VO 182.711 9. November 2012

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

Ges.)(100)	1.)(35)	2.)(20)	3.)(45)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation mit Semaphoren (35)

Gegeben sei eine Ressource, auf die wie folgt von mehreren gleichartigen Prozessen zugegriffen werden kann:

- Solange die Ressource von weniger als 3 Prozessen gleichzeitig verwendet wird, können weitere Prozesse ohne Verzögerung beginnen, die Ressource ebenfalls zu verwenden.
- Sobald die Ressource von 3 Prozessen gleichzeitig verwendet wird, müssen erst alle drei Prozesse die Ressource freigeben, bevor ein weiterer Prozess damit beginnen darf, auf die Ressource zuzugreifen.

Schreiben Sie ein Codestück zur Implementierungen des Ressourcenzugriffs der Prozesse. Verwenden Sie Semaphore zur Implementierung von Mutual Exclusion und zum Blockieren von Prozessen. Weiters dürfen Sie globale Variable verwenden, um die Anzahl der Prozesse zu zählen, die auf die Ressource zugreifen bzw. auf den Zugriff auf die Ressource warten.

Der Zugriff auf die gemeinsame Ressource soll durch Aufruf der Funktion *use_resource()* erfolgen.

Geben Sie die Initialisierungen für alle Semaphore und Variablen an.

```
/** globale Initialisierungen **/
```

```
/** Code fuer Prozess mit Ressourcenzugriff **/
```

2 Deadlock (20)

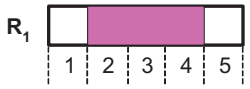
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *begin* und *end* in der Grafik eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehls.

Unterhalb bzw. links der Diagrammachsen sind Balken vorgesehen, in denen die Anforderungen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

   ```
   2: get(R1)
   3: ...
   4: free(R1)
   ```

   Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung aufgrund von Ressourcenkonflikten nicht möglich ist.

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll.

4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.

5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

   ... mit 'A', von welchem aus der Punkt *end* und welcher vom Punkt *begin* erreichbar ist.

   ... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist.

   ... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern ein solcher Punkt vorhanden ist.

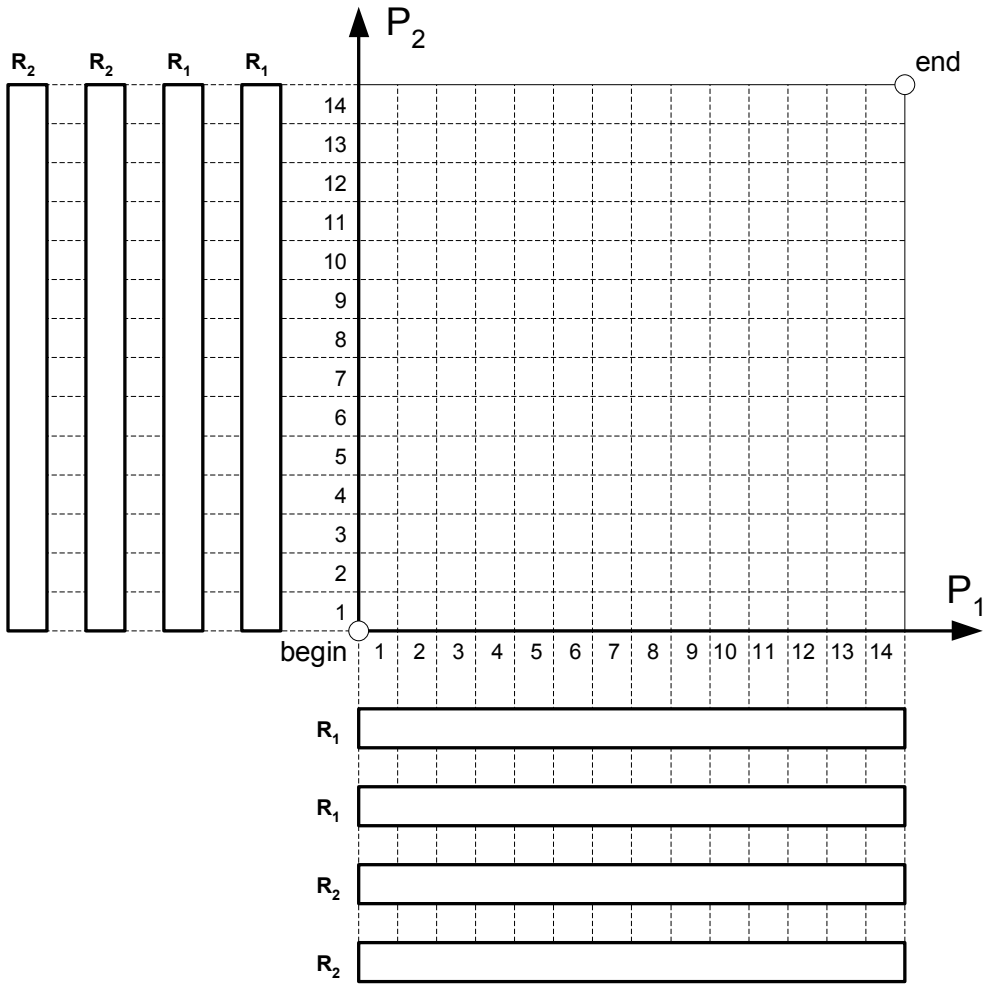
Anmerkung: Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: r1_buf=ptr;
 2: get(R1);
 3: get(R2);
 4: get(R1);
 5: copy(R1, r1_buf);
 6: clear(r1_buf);
 7: use2(R2, R1);
 8: ptr=NULL;
 9: free(R2);
10: use(R1);
11: free(R1);
12: use(R1);
13: free(R1);
14: return;
```

Program P_2:

```
 1: clear(r2_buf);
 2: get(R1);
 3: get(R2);
 4: get(R2);
 5: get(R1);
 6: use2(R1, R2);
 7: free(R2);
 8: use(R2);
 9: free(R1);
10: free(R2);
11: use(R1);
12: free(R1);
13: clear(r1_buf);
14: return;
```



3 Fragen zu Betriebssystemen (45)

Was versteht man unter einem *Process Image*? Erklären Sie, aus welchen Teilen ein Process Image besteht? (4)

Für die Lösung des Problems des geregelten Eintritts in einen kritischen Abschnitt werden drei Eigenschaften gefordert. (a) Nennen Sie diese drei Eigenschaften und erklären Sie deren Bedeutung. (b) Wodurch werden die drei Eigenschaften gewährleistet, wenn Semaphore zum Schutz eines kritischen Abschnitts verwendet werden? (5)

Gegeben sei ein Computersystem, in dem Ihnen zur Synchronisation bzw. Kommunikation von Prozessen nur Nachrichten zur Verfügung stehen (d.h., es gibt keine Semaphore oder andere Synchronisationskonstrukte). Nennen Sie zwei verschiedene Möglichkeiten, wie Sie in diesem Computersystem einen konsistenten Datenaustausch zwischen parallelen Prozessen realisieren können. (4)

Was versteht man unter *Deadlock Avoidance*? Geben Sie zwei Strategien für Deadlock Avoidance an und beschreiben Sie diese. (4)

Beschreiben Sie die folgenden Strategien für das CPU Scheduling und vergleichen Sie deren Eigenschaften: *First Come First Serve*, *Round Robin* und *Virtual Round Robin*. (4)

Wozu wird die *Clock Policy* verwendet? Beschreiben Sie deren Funktionsweise. (5)

Was versteht man unter der *Working-Set Strategie*? Beschreiben Sie deren Funktionsweise und erklären Sie, wie diese Strategie zur Optimierung eines Paging-Systems eingesetzt werden kann. (5)

Beschreiben Sie Aufgabe und Funktion eines *Translation Lookaside Buffers*? Worauf hat man bei der Betriebssystemimplementierung bei einem Process Switch zu achten, wenn man einen Translation Lookaside Buffer verwendet? (4)

Mit welcher logischen Struktur des I/O Systems versucht man bei der Realisierung von I/O Funktionen sowohl ein einheitliches Programmierinterface, als auch eine möglichst gerätespezifische Ansteuerung zu erreichen? (4)

Bei der Realisierung von Dateisystemen gibt es verschiedene Möglichkeiten, um die zu einer Datei gehörenden Datenblöcke zu organisieren bzw. auffindbar zu machen (Block-Allokierung). Nennen Sie vier verschiedene Strategien zur Block-Allokierung von Dateien und beschreiben Sie diese mit ihren Vor- und Nachteilen. (6)