Operating Systems

Tutorial 2 & 16

Michael Tänzer

os-tut@nhng.de
http://os-tut.nhng.de

Calendar Week 50

Outline

True or False

- Progress means that a process in it's critical section will eventually leave it
- On a single processor system no special atomic instructions are needed interrupts enable/disable is enough.
- Spinlocks are always useless as they're doing nothing useful while waiting they should be replaced by blocking locks or semaphores.

Vertices

- Processes – represented by circles
- Resource types – represented by rectangles
- Multiple instances of a resource – a dot per instance inside the resource type vertex

Edges

Request Edge
- Directed edge from a process to a resource vertex
- Indicates that the process wants to allocate a resource of that type

Assignment Edge
- Directed edge from a specific instance of a resource to a process vertex
- Indicates that the resource is assigned to that process

Does a cycle in a RAG always mean the a deadlock occurred?

Only if it involves resource types which only have a single instance

What is a wait-for graph (WFG)?

- A variant of a RAG
- Without resource vertices
- Edge from process P_i to P_j indicates that P_i is waiting for a resource held by P_j

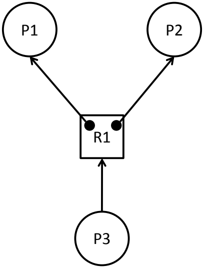
RAG $\rightarrow$ WFG

1. Remove resource vertices
2. Draw an edge from P_i to P_j if there existed an edge from P_i to R and from R to P_j for a resource R in the RAG

RAG $\leftarrow$ WFG

Not possible as the information which process is allocated/waiting for what resource is not present in a WFG.

Can a WFG be drawn for a RAG with multiple instances per resource type?



- Should the edge be drawn from P_3 to P_1 or to P_2 ot to both?

$\Rightarrow$ Not possible

What can a WFG be used for?

- Deadlock detection
- Circle in WFG $\Rightarrow$ deadlock occured

Explain the necessary conditions for deadlocks

Give an example for how to prevent each of them

Mutual Exclusion	Resources can’t be shared between processes (spooling)
Hold and Wait	A process already holding a ressource can wait to acquire another one (allocate resources atomically)
No Preemtion	Resources can’t be taken away from a process by force (save/load state)
Circular Wait	The WFG has a circle (order resources)

What is a safe state?

- All processes can run to completion
- Even if each process will request the maximum number of resources
- Processes which can't be granted their request have to wait for others to terminate

Task

(a) Allocation

	R_1	R_2	R_3	R_4
P_1	0	0	1	2
P_2	1	0	0	0
P_3	1	3	5	4
P_4	0	6	3	2
P_5	0	0	1	4

(b) Max

	R_1	R_2	R_3	R_4
P_1	0	0	1	2
P_2	1	7	5	0
P_3	2	3	5	6
P_4	0	6	5	2
P_5	0	6	5	6

(c) Available

R_1	R_2	R_3	R_4
1	5	2	0

- What is the content of the matrix ‘Need’?
- Is the system in a safe state?
- If P_2 requests $(0, 4, 2, 0)$ should it be granted?

Code vulnerable to race conditions or deadlocks?

```
Spinlock s1, s2, s3 = FREE; int counter = 0;

Thread1(){
        if (counter == 0){
                lock(s1);
                counter++;
                unlock(s1);
        }
        lock(s2); lock(s3);
        /* update some more data */
        unlock(s3); unlock(s2);
}

Thread2(){
        lock(s3);
        counter++; /* update some data */
        if (counter == 2){
                lock(s2); /* update some more data */
                unlock(s2);
        }
        lock(s1); /* update even more data */
        unlock(s3);
        unlock(s1);
}
```

A Quick Survey

Write on an anonymous piece of paper:

- At least one thing you liked
- At least one thing that could be improved about the tutorial

The End

The End