Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

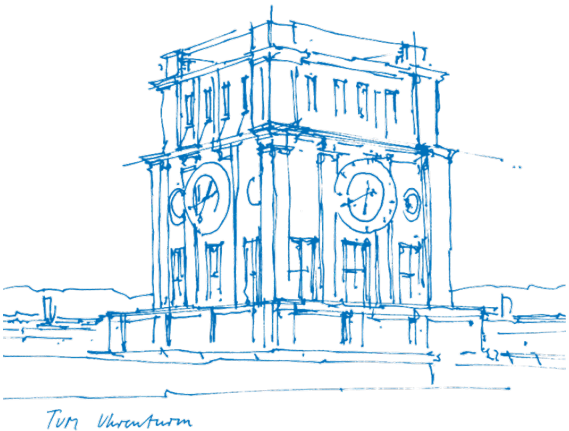
Grundlagen: Betriebssysteme und Systemsoftware

Tutorübung

Mario Delic

Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

Übungswoche 3



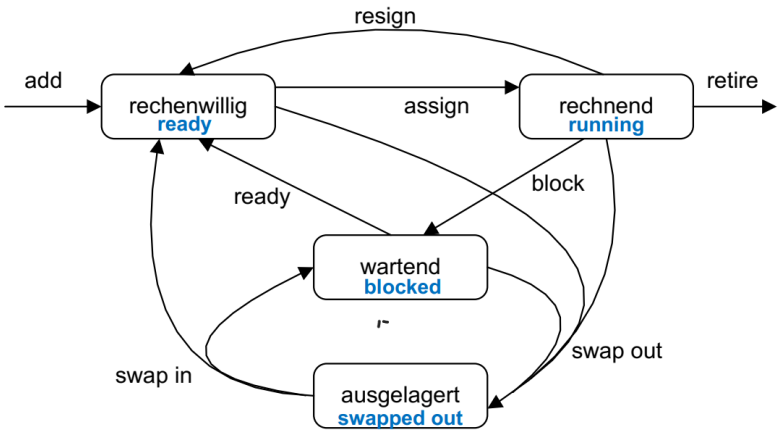
Prozesse
Basics

Ein **Prozess** stellt ein Programm in Ausführung da. Er gruppiert Ressourcen und besitzt einen Kontrollfluss.

Threads stellen einen Kontrollfluss dar. Sie sind **Aktivitätsträger**.

User-Level Threads: Durch eine Programmbibliothek im implementiert. BS hat keine Kenntnis. 'Simulierte' Nebenläufigkeit.
Kernel-Level Threads: Der Kernel sieht und verwaltet die Threads. 'Echte' Nebenläufigkeit möglich.

Prozesse
Zustände



Prozesse
Erzeugung

fork(): Systemcall zur Erzeugung eines Kindprozesses.
Der Kindprozess ist eine **Kopie** des Elternprozesses und erbt seine Daten. **Copy-on-write** memory wird dabei erst kopiert, wenn **schreibend** darauf zugegriffen wird.
Return-value: im parent = child PID; im child = 0.
(**kill(int pid, int signal)**: Sende ein Signal an Prozess pid; **exit(...)**: Beendet Prozess.)

pthread_create(...): Funktion zur erstellung eines neuen Threads.
(**phtread_join(pthread t, ...)** Aufrufernder Thread wartet bis t fertig ist; **phtread_exit(...)**: Thread wird beendet.)

Für Details → **man pages** (man 1 = shellcommands (cat, grep, git, man...), man 2 = Systemcalls (fseek, open, sched_yield...), man 3 = Programmbibliotheksfunktionen (phtread_create, malloc, puts, qsort...)

Prozesse
Attribute

Per-process items	Per-thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Aufgabe 1
Scheduling

Es seien 3 Prozesse (P_1, P_2, P_3) gegeben.
Ihre Ankunftszeiten am Scheduler (a_1, a_2, a_3) seien $(0, 5, 2)$.
Ihre Rechenzeiten (r_1, r_2, r_3) betragen $(7, 3, 4)$.

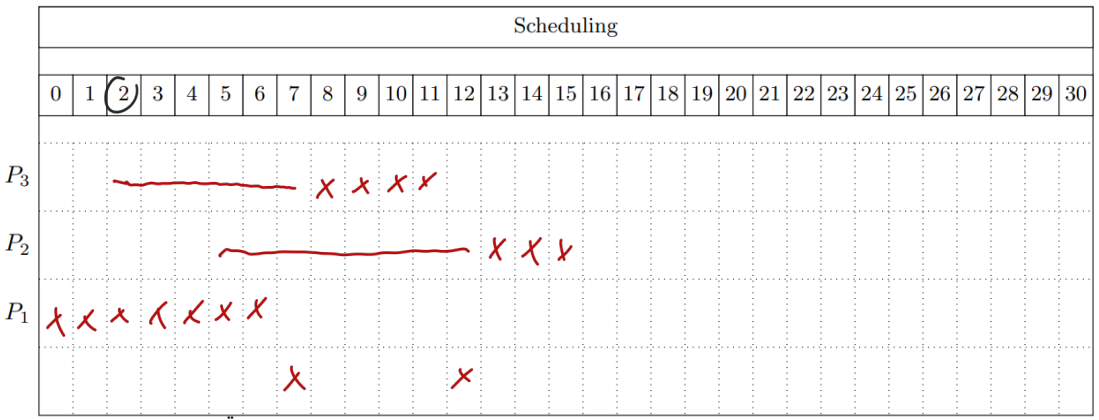
Nehmen Sie an, dass ein **Kontextwechsel eine Zeiteinheit benötigt**. Die Aktivierung des Schedulers kann in dieser Aufgabe vernachlässigt werden. Modellieren Sie den Scheduler/Dispatcher als einen eigenständigen Prozess.

Skizzieren Sie unter diesen Annahmen den Ablauf der Prozesse in einem Gantt-Diagram für folgende Schedulingstrategien.
Hinweis: Vernachlässigen Sie den initialen Kontextwechsel. Beginnen Sie im ersten Zeitslot mit dem ersten rechnenden Prozess.

Aufgabe 1a: First-Come-First-Served

FCFS: Non-preemptive, Prozesse werden in der Reihenfolge ihrer Ankunftszeiten abgearbeitet.

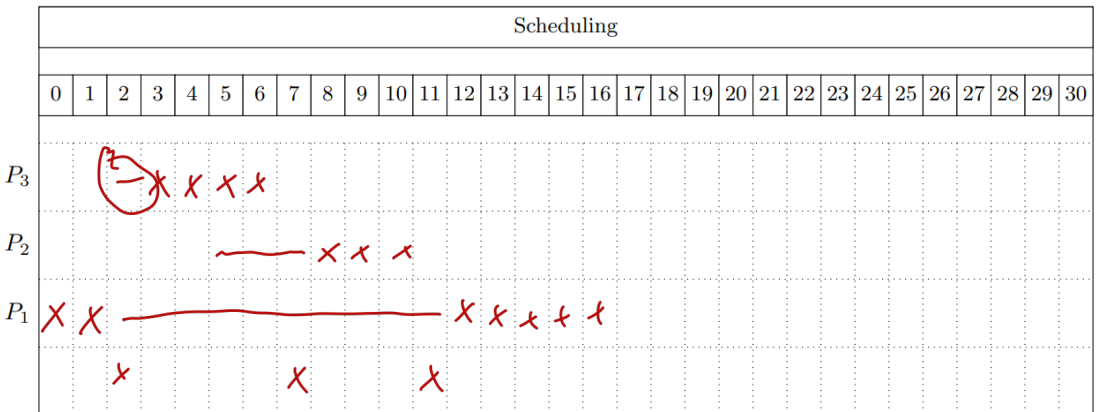
$\vec{P} = (P_1, P_2, P_3) \rightarrow \vec{a} = (0, 5, 2); \vec{r} = (7, 3, 4)$



Aufgabe 1b: Shortest Remaining Time Next

SRTN: Preemptive, Auswahl des Prozesses mit der kürzesten verbleibenden Rechenzeit, Unterbrechungen erfolgen nur beim Eintreffen eines neuen Prozesses.

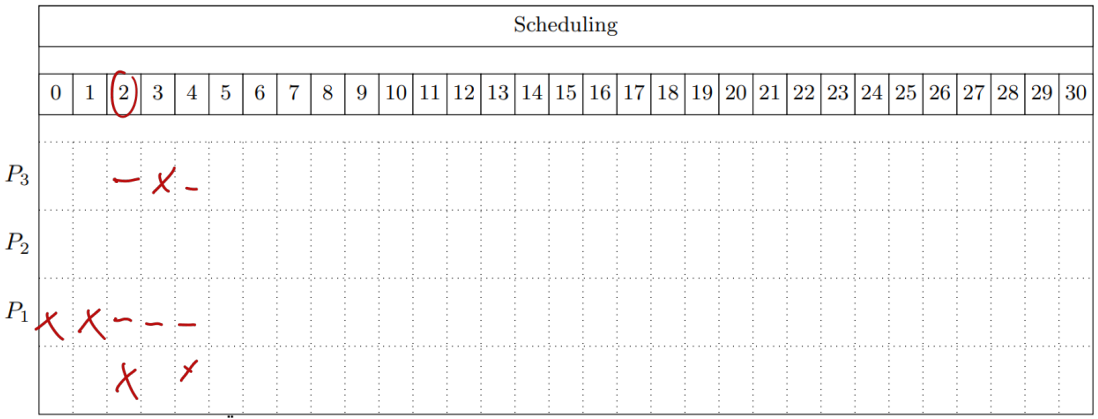
$\vec{P} = (P_1, P_2, P_3) \rightarrow \vec{a} = (0, 5, 2); \vec{r} = (7, 3, 4)$



Aufgabe 1c: Round Robin

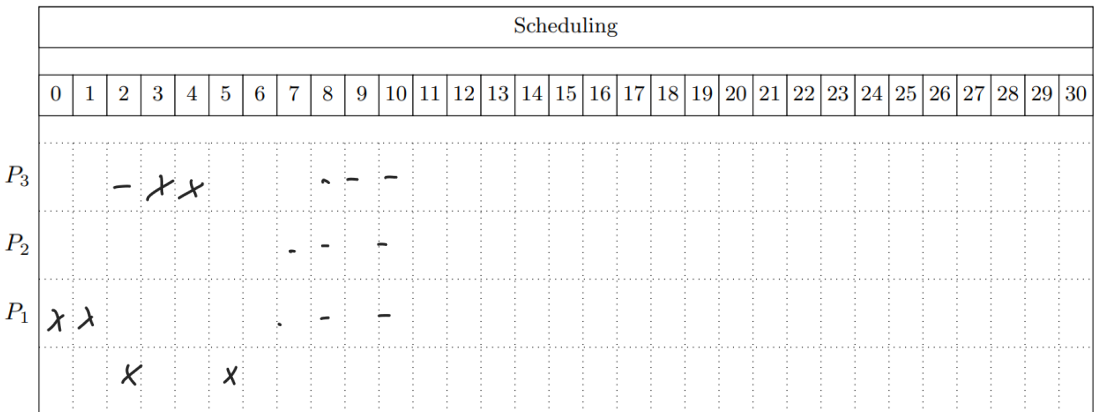
RR mit einem Zeitquantum von einer Zeiteinheit und zyklischer Abarbeitung der Prozesse (statische Prioritäten, Sortierung nach der PID (=Index))

$\vec{P} = (P_1, P_2, P_3) \rightarrow \vec{a} = (0, 5, 2); \vec{r} = (7, 3, 4)$



Aufgabe 1d: Round Robin 2

RR mit einem Zeitquantum von 2 Zeiteinheiten und zyklischer Abarbeitung der Prozesse (statische Prioritäten, Sortierung nach der PID (=Index)).
$\vec{P} = (P_1, P_2, P_3) \rightarrow \vec{a} = (0, 5, 2); \vec{r} = (7, 3, 4)$



Aufgabe 2: Priority RR Scheduling

Priorisiertes RR Verfahren:

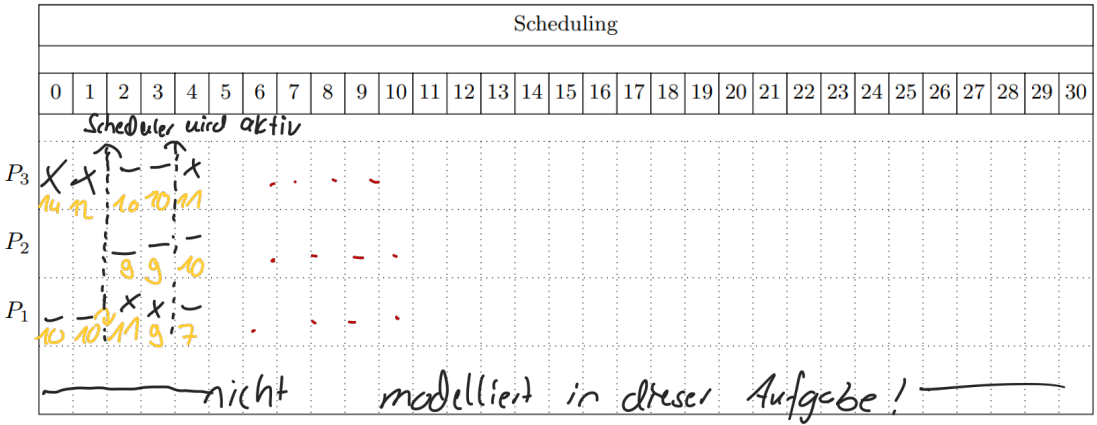
- Zeitquantum q = 2 Zeitenheiten
- Initialprioritäten $(I_1, I_2, I_3) = (10, 9, 14)$
- Ankunftszeiten: $(0, 2, 0)$
- Rechenzeiten: $(6, 6, 8)$

  - rechnend: Priorität -2/1ZE

  - wartend: Priorität +1/2 ZE

Aufgabe 2: Priority RR Scheduling

$\vec{P} = (P_1, P_2, P_3) \rightarrow \vec{I} = (10, 9, 14)$; $\vec{a} =$ (0,2,0); $\vec{r} =$ (6,6,8)



Aufgabe 2: Priority RR Scheduling

Berechnen Sie die mittlere Wartezeit $\overline{W}$ und die mittlere Verweilzeit $\overline{V}$ für dieses Szenario.

$$\overline{W} = \frac{\sum_{i=1}^{n} w_i}{n} \quad \overline{V} = \frac{\sum_{i=1}^{n} v_i}{n}$$

Mittlere Verweilzeit $\overline{V}$:

Mittlere Wartezeit $\overline{W}$:

Aufgabe 2: Priority RR Scheduling

Berechnen Sie die mittlere Wartezeit $\overline{W}$ und die mittlere Verweilzeit $\overline{V}$ für dieses Szenario.

$$\overline{W} = \frac{\sum_{i=1}^{n} w_i}{n} \quad \overline{V} = \frac{\sum_{i=1}^{n} v_i}{n}$$

Mittlere Verweilzeit $\overline{V}$:

- v_1: 16 Zeiteinheiten
- v_2: 18 Zeiteinheiten
- v_3: 18 Zeiteinheiten
- $\overline{V} = (16 + 18 + 18)/3 = 52/3$

Mittlere Wartezeit $\overline{W}$:

Aufgabe 2: Priority RR Scheduling

Berechnen Sie die mittlere Wartezeit $\overline{W}$ und die mittlere Verweilzeit $\overline{V}$ für dieses Szenario.

$$\overline{W} = \frac{\sum_{i=1}^{n} w_i}{n} \quad \overline{V} = \frac{\sum_{i=1}^{n} v_i}{n}$$

Mittlere Verweilzeit $\overline{V}$:

- v_1: 16 Zeiteinheiten
- v_2: 18 Zeiteinheiten
- v_3: 18 Zeiteinheiten
- $\overline{V} = (16 + 18 + 18)/3 = 52/3$

Mittlere Wartezeit $\overline{W}$:

- w_1: 10 Zeiteinheiten
- w_2: 12 Zeiteinheiten
- w_3: 10 Zeiteinheiten
- $\overline{W} = (10 + 12 + 10)/3 = 32/3$

Aufgabe 3
Noch mehr C

a) Betrachten Sie die nachfolgende Implementierung einer Bibliotheksfunktion. Um welche Funktion handelt es sich? Was ist natürlichsprachlich die Abbruchbedingung?

```
void fct(char *s, const char *t) {
    while(*s++ = *t++);
}
```

b) Wie unterscheiden sich die folgenden Typdeklarationen? Es gilt: `sizeof(void*)==8` und `sizeof(int)==4`

```
struct v {
    char a;
    short h;
    struct v *o;
}
```

Listing 1: Variante 1

```
struct v {
    struct v *o;
    short h;
    char a;
}
```

Listing 2: Variante 2

Aufgabe 3
Noch mehr C

a) Betrachten Sie die nachfolgende Implementierung einer Bibliotheksfunktion. Um welche Funktion handelt es sich? Was ist natürlichsprachlich die Abbruchbedingung?

```
void fct(char *s, const char *t) {
    while(*s++ = *t++);
}
```

s und t sind Anfänge von strings. Der Funktion heißt strcpy(). Der string t wird Byte für Byte in s kopiert.
Der return-value eines Assignemnts (=) ist das, was zugewiesen wurde. Die while-Schleife terminiert also sobald ein NULL-Byte assigned wurde, sprich: das Ende des strings t erreicht wurde.

Aufgabe 3
Noch mehr C

b) Wie unterscheiden sich die folgenden Typdeklarationen? Es gilt: `sizeof(void*)==8` und `sizeof(short)==2`

```
struct v {
    char a;
    short h;
    struct v *o;
}
```

Listing 1: Variante 1

```
struct v {
    struct v *o;
    short h;
    char a;
}
```

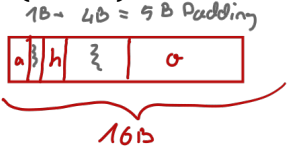
Listing 2: Variante 2

Aufgabe 3
Noch mehr C

b) Wie unterscheiden sich die folgenden Typdeklarationen? Es gilt: `sizeof(void*)==8` und `sizeof(short)==2`

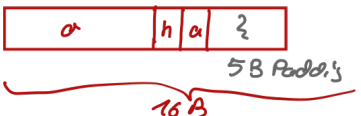
```
struct v {
    char a;
    short h;
    struct v *o;
}
```

Listing 1: Variante 1

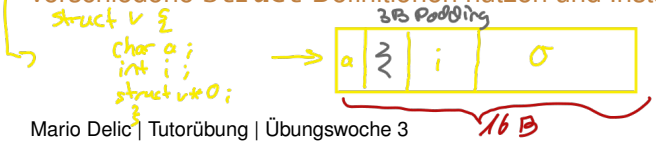



```
struct v {
    struct v *o;
    short h;
    char a;
}
```

Listing 2: Variante 2



Alignment-Anforderungen führen zu Padding. Padding richtet sich nach dem größtem Element des structs (hier: Pointer). Die Größe ist für beide `structs` identisch. Das Layout im Speicher ist aber sehr wohl unterschiedlich. C sortiert die Elemente einer Struktur nicht um. Es kann bei `structs` zu Problemen kommen, wenn Softwarekomponenten verschiedene `struct`-Definitionen nutzen und Instanzen davon austauschen.



Aufgabe 3
Noch mehr C

c) Betrachten Sie folgendes C-Programm, welches die n-te harmonische Zahl $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}$ berechnet.

Beschreiben Sie etwaige Programmierfehler, die im obigen Programm gemacht wurden und erklären Sie kurz, wie sie sich auf das Programm auswirken.

Aufgabe 3

```c
#include <stdio.h>
#include <stdlib.h>

// Returns the first n harmonic numbers
double* harmonic_numbers(unsigned int n) {
    double result[n];
    result [0] = 1.0;

    for (unsigned int i = 1; i < n; i++) {
        result[i] = result[i-1] + (1.0 / (double) (i + 1));
    }
    return result;
}

void print_harmonics(unsigned int n) {
    if (n == 0) return;
    double *result = harmonic_numbers(n);
    for (unsigned int i = 0; i < n; i++) {
        printf("%f\n", result[i]);
    }
}
// [...]
```

Aufgabe 3

Da das Array result nicht mit malloc alloziert wird landet es auf dem Stackbereich seiner Funktion (harmonic_ numbers).

Nachdem eine Funktion returned, wird der Stackbereich der Funktion wieder freigegeben! Greift man auf die Daten an der Stelle auf die der Pointer result zeigt (Pointer, der auf das erste Element des Arrays zeigen sollte) zu, so handelt es sich um undefined behavior!

Bonus: Variable Length Arrays (VIA) wie result [n] werden für bestimmte Berechnungen gern verwendet, da sie i. d. R. auf dem Stack alloziert werden und deswegen effizienter sind als auf dem heap allozierte Arrays. Aufgrund der Anfälligkeit für Stack-Overflow Angriffe sind sie in Betriebssystemen eher unvorteilhaft.