Wilhelm-Schickard-Institut für Informatik

Arbeitsbereich Symbolisches Rechnen
Prof. Dr. Wolfgang Küchlin
Jens Haußmann, M. Sc.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

07.03.2018

Wintersemester 2017/2018

Praktikum Betriebssysteme - Tag 6

IPC – Deadlocks

Aufgabe 6.1 *(Bankieralgorithmus)*

In dieser Aufgabe werden Sie sich mit Deadlocks von Prozessen beschäftigen. Mit Hilfe des Bankieralgorithmus können Deadlocks vermieden werden, indem nach jedem Resource-Request einer der beteiligten Prozesse überprüft wird, ob der Request zu einem Deadlock führen könnte. Der Bankieralgorithmus benötigt dafür die folgenden Informationen:

- Verfügbare Resourcen (Vektor Available).
- Allokierten Resourcen für jeden Prozess (Matrix Current).
- Weitere benötigte Resourcen für jeden Prozess (Matrix Request).

Ausgehend von folgendem Szenario implementieren Sie nun in einem Source-File `deadlock.c` den Bankieralgorithmus so, wie Sie ihn aus der Vorlesung Betriebssysteme kennen:

- Es laufen aktuell 5 Prozesse, genannt P0 bis P4.
- Es gibt 3 verschiedene Resourcen, genannt A, B und C.
- Von A gibt es insgesamt 7 Einheiten, von B gibt es 2 und von C gibt es 6.
- Die allokierten bzw. noch benötigten Resourcen für jeden Prozess werden am Anfang durch die folgenden Matrizen beschrieben:

	Current:			Request:		
	A	B	C	A	B	C
P0:	0	1	0	0	0	0
P1:	2	0	0	2	0	2
P2:	3	0	3	0	0	0
P3:	2	1	1	1	0	0
P4:	0	0	2	0	0	2

Berechnen Sie den Vektor Available und schreiben Sie eine Funktion `display_state`, welche die akuellen Available/Current/Request-Matrizen auf die Kommandozeile schreibt. Lassen Sie weiterhin das Programm mit Hilfe des Bankieralgorithmus berechnen, ob sich das System am Anfang in einem sicheren oder unsicheren Zustand befindet.

Überlegen Sie sich nun jeweils einen Request (also eine veränderte Request-Matrix) für die folgenden Fälle:

a) Ausgehend von den am Anfang initialisierten Available/Current/Request-Matrizen führt der Request zu einem unsicheren Zustand (einem Deadlock) und wird deswegen nicht bewilligt. In diesem Fall sollte jedoch der Request nicht gleich von Anfang an mehr Resourcen anfragen, als noch aktuell verfügbar sind. Es sollte außerdem gewährleistet sein, dass das Deadlock aus maximal 4 Prozessen besteht.

b) Ausgehend von den an Amfang initialisierten Available/Current/Request-Matrizen führt der Request zu einem sicheren Zustand, d.h. er wird vom System bewilligt.

Nach beiden Fällen soll das Programm den aktuellen System-Zustand, der durch den Bankieralgorithmus berechnet wurde, ausgeben. Ihr Programm sollte also folgende Struktur haben:

1. Initialisierung der Anzahl und Namen verfügbarer Resourcen, sowie der anfänglichen Current- und Request-Matrizen.
2. Basierend darauf die Berechnung des anfänglichen Available-Vektors.
3. Ausgabe der Available/Current/Request-Matrizen (`display_state`).
4. Ausgabe des aktuellen Zustandes (safe/unsafe) mit Hilfe des Bankieralgorithmus.
5. Veränderung der Request-Matrix für Fall a).
6. Ausgabe der Available/Current/Request-Matrizen (`display_state`).
7. Ausgabe des aktuellen Zustandes (laut Aufgabenstellung unsafe) mit Hilfe des Bankieralgorithmus.
8. Veränderung der Request-Matrix für Fall b).
9. Ausgabe der Available/Current/Request-Matrizen (`display_state`).
10. Ausgabe des aktuellen Zustandes (laut Aufgabenstellung safe) mit Hilfe des Bankieralgorithmus.

Geben Sie bitte zu Ihrem Programm ein `Makefile` mit ab!