Systemprogrammierung, Sommersemester 2016

Übungsblatt 4, Theorie

Gruppe: Hristo Filaretov, Robert Focke, Mikolaj Walukiewicz

Quellen:

- Kao, Odej. "Systemprogrammierung". 2016. Vorlesungsfolien "3. Scheduling".
- Kao, Odej. "Systemprogrammierung". 2016. Vorlesungsfolien "4. Prozesskoordination".

Aufgabe 4.1:

a) Notwendige Bedingung für Rate Monotonic Scheduling lautet:

Summe von bi / pi <= 1

Das ist hier erfüllt

Hinreichend RMS-Kriterium:

b - Bedienzeit

p - Period

n - Anzahl von Perioden

$\sum \frac{b_i}{p_i} < n(2^{\frac{1}{n}} - 1)$

Das ist hier erfüllt.

Also, Rate Monotonic Scheduling darf benutzt werden.

Beim Rate Monotonic Scheduling gilt: * Die Priorität jedes Prozesses ist umgekehrt proportional zu der Periode (Prozesse mit kleiner Periode haben höherer Priorität)

Die Hyperperiode ist also: 24

b) Würde noch ein Prozess (D: 4, 2, 6) hinzugefügt, dann ist die notwendige Bedingung nicht erfüllt:

1/3 + 2/8 + 4/12 + 4/6 = 3*1/3 + 1/4 > 1

c) Die Unterschied zwischen Hard- und Softlimits lässt sich durch die sogenannte Nutzenfunktion erläutert werden. Beschrieben zwei Zeitpunkten t_min und t_max ein Zeitbereich in dem wir eine spezifische Reaktion aus einem Prozess erwartern, und würde die Nützlichkeit dieser Reaktion von 0 bis 100% bewertet, dann haben Hard- und Softlimits die folgende Eigenschaften:

- Hardlimits: Bei einem völlig Hardlimit Echtzeitsystem ist die Reaktion nur dann nützlich, wenn sie völlig innerhalb t_min und t_max erfolgt. Also, die Nützlichkeit N zwischen t_min und t_max ist 100%, für alle andere Zeitpunkten ist es 0%.
- Softlimits: Bei einem Softlimit Echtzeitsystem steigt die Nützlichkeit graduell von 0% nach 100% als t -> t_min geht, dann bleibt die Nützlichkeit auf 100% für den Zeitbereich [t_min, t_max] und sinkt dann wieder graduell von 100% auf 0%.

Diese Nützlichkeit beschreibt die Wichtigkeit, das Ergebnis eines Prozesses innerhalb einem bestimmten Zeitbereich zu bekommen. Die Mehrheit von Systemen können als eine Mischung zwischen Hard- und Softlimit Systeme bezeichnet werden.

In dem Beispiel aus dieser Aufgabe, ist hat das System ein Hardlimit - es ist sehr wichtig die Reaktion innerhalb des Zeitbereiches zu bekommen, andererweise können Kollisionen entstehen.

Ein Programm, das ein Würfel für ein Spiel simuliert und eine zufällige Zahl ausgibt, hat ein Softlimit. Meistens wuerde eine Verspaetung zu keinen grossen Kosten oder Unfaelle fuehren.

Aufgabe 4.2

a)

- Setting 1 - Das Programm ist nebenläufig, es lässt sich also nebenläufig ausführen, jedoch nicht parallel, weil es nur einen Prozessor gibt.
- Setting 2 - Das Programm ist nebenläufig und lässt sich parallel ausführen, weil der Mehrkern-Prozessor mehrere Rechenkerne hat und kann mehrere Prozesse parallel ausführen. (in diesem Fall - 2).
- Setting 3 - Das Programm ist nebenläufig und lässt sich parallel ausführen, solange beide Prozessoren irgendwie kommunizieren (Datei übermitteln) können.

b)

- Race conition - ein Race Condition entsteht, wenn das Ergebnis oder problemlose Funktionieren eines Programms von einer unkontrollierte Reihenfolge von Einzeloperationen abhängt.
- Max' Programm In diese Implementation könnte der Fall entstehen, das beide Variable - t1_next und t2_next, gleichzeitig auf 1 gesetzt sind und dann könnte das Programm fehlerhaft funktionieren. Dass würde passieren, wenn die Instruktionen irgendwie in folgender Reihenfolge ausgeführt sind:

1. Thread 2, Line 7-8: t1_next = 1 && t2_next = 0
2. Thread 1, Line 8: t2_next = 1;

c)

Listing 1: Global

```
boolean signal_PING = true;
boolean signal_PONG = false;

void signal(boolean s){
    s = true;               // set signal
}

void wait(boolean s){
    while(s == false){ ; }  // wait for signal (s = true)
    s = false;
}
```

Listing 2: Thread 1

```
wait(signal_PING)

printf("P");
printf("I");
printf("N");
printf("G");

signal(signal_PONG);
```

Listing 3: Thread 2

```
wait(signal_PONG)

printf("P");
printf("O");
printf("N");
printf("G");

signal(signal_PING);
```

---

d) Schon mit dieser Implementation entsteht kein Race Condition. Jeder von den zwei Prozessen würde nur dann laufen, wenn der andere schon vollig ausgeführt ist.