Vorlesung Betriebssysteme 1

Aufgabenblatt 2 zu Unit OS3

(Besprechung im Übungstermin gemäß Ankündigung auf der Vorlesungs-Website)

Die Aufgaben auf diesem und den folgenden Übungsblättern sollen Ihnen helfen, Ihr Selbststudium zu strukturieren und Ihnen ein Gefühl dafür geben, welchen Wissensstand Sie bisher in der Veranstaltung erlangt haben sollten. Um die gemeinsame Zeit mit den Tutoren optimal nutzen zu können, sollten Sie versuchen, bis zum jeweiligen Tutorengespräch möglichst viele der Aufgaben zu bearbeiten.

Die Bearbeitung sollte in einer Art und Weise erfolgen, dass a) Sie und Ihre Gruppe gut mit den Ergebnissen arbeiten können und b) sich im Gespräch mit den Tutoren über den Stoff der Aufgaben austauschen können. Es bestehen keine weiteren formalen Anforderungen an die Aufgabenbearbeitung unsererseits.

1. Erklären Sie die Begriffe *Nebenläufigkeit* und *Parallelität*. Visualisieren Sie den Unterschied mithilfe einer Skizze.

2. Was ist ein *kritischer Abschnitt* (*Critical Section*)? Beschreiben Sie die **drei** Kriterien, die gültige Lösungen des *Problems des kritischen Abschnitts* (*Critical Section Problem*) erfüllen müssen und nennen Sie ein Beispiel für einen aus der Vorlesung bekannten Algorithmus, der das Problem löst.

3. Betrachten Sie den folgenden Programmablauf, in dem zwei Threads jeweils eine Funktion ausführen:

```
int shared = 0;
void thread_func_a(void)
{
  shared++;
}
```

```
extern int shared;
void thread_func_b(void)
{
  shared--;
}
```

   Welche Probleme können dabei in nebenläufiger, bzw. im paralleler Ausführung auftreten?

Betrachten Sie außerdem folgenden naiven Lösungsversuch:

```
int flags[2] = {0};
int shared = 0;
void thread_func_a(void)
{
  flags[0] = 1;
  while (flags[1] != 0)
    /* no-op */;
  shared++;
  flags[0] = 0;
}
```

```
extern int *flags;
extern int shared;
void thread_func_a(void)
{
  flags[1] = 1;
  while (flags[0] != 0)
    /* no-op */;
  shared--;
  flags[1] = 0;
}
```

Welches der Kriterien für gültige Lösungen des Problems des kritischen Abschnitts kann hier verletzt werden, und warum? Beschreiben Sie beispielhaft einen Ablauf.

4. Analysieren Sie, welche Probleme charakteristisch für rein Software-basierte Lösungen des Problems des kritischen Abschnitts sind. Übersetzen Sie dazu auch das Programm zum Vergleich von Synchronisationsmethoden von nebenläufigen Threads, und experimentieren Sie mit den verschiedenen Implementierungen:
   `https://github.com/osmhpi/concurrency-lab`

   Um diese Probleme zu beheben wurden Hardwaremechanismen zur Unterstützung der Abläufe in den Betriebssystemen eingeführt. Nennen und beschreiben Sie eine Variante davon.

5. Was ist eine Sempahore? Welche Operationen sind auf Semaphoren definiert? Analysieren Sie, welchen Vorteil die Verwendung des Betriebssystemprimitivs Semaphore gegenüber der direkten Verwendung von atomaren Instruktionen der CPU hat. An welchen Stellen im System finden Sie trotz der Vorteile von Semaphoren noch *Spinlocks*, die mit atomaren Instruktionen *Busy Waiting* implementieren?

6. Setzen Sie sich programmatisch mit Semaphoren auseinander. Identifizieren Sie die kritischen Abschnitte in den gegebenen Programmrahmen und sichern Sie Zugriffe auf die kritischen Abschnitte mit den in der Vorlesung vorgestellten Programmierschnittstellen, wie z.B. POSIX Thread Mutexe (`posix_mutex_t`) oder POSIX Semaphoren (`sem_t`) unter Linux und macOS, bzw. `CreateMutex` oder `CreateSemaphore` unter Windows.
   (a) Betrachten Sie dazu den einfachen Programmrahmen zur Synchronisation eines Produzenten Threads und eines Konsumenten Threads über eine gemeinsame Warteschlange:
   `https://github.com/osmhpi/producer_consumer`
   (b) (optional) Betrachten Sie dazu den erweiterten Programmrahmen zur Synchronisation von mehreren Produzenten Threads und mehreren Konsumenten Threads über eine gemeinsame Warteschlange:
   `https://github.com/osmhpi/producer_consumer_advanced`

7. Was ist ein *Deadlock*? Beschreiben Sie, wie Deadlocks bei der Verwendung von Sempahoren entstehen können.

8. Beschreiben Sie die Rolle der *Dispatcher Objekte* im Betriebssystem Windows. Was bedeutet es, wenn ein Dispatcher Objekt *signalisiert* oder *nicht signalisiert* ist? Beschreiben Sie für einen Typen von Dispatcher Objekten, welche Ereignisse im System einen Zustandsübergang von signalisiert nach nicht-signalisiert, bzw. von nicht signalisiert nach signalisiert auslösen können.

9. Was ist eine *Pipe*? Wofür werden Pipes in Betriebssystemen eingesetzt? Worin unterscheiden sich Pipes und *Sockets*?

10. Spielen Sie *Deadlock Empire*:
    `https://deadlockempire.github.io/`