Systemprogrammierung
Hausaufgabe 4

Tutor: Friedrich, Tutorium: Di. 8 Uhr FH 314

Gruppenmitglieder:
Julius Basler, Fabian Philipp Berner, Jannik Leander Hyun-Ho Novak

Aufgabe 4.3 Periodische Prozesse

Prozess	Dauer	Periode
A	2	6
B	1	4
C	3	9

a) Existiert ein zulässiger Schedule?

(Quelle: Foliensatz 3; Folie 37)

notwendiges Kriterium für jeden Prozess:

$$0 \leq b_i \leq d_i \leq p_i \qquad \forall i \in \{A, B, C\}$$

Anwendung auf gegebene Prozesse:

$$\begin{aligned} A: &\qquad b_A = 2 \leq 6 = p_A \\ B: &\qquad b_B = 1 \leq 4 = p_B \\ C: &\qquad b_C = 3 \leq 9 = p_C \end{aligned}$$

Schedulability Test:

$$\begin{aligned} \sum_i \frac{b_i}{p_i} &\leq 1 \\ \Rightarrow \frac{b_A}{p_A} + \frac{b_B}{p_B} + \frac{b_C}{p_C} &\leq 1 \\ \Leftrightarrow \frac{1}{3} + \frac{1}{4} + \frac{1}{3} = \frac{11}{12} &\leq 1 \end{aligned}$$

Das hinreichende Kriterium ist somit erfüllt und es existiert ein zulässiger Schedule (Schedulability Test erfolgreich).

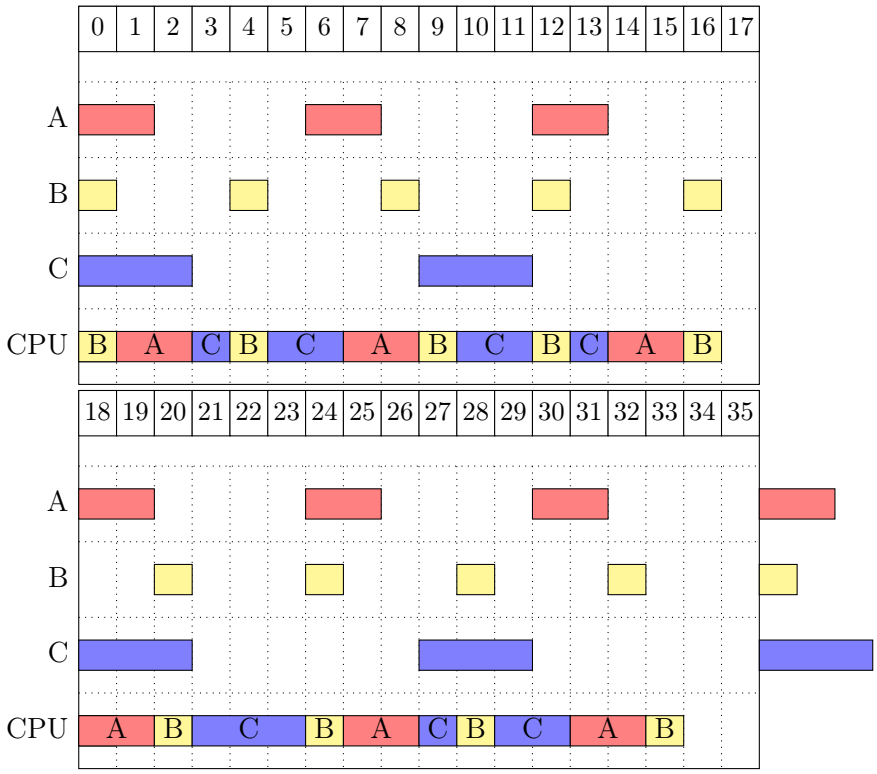
b) Wie könnte dieser Schedule aussehen?

Ratenmonotones Verfahren: (Quelle: Foliensatz 3; Folie 38)
Satz von Liu & Layland:

$$\sum_{i=1}^{n} \frac{b_i}{p_i} \leq n(2^{1/n} - 1)$$
$$\Rightarrow \frac{11}{12} \leq 3 \cdot (2^{1/3} - 1)$$
$$\Rightarrow 0,917 \leq 0,78 \quad \lightning$$

Die hinreichende Bedingung wird nicht erfüllt, somit ist eine Einplanung über ein ratenmonotones Verfahren nicht zwingend erfolgreich.
Verwendung von Earliest-Deadline-First (funktioniert wegen a))
(auch nicht unbedingt erfolgreich (Verspätungen können auftreten))

Abbildung: Gantt-Diagramm für EDF-Scheduling der gegebenen Prozesse. Das Diagramm umfasst die Hyperperiode Zeitslot $t \in [0, 35]$ (kleinstes-gemeinsames-Vielfaches der Periodendauern p_i der vorliegenden Prozesse). Leerzeiten sind durch freie Zeitslots markiert (Zeitslot $t = 17$ und $t = [34, 35]$).



c) Drei periodische Prozesse, 70% Prozessorauslastung: RMS-planbar?

In Aufgabenteil b) wurde bereits der Satz von Liu & Layland verwendet, hier noch einmal allgemein:

$$\underbrace{\sum_{j=1}^{n} \frac{b_i}{p_i}}_{\text{Prozessorauslastung}} \leq \underbrace{n(2^{1/n} - 1)}_{\text{obere Schranke für zul. Schedule}}$$

Für eine Anzahl der Prozesse von $n = 3$ ergibt sich somit für die rechte Seite der Gleichung:

$$3 \cdot (2^{1/3} - 1) \approx 0,78$$

Einsetzen der Prozessorauslastung (linke Seite) in den Satz von Liu & Layland:

$$0,7 \leq 0,78$$

Der Satz von Liu & Layland stellt eine hinreichende Bedingung für die Einplanung einer Menge periodischer Prozesse über ein ratenmonotones Scheduling-Verfahren dar. Da diese hinreichende Bedingung hier erfüllt ist, kann eine RMS-Planung der Prozesse erfolgen.

d) Hinzufügen eines weiteren Prozesses

Durch das Hinzufügen eines weiteren Prozesses D (Dauer 3, Periode 8) zur Prozessmenge aus a) und b):
Schedulability Test (vgl. Aufgabenteil a)) nicht länger erfüllt:

$$\begin{aligned} \sum_i \frac{b_i}{p_i} &\leq 1 \\ \Rightarrow \frac{b_A}{p_A} + \frac{b_B}{p_B} + \frac{b_C}{p_C} + \frac{b_D}{p_D} &\leq 1 \\ \Leftrightarrow \frac{1}{3} + \frac{1}{4} + \frac{1}{3} + \frac{3}{8} = \frac{31}{24} &\leq 1 \quad \lightning \end{aligned}$$

Aufgabe 4.4: Synchronisation/Kooperation

a) Ein praktisches Problem, welches auftreten kann:

Angenommen es steht ein Auto an der Schraubmaschine und der Schraubmaschine-Thread ist aktiv:
Schraubmaschine wird angedockt, Schrauben werden festgedreht, Schraubmaschinen-Thread wird CPU entzogen und Beförderungsband-Thread wird aktiv
→ PROBLEM: Band wird bei angedockter Schraubmaschine nach vorne bewegt (Spezifikation sagt, dass Kratzer vermieden werden sollen, genau das würde hier jedoch geschehen).

b) *Spurious-Wakeups*

(Quelle: `man`-Pages zu `p_thread_cond_wait` und Foliensatz 4: Folie 50)

Spurious Wakeups können bei der Verwendung von Monitoren in C (`PThreads`) geschehen. Bei der Verwendung von Signalen, auf die durch `p_thread_cond_wait` und/oder `p_thread_cond_timedwait` gewartet wird, kann es dazu kommen, dass ein Thread aus dem wartenden Zustand zurückkehrt und somit deblockiert, ohne dass das entsprechende Signal gesendet wurde. Die Bedingung, unter welcher der Thread den `p_thread_cond_wait`-Aufruf ausgeführt hat, muss daher in einer Schleife implementiert werden, da `p_thread_cond_wait` und/oder `p_thread_cond_timedwait` nichts über die Bedingungsvariable aussagen.

c) Verbessertes Programm

```
// Aufgabenblatt 4: Aufgabe 4.4 Synchronisation/Kooperation:

// Annahmen: 1 Beförderungsband und ein Schraubarm, immer
//              Autos vorhanden (keine Lücken auf Fließband)
// Startzustand: Frei wählbar, entweder erst Schrauben anziehen oder erst
//               Auto vorschieben


State s;                // Werte: 'belt' oder 'screw'
Mutex status_mutex;     // Schutz der status-variable
Signal car_forwarded;   // Auto wurde weitergegeben vom Fließband
Signal screws_tight;    // Schrauben angezogen

// Beförderungsband-Thread:

while(1){
        // Hier kein mutex für status notwendig, da nur 1 Band 1 Maschine
        while(status != 'belt'){        // spurious-wakeups abfangen
                wait(screws_tight);     // Schrauben angezogen
        }

        belt_move(1);   // Band nach vorne bewegen

        lock(status_mutex);
        s = screw;      // Status auf screw
        signal(car_forwarded);  // Status wurd auf screw gesetzt, signal senden
        unlock(status_mutex);
}
```

```
//Schraubarm-Thread

while(1){

        // while-Schleife um spurious wakeups abzufangen
        while(status != 'screw'){
                wait(car_forwarded);    // Wait for new car
        }

        arm_dock();     // Schraubmaschine andocken
        screw();        // Schrauben festdrehen

        arm_undock();

        lock(status_mutex);     //status locked
        s = belt();
        signal(screws_tight);   // Alle Schrauben wurden angezogen
        unlock(status_mutex);

}
```

d) Explizite Prozess-/Threadinteraktion

(Quelle: Foliensatz 4: Folie 3ff)
Bei der expliziten Prozessinteraktion lässt sich zwischen den Interaktionsmustern Konkurrenz und Kooperation unterscheiden. Bei Konkurrenz versuchen mehrere Prozesse/Threads auf die selben, exklusiv benutzbaren Betriebsmittel zuzugreifen. Die einzelnen Prozesse/Threads wissen dabei nichts von der Existenz der übrigen Prozesse/Threads bzw. über deren Inhalt. Zur Abstimmung untereinander wird Synchronisation benötigt. Im Gegensatz dazu sind Prozesse/Threads bei der Kooperation aufeinander angewiesen und wissen von der Existenz und Funktionalität ihrer kooperierenden Prozesse. Es findet Kommunikation und Synchronisation zwischen den beteiligten Prozessen statt.
Im betrachteten Beispiel aus Aufgabe 4.3 b) werden Schutzmechanismen für kritische Abschnitte (Mutex) und Signalisierungen (Signal) zur Synchronisation verwendet. Somit werden Mutex für exklusiv benutzbare Betriebsmittel (hier Status-Variable) und Signale (Position des Arms, Auto vorgeschoben) als Synchronisationsmechanismus verwendet. Es handelt sich um das Interaktionsmuster Konkurrenz, da keiner der beiden Threads (Beförderungsband, Schraubmaschine) von der Existenz oder Funktionalität des anderen Kenntnis besitzt.