olaf.neugebauer [☺] tu-dortmund.de
bjoern.boenninghoff [☺] tu-dortmund.de

Übung zur Vorlesung
Rechnerarchitektur
Sommersemester 2015

Übungsblatt 12

(9 Punkte)

Besprechung ab Montag, 6. Juli 2015

12.1 Barrieren (2 Punkte)

Der folgende Quelltext aus den Vorlesungsfolien implementiert eine Variante der Barrierensynchronisation.

```
class Barrier {
  int counter;
  Mutex mutex;
  boolean flag = false;

  void barrier( int p );
}
```

```
void barrier( int p ) {
  mutex.lock();
  if ( counter == 0 ) flag = false;
    int mycount = ++counter;
  mutex.unlock();
  if ( mycount == p ) {
    counter = 0;
    flag = true;
  } else
    while ( flag == false ) {};
}
```

Verändern Sie den Quelltext, gegebenenfalls durch Hinzufügen weiteren Pseudocodes, so dass die folgenden Ziele erreicht werden. Es ist eine Lösung gesucht, die folgende Forderungen erfüllt.

a) Ermöglichung der n-fachen / mehrfachen Nutzung der Barriere durch die teilnehmenden Threads.

b) Verhindern von Deadlocks, falls ein teilnehmender Thread vor Erreichen der Barriere terminiert (z. B. durch unbehandelte Ausnahme).

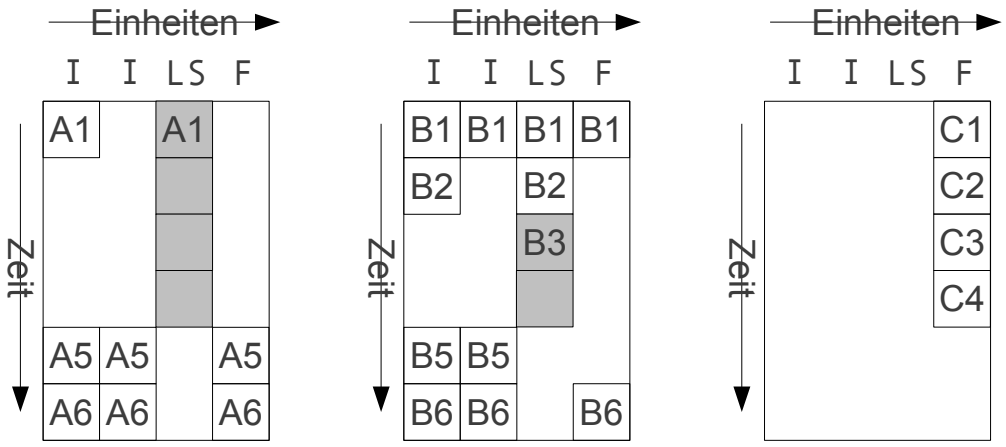
12.2 Multithreading (5 Punkte)

In dieser Aufgabe sollen Sie verschiedene Multithreading-Modelle exemplarisch anwenden. Gegeben sei dazu ein System mit

- einem Prozessor mit dynamischem Scheduling und
  - 2 Integer-Ausführungseinheiten (Latenz 1 Takt)
  - 1 Load/Store-Einheit (Latenz 1-4 Takte, je nach Cacheverhalten)
  - 1 Floating-Point-Einheit (Latenz 1 Takt)
- einem L1-Cache und einem L2-Cache. Die Latenzen der Zugriffe sind wie folgt:

L1-Zugriff	L2-Zugriff	Latenz (Takte)
Hit	-	1
Miss	Hit	2
Miss	Miss	4

Beide Caches unterstützen *Hit-Under-1-Miss*, d.h. während ein Miss bearbeitet wird, können weitere Hit-Zugriffe abgewickelt werden. Ein Zugriff, der einen weiteren Miss produzieren würde, wird abgebrochen und muss vom Prozessor wiederholt werden.



(a) Thread A (Priorität 0) (b) Thread B (Priorität 1) (c) Thread C (Priorität 2)

Abbildung 1: Ausführungsverlauf dreier Threads auf der gegebenen Plattform

Für dieses System sind die Ausführungsverläufe dreier Programme A-C in Abbildung 1 skizziert, wie sie sich bei isolierter Ausführung auf dem System ergeben. Operationen, die mehr als einen Takt benötigen, sind in hellgrau markiert. Außerdem ist die Priorität jedes Threads angegeben (nur bei CGMT und SMT nötig), wobei niedrigere Zahlen höherer Priorität entsprechen.

a) Zeichnen Sie für eine Ausführung mit *Coarse-Grained Multithreading (CGMT)*, *Fine-Grained Multithreading (FGMT)* und *Simultaneous Multithreading (SMT)* jeweils den zeitlichen Verlauf der Abarbeitung auf dem gegebenen System. Richten Sie sich bei der Notation nach Abbildung 1, d.h. tragen Sie die Verarbeitungseinheiten horizontal und die Zeit vertikal auf. Der *Context-Switch-Overhead* sei 1 Takt für CGMT und 0 Takte für die anderen Modelle.

   Sie können davon ausgehen, dass die Speicherzugriffe der Threads auf disjunkte Cache-Bereiche abgebildet werden, d.h. dass diese Zugriffe interferenzfrei sind. Außerdem sei der verwendete Scheduler ein ASAP-Scheduler (*As Soon As Possible*). Falls nicht alle Instruktionen aus einem Zeitschritt aus den Abbildungen 1(a) bis 1(c) gleichzeitig gestartet werden können, führt der Scheduler die verbleidenden so bald wie möglich danach aus. Nehmen Sie in diesem Fall außerdem an, dass erst *alle* Instruktionen eines Zeitschritts i aus Abbildung 1 ausgeführt worden sein müssen, bevor Instruktionen aus Schritt $i+1$ desselben Threads ausgeführt werden können.

b) Was sind die hardwareseitigen Voraussetzungen für die einzelnen Modelle?

12.3 Grobgranulares Multithreading (2 Punkte)

Für die Ausführungszeiten der Instruktionen aller Threads einer Anwendung gilt diese Verteilung:

- 40% der Taktzyklen entfallen auf die Ausführung der Instruktionen im Prozessor.
- 30% der Taktzyklen sind Warte-Zyklen, die durch L1-Cache-Misses (aber L2-Cache-Hits) entstehen. Die L1-Miss-Bearbeitung benötigt 10 Taktzyklen.
- 30% der Taktzyklen sind Warte-Zyklen, die durch L2-Cache-Misses hervorgerufen werden (jeweils 30 Taktzyklen).

Im Durchschnitt tritt pro Thread nach jeweils 10 Rechen-Takten ein Cache-Miss auf. Beantworten Sie folgende Fragen unter der Annahme, dass ausreichend viele Threads sowohl von der Hardware als auch durch die Anwendung unterstützt werden.

a) Wie hoch ist die maximale Auslastung, wenn ein Threadwechsel 5 Taktzyklen dauert?

b) Wie lange darf ein Threadwechsel höchstens dauern, so dass die Auslastung mindestens 50% beträgt?