Sammanfattning EDAF05

Meris Bahti & Felix Mul

16 oktober 2013

1 Deadlock Analysis

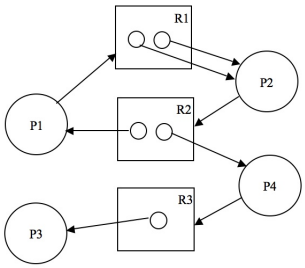
Resource allocation graphs are used to determine if a program can deadlock. For a program to end up in a deadlock there are a few requirements.

- Mutual exclusion: at least one resource is held in a non-shareable mode.
- Hold and wait: there must exist a process that is holding at least one resource and simultaneously waiting for resources that are held by other processes.
- No preemption: resources cannot be preempted; the resource can only be released voluntarily by the resource holding it.
- Circular wait: There must exist a set of processes waiting for each other in a circular structure. I.e: p1 waits for p2, p2 waits for p3, p3 waits for p1.

To draw a resource allocation graph from source code:

1. Draw boxes for each resource.
2. For each thread (i) and line (j), draw a bubble with T_{ij}. If a thread takes, then draw a line to the resource. For $T_{i(j+1)}$ draw a line from the resource to the thread.
3. If T_{ij} only emits or only absorbs arrows, you don't have to keep it in the graph.
4. For resources that exist as multiple instances, draw dots inside the resource. If a cycle exists containing a multiple instance resource, then it may be a false cycle.

Cycles in the graph indicate the possibility of deadlocks.



2 Komplexitet