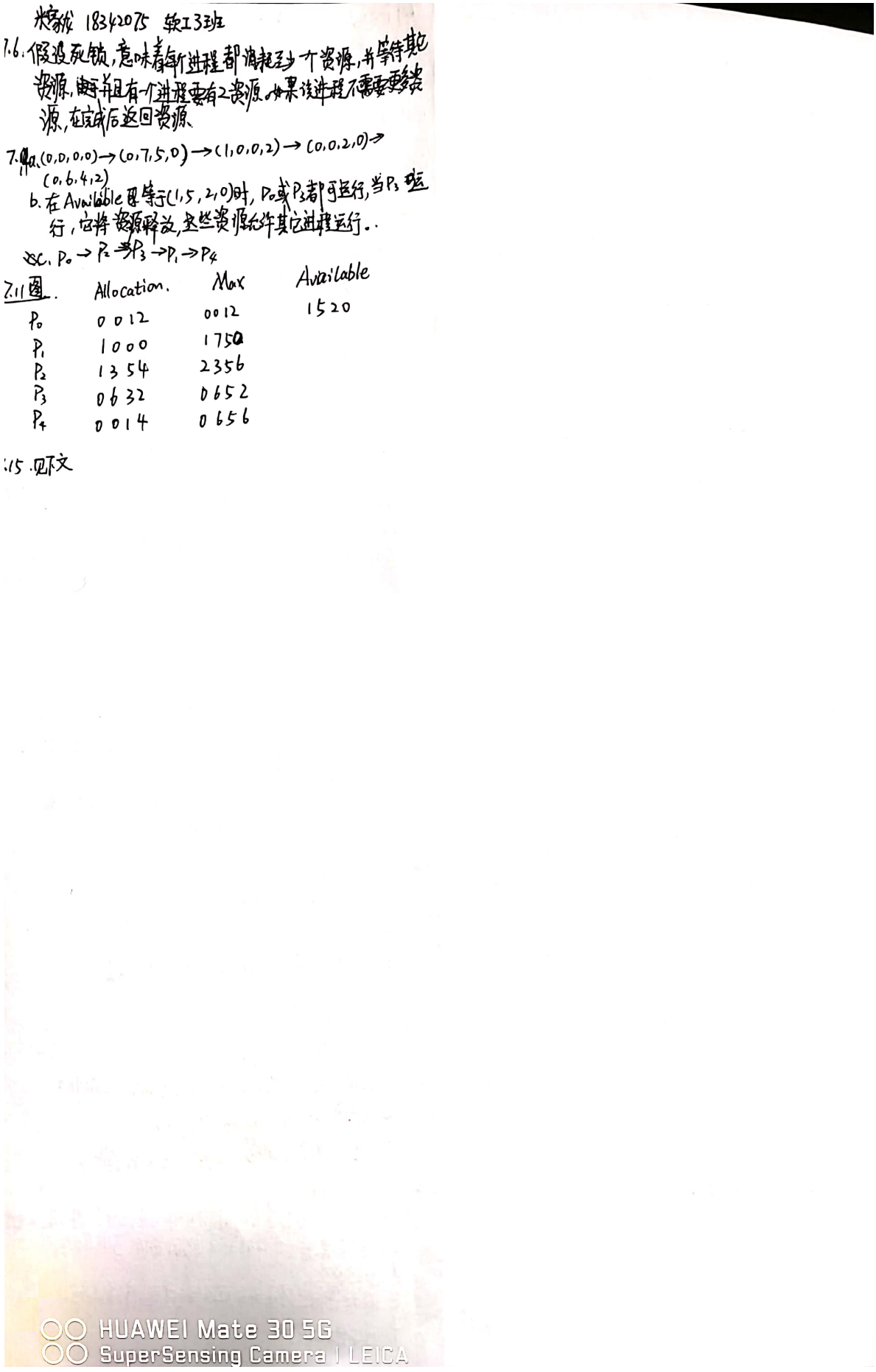
b. 假设有在一定数量的可用资源时，在系统安全时可能死锁

c. 该进程需要比允许的资源多的资源，可能导致

d. 可安全更改

e. 假设资源已经重新分配到新进程，可保证安全

f. 可安全解决



6.7

```
do
{
  waiting[1] = true;
  key = true;
```

```
    while(waiting[1] && key)
      key = Swap(&lock, &key);

    waiting[1] = false;

    j = (i + 1) % n;
    while((j != 1) && !waiting[j])
      j = (j + 1) % n;

    if (j == 1)
      lock = false;
    else
      waiting[j] = false;
  }
  while(true)
```

6.10

```
int guard = 0;
int semaphore_value = 0;

wait()
{
  while (TestAndSet(&guard) == 1)
    ;
  if (semaphore_value == 0)
  {
    // 自动将进程加入进程等待队列
    guard = 0;
  }
  else
  {
    semaphore_value--;
    guard = 0;
  }
}
signal()
{
  while (TestAndSet(&guard) == 1)
    ;
  if (semaphore_value == 0 && process_on_wait_queue)
    ; // 唤醒队列的第一个进程
  else
    semaphore_value++;
  guard = 0;
}
```

6.10

1. 对于 BarberShop

```
 public class BarberShop
 {
   private int chairNum;
   private int barber;
   private int chairState[];

   // 使用静态变量来表示状态
```

```
    static final int FULL = -1;
    static final int EMPTY = 0;
    static final int OCUPIED = 1;
    static final int SLEEPING = 2;
    static final int DONE = 3;
    static final int WAITED = 4;

    // 初始化
    public BarberShop(int chairNum)
    {
      this.chairNum = chairNum;
      this.chairState = new int[chairNum];
      this.barber = SLEEPING;

      for (int i = 0; i < this.chairNum; i++)
        this.chairState[i] = EMPTY;
    }

    // 找椅子
    private boolean findChair(int customer)
    {
      int tmp = getFirstEmptyChair();
      if (tmp == FULL) // 如果彻底满了
        return false;
      else             // 如果有位置
        this.chairState[tmp] = OCUPIED; // 占这个位置
    }

    public synchronized int getHairCut(int customer)
    {
      if (this.barber == SLEEPING) // 如果在休眠
      {
        this.barber = OCUPIED;
        return SLEEPING;
      }
      else if (this.barber == OCUPIED) // 如果拥塞
      {
        boolean tmp = findChair(customer);
        if (!tmp) // 人满
          return FULL;
        else
        {
          while(this.barber == OCUPIED) // 当处于拥挤的情况下，等待
          {
            try
            {
              this.wait();
            }
            catch(InterruptedException err)
            {}
          }

          for (int i = 0; i < this.chairNum; i++)
          {
            if (this.chairState[i] == OCUPIED) // 位置准备释放
            {
              this.chairState[i] = EMPTY;
              break;
```

```
            }
          }

          this.barber = OCUPIED;
          return WAITED;
        }
      }
      else
      {
        this.barber = OCUPIED;
        return DONE;
      }
    }

    public synchronized void leaveBarberShop(int customer)
    {
      boolean tmp = isAnyoneWaiting(); // 是否有人在等待
      if (tmp == true)
        this.barber = DONE;
      else
        this.barber = SLEEPING;

      notify();
    }

    private int getFirstEmptyChair()
    {
      // 循环找位置
      for (int i = 0; i < this.chairNum; i++)
      {
        if (this.chairState[i] == EMPTY)
          return i;
      }

      return FULL;
    }

    // 查看是否有人在等待 == 是否有位置被占用
    private boolean isAnyoneWaiting()
    {
      for (int i = 0; i < this.chairNum; i++)
      {
        if (this.chairState[i] == OCUPIED);
          return true;
      }
      return false;
    }
  }
```

2. 对于 Customer.java

```
import java.util.*;

public class Customer implements Runnable
{
  private BarberShop barberShop;
  private int customer;
  private int HAIRCUT_TIME = 5; // 剪发花费时间（等待时间）
```

```
8
9     public Customer(int customer, BarberShop bShop)
10    {
11      this.customer = customer;
12      this.barberShop = bShop;
13    }
14
15    public void run()
16    {
17      int sleepTime = (int =) (this.HAIRCUT_TIME * Math.random());
18      System.out.println("Customer " + this.customer + "enter barber shop");
19      int tmp = this.barberShop.getHairCut(this.customer); // 尝试剪发
20
21      if (tmp == BarberShop.WAITED) // 如果阻塞
22      {
23        System.out.println("Customer " + this.customer + "waiting for haricut");;
24      }
25      else if (tmp == BarberShop.FULL) // 店满了
26      {
27        System.out.println("Customer " + this.customer + "cannot get haircut");
28        return ;
29      }
30      else // 准备就绪，可以剪发
31      {
32        System.out.println("Customer " + this.customer + "start to haircut");
33        SleepUtilities.nap(); // 剪发花费时间
34        System.out.println("Customer " + this.customer + "leave barber shop");
35        this.barberShop.leaveBarberShop(this.customer);
36      }
37    }
38  }
```

3. 关于 sleepUtilities.java

```
1   public class SleepUtilities
2   {
3     private static final int NAP_TIME = 5;
4
5     public static void nap()
6     {
7       nap(NAP_TIME);
8     }
9
10    public static void nap(int duration)
11    {
12      int sleepTime = (int) (NAP_TIME * Math.random());
13      try
14      {
15        Thread.sleep(sleepTime * 1000);
16      }
17      catch (InterruptedException err)
18      {}
19    }
20  }```
21
22  4. 创建测试
23
24  ```java
```

```
public class CreateBarberShopTest implements Runnable
{
  static final private int WAIT_TIME = 3;
  static public void main(String [] args)
  {
    new Thread(new CreateBarberShopTest()).start();
  }

  public void run()
  {
    BarberShop newShop = new BarberShop(15);
    int customer = 1;

    while (customer <= 10000)
    {
      new Thread(new Customer(customer, newShop)).start();
      customer++;
    }
  }
}
```

6.13

```
monitor bounded_buffer
{
  int items[MAX_ITEMS];
  int numItems = 0;
  condition full, empty;

  void produce(int v)
  {
    while (numItems == MAX_ITEMS)
      full.wait();
    items[numItems++] = v;
    empty.signal();
  }

  int consume()
  {
    int res;
    while (numItems == 0)
      empty.wait();
    res = items[--numItems];
    full.signal();
    return res;
  }
}
```

7.15

```
semaphore ok to cross = 1;
void enter bridge()
{
  ok to cross.wait();
}

void exit bridge()
{
  ok to cross.signal();
}
```