TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

Betriebssysteme und Sicherheit, WS 2018/19

2. Aufgabenblatt – Threads

Geplante Bearbeitungszeit: drei Wochen

Teil A Threads und Prozesse

Prozess (nach Nehmer): dynamisches Objekt, das selbständige, voneinander isolierte sequentielle Aktivitäten bezüglich Anfordern und Besitz von Betriebsmitteln in einem Rechensystem repräsentiert.

Ein Prozess ist gegeben durch:

Adressraum: Behälter für die Aufnahme abgegrenzter Programme mit zugeordneten Daten, Abstraktion des physischen Speichers

Handlungsvorschrift: im Adressraum in Form eines sequentiellen Programms gespeichert

Aktivitätsträger oder Thread: führt Handlungsvorschrift aus, Abstraktion des physischen Prozessors

Aufgabe 2.1 Mit welchem Ziel wurde der Prozessbegriff für Betriebssysteme eingeführt? Welche grundlegenden Konsequenzen ergeben sich daraus? Wie unterscheiden sich Threads von Prozessen?

Aufgabe 2.2 Erklären Sie Aufgabe und Struktur eines Thread-Steuerblocks (TCB). Welche Informationen sind für einen Prozess-Steuerblock (PCB) erforderlich?

Aufgabe 2.3 Threads können sich in einem von mehreren Zuständen befinden. Entwerfen Sie ein Zustandsmodell und diskutieren Sie, welche Zustandsübergänge notwendig und welche sind darüber hinaus sinnvoll sind. Erläutern Sie dabei auch die verwendeten Zustände und Übergänge.

Aufgabe 2.4 Betrachtet werde ein Mail-Server, der in einer Liste vorliegende Nachrichten an den jeweiligen Empfänger versenden soll; jede Nachricht habe nur einen einzigen Empfänger. Der Server arbeite gemäß dem nebenstehenden Pseudocode. Dabei werde angenommen, dass stets zu versendende Nachrichten vorliegen.

```
1 while(true) {
2    Nachricht <- Entnehmen(Liste)
3    Empfänger <- Extrahieren(Nachricht)
4    IP <- IP_Ermitteln(Empfänger)
5    Verbindung <- Verbindung_aufbauen(IP)
6    Senden(Nachricht, Verbindung)
7    Verbindung_schließen(Verbindung)
8 }
```

(a) Warum ist diese Lösung ineffizient? Welche Möglichkeiten gibt es, sie zu verbessern? Bietet sie dennoch Vorteile?

(b) Beschreiben Sei eine Implementation, bei der mehrere Threads innerhalb eines Prozesses genutzt werden können. Welche Vor- und Nachteile hat dieses Vorgehen?

(c) Welche Konsequenzen hat es, wenn anstelle der Threads selbst wieder sequentielle Prozesse (mit nur jeweils einem Thread) verwendet werden?

Aufgabe 2.5 In einem System paralleler Prozesse erfolge das Ausdrucken von Dateien folgendermaßen: Ein Prozess, der eine Datei drucken möchte, schreibt den Dateinamen in ein (genügend großes) Feld; jedes Feldelement enthält also genau einen Dateinamen. Die Einträge erfolgen fortlaufend. Eine Variable `frei` zeigt den nächsten freien Platz an, eine weitere Variable `drucke` die nächste zu druckende Datei. Die Variablen `frei` und `druckliste` können von allen Prozessen genutzt werden. Ein Prozess `druckprozess` ermittelt periodisch, ob eine Datei auszudrucken ist, druckt sie gegebenenfalls und aktualisiert die Variable `drucke`.

Erklären Sie an diesem konkreten Beispiel die Begriffe „Wettlaufsituation“, „kritischer Abschnitt“ und „wechselseitiger Ausschluss“.

TEIL B WECHSELSEITIGER AUSSCHLUSS

Wettlaufsituation (race condition): Gegeben seien mehrere Prozesse, die sich *unabhängig* voneinander um die zeitweise exklusive Nutzung derselben Betriebsmittel bewerben. Seien `A` und `B` Code-Abschnitte innerhalb dieser Prozesse, die bei der Ausführungsreihenfolge `A;B` das System in den Zustand 1 überführen und bei der Reihenfolge `B;A` in den Zustand 2. Eine Wettlaufsituation liegt vor, wenn die parallele Ausführung von `A` und `B` zu einem anderem Zustand als 1 oder 2 führen kann.

Kritischer Abschnitt (critical section): Abschnitt eines zu einem Prozess gehörenden Programms, innerhalb dessen auf die gemeinsam genutzten Betriebsmittel zugegriffen wird.

Wechselseitiger Ausschluss (mutual exclusion): Koordinierung der Abläufe der beteiligten Prozesse so, dass die kritischen Abschnitte jeweils nur von einem Prozess betreten werden können.

Aufgabe 2.6 Wie wird wechselseitiger Ausschluss prinzipiell realisiert? Welche allgemeinen Anforderungen muss jede Realisierung erfüllen?

Aufgabe 2.7 Erklären Sie, warum die folgende Lösung zum wechselseitigen Ausschluss mittels Sperrvariablen (vgl. Folie 44 des Foliensatzes „Threads“) das Problem nicht löst. Worin liegt die prinzipielle Ursache?

```
void enter(int *lock) {
  while (*lock != 0) { /* wait */ }
  *lock = 1;
}
```

```
void leave(int *lock) {
  *lock = 0;
}
```

Aufgabe 2.8 Welche Möglichkeiten zur Realisierung des wechselseitigen Ausschlusses gibt es auf Maschinenebene? Welche Vor- und Nachteile haben sie?

Aufgabe 2.9 Im Gegensatz zu Aufgabe 7 ist der wechselseitige Ausschluss mehrerer Threads bezüglich eines kritischen Abschnitts mittels einer einfachen Schleife beispielsweise dann korrekt möglich, wenn in der zugrundeliegenden Hardware ein Maschinenbefehl der Form `xchg R adr` bereitgestellt wird, durch den atomar der Inhalt eines Registers `R` und einer Hauptspeicherzelle `adr` miteinander ausgetauscht werden. Diskutieren Sie unter diesem Gesichtspunkt die nachstehende Implementation auf der Basis eines i386-Systems.

Assembler-Implementation

```
/* Funktionen extern zugänglich machen */
.globl lock, unlock

/* Variable 'mutex' mit Inhalt 0 */
mutex:  .long  0

lock:   movl   $1, %eax
loop:   xchg   %eax, mutex
        cmp    $0, %eax
        jne    loop
        ret
unlock: movl   $0, mutex
        ret
```

Semantik

```
long mutex = 0;

void lock() {
  long eax = 1;

  do {
    // atomare Tupelzuweisung
    (eax, mutex) = (mutex, eax);
  } while (eax != 0);
}

void unlock() { mutex = 0; }
```

Aufgabe 2.10 Die für die Lösung des Wettlaufproblems geforderte Bedingung der Lebendigkeit kann folgendermaßen untergliedert werden: Lebendigkeit ist das *Nicht*-Vorliegen von

- Fernwirkung: Ein Thread außerhalb seines kritischen Abschnitts (und außerhalb der Dienste `enter_section`, `leave_section`) behindert den Thread-Ablauf.
- Ausgrenzung: Ein Thread wird ständig am Eintritt in seinen kritischen Abschnitt gehindert.
- Verklemmung: Zwei Threads behindern sich gegenseitig am Eintritt in ihren kritischen Abschnitt.

Diskutieren Sie unter diesem Gesichtspunkt, inwieweit die folgenden fünf Versuche das Wettlaufproblem zwischen zwei Threads T_1, T_2 lösen. (Die Threads sollen die angegebenen Befehlsfolgen jeweils im Zyklus „ewig“ durchlaufen, sofern möglich.)

- Voraussetzung für 1. und 5. Versuch:

```
int next = 1;
```

- Voraussetzung für 2. – 5. Versuch:

```
bool req1 = false, req2 = false;
```

	Thread 1	Thread 2

1. Versuch	`while (next == 2) {`
	`}`
	`/* kritischer Abschnitt */`
	`next = 2;`
		`while (next == 1) {`
		`}`
		`/* kritischer Abschnitt */`
		`next = 1;`

2. Versuch	`while (req2 == true) {`
	`}`
	`req1 = true;`
	`/* kritischer Abschnitt */`
	`req1 = false;`
		`while (req1 == true) {`
		`}`
		`req2 = true;`
		`/* kritischer Abschnitt */`
		`req2 = false;`

3. Versuch	`req1 = true;`
	`while (req2 == true) {`
	`}`
	`/* kritischer Abschnitt */`
	`req1 = false;`
		`req2 = true;`
		`while (req1 == true) {`
		`}`
		`/* kritischer Abschnitt */`
		`req2 = false;`

4. Versuch	`req1 = true;`
	`while (req2 == true) {`
	`  req1 = false;`
	`  req1 = true;`
	`}`
	`/* kritischer Abschnitt */`
	`req1 = false;`
		`req2 = true;`
		`while (req1 == true) {`
		`  req2 = false;`
		`  req2 = true;`
		`}`
		`/* kritischer Abschnitt */`
		`req2 = false;`

5. Versuch	`req1 = true;`
	`while (req2 == true) {`
	`  if (next == 2) {`
	`  req1 = false;`
	`  while (next == 2) {`
	`  }`
	`  req1 = true;`
	`  }`
	`}`
	`/* kritischer Abschnitt */`
	`next = 2;`
	`req1 = false;`
		`req2 = true;`
		`while (req1 == true) {`
		`  if (next == 1) {`
		`  req2 = false;`
		`  while (next == 1) {`
		`  }`
		`  req2 = true;`
		`  }`
		`}`
		`/* kritischer Abschnitt */`
		`next = 1;`
		`req2 = false;`

Aufgabe 2.11 Untersuchen Sie, ob der folgende Algorithmus die Bedingungen zum Schutz kritischer Abschnitte erfüllt.

```
int s1 = s2 = next = 1;
```

Thread 1	Thread 2

`  s1 = 0;`
`M1:`
`  if (s2 == 0) {`
`  if (next == 1)`
`  goto M1;`
`  s1 = 1;`
`  while (next == 2) {}`
`  }`
`  /* kritischer Abschnitt */`
`  s1 = 1;`
`  next = 2;`
	`  s2 = 0;`
	`M2:`
	`  if (s1 == 0) {`
	`  if (next == 2)`
	`  goto M2;`
	`  s2 = 1;`
	`  while (next == 1) {}`
	`  }`
	`  /* kritischer Abschnitt */`
	`  s2 = 1;`
	`  next = 1;`

TEIL C SEMAPHORE

Aufgabe 2.12 Was versteht man unter einem Semaphor? Welche Vor- und Nachteile bieten Semaphore gegenüber den in Aufgabe 8 besprochenen Mechanismen?

Aufgabe 2.13 Geben Sie eine Implementation für einen zählenden Semaphor in Form eines Programmablaufplans an. Die interne Zählvariable `count` soll dabei auch negative Werte annehmen können, mit folgender Bedeutung:

- Ist `count` ≥ 0, so gibt `count` an, wie viele Threads den kritischen Abschnitt noch betreten können.
- Ist `count` ≤ 0, so gibt `-count` an, wie viele Threads derzeit auf das Betreten des kritischen Abschnitts warten.

Sie können vom Vorhandensein einer geeigneten Implementierung für die benötigte Warteschlange ausgehen.

Aufgabe 2.14 Beschreiben Sie die Steuerung für ein System, bei dem Fahrzeuge über eine Brücke wollen, für folgende Fälle der maximalen Belastbarkeit der Brücke:

(a) ein Fahrzeug, unabhängig von der Richtung (1 Fahrspur)

(b) ein Fahrzeug je Richtung gleichzeitig (2 Fahrspuren)

(c) drei Fahrzeuge je Richtung gleichzeitig (2 Fahrspuren)

(d) drei Fahrzeuge, unabhängig von der Richtung (1 Fahrspur; die Fahrzeuge aus einer Richtung müssen so lange warten, bis alle Fahrzeuge der Gegenrichtung die Brücke passiert haben)

Verwenden Sie Semaphore. Welches Problem tritt in (d) auf, wie lässt es sich lösen?

Aufgabe 2.15 Erläutern Sie das Erzeuger-Verbraucher-Problem und seine prinzipiellen Lösungsmöglichkeiten.

(a) Der Puffer sei einelementig. Geben Sie eine Lösung dieses speziellen Problems mittels Semaphoren an. Verdeutlichen Sie die Korrektheit Ihrer Lösung anhand eines repräsentativen Ablaufbeispiels.

(b) Inwieweit lässt sich das spezielle Problem auch ohne Semaphore (allein auf Maschinenebene) lösen?

(c) Wie unterscheidet sich das Leser-Schreiber-Problem vom Erzeuger-Verbraucher-Problem? Welche grundlegende Erwägung spielt bei der Lösung eine Rolle?

Aufgabe 2.16 Ein Ringpuffer werde durch einen Thread A mit Elementen vom Typ `elem_t` gefüllt und durch einen Thread B geleert (gemäß FIFO); der Puffer soll dabei als n-elementiges Feld (n konstant) implementiert werden.

(a) In welchen Fällen können Konflikte auftreten?

(b) Beschreiben und diskutieren Sie eine Lösung des Problems in C mittels Semaphoren.

Aufgabe 2.17 Erklären Sie den Begriff „Prioritätsumkehr“ (priority inversion) in einem System ohne beziehungsweise mit Semaphoren.

Klausuraufgabe I

Folgender C-Code sei gegeben:

```
1  int x = 0;
2
3  int inc(void)
4  {
5      x++;
6  }
```

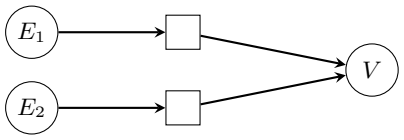
Die Funktion `inc()` werde von zwei Threads gleichzeitig aufgerufen.

a) Erläutern Sie an diesem Beispiel den Begriff der Wettlaufsituation (englisch „Race Condition“)! Nennen Sie mögliche Ergebnisse nach Ausführung der Funktion durch beide Threads!

b) Nennen Sie vier Techniken, die zu einem deterministischen Ergebnis führen! Erwähnen Sie für jede Technik, ob diese mit Busy Waiting arbeitet!

HINWEIS: Die Funktionalität von `inc()` soll erhalten bleiben; ebenso die Parallelausführung der Funktion durch zwei Threads.

Ein Erzeuger-Verbraucher-System bestehe aus zwei Erzeugern E_1 und E_2, die wiederholt jeweils einen einelementigen Puffer befüllen. Diese Puffer werden durch einen gemeinsamen Verbraucher V geleert. Dabei kann V die Puffer in beliebiger Reihenfolge bearbeiten. Insbesondere kann V arbeiten, sobald einer der beiden Puffer gefüllt ist.



c) Geben Sie Implementierungen für Erzeuger und Verbraucher in Pseudocode an, bei der kein Busy Waiting auftritt und die maximale Parallelität gewährleistet!

Klausuraufgabe II

In einem Programm mit 2 Threads werden die unten folgenden Codeteile zur Absicherung eines kritischen Abschnitts verwendet. Dabei bezeichnet `/* kA */` den kritischen Abschnitt, für dessen Ausführung die Kriterien des wechselseitigen Ausschlusses garantiert werden soll.

Globale Variablen

```
1  int next = 1;
2  int t1 = 0, t2 = 0;
```

Thread 1

```
1  t1 = 1;
2  next = 2;
3  while(t2 == 1 && next == 1) {}
4  /*
5      kA
6  */
7  t1 = 0;
```

Thread 2

```
1  t2 = 1;
2  next = 1;
3  while (t1 == 1 && next == 2) {}
4  /*
5      kA
6  */
7  t2 = 0;
```

a) Bei einem Codereview wird festgestellt, dass der verwendete Code wechselseitigen Ausschluss nicht gewährleistet. Begründen Sie wieso der Reviewer Recht hat.

b) Schlagen Sie eine Änderung des oben stehenden Codes vor, so dass das gefundene Problem behoben wird und der Code die Anforderungen an wechselseitigen Ausschluss erfüllt.

Klausuraufgabe III

a) Nennen Sie jeweils einen Nachteil der folgenden Mechanismen für wechselseitigen Ausschluss.

- Interrupts ausschalten
- Atomare Instruktionen
- Dekker-Algorithmus

b) In einem Supermarkt wird neue Ware von **einem** Lieferant angeliefert und von **zwei** Lageristen eingeräumt. Dabei stehen auf der Laderampe insgesamt 10 Plätze zum Ablegen von Paketen zur Verfügung.
Ergänzen Sie folgenden Code um die nötige Synchronisierung zwischen dem Lieferant und den Lageristen mittels Semaphoren (Datentyp `sem_t`). Fügen Sie außerdem Code ein, um die Pakete auf der Laderampe abzulegen bzw. dieser zu entnehmen. Dabei sollen alle Beteiligten maximal parallel arbeiten und kein aktives Warten (*busy waiting*) auftreten.

```
int n_frei =    ; int n_voll =    ;

paket_t plaetze[    ];


void lieferant(paket_t ware) {

    plaetze[n_frei] = ware;

}

void lagerist() {

    paket_t ware = plaetze[n_voll];

    einraeumen(ware);

}
```