Mo Tu We Th Fr Sa Su

Memo No.

Date / /

E03 并发进程

一、1. D

2. B

3. D

4. B

5. B

6. C

7. D

8. B

9. D

10. B

11. C

12. D

13. ① 一次只能允许一个进程使用的资源

② 执行操作临界资源的程序段

14. 停止访问

15. ① P ② V

16. [1-m, 1]

17. S.mutex < 0

18. 互斥

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

19. ①P操作 ②V操作

20. 一个

21. 同步

22. ①P操作 ②V操作

23. PV操作是指用于实现进程间通信和同步的一种机制，P操作也称等待操作，用于申请资源或进入临界区，如果资源不可用进程将被阻塞，直至进程可用；V操作也称释放操作，用于释放资源或退出临界区，当资源释放后，其他等待的资源可以获取资源并继续执行。

S.value > 0 说明资源可用，进程可直接使用资源，

S.value = 0 说明资源恰好用完且无进程在等待.

S.value < 0 说明资源用完且有进程在等待.

24. int waiting = 0;

semaphore mutex.value = 1

semaphore customer customer.value = 1

semaphore operator operator.value = 1

CHAIRS = 5

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

```
main( )
{
   cobegin
   {
      operator( );
      customer( );
   }
}
void operator( )
{
      while (TRUE){
         P(customer);
         P(mutex);
         waiting = waiting-1;
         V(operator);
         V(mutex);
         print( );
      }
}
```

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

```
void customer( )
{
    P(mutex);
    if(waiting < CHAIRS){
      waiting = waiting+1;
      V(customer);
      V(mutex) ;
      P(operator);
      get_print( );
    }
   elee {
        V(mutex);
    }
}
```

25. C
26. AB
27. A
28. D
29. C.
30. D
31. B
32. B
33. ①安全状态、②不安全状态.
34. 请求和保持
35. 避免 解除 预防 解除

36.

(1). 利用 safety() 算法寻找可能的进程执行队列.

~~首先 P2 可执行, Allocation 变为~~

首先计算各进程 Need 数组

P1 <0,0,0,0>

P2 <0,7,5,0>

P3 <6,6,2,2>

P4 <2,0,0,2>

P5 <0,3,2,0>

首先执行 P1, Allocation 变为 <2,1,1,2>

之后执行 P4, Allocation 变为 <4,4,6,6>

之后执行 P5, Allocation 变为 <4,7,9,8>

之后执行 P2, Allocation 变为 ~~<4,7,12,12>~~ <6,7,9,8>

最后执行 P3,

存在 <P1, P4, P5, P2, P3> 所以初始状态安全.

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

<2> 首先尝试分配.

Allocation变为<2,0,0,0>

Need数组

P_1 <0,0,0,0>

P_2 <0,7,5,0>

P_3 <6,5,2,2>

P_4 <2,0,0,2>

P_5 <0,3,2,0>

首先执行 P_1 ⟶ <2,0,1,2>

执行 P_4 ⟶ <4,3,6,6>

执行 P_5 ⟶ <4,6,9,8>

~~执行 P_3~~

此时, P_3, P_2 均无法执行

所以系统不能选择分配资源..