Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

WS 2006/2007
15. Februar 2007

Lösungsvorschläge der Klausur zu Echtzeitsysteme

Aufgabe 1 Nebenläufigkeit (20 Punkte)

a) Lösung siehe Algorithmus (Zur Vereinfach der Realisierung wird angenommen, dass der Benutzer sicherstellt, dass nie mehr als size Elemente enthalten sind. Außerdem soll die Funktion *init*() nur einmal und zum Beginn aufgerufen werden.)

Algorithm 1 Realisierung Warteschlange mit Semaphor

```
 1: Array storage; //Speicherplatz
 2: int pointer; //aktueller Speicheranfang
 3: int size; //Speicherplatzgröße
 4: int count; //aktuell enthaltene elemente
 5: Semaphor sem1=Semaphor(1); //Semaphor zum Schutz der Nachrichtenwarteschlange
 6: Semaphor sem2=Semaphor(0); //Semaphor zur Realisierung der Blockade bei read
 7:
 8: procedure INIT(int max_msg)
 9:     storage=newArray(max_msg);
10:     pointer=0;
11:     size=max_msg;
12:     count=0;
13: end procedure
14:
15: procedure SEND(msg)
16:     down(sem1); //Nur eine Funktion darf zeitgleich schreiben
17:     storage[(pointer+count) modulo size]=msg; //siehe Kommentar zur Vereinfachung
18:     count++;
19:     up(sem2)
20:     up(sem1);
21: end procedure
22:
23: function READ
24:     down(sem2); //Wenn kein Element vorhanden ist, wird blockiert
25:     down(sem1);
26:     count–;
27:     ret= array[pointer];
28:     pointer=(pointer+1) modulo size;
29:     up(sem1);
30:     return ret;
31: end function
```

Algorithm 2 Realisierung Semaphor mit Warteschlange

```
 1: MessageQueue mq;
 2: procedure INIT(int initValue,int max)
 3:     mq.init(max);
 4:     for i=1,...,initValue do
 5:         mq.send(0); //senden eines Dummy-Wertes
 6:     end for
 7: end procedure
 8:
 9: procedure UP(msg)
10:     mq.send(0);
11: end procedure
12:
13: procedure DOWN
14:     mq.read();
15: end procedure
```

b) Lösung siehe Algorithmus (Zur Vereinfachung wird angenommen, dass es einen maximal Wert *max* für den Semaphor gibt, der nicht überschritten wird.)

c) Lösung siehe Algorithmus

Algorithm 3 Realisierung Barriere mit Semaphor

```
 1: Semaphore block=Semaphor(0); //Semaphor zur Blockade der Prozesse
 2: Semaphore critical=Semaphor(1); //Semaphor zum Schutz des Zählers
 3: int count;
 4: int processNumber;
 5: procedure INIT(int value)
 6:     processNumber=init;
 7: end procedure
 8:
 9: procedure REACHED
10:     down(critical)
11:     count++;
12:     if count==processNumber then //falls alle anderen Prozesse die Barriere bereits erreicht
    haben
13:         for i=1,...,processNumber-1 do
14:             up(block); //alle anderen Prozesse freigeben
15:         end for
16:         up(critical);
17:     else
18:         up(critical);
19:         down(block); //Blockieren
20:     end if
21: end procedure
```

Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

WS 2007/2008
08. Februar 2008

Lösungsvorschläge der Klausur zu Echtzeitsysteme

Aufgabe 1 Modellierung (Lösungsvorschlag) (20 Punkte)

a) Rendezvous-Konzept 1, siehe Abbildung: die beiden Prozesse schicken jeweils eine Nachricht (der akzeptierende schickt *ACCEPT*, der eintretende *REACHED*), erst wenn beide Prozesse angekommen sind, wird das Signal zur Weiterverarbeitung (*RENDEZVOUS_COMPLETED*) gegeben. Die Unterscheidung zwischen akzeptierenden Prozess und eintretenden Prozess ist aber nicht notwendig.

b) Leser-Schreiber 2: solange kein Schreiber vorhanden ist, werden Lesezugriffe zugelassen (Signal: *reading_start*) und die Zählervariable hochgezählt. Gibt es einen Schreiber, so wird kein neuer Lesezugriff zugelassen, allerdings muss man die blockierten Leser mitzählen, da sie beim nächsten *reading_start* Signal mit dem Lesen beginnen werden.

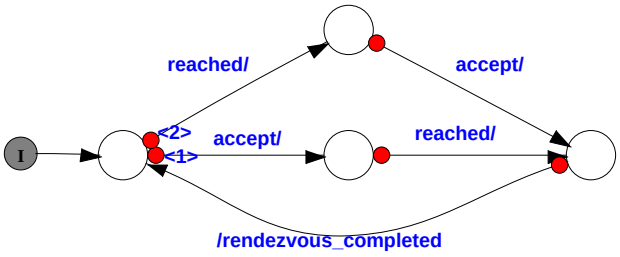

Abbildung 1: Aufgabe Modellierung: Rendezvous

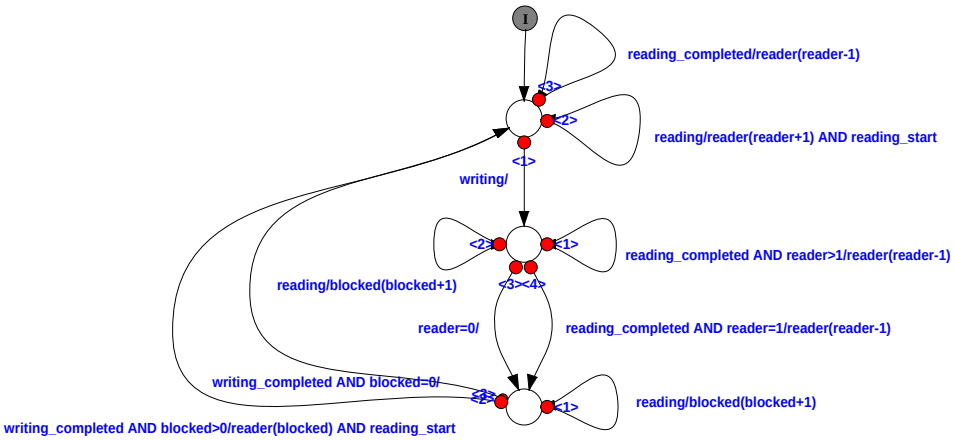

Abbildung 2: Aufgabe Modellierung: Leser-Schreiber