Übungen zur Vorlesung
Betriebssysteme I
Studiengang Informatik
WS 2011/2012

AG Betriebssysteme FB3
Prof. Dr. Jan Peleska, Florian Lapschies

Abgabetermin: 15.12.2011

Übungszettel 3

1 Deadlock-Erkennung mit Wartegraphen

Bei der unachtsamen Verwendung von Semaphoren kann es schnell zu Deadlocks kommen. In der Vorlesung habt Ihr einen Algorithmus zur Erkennung von Deadlocks kennen gelernt, der mit einem sogenannten Wartegraph arbeitet.

Sei P die Menge der Prozesse und S die Menge der Semaphoren. Die Knoten des Wartegraphen sind aus der Menge P. Für jeden Prozess wird mit $ups : P \longrightarrow \mathbb{P}(S)$ eine Menge von Semaphoren vermerkt, auf die der Prozess in Zukunft ein Up machen möchte. Wenn nun ein Prozess $p_0 \in P$ ein Down auf eine Semaphore $s \in S$ mit einem Zählerwert von 0 machen möchte, muss p_0 so lange warten, bis einer der Prozesse $p_1 \in \{p \in P \mid s \in ups(p)\}$ diese Semaphore für ihn freigibt. Diese Warteabhängigkeit wird durch je eine mit s markierte Kante von p_0 hin zu p_1 in dem Graphen erfasst. Die Kanten sind daher aus der Menge $E = P \times S \times P$.

Zur Deadlock-Erkennung werden in dem Algorithmus die nicht-trivial starken Zusammenhangskomponenten untersucht.

Dieser Algorithmus soll nun in Form einer C++-Klasse implementiert werden, dessen Grundgerüst in Listing 1 vorgegeben ist.

Listing 1: DeadlockDetection Klasse

```
class DeadlockDetection {
public:
  typedef int PId;
  typedef int SemId;
  typedef ::std::set<SemId> SemSet;

  void update(PId p, const SemSet &sems);
  void wait(PId p, SemId sem);
  void wakeup(PId p);
  bool deadlock();
};
```

Die Klasse soll die folgenden Funktionen bereitsstellen:

- `update(p, sems)`: Mit dieser Methode wird dem Algorithmus mitgeteilt, dass ein Prozess p im weiteren Verlauf die Semaphoren in *sems* inkrementieren wird. Dabei kann die Menge `sems` auch leer sein.
- `wait(p, sem)`: Mit dieser Methode wird dem Algorithmus mitgeteilt, dass der Prozess p ein blockierendes Down auf die Semaphore *sem* ausgeführt hat und nun wartet. In unserem Szenario kann ein Prozess nur auf eine Semaphore zur Zeit warten.
- `wakeup(p)`: Mit dieser Methode wird dem Algorithmus mitgeteilt, dass der Prozess p nun nicht mehr blockiert ist.
- `deadlock()`: In dieser Methode sollen die starken Zusammenhangskomponenten des Wartegraphens aufuntersucht werden. Hierzu nutzt Ihr bitte den Algorithmus von Tarjan [WTFE11]. Die Methode gibt genau dann `true` zurück, wenn es einen Deadlock gibt.

Testet Eure Implementierung mit Unit-Tests.

Hinweise

- Alle in dieser Aufgabe geforderten Programme sollen mit einem Makefile übersetzt werden können. In diesem Makefile sollen Abhängigkeiten des Build-Prozesses richtig erfasst sein. Des weiteren soll das Makefile auch eine Regel `clean` enthalten, die alle Komplilate wieder löscht.
- Achtet darauf, dass Ihr allen Speicher, den Ihr mit `new` alloziert habt, auch zum Programmende wieder mit `delete` freigegeben habt. Bei der Behandlung von Fehlerfällen, bei denen das Programm mit einem Fehlercode beendet wird, darf darauf verzichtet werden. Hilfreich zum Aufdecken von Speicherlecks und Speicherzugriffsfehlern ist das Tool `valgrind`.
- Die Abgabe erfolgt als Ausdruck am Ende der Übung und zusätzlich elektronisch über das Subversion Repository.
- Die Dokumentation der Aufgabenlösung ist in LaTeX anzufertigen.

Literatur

[WTFE11] Wikipedia, The Free Encyclopedia. Tarjan's strongly connected components algorithm, 2011. `http://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm`, abgerufen am 30.11.2011.