Universität Konstanz
Fachbereich Informatik & Informationswissenschaft
Prof. Dr. Marcel Waldvogel / Tino Klingebiel

Betriebssysteme
WS 2016/2017

7. Übungsblatt

Abgabe: 15. Dezember 2016, 4 Uhr

Aufgabe 1: Allgemeines — 3 Punkte

a) Nennen Sie zwei Beispiele für die Nutzung von Threads in heutigen Systemen? (1)

b) Erläutern Sie kurz die Unterschiede zwischen Short-, Mid- und Longtermscheduling. (1)

c) Einige Schedulingverfahren nutzen verschiedene Zeitquanta in verschiedenen Schedulingqueues. Für welche Situationen könnte dies sinnvoll sein? (1)

Aufgabe 2: Deadlock Prevention — 6 Punkte

```
void P0()
{
  while (true) {
    get(A);
    get(B);
    get(C);
    // critical region:
    // use A, B, C
    release(A);
    release(B);
    release(C);
  }
}
```

```
void P1()
{
  while (true) {
    get(D);
    get(E);
    get(B);
    // critical region:
    // use D, E, B
    release(D);
    release(E);
    release(B);
  }
}
```

```
void P2()
{
  while (true) {
    get(C);
    get(F);
    get(D);
    // critical region:
    // use C, F, D
    release(C);
    release(F);
    release(D);
  }
}
```

Betrachten Sie die drei dargestellten Prozessabläufe und beantworten Sie dazu folgende Fragen:

1. Es ist bekannt, dass es gelegentlich zu Deadlocks bei der Ausführung dieser Prozesse kommen kann. Warum ist das so? (Begründen Sie z.B. an Hand einer Ausführungsreihenfolge, die zu einem Deadlock führt) (2)
2. Welche der Vorraussetzungen für einen Deadlock sind ihrer Meinung nach in diesem Beispiel vorhanden, so dass ein solcher auftreten kann? Zählen Sie diese auf. (1)
3. Verändern Sie den Code so, dass Deadlocks ausgeschlossen werden können. Nutzen Sie hierfür nur schon vorhandene Ressourcen. Sperren Sie nicht alle Prozesse durch ein vorheriges Belegen einer gemeinsamen, neuen, Ressource. Begründen Sie, wieso es nun nicht mehr zu einem Deadlock kommen kann. (3)

Aufgabe 3: Race Condition: Mutual Exclusion **7 Punkte**

Gegeben Sei folgender Pseudocode

```
1  int n=50;
2  int anzGesamt = 1;
3  Semaphore lock = 1;
4
5  void total(){
6    int count;
7    for(count=0; count < n; n--) {
8      semWait(lock);
9      anzGesamt++;
10     semSignal(lock);
11   }
12 }
13
14 int main(void) {
15   anzGesamt=0;
16   parbegin(total(), total());
17   write(anzGesamt);
18   return 0;
19 }
```

a) Warum liegt hier eine Race Condition vor? (1)

b) Bestimmen Sie die korrekte untere und obere Grenze des Endwerts der Ausgabe der gemeinsamen Variablen *anzGesamt* durch dieses nebenläufige Programm. Begründen Sie ihre Ergebnisse. Nehmen Sie dafür an, dass Prozesse mit jeder beliebigen relativen Geschwindigkeit ausgeführt werden können und dass ein Wert nur verändert werden kann, nachdem er mit Hilfe eines separaten Maschinenbefehls in ein Register geladen wurde. (2)

c) Nehmen Sie an, dass unter Anwendung der Voraussetzungen in Teil (a) eine willkürliche Anzahl dieser Prozesse parallel ausgeführt werden darf. Welche Auswirkungen hat diese Änderung auf die Bandbreite der Endwerte von *anzGesamt*? Begründen Sie ihre Ergebnisse. (2)

d) Lösen Sie die Race Condition durch die Nutzung eines Monitors auf. (2)

Anmerkung:
parbegin($P_1, P_2, \ldots, P_n$) bedeutet: Suspendiere die Ausführung des Main-Programms, starte die Prozeduren $P_1, P_2, \ldots, P_n$ nebenläufig und führe das Main-Programm weiter aus, nachdem alle $P_1, P_2, \ldots, P_n$ terminiert sind.

Aufgabe 4: Selbststudium: Threading **4 Punkte**

Reine User-Level-Threads waren lange Zeit die einzig möglichen Threads in Java (unter der Bezeichnung "Green Threads". Seit einiger Zeit werden Sie jedoch in der Sun Referenzimplementierung von Sun/Oracle der JVM nicht mehr unterstützt. Wodurch wurden Sie abgelöst? Wie wird Threading in der Referenz-JVM heutzutage bewerkstelligt?

Hinweis: https://en.wikipedia.org/wiki/Thread_(computing)#Models