UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Aufgabenblatt 4
Musterlösung

Vorlesung Betriebssysteme und nebenläufige Programmierung
Sommersemester 2025

Aufgabe 1: Thread-Zustände und Zustandsübergänge

a) **bereit:** Thread kann ausgeführt werden, rechnet aber im Moment nicht auf einer CPU.

rechnend: Thread wird momentan auf einer CPU ausgeführt.

blockiert: Thread kann nicht ausgeführt werden, da er auf ein Ereignis oder eine Ressource wartet.

b) **bereit → rechnend:** Der Scheduler des BSs hat den Thread zur Berechnung auf einer CPU ausgewählt.

rechnend → bereit: Der Scheduler des BSs hat dem Thread die CPU entzogen. Dies kann passieren, wenn der Thread einen Systemaufruf durchführt bzw. eine Ausnahme ausgelöst hat, oder wenn die CPU einen Interrupt behandelt. Der Thread kann die CPU auch freiwillig abgeben (s. `man 2 sched_yield`).

rechnend → blockiert: Der Thread hat z.B. einen Systemaufruf durchgeführt, um Daten von der Festplatte zu lesen. Bis die Daten verfügbar sind, wird er am Weiterrechnen gehindert.

blockiert → bereit: Ein Interrupt eines E/A-Gerätes zeigt das Ende einer Operation an. Ein wartender Thread ist nun wieder bereit zu rechnen.
Neben einem Interrupt kann auch ein Systemaufruf eines Threads dazu führen, daß ein anderer Thread wieder rechenbereit wird (wenn dieser z.B. auf die Freigabe einer Ressource gewartet hat).

Aufgabe 2: Thread-Zustandsübergänge

a) Beim Eintritt in den BS-Kern bzw. während der Abarbeitung des Systemaufrufs wird **T2** von **rechnend** in den Zustand **blockiert** überführt. (Beim Eintritt wird zunächst auch der gesamte Prozessorzustand, also die Inhalte aller CPU-Register im Threadkontrollblock von **T2** gespeichert.)

b) Beim Austritt aus dem BS-Kern wählt der Scheduler **T1** aus, da er jetzt der einzige rechenbereite Thread ist. **T1** geht also von **bereit** nach **rechnend**. Gleichzeitig werden die im Threadkontrollblock von **T1** gesicherten Registerinhalte wieder in die CPU geladen.

Aufgabe 3: Elemente von Prozessen und Threads

a) Prozeß
b) Prozeß
c) Thread (obwohl es ggf. auch noch eine gemeinsame Priorität für alle Threads des Prozesses geben kann, sollte jeder Thread eine eigene Priorität haben können)
d) Thread
e) Prozeß
f) Thread
g) Thread
h) Thread

Aufgabe 4: Timer-Interrupt

Zumindestens auf einem Einprozessorsystem (mit einem Core) kann die CPU nur entweder Anwendungs- oder Betriebssystem-Code ausführen. Wenn sie gerade ein Anwendungsprogramm ausführt, kann das Betriebssystem nur dann die Kontrolle erlangen, wenn

a) das Anwendungsprogramm eine Ausnahme verursacht,
b) das Anwendungsprogramm einen Systemaufruf ausführt, oder
c) die Interruptleitung gesetzt wird.

Da eine Anwendung die Fälle **a)** bzw. **b)** vermeiden kann (also das Betriebssystem nie „freiwillig“ aufzurufen braucht), kann man nur durch einen regelmäßigen Interrupt sicherstellen, daß das Betriebssystem regelmäßig aufgefufen wird. Nur dann kann das Betriebssystem beispielsweise entscheiden, daß ein Thread die CPU zugunsten eines anderen Threads abgeben **muß**.

Aufgabe 5: Threadwechsel

Seit **T1** der aktuell laufende und **T2** der nächste auszuführende Thread.

Zuerst muss dafür gesorgt werden, daß ins Betriebssystem verzweigt wird, z.B. durch einen Interrupt. Die Hardware sichert dabei den Inhalt des Befehlszählers (PC) und schaltet in den Systemmodus. Das Betriebssystem muss nun zuerst den von der Hardware gesicherten PC sowie die die Inhalte **aller** anderen CPU-Register im Threadkontrollblock von **T1** abspeichern (damit **T1** später einmal weiterlaufen kann).

Dann müssen die im Threadkontrollblock von **T2** gesicherten Registerinhalte in die CPU geschrieben werden. Genau genommen werden zunächst einmal alle Register bis auf den PC beschrieben. Mit dem speziellen Befehl zur Rückkehr aus dem Betriebssystem wird dann der gesicherte Befehlszählerstand von **T2** in das PC-Register geladen und gleichzeitig der Benutzermodus eingeschaltet. Die CPU arbeitet jetzt also wieder die Befehle von **T2** im Benutzermodus ab.

Aufgabe 6: Threads mit gemeinsamen Variablen

```
// Diese Klasse enthält eine Integer Variable, die von allen
// Threads genutzt werden kann.
class CommonVar
{
    public int i;
    public CommonVar()
    {
        i = 0;
    }
}

// Thread-Klasse
public class Test extends Thread
{
    private CommonVar cvar;

    // Konstruktor initialisiert Referenz auf gemeinsame Variable
    public Test(CommonVar c)
    {
        cvar = c;
    }

    // Implementierung der run() Methode
    public void run()
    {
        for (int i = 0; i < 1000000; i++) {
            cvar.i++;
```

```
        }
    }

    public static void main(String[] args) throws InterruptedException
    {
        CommonVar c = new CommonVar();
        Thread t1 = new Test(c);
        Thread t2 = new Test(c);

        // Starte Threads
        t1.start();
        t2.start();

        // Warte auf Thread-Ende
        t1.join();
        t2.join();

        // Drucke c.i
        System.out.println(c.i);
    }
}
```

Programm ausführen:

```
bslab01% java Test
1019051
bslab01% java Test
2000000
bslab01% java Test
1560671
```

Der Prozessor führt das Hochzählen in drei Schritten durch: erst wird der alte Wert aus dem Speicher geladen, dann wird er hochgezählt, dann wieder in den Speicher geschrieben. Es kann daher vorkommen, daß zwei Threads (fast) gleichzeitig den alten Wert holen (also bevor der andere Thread ihn wieder speichern konnte). Dann geht mindestens eine Änderung (also eine Inkrementierung) verloren, so daß das Ergebnis kleiner als 2000000 wird.

Aufgabe 7: Java Threads (Pflichtaufgabe für die Studienleistung! Abgabe bis So., 18.05., 23:59 über moodle)