Operating Systems 2016/17
Solutions for Assignment 8

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

T-Question 8.1: Kernel Synchronization

a. On a multi-processor system, when must local interrupts be disabled for kernel spinlocks? **1 T-pt**

Solution:
Local interrupts must be disabled if the code executed due to interrupts (e.g., the interrupt handler) accesses the same shared data.

b. Why is disabling interrupts a privileged instruction? **1 T-pt**

Solution:
Preemptive scheduling is realized by using the timer interrupt to involuntarily enter the kernel while a user-space process is executing. If a process in user-space could disable interrupts, it could gain infinite CPU time, thereby taking over the system.

Furthermore, interrupts are the common way to signal the completion of I/O operations. Without interrupts the OS misses these completions and potentially waits indefinitely for devices to respond. If interrupts are disabled for too long, input may also be lost (e.g., network packets).

T-Question 8.2: Deadlocks

a. Enumerate and explain the 4 necessary conditions for a deadlock. **2 T-pt**

Solution:

Mutual exclusion (aka exclusiveness) *Resources cannot be shared between processes.*

Hold and wait *A process already holding a resource can wait to acquire more resources.*

No preemption *Resources cannot be taken away forcibly from processes.*

Circular wait *There exists a set of processes $\{P_0, \dots P_n\}$ where P_0 is waiting for a resource held by P_1, P_1 is waiting for a resource held by P_2, ..., P_n is waiting for a resource held by P_0.*

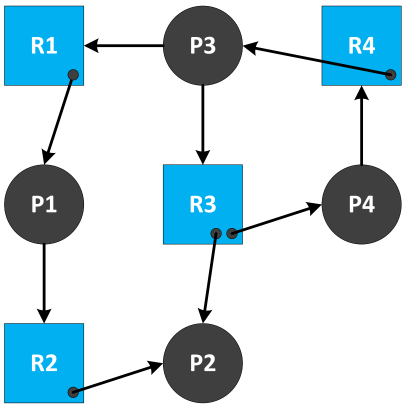
Note that these four conditions are not truly independent, as circular wait implies hold and wait.

b. How can periodic process snapshots be used to recover from deadlocks? What is a major disadvantage of this method? **2 T-pt**

Solution:
Whenever a deadlock has been detected, a recovery mechanism can select one of the processes involved in the deadlock and abort it. This way the deadlock can be solved. The process is later restarted from the last snapshot.

This mechanism is very expensive, because periodic snapshot creation is necessary. To take a snapshot the process must be paused for a short period of time and its complete state must be saved (entire virtual address space, PCB, TCBs, etc.). A major problem here is the interaction of the process with other processes and the system. The process could for example be a database application, which regularly works with database files on disk. Restoring only the process state and not the state of the database files on disk, probably leads to corruption. A snapshot-based recovery mechanism thus will most certainly require support from the application, not only complicating the operating system but also making the life of application developers harder.

T-Question 8.3: Resource Allocation Graph



a. Describe the situation depicted in the resource allocation graph. **2 T-pt**

Solution:
We have 4 resources $\{R1, R2, R3(2), R4\}$ *and 4 processes* $\{P1, P2, P3, P4\}$. $R3$ *provides two instances, all other resources one. We define the following triple:*

$< processP, \{resourcesPowns\}, \{resourcesPwants\} >$.

We can then describe the situation with:
$< P1, \{R1\}, \{R2\} >$
$< P2, \{R2, R3_1\}, \{\} >$
$< P3, \{R4\}, \{R1, R3\} >$
$< P4, \{R3_2\}, \{R4\} >$

b. Has a deadlock occurred in the above situation? Why, or why not? **1 T-pt**

Solution:
No deadlock occurred, although there is a cycle in the graph ($P3 \rightarrow R3 \rightarrow P4 \rightarrow R4 \rightarrow P3$). $P2$ *has all resources to finish execution. It will thus eventually release* $R2$ *and* $R3_1$. $P1$ *can then acquire* $R2$, *run to completion and release* $R1$ *and* $R2$. *Then* $P3$ *can get* $R1$ *and* $R3$, *and finally releases* $R4$. $P4$ *can now run.*

c. What changes if $P1$ also requests $R3$? **1 T-pt**

Solution:
It depends on the scheduling order between $P3$ *and* $P1$. *If* $R3_1$ *is released by* $P2$ *and assigned to* $P1$, *all processes can run to completion. Nothing changes. However, if the scheduling will prefer* $P3$ *and grant it the access to* $R3_1$, *we end in a deadlock with a cycle* $R1 \rightarrow P1 \rightarrow R3_1 \rightarrow P3 \rightarrow R1$.

Total: 10 T-pt