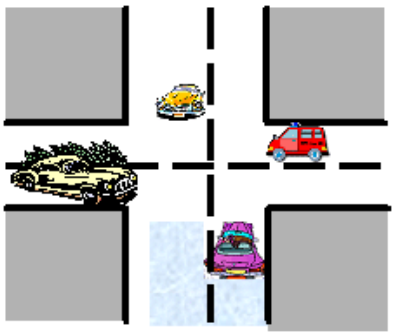
Kapitel 8: Verklemmungen

- Motivation und Einführung
- Betriebsmittelverwaltung (Resource Management)
- Verklemmungsbedingungen) (Deadlock Conditions)
- Strategien gegen Verklemmungen (Policies against Deadlocks)

8.1 Motivation und Einführung

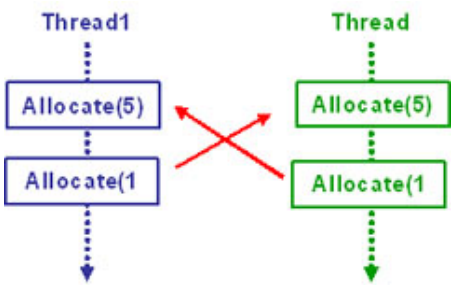
Insbesondere beim Wettbewerb um exklusive Ressourcen aber auch bei wechselseitigem Ausschluss an kritischen Abschnitten kann es bei ungünstiger Programmierung zu Wettbewerbssituationen kommen, die zu einer Verklemmung (*deadlock*) führen können. Im täglichen Leben begegnen uns immer wieder Situationen, die zu einer Verklemmung führen können, z.B. an einer Kreuzung ohne Vorfahrtsberechtigung.



Würde jeder der beteiligten Autofahrer auf seinem "Recht" bestehen, dann würde i.d.T. obige Situation auf Dauer verklemmt sein. Andere Beispiele für Verklemmungen sind:

```
semaphore s1 = 1, s2 = 1;
Thread T1 {                          Thread T2 {
...                                  ...
p(s1); /* outer CS */                p(s2); /* outer CS */
  p(s2); /* inner CS */                p(s1); /* inner CS */
  ...                                  ...
  ...                                  ...
  ...                                  ...
  v(s2);                               v(s1);
v(s1);                               v(s2);
}                                    }
```

Wie man dem obigen Beispiel entnehmen kann, tritt eine Verklemmung an den beiden Semaphoren auf, weil die beiden Threads die geschachtelten kritischen Abschnitte nicht in der gleichen Reihenfolge durchlaufen, so dass sie dann wechselseitig an ihren jeweils inneren kritischen Abschnitten hängen bleiben. Wie man am folgenden Beispiel sieht, können Threads auch an einem einzigen Betriebsmitteltyp (hier Speicherkacheln) in eine Verklemmung geraten, wobei wir annehmen wollen, dass im Folgenden nur 10 Kacheln zur Verfügung stehen.

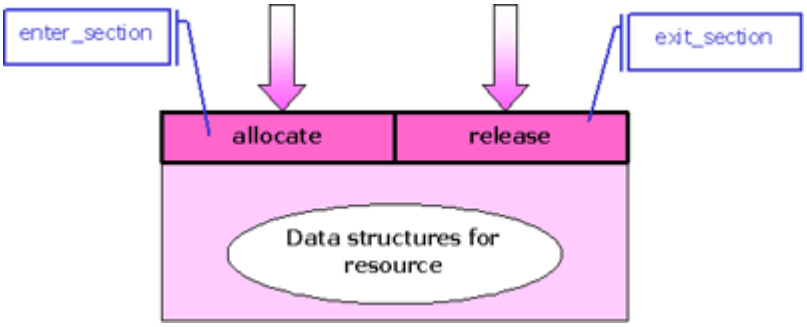


Man beachte insbesondere, dass es bei den Applikationen, die durch fehlerhafte Programmierung zu Verklemmungen führen, nicht notwendigerweise zu einer Verklemmung kommen muss. Im letzten Beispiel könnte Thread 1 solange rechnen, dass er beide Speicheranforderungen durchführt, bevor Thread 2 zu seiner ersten Speicheranforderung kommt. In diesem Fall müsste Thread 2 solange warten, bis Thread1 zumindest wieder eine Kachel freigibt.

Hinweis: In der Literatur wird sehr häufig der Begriff Verklemmung bzw. Deadlock bereits für Situationen verwendet, in denen nur ein Prozess oder ein KLT unbegrenzt warten muss. Solange nicht mindestens zwei KLTs oder zwei Prozesse beteiligt sind, wollen wir noch nicht von einer Verklemmung sprechen, ein quasi unbegrenzt wartender Prozess ist am Verhungern oder im Livelock.

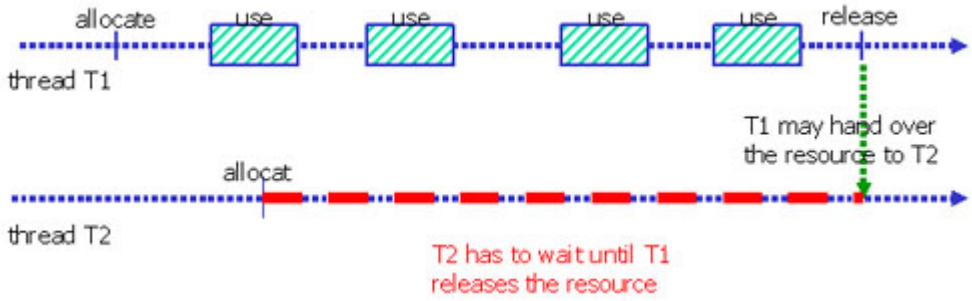
8.2 Ressourcenmanagement

Der Wettbewerb um eine exklusive Ressource gleicht dem Wettbewerb um einen kritischen Abschnitt, ja man kann sagen, dass die Nutzung einer exklusiven Ressource ein besonders kritischer Abschnitt ist.



Threads (Prozesse), die sich um eine exklusive Ressource mittels `allocate()` vergeblich bemühen, werden in den Zustand blockiert überführt, wobei eine Warteschlange in der Verwaltung dieser Ressource den wartenden Thread (Prozess) aufnimmt.

Nutzungsprotokoll eines nur exklusiv zu nutzenden Betriebsmittels:



Formale Definition einer Verklemmung (*deadlock*):

Eine Menge von Threads (Prozessen) heißt verklemmt, wenn jeder Thread (Prozess) dieser Menge auf ein Ereignis "im Zustand blockiert" wartet, das nur durch einen anderen Thread (Prozess) dieser Menge ausgelöst werden kann. Im Prinzip warten dann diese Threads ewig, da ja keiner dieser Threads jemals wieder auf den Prozessor zugeordnet wird, denn jeder dieser Threads ist ja blockiert.

8.3 Notwendige Verklemmungsbedingungen

Während Verklemmungssituationen innerhalb einer multi-threaded Anwendung ihre Ursache in einer falschen Programmierung derselben haben und sich nur bedingt auf das Restsystem auswirken, sind

Verklemmungssituationen zwischen unabhängigen Prozessen auf eine wenig durchdachte Ressourcenverwaltung im System zurückzuführen, die sehr wohl Auswirkungen auf das Gesamtsystem haben kann.

Im ersten Fall kann man diese Verklemmungssituation aus Anwendersicht relativ einfach mit Hilfe eines Timeouts erkennen. Diese Verklemmungssituation ist u.U. sogar reproduzierbar, während im zweiten Fall eine Verklemmung mal auftritt, mal auch wieder nicht, je nachdem welche sonstigen Anwendungen, die jede für sich Ressourcen benötigt, gleichzeitig im System abgearbeitet werden. Folgende Bedingungen müssen erfüllt sein, damit es zu einer Verklemmung kommen kann:

1. Exklusivität
2. Halten von Ressourcen beim Warten auf weitere Ressourcen
3. Keine Verdrängungsmöglichkeit von Ressourcen
4. Zirkuläre Wartebedingung

Man beachte, dass jede dieser Bedingungen notwendig ist für das Eintreten einer Verklemmung, aber keine von ihnen hinreichend. Mittels des so genannten "Resource Allocation Graphs" könnte man feststellen, ob er Zyklen enthält oder nicht. Im letzteren Fall würde auf keinen Fall eine Verklemmung vorliegen, im ersten Fall könnte eine Verklemmung vorliegen, muss aber nicht.

8.4 Strategien gegen Verklemmungen

1. Prävention
2. Vermeidung
3. Entdeckung und Beseitigung
4. Ignorierung
5. (Vogel Strauss Politik)

8.4.1 Prävention

Hierzu muss man versuchen, eine der drei notwendigen Bedingungen für das Eintreten von Verklemmungen zu verhindern. Die meisten vorgeschlagenen Modelle sind zu restriktiv, was die Ressourcennutzung angeht. Eine Restriktion hinsichtlich der Ordnung, in der bestimmte Ressourcen dynamisch angefordert werden dürfen, verspricht zumindest eine gewisse Akzeptanz bei den Anwenderprogrammierern. Ansonsten kann man mit Einführung von möglichst vielen virtuellen Ressourcen das Problem der exklusiven Ressourcennutzung weitgehend umgehen.

8.4.2 Vermeidung

Hierbei geht es darum, bei der Ressourcenvergabe so vorsichtig zu sein, dass man erst gar nicht in einen unsicheren Zustand des Systems kommt. Obwohl dieses Verfahren auch beliebige Ressourcenanforderungsfolgen unterstützt, erfreut es sich in der Praxis wenig Beliebtheit, da man hierbei pro Prozess dessen potentielle Gesamtforderungen vor dessen Start kennen muss.

Ein System ist dabei solange in einem sicheren Zustand, wie es zumindest noch eine terminierende serielle Ausführungsreihenfolge aller aktiven Prozesse gibt, selbst wenn jeder von diesen Prozessen noch seine Gesamtforderungen stellen sollte.

Es sollte klar sein, dass nicht jeder unsichere Systemzustand automatisch zu einer Verklemmung führen muss, da nicht jeder der beteiligten Prozesse noch seine Gesamtforderungen stellen muss, manche dieser Prozesse sind möglicherweise bereits in ihrer Terminierungsphase, in der sie Ressource für Ressource zurückgeben.

Mittels des Bankieralgorithmus kann man in $O(n^2*m)$ ermitteln, ob im Zuge einer aktuellen Ressourcenanforderung ein System noch sicher ist oder bereits unsicher werden würde. Im letzteren Fall

würde man trotz evtl. freier Ressourcen diese aktuelle Ressourcenanforderung zurückweisen und den Prozess zunächst auf "Eis legen".

Wie würden Sie in diesem Fall den anfordernden Prozess auf's Eis legen?

8.4.3 Entdeckung und Beseitigung

Mittels eines leicht geänderten Bankieralgorithmus kann man in einem System ohne Verklemmungsverhinderung bzw. -vermeidung zumindest eine Verklemmung entdecken. Es bleibt zu klären, wann der Entdeckungsalgorithmus gestartet werden sollte, da er ähnlich wie der Vermeidungsalgorithmus einen Aufwand von $O(n^2*m)$ besitzt.

Wäre dieser Algorithmus ein geeigneter Kandidat für den Leerlaufprozess (idle process)?
Könnte ein Überwachungsprozess periodisch gestartet werden, der diese Aufgabe übernimmt?
Wie groß müsste dann die Periode dieses Prozesses sein?

Die Frage erhebt sich nun, was tut man dann, nachdem eine Verklemmung entdeckt worden ist. Es mag viele vergleichsweise brauchbare Kriterien geben, nach denen man einen oder mehrere der verklemmten Prozesse in der Hoffnung abbricht, dass mit den zusätzlich verfügbaren Ressourcen die restlichen, ehemals verklemmten Prozesse terminieren können, ehe dann die abgebrochenen Prozesse neu gestartet werden. Kein gutes Kriterium wäre es, wenn dadurch ausgerechnet der Prozess abgebrochen würde, der nur noch am wenigsten zu rechnen hätte.

8.4.4 Ignorierung

Dieses "Verfahren" wird zwar in einigen Desktopbetriebssystemen verwendet, was aber eher einer Bankrotterklärung von Systembastlern gleichkommt.