Institut für Informatik

Arbeitsgruppe Autonome Intelligente Systeme

Freiburg, 7. Februar 2017

Systeme I
Musterlösung zu Übungsblatt 14 - Wiederholung

Aufgabe 1 (2+1 Punkte)

Wechselseitiger Ausschluss - Petersons Algorithmus

In der Vorlesung wurden mehrere Lösungsversuche vorgestellt, mit denen eine Softwarelösung für den wechselseitigen Ausschluss gefunden werden sollte. Hier geht es um den *Peterson-Algorithmus*:

Gemeinsame Initialisierung

```
1  flag[0] := false;
2  flag[1] := false;
3  turn := 0;
```

Prozess 0

```
4  wiederhole
5  {
6    flag[0] := true;
7    turn := 1;
8    solange (flag[1] = true und turn = 1)
9      tue nichts;
10
11   Anweisung 1      } kritische Region
12   Anweisung 2      }
13   ...              }
14
15   flag[0] := false;
16
17   Anweisung 3      } nichtkritische Region
18   Anweisung 4      }
19   ...              }
20 }
```

Prozess 1

```
wiederhole
{
  flag[1] := true;
  turn := 0;
  solange (flag[0] = true und turn = 0)
    tue nichts;

  Anweisung 5      } kritische Region
  Anweisung 6      }
  ...              }

  flag[1] := false;

  Anweisung 7      } nichtkritische Region
  Anweisung 8      }
  ...              }
}
```

Die Anweisungen in den kritischen und nichtkritischen Regionen ändern nichts an den Variablen `flag[0]`, `flag[1]` und `turn`.

Im Folgenden soll per Widerspruchsbeweis gezeigt werden, dass diese Lösung den wechselseitigen Ausschluss auf die kritische Region garantiert, d.h., dass die beiden Prozesse niemals gleichzeitig in der kritischen Region sein können.

Nehmen Sie dazu an, dass die Prozesse 0 und 1 zu einem Zeitpunkt t beide in der kritischen Region seien. Die Zeitpunkte, zu denen die Prozesse 0 und 1 die `solange`-Schleife zuletzt verlassen haben, seien t_0 und t_1. Ohne Beschränkung der Allgemeinheit sei $t_0 > t_1$.

Der Beweis beruht auf einer Fallunterscheidung darüber, aus welchem Grund Prozess 0 die `solange`-Schleife zum Zeitpunkt t_0 verlassen konnte. Den Fall `turn` $\neq$ `1` brauchen Sie an dieser Stelle nicht zu betrachten.

a) Betrachten Sie den Fall, dass `flag[1] = false` zum Zeitpunkt t_0. Zeigen Sie, dass diese Annahme zum Widerspruch führt.

b) Welchen Nachteil hat Petersons Lösungsversuch? Wie kann man diese Art von Nachteil umgehen (allgemein, nicht auf Petersons Algorithmus bezogen)?

Lösung:

a) Gilt zum Zeitpunkt t_0, dass `flag[1] = false` ist, so muss sich P1 noch vor Zeile 6 befinden, da P0 `flag[1]` nicht verändert und P1 vor dem Betreten der kritischen Region nur in Zeile 6 auf dieses Flag zugreift und `flag[1] = true` setzt. In der kritischen Region selbst, wird `flag[1]` nicht verändert. Die Tatsache, dass sich P1 noch vor Zeile 6 befindet, ist ein Widerspruch zur Voraussetzung $t_0 > t_1$. [2]

b) Nachteil: Busy Waiting [0.5]
Umgehung durch Blockierung von Prozessen durch das Betriebssystem z.B. mit Mutexen oder Semaphoren. [0.5]

Aufgabe 2 (3 Punkte)

Produzenten/Konsumenten-Problem

In der Vorlesung haben Sie das *Produzenten/Konsumenten-Problem* kennengelernt. Sie sehen hier eine Variante der Lösung aus der Vorlesung. Es wurden lediglich bei der Prozedur `producer` die Reihenfolge der Befehle `down(empty);` und `down(mutex);` vertauscht:

```
Semaphore mutex; count_mutex := 1;
Semaphore empty; count_empty := MAX_BUFFER;
Semaphore full;  count_full  := 0;

1  Prozedur producer
2  {
3    wiederhole
4    {
5      item := produce_item();
6
7      down(mutex);     /* Reihenfolge */
8      down(empty);     /* vertauscht  */
9
10     insert_item(item);
11
```

```
12      up(mutex);
13      up(full);
14    }
15  }
16
17  Prozedur consumer
18  {
19    wiederhole
20    {
21      down(full);
22      down(mutex);
23
24      item := remove_item();
25
26      up(mutex);
27      up(empty);
28
29      consume_item(item);
30    }
31  }
```

Funktioniert diese Variante der ursprünglichen korrekten Lösung fehlerfrei? Beweisen Sie entweder, dass Deadlocks bei diesem Algorithmus garantiert ausgeschlossen sind, oder geben Sie eine Ausführungsreihenfolge an, die zu einem Deadlock führt.

Lösung:

Diese Variante funktioniert nicht fehlerfrei [1]. Beispiel für eine Ausführungsreihenfolge, die zum Deadlock führt [2]:

- Am Anfang ist der Puffer leer.
- Ein Produzent läuft zunächst alleine und füllt den gesamten Puffer. Dann gilt `count(full) = MAX_BUFFER` und `count(empty) = 0`.
- Da `count(empty) = 0` ist, wird der Produzent nun bei `down(empty)` schlafen gelegt. Er hält jedoch noch den Mutex, da dieser zuvor gesperrt wird.
- Ein Konsument startet, kann `down(full)` noch ausführen, bleibt aber bei `down(mutex)` hängen, da der Mutex gesperrt ist. Alle weiteren Konsumenten bleiben ebenfalls hängen.
- Da alle Prozesse warten, kann kein Ereignis mehr eintreten, also befinden sich die Prozesse im Deadlock.

Aufgabe 3 (2+2+2 Punkte)

Deadlocks

Zwei Prozesse wollen auf vier Ressourcen `A`, `B`, `C` und `D` zugreifen. Folgende Tabelle zeigt, in welcher Reihenfolge die Prozesse Ressourcen anfragen und freigeben. Wir vernachlässigen hier Befehle, die zwischen den Anforderungen und Freigaben stehen.

Prozess 1

```
1: Anforderung B
2: Anforderung A
3: Anforderung C
4: Freigabe B
5: Anforderung D
6: Freigabe C
7: Freigabe D
8: Freigabe A
```

Prozess 2

```
1: Anforderung A
2: Anforderung D
3: Freigabe A
4: Anforderung B
5: Anforderung C
6: Freigabe B
7: Freigabe C
8: Freigabe D
```

a) Zeichnen Sie in das Diagramm in Abbildung 1 auf Seite 5 die Bereiche ein, in der beide Prozesse auf eine Ressource zugreifen würden. Die horizontale Achse repräsentiert den Programmfortschritt von Prozess 1 und die vertikale Achse repräsentiert den Programmfortschritt von Prozess 2. Die mit einer Nummer versehenen horizontalen und vertikalen Linien sind die Zeitpunkte in der Programmausführung, bei deren Überschreitung die entsprechend nummerierte Zeile des Prozesses ausgeführt wird.

b) In diesem Szenario kann es zu einem Deadlock kommen. Zeichnen Sie den Bereich ein, in dem ein Deadlock unvermeidlich ist und markieren Sie die Stelle, an dem der Deadlock eintritt. Begründen Sie kurz, wieso an dieser Stelle der Deadlock eintritt.

c) Geben Sie schriftlich in Form von Stichpunkten eine Ausführungsreihenfolge an, die deadlockfrei ist, und eine, die zu einem Deadlock führt.

Lösung:

a) Siehe Abbildung 2, pro korrekt eingezeichneter Ressource [0.5] = insgesamt [2]

b) Bereich für unvermeidbaren Deadlock richtig eingezeichnet [0.5], Stelle des Deadlocks richtig eingezeichnet [0.5].
Begründung Deadlock: Der Deadlock tritt ein, da Prozess P2 Ressource C anfordert, diese aber noch von P1 reserviert ist. Prozess 1 benötigt Ressource D, diese ist aber noch von P2 reserviert. Aus diesem Grund kann keiner der beiden Prozesse weiter ausgeführt werden, sodass es zu einem Deadlock kommt [1].

c) deadlockfreier Ablauf [1]: z.B. Prozess 1 führt alle Schritte durch, danach führt Prozess 2 alle Schritte durch.
Ablauf mit Deadlock [1] (Erklärungen dazu sind nicht gefordert!): z.B.

1) Prozess 2 führt Schritte 1–3 aus, hält danach Ressource D
2) Prozess 1 führt Schritte 1–4 aus, hält danach Ressourcen A und C
3) Prozess 2 führt Schritt 4 aus und erhält zusätzlich Ressource B
⇒ alle Ressourcen sind jetzt gesperrt: A und C von Prozess 1, B und D von Prozess 2
4) Prozess 1 will Schritt 5 ausführen und wird blockiert, da D von Prozess 2 gehalten wird
5) Prozess 2 will Schritt 5 ausführen und wird blockiert, da C von Prozess 1 gehalten wird
⇒ Deadlock

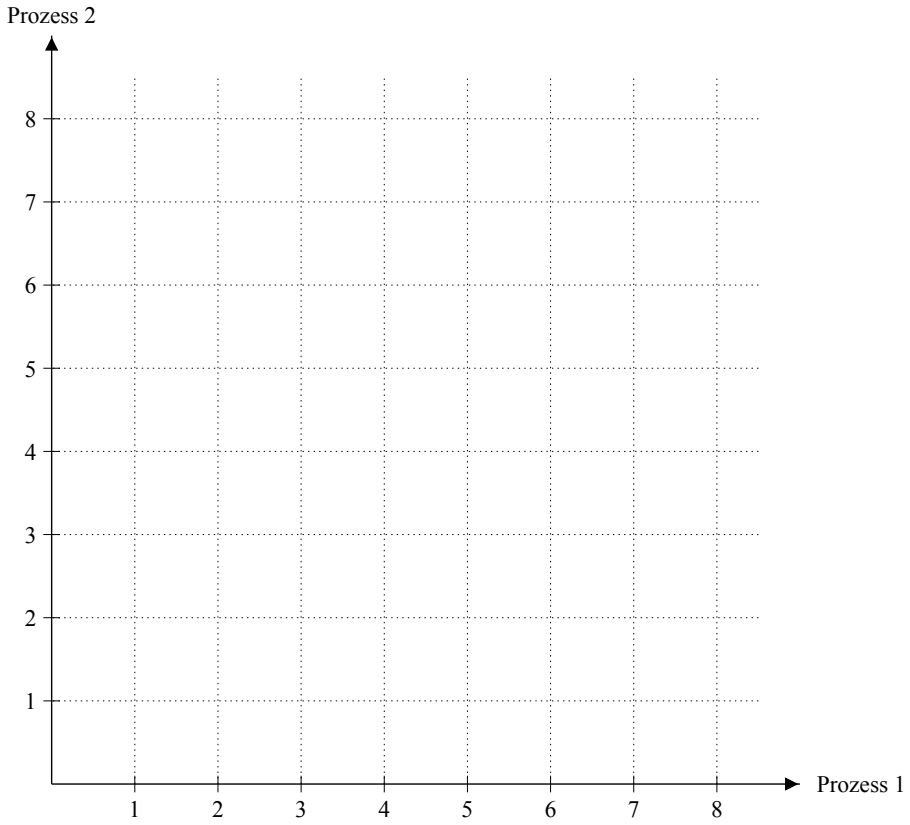

Abbildung 1: Zeichnen Sie hier die Bereiche ein.

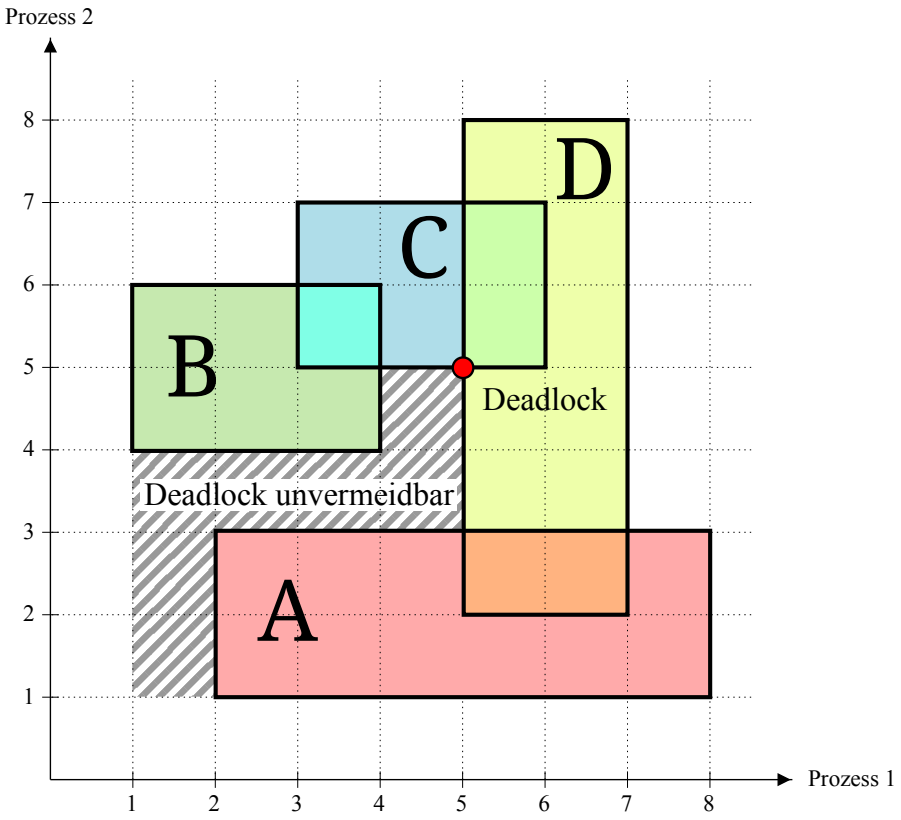

Abbildung 2: Musterlösung

Aufgabe 4 (4+1+1+1 Punkte)

Deadlocks und Bankieralgorithmus

a) Nennen Sie die vier Voraussetzungen, die erfüllt sein müssen, damit ein Ressourcen-Deadlock auftreten kann und erklären Sie jeden Punkt kurz.

b) Beim Bankieralgorithmus wurde der Begriff des „sicheren Zustands“ verwendet. Wie lautet die Definition eines sicheren Zustandes?

c) Führt jeder unsichere Zustand unweigerlich in einen Deadlock? Begründen Sie Ihre Antwort.

d) Drei Prozesse p_1, p_2 und p_3 greifen auf Ressourcen einer einzigen Ressourcenklasse zu. Insgesamt stehen $V = 7$ Ressourcen zur Verfügung. Für die maximale Anzahl M_i von Ressourcen, auf die die Prozesse zugreifen werden, und für die Anzahl von Ressourcen E_i, die die Prozesse schon erhalten haben, gilt:

	M_i	E_i
p_1	6	2
p_2	3	2
p_3	6	2

Ist dieser Zustand ein „sicherer Zustand“? Begründen Sie Ihre Antwort.

Lösung:

a) Je [0.5] für Begriff und [0.5] für Erklärung:
   - Wechselseitiger Ausschluss: Jede Ressource ist entweder verfügbar oder genau einem Prozess zugeordnet.
   - Besitzen und Warten/Hold-and-wait: Prozesse, die schon Ressourcen reserviert haben, können noch weitere Ressourcen anfordern.
   - Kein Ressourcenentzug/Ununterbrechbarkeit: Ressourcen, die einem Prozess bewilligt wurden, können nicht gewaltsam wieder entzogen werden.
   - Zyklische Wartebedingung: Es muss eine zyklische Kette von Prozessen geben, von denen jeder auf eine Ressource wartet, die dem nächsten Prozess in der Kette gehört.

b) Ein Zustand ist sicher, wenn es auf jeden Fall eine deadlockfreie Restausführung aller Prozesse gibt, auch wenn die Prozesse ihre restlichen Anforderungen auf einen Schlag stellen [0.5] und Freigaben erst bei Prozessbeendigung durchführen [0.5].

c) Nein [0.5]. Der Bankier-Algorithmus betrachtet das Worst Case-Szenario. Dieses Szenario muss nicht zwangsläufig auftreten [0.5]. (Worst case: Jeder Prozess fordert seinen maximal angegebenen Ressourcenverbrauch sofort und komplett an und gibt alle Ressourcen erst am Schluss wieder frei.)

d) Der Zustand ist nicht sicher [0.5]. Der Prozess p_2 kann zwar ausgeführt werden, aber danach stehen nicht genug Ressourcen für die Ausführung von p_1 oder p_3 zur Verfügung. Nach der Ausführung von p_2 stehen drei Ressourcen zur Verfügung, aber sowohl p_1 als auch p_3 fordern maximal noch zusätzlich bis zu vier Ressourcen an [0.5].

Aufgabe 5 (3+2+3 Punkte)

Sicherheit

a) Entscheiden Sie, ob die folgenden Aussagen richtig oder falsch sind.

	Behauptung	richtig	falsch
1.	Bei der Caesar-Chiffre lässt sich der Schlüssel bereits aus einem Klartext-Chiffrat-Paar berechnen.	☐	☐
2.	AES gilt für Schlüssellängen ab 192bit als empirisch sichere Stromchiffre.	☐	☐
3.	Eine SSH-Verbindung ist stets gegen Man-in-the-middle-Attacken sicher.	☐	☐
4.	Blockchiffren werden wegen ihrer beweisbaren Sicherheit verwendet.	☐	☐
5.	Pseudozufallszahlengeneratoren erzeugen deterministische Zahlenfolgen basierend auf einer zufälligen Initialisierung.	☐	☐
6.	Ein mit Hilfe des Diffie-Hellman-Verfahrens berechneter Schlüssel wird häufig im Nachgang für symmetrische Verschlüsselung verwendet.	☐	☐

b) Alice möchte mit dem ElGamal-Verfahren den Klartext $P = 2$ an Bob schicken. Bobs öffentlicher Schlüssel lautet $K_{\text{public, Bob}} = (p, g, h) = (17, 3, 12)$.

1) Nehmen Sie an, dass Alice die Zufallszahl $y = 2$ wählt. Geben Sie den Inhalt der Nachricht von Alice an Bob an.

2) Angenommen, Alice verwendet stets das gleiche y. Welches Problem ergibt sich?

c) Wir betrachten eine Blockchiffre, für die als Betriebsmodus ein modifiziertes Cipher-Block-Chaining (CBC)-Verfahren verwendet wird, in dem der Initialisierungsvektor nicht zufällig ist, sondern aus dem kryptographischen Hash des Klartexts besteht.

Angenommen, es existieren nur zwei verschiedene Klartexte (z.B. „ja“ oder „nein“). Die beiden möglichen Klartexte seien dem Angreifer bekannt. Der Angreifer besitzt außerdem Zugriff auf ein Klartext-Chiffrat-Paar für einen der möglichen Klartexte aus früherer Kommunikation. Ist die Kommunikation noch vertraulich? Begründen Sie oder geben Sie einen Angriff an.

Lösung:

a) Die richtigen Antworten lauten ([0.5] pro korrekter Antwort):

	Behauptung	**richtig**	**falsch**
1.	Bei der Caesar-Chiffre lässt sich der Schlüssel bereits aus einem Klartext-Chiffrat-Paar berechnen.	☒	☐
2.	AES gilt für Schlüssellängen ab 192bit als empirisch sichere Stromchiffre.	☐	☒
3.	Eine SSH-Verbindung ist stets gegen Man-in-the-middle-Attacken sicher.	☐	☒
4.	Blockchiffren werden wegen ihrer beweisbaren Sicherheit verwendet.	☐	☒
5.	Pseudozufallszahlengeneratoren erzeugen deterministische Zahlenfolgen basierend auf einer zufälligen Initialisierung.	☒	☐
6.	Der mittels des Diffie-Hellman-Verfahren berechnete Schlüssel wird häufig im Nachgang für symmetrische Verschlüsselung verwendet.	☒	☐

Anmerkung zu (3): vgl. ÜB 13 Aufgabe 2b (2): Bei der ersten Verbindung ist der öffentliche Schlüssel des Servers noch nicht bekannt; ohne einen sicheren separaten Kanal sind deswegen MITM-Attacken möglich.

b) 1) Alice berechnet $C \equiv h^y P \equiv 12^2 \cdot 2 \equiv 288 \equiv 16 \pmod{17}$ [0.5] sowie $g^y \equiv 3^2 \equiv 9 \pmod{17}$ [0.5] und schickt die Nachricht $(9, 16)$ an Bob.

2) Zunächst ergibt sich das Problem, dass gleiche Klartexte stets auf gleiche Chiffrate abgebildet werden, bei sich wiederholenden Klartexten sind also auf Basis der Chiffrate Rückschlüsse auf die Klartexte möglich (vgl. Pinguin-Bild auf Folie 40). [1]

Bemerkung: Es gibt jedoch noch ein weiteres und gravierenderes Problem: Ist ein Klartext-Chiffrat-Paar (P_1, C_1) bekannt, so lässt sich $h^y \equiv P_1^{-1} C \pmod{p}$ berechnen. Bei konstantem y lässt sich dann also aus jedem Chiffrat C_2 der Klartext mittels $P_2 \equiv (h^y)^{-1} C$ berechnen, die Vertraulichkeit der Kommunikation ist nicht mehr gewährleistet.

Für die Lösung der Aufgabe reicht die Schilderung des ersten Problems.

c) Wird im CBC-Modus ein statischer Initialisierungsvektor gewählt, so werden Nachrichten mit gleichem Präfix (Anfang) auf Chiffrate mit gleichem Präfix abgebildet, insbesondere also auch gleiche Nachrichten auf gleiche Chiffrate. [1] Da kryptographische Hashfunktionen deterministisch sind, ist beim vorliegenden modifizierten CBC-Verfahren der Initialisierungsvektor für gleiche Nachrichten identisch. [1] Dementsprechend werden gleiche Klartexte auf gleiche Chiffrate abgebildet.

Die Kommunikation ist nicht vertraulich. Der Angreifer kennt ein Klartext-Chiffrat-Paar (P_1, C_1). Gegeben ein Chiffrat C_2, weiß der Angreifer, dass für den Klartext genau dann $P_2 = P_1$ gilt, wenn $C_2 = C_1$. Sonst entspricht P_2 dem entsprechend anderen möglichen Klartext. [1]

Abgabe: Als PDF-Datei im Ilias bis zum 13. Februar 2017, 23:59 Uhr; zu diesem Übungsblatt findet kein Tutorium statt