(a) 算法是这样的

- 按完成时间不递减的顺序排列所有工序
- 而某些进程仍未被覆盖：在第一个未覆盖进程的结束时间插入状态检查

首先，在覆盖所有进程之前，算法不会停止；在每次迭代中，至少有一个未覆盖的进程被覆盖，因此算法会终止。因此，它确实能计算出一套有效的状态检查，覆盖所有敏感进程。

其次，我们将通过一个 "保持领先 "类型的论证来证明，该算法计算出的状态检查集合具有尽可能小的规模。 我们把上述算法计算出的状态检查集合称为 。我们将通过归纳法证明，在"...... "到"...... "的 #ʰ 状态检查区间内，"...... "也至少会有 # 次状态检查。

- 基本情况：'必须在不晚于第一次状态检查的情况下进行状态检查，因为敏感进程会在此时结束。
- 归纳步骤：假设在ʰ 的 # 个状态检查之前，'中至少有 # 个状态检查。我们已经知道，根据我们的算法，导致插入 (#1 I)-th 状态检查 in 的过程没有被前面的所有 # 个状态检查覆盖，所以 ' 必须在 #ʰ 和 (#1 I)" 状态检查 in 之间放入一个状态检查来覆盖该过程。因此，在 (#1 I)" 状态检查之前，'至少有 # 1 次状态检查。

因此，我们知道算法是正确的。

对所有进程的开始和结束时间进行排序的运行时间为 fi(n log n)，然后插入所有状态检查的运行时间为 fi(n)。为了在线性时间内完成这项工作，我们需要保留一个数组，记录迄今为止已覆盖的进程，以及一个队列，记录自上次状态检查以来我们看到的所有进程的 开始时间。当我们第一次看到某个未覆盖进程的结束时间时，我们就会插入一次状态检查，并将当前队列中的所有进程标记为已覆盖。这样，在这部分算法的运行过程中，每个进程的工作时间都是不变的。

(b) 这种说法是正确的。

让我们用 来表示我们的算法所提供的解。由于 #* 是一组没有两个进程同 时 运行的敏感进程的*最大*大小，因此只要找到 | | 这样的 "不相交 "进程就能证明 | | # *。我们在（a）部分的算法实际上提供了这样一组不相交的进 程： 其完成时间导致插入状态检查的进程是不相交的

，因为每个进程都没有被前面所有的状态检查覆盖。

!ex559.1T6.225