# Threads: Probleme des Multithreading

# Safety Hazards - Race Condition

Während der Verwendung mehrerer Threads kann es vorkommen, dass die End- bzw. Zwischenergebnisse des Programms nicht dem Erwartungswert entsprechen. Gewöhnlicherweise liegt es an mangelhafter oder gar nicht vorhandener synchronisierung eines kritischen Bereiches. Da parallel laufende Threads gleichzeitig den selben Wert auslesen, verändern und erst anschließend abspeichern (könnten), führt das nicht selten zum Informationsverlust und verfälschtem Ergebnis.

Es ist daher klug, kritische Bereche des Programms durch Monitore zu schützen. So erhält nur ein Thread den Zugriff auf einen kritischen Bereich, während andere Threads kurzzeitig warten – was wiederum zu einem georneten und sauberen Programmablauf führt.

# Liveness Hazards - Deadlock

Parallel laufende Threads sind nicht selten voneinander anhängig und darauf ausgelegt miteinander zu kommunizieren. Im einfachsten Fall wartet ein Thread auf die Antwort seines Thread-Freundes, um anschließend seine ausgelegte Aufgabe fortzusetzen. Eine Deadlock- bzw. Pattsituation entsteht dann, wenn beide Threads gegenseitig auf die Antwort ihres Thread-Freundes warten müsen, während das Warten erst dann aufhören kann, wenn die Antwort des Freundes ankommt. So machen beide Threads keine Fortschritte mehr und das Programm kann nicht mehr terminieren.

Es ist also die Aufgabe des Programmierers sicherzustellen, dass die Kommunikation der Threads diese nicht an der Ausführung ihrer Aufgabe hindern kann.

# Liveness Hazards – Livelock

Parallel laufende Threads sind nicht selten voneinander anhängig und darauf ausgelegt miteinander zu kommunizieren. Nicht selten ist der weitere Arbeitsverlauf eines Threads abhängig von der Antwort seines Thread-Freundes. Ein Livelock entsteht dann, wenn der weitere Arbeitsverlauf beider Threads jeweils von von der Antwort des anderen abhängt. Das wiederum kann dazu führen, dass die Threads nicht mehr aufhören zu kommunizieren, währen sie ihre Arbeitsverläufe gegenseitig verändern. Obwohl aus verschiedenen Gründen, führen sowohl Dead- als auch Livelock zum selben Resultat - das Programm macht keine Fortschritte und kann nicht mehr terminieren.

# Liveness Hazards – Starvation

Starvation oder Aushungern passiert dann, wenn ein Thread durch das Konzept des Monitorings keinen Zugriff auf essenzielle Recourcen erhalten kann, da andere gleich/ höher prioritisierte Threads den Zugriff für eine sehr lange Zeit für sich beanspruchen. Diese benachteiligten Threads werden eine sehr lange Zeit ausgeschlossen und können ihre Aufgabe nicht mehr fortsetzen.

# Performance Hazards – Bad Performance

Während das Konzept der parallelen Arbeitsverteilung ihre Vorteile hat, so ist sie ebenfalls systemlastig. Da Threads einen Teil des Arbeitsspeichers für sich beanspruchen und für ordentliche Ergebnisse mit dem Konzept des Monitors verwendet werden sollten, kann dies zu einer Situation führen bei welcher zu viele aktive Threads sich gegenseitig am raschen Erledigen ihrer Aufgabe hindern - wohingegen weniger Threads die selbe Aufgabe in einer viel kürzeren Zeit erledigen könnten.