



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. \_\_\_\_\_

Date      /      /

## E03 并发进程

一、1. D

2. B

3. ~~D~~ A P, V进程是两条低级进程通信原语

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 等待

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. ① P 操作      ② V 操作

20. 一个

21. 同步

22. ① P 操作      ② V 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;` 等待顾客数量

`semaphore mutex.value = 1` 对 `waiting` 的互斥访问

`semaphore customer` `customer.value = 1` 等待复印的顾客

`semaphore operator` `operator.value = 1` 等待顾客的操作员

`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( );
    }
}
```



扫描全能王 创建



Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. \_\_\_\_\_  
Date        /        /

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

25. C

26. AB

27. A

28. D

29. C.

30. B.

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



扫描全能王 创建





Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Memo No. \_\_\_\_\_

Date     /     /

36.

(1) 利用 safety (1) 算法寻找可能的进程执行队列.

首先  $P_2$  可执行, Allocation 变为

首先计算各进程 Need 数组

$$P_1 < 0, 0, 0, 0 >$$
$$P_2 < 0, 7, 5, 0 >$$
$$P_3 < 6, 6, 2, 2 >$$
$$P_4 < 2, 0, 0, 2 >$$
$$P_5 < 0, 3, 2, 0 >$$

首先执行  $P_1$ , Allocation 变为  $< 2, 1, 1, 2 >$

之后执行  $P_4$ , Allocation 变为  $< 4, 4, 6, 6 >$

之后执行  $P_5$ , Allocation 变为  $< 4, 7, 9, 8 >$

之后执行  $P_2$ , Allocation 变为  $< 4, 7, 12, 12 >$   $< 6, 7, 9, 8 >$

最后执行  $P_3$ ,

存在  $< P_1, P_4, P_5, P_2, P_3 >$  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  $\langle 2, 0, 0, 0 \rangle$

Need 数组

$P_1 \langle 0, 0, 0, 0 \rangle$

$P_2 \langle 0, 7, 5, 0 \rangle$

$P_3 \langle 6, 5, 2, 2 \rangle$

$P_4 \langle 2, 0, 0, 2 \rangle$

$P_5 \langle 0, 3, 2, 0 \rangle$

首先执行  $P_1 \longrightarrow \langle 2, 0, 1, 2 \rangle$

执行  $P_4 \longrightarrow \langle 4, 3, 6, 6 \rangle$

执行  $P_5 \longrightarrow \langle 4, 6, 9, 8 \rangle$

~~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..

