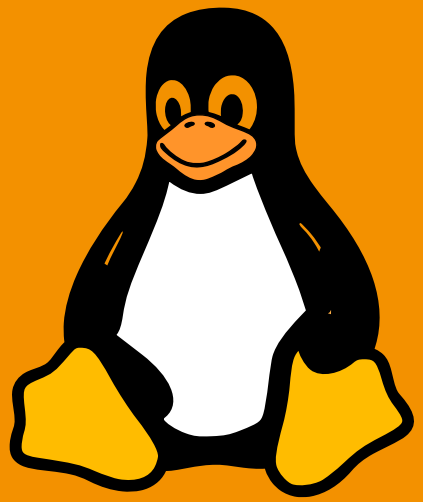
Technische Hochschule Rosenheim
Technical University of Applied Sciences

Prof. Florian Künzner
Technical University of Applied Sciences Rosenheim, Computer Science

OS 6 – Synchronisation 1



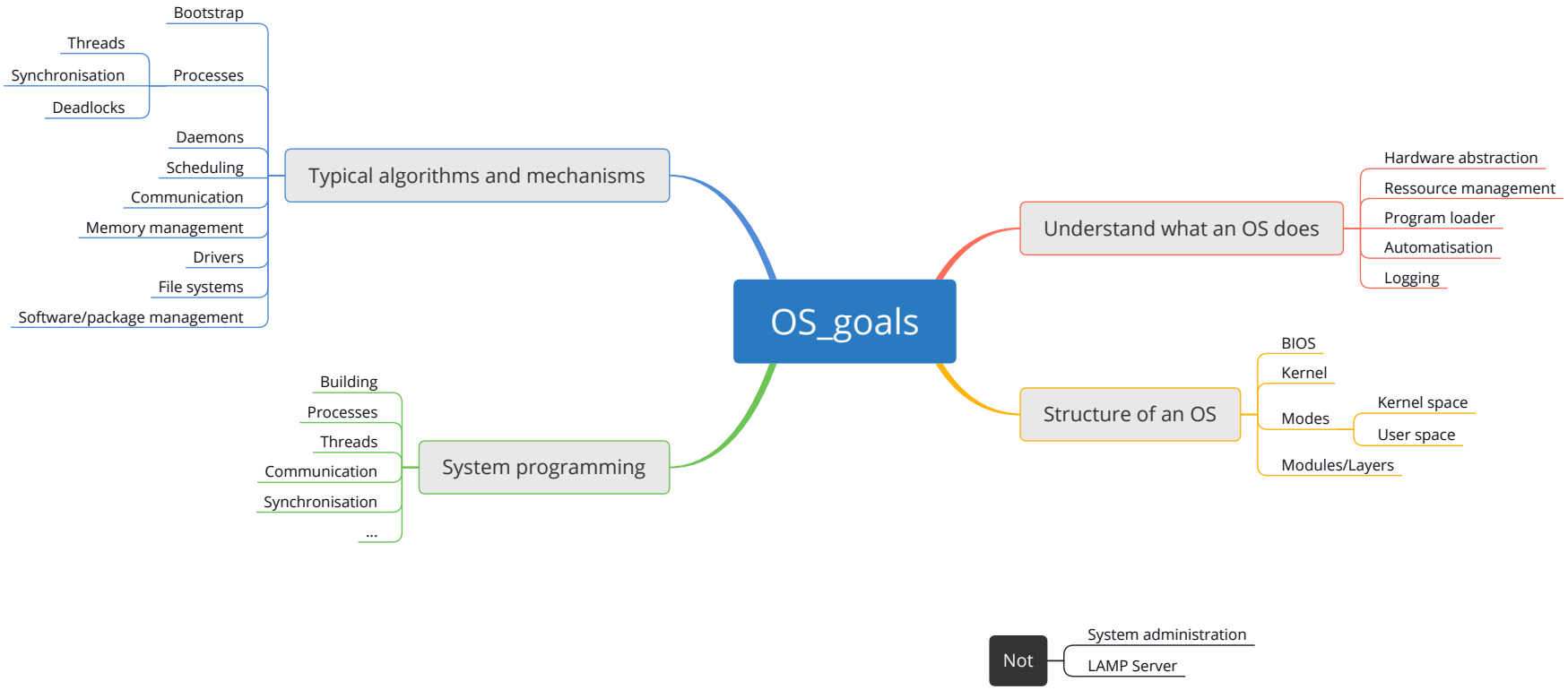
source: iconspng.com

The lecture is based on the work and the documents of Prof. Dr. Ludwig Frank

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal

OS::Synchronisation

- Understand synchronisation problem
- Mutual exclusion
- Semaphore (theoretical, practical)
- Lock-Files

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intro

Parallelisation with processes and threads is nice, but...

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

The problem (1)

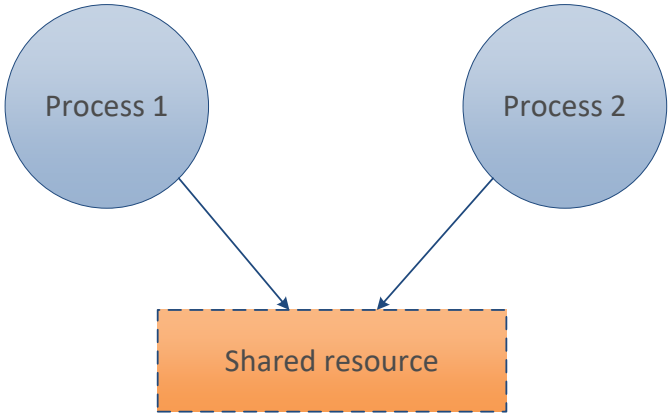
- Cause
  - Parallel read/write
  - Parallel use
- Problem
  - Read of unfinished data
  - (Partial) overwrite of data
- May occur **sporadic**: looks like undefined behaviour
- These kind of **bugs** are often **very hard to find**

It is called **race condition** (Konkurrenzbedingung)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

The problem (2)



Critical section

```
1 { //critical_section
2   //work with shared resource:
3   //- read/write
4   //- use
5 }
```

Solution: Mutual exclusion

"Gegenseiter Ausschluss"

- **Only one** process can **access** the **critical section**
- **Others** have to **wait**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

A problematic example

```
1  int global_counter = 0;

2  void* thread1() {
3    while(1) {
4      //increase counter
5      int counter = global_counter;
6      counter = counter + 1;
7      global_counter = counter;
8
9      produce_something(counter);
10   }
11 }
22 int main() {
23     //start threads...
24 }
```

```
12 void* thread2() {
13   while(1) {
14     //increase counter
15     int counter = global_counter++;
16
17
18
19     produce_something(counter);
20   }
21 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Towards synchronisation

Towards a synchronisation solution...

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Idea 1: Lock variables

```
 1 int global_counter = 0;
 2 int global_lock = 0;

 3 void* thread1() {
 4   while(1) {
 5     while(lock == 1) {} //busy wait
 6
 7     lock = 1;
 8       //increase counter
 9       int counter = global_counter;
10       counter = counter + 1;
11       global_counter = counter;
12     lock = 0;
13   }
14 }
```

```
15 void* thread2() {
16   while(1) {
17     while(lock == 1) {} //busy wait
18
19     lock = 1;
20       //increase counter
21       int counter = global_counter;
22       counter = counter + 1;
23       global_counter = counter;
24     lock = 0;
25   }
26 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Idea 1: Lock variables (analysis)

```
1  int global_counter = 0;
2  int global_lock = 0;

3  void* thread1() {
4    while(1) {
5      while(lock == 1) {} //busy wait
6      //thread1 see: lock==0
7      //!! INTERRUPT: activate thread2 !!
8
9
10
11
12
13
14     lock = 1;
15     //...
16   }
17 }
```

```
18 void* thread2() {
19    while(1) {
20
21
22
23       while(lock == 1) {} //busy wait
24
25       lock = 1;
26         //increase counter
27         int counter = global_counter;
28         //!! INTERRUPT: activate thread1 !!
29
30
31    }
32 }
```

Problem: Both threads are in the critical section. **Solution useless!!!**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Idea 2: Disable interrupts

```
1  int global_counter = 0;
2
3  void* thread1() {
4    while(1) {
5      disable_interrupts();
6
7      //increase counter
8      int counter = global_counter;
9      counter = counter + 1;
10     global_counter = counter;
11
12     enable_interrupts();
13
14     produce_something(counter);
15   }
16 }
```

Pro

- Easy solution

Con

- Only works on single core CPUs
- May disturb the scheduling
- May disturb the realtime behaviour
- Some interrupts can't be deactivated (depends on hardware)
- Danger: A process/thread doesn't activate interrupts again
- Program error in critical section

Conclusion

- Only in some parts of the OS kernel possible

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Idea 3: Test-and-set CPU command

- **Problem** of the previous ideas: **Read and write** (set) of lock **can be interrupted**.
- **Solution**: Make read and write (set) to an **atomic operation**.
- `If lock==0: condition==0: then set lock=1`
- `If lock!=0: condition==1`

```
1 //HW specific assembler command
2 Loop: TS  lock              //test and set variable lock
3       BNZ Loop              //branch on not zero : lock==1
4       //critical section    //can be entered if : lock==0
5       //...
6       MVI lock, 0           //set zero back
```

Pro

- Easy to implement if HW supports it
- Can be used for any number of processes/threads
- Can be used on multicore CPUs

Con

- Busy wait: Waste of computing time
- Depending on the good behaviour of all processes/threads
- It is still assembler code: Not C
- Problematic if a process exits inside the critical area

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Semaphore

A working solution with semaphores...

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Semaphore: Idea

Idea Instead of busy wait, a process/thread blocks (sleep) until the critical area is free.

Operations

Operation	Description
`seminit(s, value)`	Creates and **initialises a semaphore** with a value. The value is a number that specifies the number of processes that can simultaneously enter the critical area.
`P(s)`	**Wait** until the critical area is free (`value--`).
`V(s)`	**Releases** the critical area (`value++`).

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Semaphore: Usage

Basic usage

```
1 seminit(s, 1);
2
3 P(s);
4 //critical area..,
5 V(s);
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Semaphore: Types

Types	Initialisation	Description
Mutex	`seminit(s, 1)`	A **mutex** semaphore is used for **mutual exclusion**. Typically initialised with 1.
Binary	`seminit(s, 0)`	A **binary** semaphore is used when there is only **one shared resource**. Initialisation with 0/1 possible.
Counting	`seminit(s, N)`	A **counting semaphores** is used to handle more than one shared resource. Typically initialised with the number N of shared resources. Initialisation with 0/N possible.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Semaphore: The role of the OS

The OS

- provides semaphores.
- ensures that the `P()/V()` operations are atomic.

The OS can reach this with

- disable process changes (temporarily).
- disable interrupts (temporarily) (also process changes are not possible than).
- **use of a test-and-set CPU command**.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Semaphore: Example implementation

Pseudo C code

```
1  //Semaphore struct with a value and
2  //an internal list of waiting
3  //processes/threads
4  struct Semaphore
5  {
6    int value;
7    struct ProcessList process_list;
8  };
9  //initialises a semaphore with a value
10 void seminit(struct Semaphore* s, int value)
11 {
12   s->value=value;
13 }
14 void P(struct Semaphore* s)
15 {
16   if (s->value > 0) {
17     s->value--;
18   } else {
19     append_to(pid, s->process_list);
20     sleep(); //sleep indefinitely
21   }
22 }
23 void V(struct Semaphore* s)
24 {
25   if (is_empty(s->process_list)) {
26     s->value++;
27   } else {
28     int pid=pop_any(s->process_list);
29     wakeup(pid);
30   }
31 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Mutual exclusion: Pseudo C code

```
 1 int global_counter = 0;
 2 seminit(s, 1); //declare and initialise semaphore

 3 void* thread1() {
 4   while(1) {
 5     P(s);
 6       //increase counter
 7       int counter = global_counter;
 8       counter = counter + 1;
 9       global_counter = counter;
10     V(s);
11   }
12 }
13 void* thread2() {
14     while(1) {
15       P(s);
16         //increase counter
17         int counter = global_counter;
18         counter = counter + 1;
19         global_counter = counter;
20       V(s);
21     }
22 }
23 int main() {
24     //start threads...
25 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Mutual exclusion: Example C code

Mutual exclusion with the POSIX semaphore API and named semaphores.

`sem_overview`: `http://man7.org/linux/man-pages/man7/sem_overview.7.html`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Mutual exclusion: Example C code

Includes and definitions

```
1  #include <stdio.h>     //printf, perror
2  #include <stdlib.h>    //EXIT_FAILURE, EXIT_SUCCESS
3  #include <fcntl.h>     //flags: O_CREAT, O_EXCL
4  #include <semaphore.h> //sem_open, sem_wait, sem_post, sem_close
5  #include <pthread.h>   //pthread_*
6
7  #define SEMAPHORE_NAME "/global_counter" //name of semaphore
8  sem_t*   semaphore = NULL;               //pointer to semaphore
9  const int     PERM = 0600;               //Permission to the semaphore
10
11 int global_counter = 0;                  //global counter
12 const int         N = 100000;           //Number of iterations per thread
```

- semaphore.h: http://man7.org/linux/man-pages/man7/sem_overview.7.html
- https://www.softprayog.in/programming/posix-semaphores

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Mutual exclusion: Example C code

Semaphore functions

```
15 void create_semaphore() {
16     semaphore = sem_open(SEMAPHORE_NAME, O_CREAT, PERM, 1);
17     if(semaphore == SEM_FAILED){
18         perror("Error when creating the semaphore ...\n");
19         exit(EXIT_FAILURE);
20     }
21 }
22
23 void delete_semaphore() {
24     if(sem_close(semaphore) == -1){
25         perror("Error can't close semaphore ...\n");
26         exit(EXIT_FAILURE);
27     }
28
29     if(sem_unlink(SEMAPHORE_NAME) == -1) {
30         perror("Error can't delete (unlink) semaphore ...\n");
31         exit(EXIT_FAILURE);
32     }
33 }
```

- sem_open(): http://man7.org/linux/man-pages/man3/sem_open.3.html
- sem_close(): http://man7.org/linux/man-pages/man3/sem_close.3.html
- sem_unlink(): http://man7.org/linux/man-pages/man3/sem_unlink.3.html

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Mutual exclusion: Example C code

Thread function

```
35 //thread function
36 void* thread() {
37     for(int i = 0; i < N; ++i) {
38         sem_wait(semaphore); //P(s)
39         int counter = global_counter++;
40         sem_post(semaphore); //V(s)
41
42         //produce_something(counter);
43     }
44
45     return NULL;
46 }
```

- sem_wait(): http://man7.org/linux/man-pages/man3/sem_wait.3.html
- sem_post(): http://man7.org/linux/man-pages/man3/sem_post.3.html

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Mutual exclusion: Example C code

Main function

```
48 int main(int argc, char** argv) {
49     create_semaphore();
50
51     //start threads
52     pthread_t thread_id1, thread_id2;
53     pthread_create(&thread_id1, NULL, &thread, NULL); //error handling as usual...
54     pthread_create(&thread_id2, NULL, &thread, NULL);
55
56     //join threads
57     pthread_join(thread_id1, NULL); //error handling as usual...
58     pthread_join(thread_id2, NULL);
59
60     delete_semaphore();
61
62     //print result
63     printf("Counter: %d\n", global_counter);
64
65     return EXIT_SUCCESS;
66 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Linux commands

Named semaphores can be found on
`/dev/shm`

Example:

```
ls -l /dev/shm
-rw------- 1 flo flo 32 Nov 4 15:18 sem.global_counter
```

Remove a semaphore on the shell:

```
rm /dev/shm/sem.global_counter
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Mutual exclusion with lock files

Idea

- Use a file to simulate `P()/V()` operations
- The process/thread that can acquire the file lock can enter the critical section
- Not discussed in detail in this lecture.

flock function

```
1 //flock - apply or remove an advisory lock on an open file
2 int flock(int fd, int operation);
```

For the interested people: `https://linux.die.net/man/2/flock`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary and outlook

Summary

- Synchronisation problems
- Mutual exclusion
- Semaphore (theoretical, practical)
- (Lock-Files)

Outlook

- Producer-consumer problem
- Reader-writer problem
- Monitor concept