1. 从就绪到阻塞不太可能实现，因为就绪状态的进程不会遇到要求进程阻塞的中断；从阻塞到运行可能实现，即阻塞状态结束后的进程刚好遇上CPU空闲的情况，阻塞结束后就进入运行状态。
2. 要实现进程切换，只要用一个寄存器保存当前进程在进程表中的首地址；需要切换进程时，只要将当前的机器状态存入对应的进程表项，使存放进程地址的寄存器指向新的进程。
3. 大部分高级程序设计语言要实现平台通用性，就不会涉及硬件的编程，但中断处理程序要涉及计算机中断的开启与禁用，以及中断服务程序的跳转与执行等，高级语言就无法实现了。

5、 顺序执行需要时间：

并行执行耗时：

7、 不会。单线程进程创建新进程需要该进程在运行态，才能执行fork()命令。

10、 线程执行过程需要使用寄存器和保存数值到寄存器中，当一个线程没有执行结束就要被切换时，寄存器保存了该线程的中间计算结果，因此需要利用堆栈对这些值进行保护。

11、 在有些场合需要让线程进行切换，使用得当可以提高一个程序的执行效率。

12、 内核级线程可以被时间中断抢占，用户级线程无法被中断抢占。

14、 优点：线程切换时，保存线程状态的过程和调度程序都是本地过程，所以启动他们比进行内核调用效率高，而且不需要陷入内核，不需要上下文切换，不需要对内存高速缓存进行刷新，使线程调度十分快捷；

允许每个进程有自己定制的线程调度算法，而且用户级线程具有比较好的可扩展性。

缺点：难以实现阻塞系统调用，一个线程进行系统调用会停止所有的线程；由于没有时钟中断，无法实现轮转调度，一个线程开始运行后其他线程就得等到它自动放弃CPU才能运行。

30、 消息传递。顾客点的菜单、饭菜、食品袋作为消息进行传递，每个雇员可以看作执行各自的进程，传递物品的过程就像UNIX通过管道进行消息传递一样。

35、 (a)和(b)执行完所有程序的时间为N(S+T), 从而CPU的效率都是。

(c)中要完成一个进程需要进行T/Q次进程切换，完成N个线程的总时间是

，所以CPU效率为。

(d)则是(c)在S=Q时的情况，此时CPU效率为50%。

(e)是(c)在Q=0的情况，CPU效率为0，因为CPU一直在进行进程切换而不执行计算任务。

36、 采用最短作业有限的调度程序：

|  |  |
| --- | --- |
| 0<X<3 | X, 3, 5, 6, 9 |
| 3<X<5 | 3, X, 5, 6, 9 |
| 5<X<6 | 3, 5, X, 6, 9 |
| 6<X<9 | 3, 5, 6, X, 9 |
| X>9 | 3, 5, 6, 9, X |

37、 (a)：进程的执行时间与完成顺序如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 时间min | 剩余时间 | | | | |
| A(min) | B(min) | C(min) | D(min) | E(min) |
| 0-10 | 8 | 4 | 0 | 2 | 6 |
| 10-18 | 6 | 2 | 0 | 0 | 4 |
| 18-24 | 4 | 0 | 0 | 0 | 2 |
| 24-28 | 2 | 0 | 0 | 0 | 0 |
| 28-30 | 0 | 0 | 0 | 0 | 0 |
| 完成时间min | 30 | 24 | 10 | 18 | 28 |

平均周转时间为(30+24+10+18+28)/5=22min.

(b) 按照B、E、A、C 、D依次执行，进程的执行时间与完成顺序如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 时间min | 剩余时间 | | | | |
| A(min) | B(min) | C(min) | D(min) | E(min) |
| 0-6 | 10 | 0 | 2 | 4 | 8 |
| 6-14 | 10 | 0 | 2 | 4 | 0 |
| 14-24 | 0 | 0 | 2 | 4 | 0 |
| 24-26 | 0 | 0 | 0 | 4 | 0 |
| 26-30 | 0 | 0 | 0 | 0 | 0 |
| 完成时间min | 24 | 6 | 26 | 30 | 14 |

平均周转时间为(24+6+26+30+14)/5=20min

(c) 按照A、B、C、D、E依次执行，进程的执行时间与完成顺序如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 时间min | 剩余时间 | | | | |
| A(min) | B(min) | C(min) | D(min) | E(min) |
| 0-10 | 0 | 6 | 2 | 4 | 8 |
| 10-16 | 0 | 0 | 2 | 4 | 8 |
| 16-18 | 0 | 0 | 0 | 4 | 8 |
| 18-22 | 0 | 0 | 0 | 0 | 8 |
| 22-30 | 0 | 0 | 0 | 0 | 0 |
| 完成时间min | 10 | 16 | 18 | 22 | 30 |

平均周转时间为(10+16+18+22+30)/5=19.2min

(d)按照C、D、B、E、A依次执行，进程的执行时间与完成顺序如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 时间min | 剩余时间 | | | | |
| A(min) | B(min) | C(min) | D(min) | E(min) |
| 0-2 | 10 | 6 | 0 | 4 | 8 |
| 2-6 | 10 | 6 | 2 | 0 | 8 |
| 6-12 | 10 | 0 | 0 | 0 | 8 |
| 12-20 | 10 | 0 | 0 | 0 | 0 |
| 20-30 | 0 | 0 | 0 | 0 | 0 |
| 完成时间min | 30 | 12 | 2 | 6 | 20 |

平均周转时间为(30+12+2+6+20)/5=14.0min

40、 T0 = 40;

0.5xT0+0.5xT1=30;

0.25xT0+0.25xT1+0.5T2=35;

0.125xT0+0.125xT1+0.25T2+0.5T3=25。所以下一次的预测时间为25分钟。

41、 解方程：

得。

43、 机制由操作系统确定，比如严格使用优先级调度来执行进程；策略由用户自行确定，比如用户根据实际需要分配各个线程的优先级。

45、 在take\_forks函数中设置state为HUNGRY，随即又在test函数检查state是否为HUNGRY，这样看来貌似没有意义，但是如果第一次调用test函数无法获得叉子，会在最后阻塞，此时哲学家的状态一直是HUNGRY。这样就使得别的哲学家在put\_forks函数中可以通过test函数决定是否需要唤醒旁边的哲学家进程。

46、 这样的话两次调用test都是没有意义的，因为执行put\_forks函数的哲学家在调用test函数是还是EATING状态，所以他左右两边的哲学家的左边或右边都至少有一个哲学家处于EATING状态（即调用put\_forks函数的哲学家）。最终的后果是，如果某个哲学家在put\_forks函数中被阻塞，他将一直被阻塞。