Bitte hier tackern!

SoSe 20 Systemprogrammierung

Aufgabenblatt 4

Abgabe von

Michelle Döring

Jannik Heinze

Iryna Novytska

Charlotte Fritsch

Aufgabe 4.4: Sychronisation/Kooperation

a) Die beiden Threads sollen nun nebenläufig ausgeführt werden. Nennen Sie ein praktisches Problem, das dabei auftreten kann.

Während die Wettervorhersage von $calculate\_forecast$ kalkuliert wird, kann es passieren, dass $gather\_data$ die Daten in dem gemeinsam genutzten Betriebsmittel verändert. Dadurch wird die Wettervorhersage verfälscht.

b) Was sind Spurious Wakeups? Und wie kann man sicher stellen, dass die eigentliche Bedingung erfüllt ist, auch im Fall eines Spurious Wakeups?

Ein Spurious Wakeup tritt auf, wenn ein Thread aus dem Warten auf eine Bedingung aufwacht, nur um festzustellen, dass die Bedingung auf die er wartet gar nicht erfüllt ist. Man nennt diesen Wakeup spurious (deutsch: unberechtigt, falsch), da der Thread scheinbar ohne Grund geweckt wurde.
Solche Wakeups passieren keinesfalls grundlos, häufig treten sie auf, da in der Zeit zwischen dem Signal der Bedingungsvariable und dem tatsächlichen Laufen des wartenden Threads ein anderer Thread bereits angelaufen ist und die Bedingung wieder geändert hat.
Die Threads treten also in ein Rennen bei dem üblicher Weise einer der Threads als erstes läuft und 'gewinnt' und der andere auf eine unerfüllte Variable stößt.
Auf einigen Systemen, besonders Multiprozessor Systemen, tritt das Problem der Spurious Wakeups verschärft auf, da dort viele Threads auf eine Bedingungsvariable warten. Hat man also 10 Threads warten, so wird nur einer 'gewinnen' und die anderen 9 werden einen Spurious Wakeups erleben.
[Quelle: https://www.modernescpp.com/index.php/c-core-guidelines-be-aware-of-the-traps-of-condition-variables, letzter Abruf: 22.06.2020, 05:40]

c) Verbessern Sie obiges Programm mit $signal/wait$ und $mutex$, sodass Probleme verhindert werden. Da die Datenerhebung($gather\_data$) sehr lange dauert, sollte damit schon begonnen werden, während der alte Datensatz ausgewertet wird($calculate\_forecast$)(siehe Beispielablauf). Außerdem sollten beim Auswerten keine Datensätze übersprungen werden, um stets aktuelle Daten zu gewährleisten. Ihre Lösung sollte auch Spurious Wakeups berücksichtigen. Hinweis: Sie können zusätzliche globale Variablen verwenden.

Global:

```
char buffer[1024];
state s= gather;  //Werte: 'gather' oder 'copy_data'
Mutex data_m;    // mutex auf die Daten
Mutex s_m;
Signal dataFinished_S;
Mutex calculate_block_m;
```

Sensor-Thread:

```
while(1)          {
        lock(data_m);
        gatherData(&buffer);
        unlock(data_m);
        lock(s_m);
        s = copy_data;
        signal(dataFinished_S);
        unlock(s_m);
}
```

Berechnungs-Thread:

```
lock(calculate_block_m);
lock(s_m);
If (s=gather) {
        unlock(s_m);
        wait(dataSinished_S);
}
lock(s_m);
char internal_buffer[1024];
memcpy(internal_buffer,buffer, 1024);
s = gather;
unlock(s_m);
unlock(calculate_block_m);
calculate_forecast(internal_buffer);
```

d) Welche Arten der expliziten Prozess-/Threadinteraktion gibt es? Um welche Art der Interaktion handelt es sich hier?

Prozesse als Teile komplexer Programmsysteme müssen Daten austauschen, z. B. sich beauftragen, aufeinander warten, ...

Konkurrenz
Tritt auf, wenn sich zwei oder mehrere Prozesse sich gleichzeitig um ein Betriebsmittel bewerben, welches nur in beschränkter Anzahl zur Verfügung steht und bei dem die Nutzung nur exklusiv durch einen Prozess möglich ist, da es andernfalls zu fehlerhaften Ergebnissen oder inkonsistenten Zuständen kommt, d. h. wenn es kritische Abschnitte in den Programmen der Prozesse gibt. In diesem Fall sind Synchronisationsmechanismen notwendig.

Kooperation
Tritt auf, wenn Prozesse gezielt Informationen untereinander austauschen, z. B. wenn Sie ihre Aktionen bewusst aufeinander abstimmen. Die kooperierenden Prozesse müssen von der Existenz aller anderen beteiligten Prozesse wissen und ausreichend Informationen über diese besitzen, z. B. Art und Funktionalität der Schnittstellen. In diesem Fall werden Synchronisations- und Kommunikationsmechanismen verwendet.
[Quelle: Vorlesung 4, S.3 ff., https://docplayer.org/40741358-Interaktionsarten-zusammenhang-arten-der-interaktion-7-kapitel-prozesse-im-zusammenspiel-prozessinteraktion.html]

In diesem Fall liegt sowohl Konkurrenz und Kooperation vor. Ein Sensor-Thread steht mit einem Berechnungs-Thread in Kooperation, da sie gezielt die Daten untereinander austauschen. Verschiedene Berechnungs-Threads stehen in Konkurrenz miteinander, genauso wie Sensor-Threads untereinander.

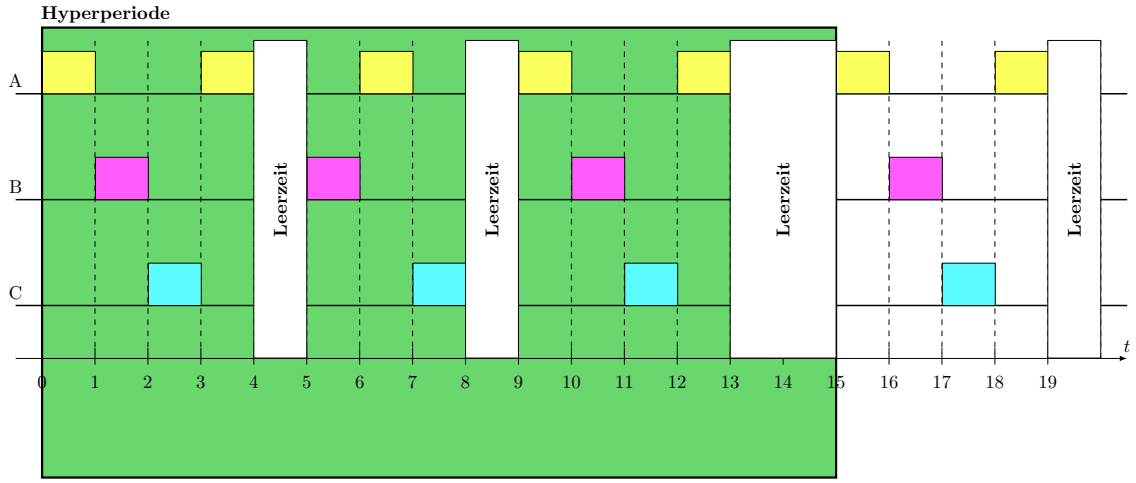
Aufgabe 4.5: Periodische Prozesse

a) Existiert für diese Prozesse ein zulässiger Schedule? Wird das notwendige Kriterium erfüllt?

Offensichtlich gibt es einen zulässigen Schedule, sonst würden die folgenden Teilaufgaben wenig Sinn ergeben. ;)
Ja. Siehe Teilaufgabe b) und c). Insbesondere ist aber auch der schedulability, feasibility test erfüllt:

$$\sum_{i=1}^{3} \frac{D_i}{P_i} = \frac{1}{3} + \frac{1}{5} + \frac{1}{5} = \frac{11}{15} \leq 0.74 \leq 1$$

b) Wie könnte dieser aussehen? Geben Sie etwaige Leerzeiten an und markieren Sie die Hyperperiode.



c) Ist Rate-Monotonic-Scheduling (RMS) ein gültiger Schedule? Begründen Sie Ihre Antwort.

Ja. Die hinreichende Bedingung ist erfüllt:

$$\sum_{i=1}^{3} \frac{D_i}{P_i} = \frac{1}{3} + \frac{1}{5} + \frac{1}{5} = \frac{11}{15} \leq 0.74 \leq 0.75 = 3 \cdot (1.25 - 1) \leq 3 \cdot (\sqrt[3]{2} - 1)$$

Damit können die gegebenen periodischen Prozesse durch ein ratenmonotones Verfahren eingeplant werden.

Was passiert, wenn zu den Prozessen ein weiterer Prozess, Prozess D mit ($D = 2, P = 9$), hinzugefügt wird?

Die hinreichende Bedingung ist nun nicht mehr erfüllt und RMS ist damit kein gültiger Scheduler (mehr).

$$\sum_{i=1}^{4} \frac{D_i}{P_i} = \frac{1}{3} + \frac{1}{5} + \frac{1}{5} + \frac{2}{9} = \frac{43}{45} \leq 0.9556 \nleq 0.756 = 3 \cdot (1.189 - 1) \leq 3 \cdot (\sqrt[4]{2} - 1)$$

Das ist mathematisch korrekt, da die Abschätzungen nach oben bzw unten höchstens um ein Hundertstel sind, aber $0.9556 - 0.756 = 0.1996$. Sonst könnte man argumentieren, dass einfach nicht gut genug abgeschätzt wurde: $0.5 \leq 2 \nleq 1 \leq 1.5$, aber $0.5 \leq 1.5$.