Concurrent Systems
*Nebenläufige Systeme*

III. Processes

Wolfgang Schröder-Preikschat

November 8, 2016

---

**Subject Matter**

- discussion on **abstract concepts** as to multiplexing machines:
  - **program**
    - concretized form of an algorithm
    - static sequence of actions to be conducted by a processor
    - of sequential or non-sequential structure
  - **process**
    - a program in execution
    - dynamic sequence of actions conducted by a processor
    - of parallel, concurrent, simultaneous, or interacting nature
Subject Matter

- discussion on **abstract concepts** as to multiplexing machines:
  - program
    - concretized form of an algorithm
    - static sequence of actions to be conducted by a processor
    - of sequential or non-sequential structure
  - process
    - a program in execution
    - dynamic sequence of actions conducted by a processor
    - of parallel, concurrent, simultaneous, or interacting nature
- explanation of **process characteristics** in physical and logical terms
  - appearance of a process as kernel thread and/or user thread
  - sequencing of processes, process states, and state transitions

Process – The Course of Being Done acc. [9], cf. p. 33

Operating systems bring programs to execution by creation, releasing, controlling and timing of processes

Subject Matter

- discussion on **abstract concepts** as to multiplexing machines:
  - program
    - concretized form of an algorithm
    - static sequence of actions to be conducted by a processor
    - of sequential or non-sequential structure
  - process
    - a program in execution
    - dynamic sequence of actions conducted by a processor
    - of parallel, concurrent, simultaneous, or interacting nature
- explanation of **process characteristics** in physical and logical terms
  - appearance of a process as kernel thread and/or user thread
  - sequencing of processes, process states, and state transitions
- a bridging of concurrency/simultaneity concepts and mechanisms
  - on the one hand, program as the means of specifying a process
  - on the other hand, process as medium to reflect simultaneous flows

Operating systems bring programs to execution by creation, releasing, controlling and timing of processes

in computer sciences, a process is unimaginable without a program
- as coded representation of an algorithm, the program specifies a process
- thereby, the program manifests and dictates a specific process
- if so, it even causes, controls, or terminates other processes

Provided that the operating system offers all necessary commands.
Operating systems bring programs to execution by creation, releasing, controlling and timing of processes.

- In computer sciences, a process is unimaginable without a program as coded representation of an algorithm, the program specifies a process thereby, the program manifests and dictates a specific process if so, it even causes, controls, or terminates other processes.
- A program (also) describes the kind of flow (Ger. *Ablauf*) of a process:
  - **Sequential**: a sequence of temporally non-overlapping actions
  - **Parallel**: non-sequential

A program consists of actions (p. 7 ff.).

Consider: Program Flow and Level of Abstraction

*One and the same program flow may be sequential on one level of abstraction and parallel on another.* [8, 10]

Program I

**Definition**

For a certain machine concretised form of an algorithm.
Program I  Problem-Oriented Language Level

Definition
For a certain machine concretised form of an algorithm.

- virtual machine C
  - after editing and
  - before compilation

```c
#include <stdint.h>
void inc64(int64_t *i) {
  (*i)++;
}
```

1 action (line 4)

Program I  Problem-Oriented/Assembly Language Level

Definition
For a certain machine concretised form of an algorithm.

- virtual machine ASM (x86)
  - after compilation
  - before assembly

```assembly
inc64:
  movl 4(%esp), %eax
  addl $1, (%eax)
adcl $0, 4(%eax)
ret
```

3 actions (lines 7–9)

Definition (Action)
The execution of an instruction of a (virtual/real) machine.

```c
gcc -O4 -m32 -static -fomit-frame-pointer -S, also below
```

© wosch CS (WS 2016, LEC 3) Fundamentals–Program

Program I  Problem-Oriented Language Level

Definition
For a certain machine concretised form of an algorithm.

- virtual machine C
  - after editing and
  - before compilation

```c
#include <stdint.h>
void inc64(int64_t *i) {
  (*i)++;
}
```

1 action (line 4)

Program I  Problem-Oriented/Assembly Language Level

Definition
For a certain machine concretised form of an algorithm.

- virtual machine ASM (x86)
  - after compilation
  - before assembly

```assembly
inc64:
  movl 4(%esp), %eax
  addl $1, (%eax)
adcl $0, 4(%eax)
ret
```

3 actions (lines 7–9)

Definition (Action)
The execution of an instruction of a (virtual/real) machine.

```c
gcc -O4 -m32 -static -fomit-frame-pointer -S, also below
```

© wosch CS (WS 2016, LEC 3) Fundamentals–Program
Program II
Operating-System Machine Level

- address space and virtual machine SMC³
  - text segment
  - Linux
  - after linking/binding and
  - before loading

1 0x80482f0: mov 0x4(%esp),%eax
2 0x80482f4: add $0x1,(%eax)
3 0x80482f7: adc $0x0,0x4(%eax)
4 0x80482fb: ret

Virtual machine SMC³ after linking/binding and before loading

mov 0x4 (% esp ) ,% eax
add $0x1 ,(% eax )
adc $0x0 ,0 x4 (% eax )
ret

Real machine after loading executable

8b 44 24 04
83 00 01
83 50 04 00
c3

Same number of actions (lines 1–3, resp.), but different forms of representation

Hint (ret or c3, resp.)

The action for a subroutine return corresponds to the action of the corresponding subroutine call (gdb, disas /rm main):

1 0x80481c9: c7 04 24 b0 37 0d 08 movl $0x80d37b0,(%esp)
2 0x80481d0: e8 1b 01 00 00 call 0 x80482f0 <inc64>

³Symbolic machine code: x86 + Linux.

© wosch CS (WS 2016, LEC 3) Fundamentals–Program 8

Non-Sequential Program I

Definition
A program P specifying actions that allow for parallel flows in P itself.
Non-Sequential Program I

Definition
A program $P$ specifying actions that allow for parallel flows in $P$ itself.

An excerpt of $P$ using the example of POSIX Threads [4]:

```c
pthread_t tid;
if (!pthread_create(&tid, NULL, thread, NULL)) {
    /* ... */
    pthread_join(tid, NULL);
}
```

The parallel flow allowed in $P$ itself:

```c
void * thread(void * null) {
    /* ... */
    pthread_exit(NULL);
}
```

Non-Sequential Program II

Despite actions of parallelism, sequential flows of the same program:

```c
pid_t pid;
if (!(pid = fork())) {
    /* ... */
    exit(0);
} else {
    fork duplicates the address space $A$ of $P$, creates $A'$ as a copy of $A$
    within $A$ as source address space arises thereby no parallel flow, however
    independent of the degree of parallelism within $P$, fork sets it to 1 for $A'$
    sequential flows can establish parallel ones within a domain that logically comprises those sequential flows
}
```
Non-Sequential Program II

- Despite actions of parallelism, sequential flows of the same program:

```c
pid_t pid;

if (!(pid = fork())) {
    /* ... */
    exit(0);
}
wait(NULL);
```

- Sequential flows can establish parallel ones within a domain that logically comprises those sequential flows.

- The shown actions cause parallel flows within an operating system.

- Multiprocessing (Ger. *Simultanbetrieb*) of sequential programs requires the operating system in the shape of a non-sequential program.

- Serviceable characteristic is multithreading within the operating system.

The exception (strictly cooperative systems) proves the rule.

© wosch CS (WS 2016, LEC 3) Fundamentals – Program 10

---

Multiprocessing of Sequential Programs

- **address space A**

  - Directions
    - fork()
    - wait(NULL)

  - fork duplicates the address space A of P, creates A′ as a copy of A.

  - Independent of the degree of parallelism within P, fork sets it to 1 for A′.

  - Sequential flows can establish parallel ones within a domain that logically comprises those sequential flows.

  - The shown actions cause parallel flows within an operating system.

  - Multiprocessing (Ger. *Simultanbetrieb*) of sequential programs requires the operating system in the shape of a non-sequential program.

  - Serviceable characteristic is multithreading within the operating system.

  - Concept "operating system" is epitome of "non-sequential program".

© wosch CS (WS 2016, LEC 3) Fundamentals – Program 11

---
Multiprocessing of Sequential Programs

address space A

parent

fork()

wait(NULL)

child

sequential program flows

address space A'

duplicate

/* ... */

extit(0)

operating system

address space OS

operating system

non-sequential program flow

processor (core) characteristic:

Uni  operated by a process-based operating system, namely:
    pseudo-parallelism by means of processor (core) multiplexing

Multi  ditto; but also event-based operating system, namely:
    real parallelism by means of processor (core) multiplication

© wosch CS (WS 2016, LEC 3) Fundamentals–Program 11
Multiprocessing of Sequential Programs

parent
fork()
wait(NULL)
/* ... */
exit(0)

operating system non-sequential program flow
sequential program flows
address space A address space A'
duplicate
child
address space OS

processor (core) characteristic:
- Uni: operated by a process-based operating system, namely: pseudo-parallelism by means of processor (core) multiplexing
- Multi: ditto; but also event-based operating system, namely: real parallelism by means of processor (core) multiplication
- both cause parallel processes (p. 16) within the operating system

Process

Definition (Program flow)
A program in execution.

the program specifies a sequence of actions that are to be executed
- its kind depends on the particular level of abstraction (cf. p. 34)
  - level 5 -> program statement
  - level 4 -> assembly mnemonic
  - level 3 -> machine instruction
  - level 2 -> microprogram directive
- the actions of a processor thus are not imperatively indivisible (atomic)
  - this particularly holds both for the abstract (virtual) and real processor
Process

Definition (Program flow)
A program in execution.

- the program specifies a sequence of actions that are to be executed
- its kind depends on the particular level of abstraction (cf. p. 34)
  - level \(5\) \(\mapsto\) program statement \(\geq 1\) assembly mnemonics
  - level \(4\) \(\mapsto\) assembly mnemonic \(\geq 1\) machine instructions
  - level \(3\) \(\mapsto\) machine instruction \(\geq 1\) microprogram directives
  - level \(2\) \(\mapsto\) microprogram directive

- the actions of a processor thus are not imperatively indivisible (atomic)
  - this particularly holds both for the abstract (virtual) and real processor
- this sequence is static (passiv), while a process is dynamic (active)

Hint (Process \(\neq\) Process instance)
A process instance (Ger. Exemplar) is incarnation of a process.¹

*Just as an object is a “core image” of a class.

Indivisibility I

Definition
Being indivisible, to keep something appear as unit or entireness.

- a question of the “distance” of the viewer (subject) on an object

<table>
<thead>
<tr>
<th>level</th>
<th>action</th>
<th>sequence of actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>(i++)</td>
<td></td>
</tr>
<tr>
<td>4–3</td>
<td>incl (i^*)</td>
<td>movl (i,%r)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addl (1,%r^*)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>movl (%r, i)</td>
</tr>
<tr>
<td>2–1</td>
<td>* read from memory into accumulator</td>
<td>modify contents of accumulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>write from accumulator into memory</td>
</tr>
</tbody>
</table>

¹ typical for a complex instruction of an “abstract processor” (C, CISC)
Entireness or unit of a sequence of actions whose solo efforts all will happen apparently simultaneous (i.e., are synchronised)

- an/the essential non-functional property of an atomic operation \(^5\)
  - logical togetherness of a sequence of actions in terms of time
  - by what that sequence appears as elementary operation (ELOP)

Examples of (critical) actions for incrementation of a counter variable:

- level \(_{5 \rightarrow 3}\)

  C/C++          ASM
  1 i++; 2 movl i, %eax
  3 addl $1, %eax
  4 movl %eax, i

- level \(_{3 \rightarrow 2}\)

  C/C++          ASM          ISA
  1 i++; 5 incl i
  6 read A from <i>
  7 modify A by 1
  8 write A to <i>

\(^5\)from (Gr.) átomo “indivisible”. 
© wosch CS (WS 2016, LEC 3) Fundamentals–Process 14
Indivisibility II

Entireness or unit of a sequence of actions whose solo efforts all will happen apparently simultaneous (i.e., are synchronised)

- an/the essential non-functional property of an atomic operation
  - logical togetherness of a sequence of actions in terms of time
  - by what that sequence appears as elementary operation (ELOP)

- examples of (critical) actions for incrementation of a counter variable:
  - level $5 \rightarrow 3$
  - level $3 \rightarrow 2$

  C/C++   | ASM       | ASM       | ISA
  ---     | ---        | ---        | ---
  1 i++;  | movl i, %eax | 5 incl i  | 6 read A from <i>
  3 addl $1$, %eax | 7 modify A by 1
  4 movl %eax, i | 8 write A to <i>

- points (i++, incl) in case of merely conditionally atomic execution
  - namely uninterruptible operation (level $5 \rightarrow 3$), uniprocessor (Ebene $3 \rightarrow 2$)
  - problem: overlapping in time of the sequence of actions pointed here

Sequential Process

Definition

A process that is composed exclusively of a sequence of temporally non-overlapping actions.

- the sequence of actions forms a unique execution thread
  - of which always only a single one exists within a sequential process
  - but which may develop differently with each restart of that process
    - other input data, program change, ..., transient hardware errors
  - the sequence is defined by a total order of its actions
  - it is reproducible given unmodified original conditions
Sequential Process

Definition
A process that is composed exclusively of a sequence of temporally non-overlapping actions.

- the sequence of actions forms a unique execution thread
  - of which always only a single one exists within a sequential process
  - but which may develop differently with each restart of that process
    - other input data, program change, ..., transient hardware errors
- the sequence is defined by a total order of its actions
  - it is reproducible given unmodified original conditions

Hint (Execution Thread ≠ Thread)
Assumptions about the technical implementation of the sequence of actions are not met and are also irrelevant here. A thread is only one option to put the incarnation of a sequential process into effect.

Non-Sequential Process

Definition
Also referred to as “parallel”, namely a process that is composed of a sequence of temporally overlapping actions.

- requirement is a non-sequential program (cf. p. 9)
  - that allows for at least one more process incarnation (child process) or
  - that makes arrangements for the handling of events of external processes
- whereby sequences of actions may overlap in the first place:
  i multithreading (Ger. simultane Mehrfädigkeit), in fact:
    - pseudo-parallel = multiplex mode of a single processor (core)
    - real parallel = parallel mode of a (multi-core) multiprocessor
  ii asynchronous program interrupts

© wosch CS (WS 2016, LEC 3) Fundamentals–Process 15

© wosch CS (WS 2016, LEC 3) Fundamentals–Process 16
Non-Sequential Process

Definition
Also referred to as “parallel”, namely a process that is composed of a sequence of temporally overlapping actions.

- requirement is a non-sequential program (cf. p. 9)
  - that allows for at least one more process incarnation (child process) or
  - that makes arrangements for the handling of events of external processes
- whereby sequences of actions may overlap in the first place:
  - multithreading (Ger. simultane Mehrfädigkeit), in fact:
    - pseudo-parallel – multiplex mode of a single processor (core)
    - real parallel – parallel mode of a (multi-core) multiprocessor
  - asynchronous program interrupts
- consequently, the sequence of all actions is defined by a partial order
- as external processes may enable temporal/causal independent actions

*Interrupt requests issued by some device (IRQ) oder process (signal).*

Concurrent Processes (Ger.) gleichzeitige Prozesse [3]

Definition (in a broader sense: “simultaneous processes”)
One or more (non-sequential) processes whose sequences of actions will overlap in time area by area (Ger. Bereichsweise).

- areas are concurrent (Ger. nebenläufig) only if they are independent
  - none of these concurrent processes is cause or effect of the other
  - none of these actions of these processes requires the result of any other

Concurrent Processes

Definition (in a broader sense: “simultaneous processes”)
One or more (non-sequential) processes whose sequences of actions will overlap in time area by area (Ger. Bereichsweise).

- areas are concurrent (Ger. nebenläufig) only if they are independent
  - none of these concurrent processes is cause or effect of the other
  - none of theses actions of these processes requires the result of any other
  - to proceed, concurrent processes compete for reusable resources
    - they share the processor (core), cache (line), bus, or devices
    - outcome of this is interference (Ger. Interferenz) in process behaviour

*Derived from (Fre.) s’entreferir “to brawl each other”.*
Interacting Processes I  (Ger.) gekoppelte Prozesse [3, p.77]

Definition (also: “depending processes”)
Simultaneous processes that, directly or indirectly, interact with each other through a shared variable or by accessing a shared resource.

- their actions get into conflict if at least one of these processes... 
  - will change the value of one of the shared variables (access pattern) or 
  - already occupies a shared non-preemptable resource\(^8\) (resource type)
  \(^8\)printer, mouse, plotter, keyboard.

- this may emerge as a race condition (Ger. Wettlaufsituation) 
  - for shared variables or (reusable/consumable) resources, resp. 
  - for starting or finishing an intended sequence of actions

Concurrent Processes  (Ger.) gleichzeitige Prozesse [3]

Definition (in a broader sense: “simultaneous processes”)
One or more (non-sequential) processes whose sequences of actions will overlap in time area by area (Ger. bereichsweise).

- areas are concurrent (Ger. nebenläufig) only if they are independent
- none of these concurrent processes is cause or effect of the other
- none of these actions of these processes requires the result of any other

- to proceed, concurrent processes compete for reusable resources
  - they share the processor (core), cache (line), bus, or devices
  - outcome of this is interference\(^7\) (Ger. Interferenz) in process behaviour
  - the effective degree of overlapping is irrelevant for the simultaneity
    - apart from time-dependent processes that have to keep deadlines
    - note that the larger the overlapping, the larger the time delay
    - and the more likely will a delayed process miss its deadline
- just as interference, which may also cause violation of timing constraints

\(^7\)Derived from (Fre.) s’entreferer “to brawl each other”.

- printer, mouse, plotter, keyboard.
Interacting Processes I

Definition (also: “depending processes”)

Simultaneous processes that, directly or indirectly, interact with each other through a shared variable or by accessing a shared resource.

- their actions get into conflict if at least one of these processes... 
  - will change the value of one of the shared variables (access pattern) or 
  - already occupies a shared non-preemptable resource \(^8\) (resource type)
- this may emerge as a race condition (Ger. Wettlaufsituation)
  - for shared variables or (reusable/consumable) resources, resp.
  - for starting or finishing an intended sequence of actions
- conflicts are eliminated by means of synchronisation methods:
  - blocking prevent from executing an intended sequence of actions
  - non-blocking let a process abort and retry a started sequence of actions
- founds coordination of cooperation and competition of processes

---

Interacting Processes II

Race Conditions

```c
int64_t cycle = 0;

void *thread_worker(void *null) {
    for (;;) {
        /* ... */
        inc64(&cycle);
    }
}

void *thread_minder(void *null) {
    for (;;) {
        printf("worker cycle %lld\n", cycle);
        pthread_yield();
    }
}
```

\(^8\) printer, mouse, plotter, keyboard.

---
Interacting Processes II

```c
int64_t cycle = 0;

void * thread_worker (void * null) {
    for (;;) {
        /* ... */
        inc64 (& cycle);
    }
}

void * thread_minder (void * null) {
    for (;;) {
        printf (" worker cycle %lld \n", cycle);
        pthread_yield();
    }
}
```

which cycle values prints the minder thread (Ger. Aufpasserfaden)?

which are produced by multiple worker threads (Ger. Arbeiterfäden)?

in case thread_worker exists in several identical incarnations

inc64: see p.7

---

Interacting Processes III

1. Race Condition

assuming that the non-sequential program runs on a 32-bit machine
- instances of int64_t then form a pair of 32-bit words: double word
- operations on instances of int64_t cease to be solo efforts

inc64:
```c
movl 4(% esp), % eax
addl $1, (% eax)
adc1 $0, 4(% eax)
ret
```

.L6:
```c
movl $cycle, (% esp)
call inc64
jmp .L6
```
Interacting Processes III

1. Race Condition

assuming that the non-sequential program runs on a 32-bit machine
instances of int64_t then form a pair of 32-bit words: double word
operations on instances of int64_t cease to be solo efforts

worker thread

```
1  inc64:
2    movl 4(%esp), %eax
3    addl $1, (%eax)
4    addcl $0, 4(%eax)
5    ret

.L6:
6    movl $cycle, (%esp)
7    call inc64
8    jmp .L6
```

minder thread

```
10   movl cycle+4, %edx ; high &
11   movl cycle, %eax ; low word
12   movl $.LC0, (%esp)
13   movl %edx, 8(%esp)
14   movl %eax, 4(%esp)
15   call printf
```

```
11    movl cycle+4, %edx ; high &
12    movl cycle, %eax ; low word
13    movl $.LC0, (%esp)
14    movl %edx, 8(%esp)
15    movl %eax, 4(%esp)
```

```
.assume cycle = 2^{32} - 1
```

```
.assume cycle = 2^{32} - 1
```

movl cycle +4, %edx ; high &
movl cycle, %eax ; low word
movl $.LC0, (%esp)
movl %edx, 8(%esp)
movl %eax, 4(%esp)
call printf
```
Interacting Processes IV

2. Race Condition

- assuming that the development or run-time environment varies
  - different compilers, assemblers, linker, or loaders
  - different operating systems—but the same real processor (x86)

- GCC 4.7.2, Linux

```
1 inc64:        6 _inc64:
  2 movl 4(%esp), %eax            7 movl 4(%esp), %eax
  3 addl $1, (%eax)              8 movl (%eax), %ecx
  4 addcl $0, 4(%eax)            9 movl 4(%eax), %edx
  5 ret                           10 addl $1, %ecx
                                  11 addcl $0, %edx
                                  12 movl %edx, 4(%eax)
                                  13 movl %ecx, (%eax)
                                  14 ret
```

- GCC 4.2.1, MacOSX

```
1 inc64:        6 _inc64:
  2 movl 4(%esp), %eax            7 movl 4(%esp), %eax
  3 addl $1, (%eax)              8 movl (%eax), %ecx
  4 addcl $0, 4(%eax)            9 movl 4(%eax), %edx
  5 ret                           10 addl $1, %ecx
                                  11 addcl $0, %edx
                                  12 movl %edx, 4(%eax)
                                  13 movl %ecx, (%eax)
                                  14 ret
```

- pseudo-parallel actions (case 4.2.1)
  - (UNIX-) signal
  - asynchronous program interrupt
Interacting Processes IV

2. Race Condition

assuming that the development or run-time environment varies
- different compilers, assemblers, linker, or loaders
- different operating systems—but the same real processor (x86)

GCC 4.7.2, Linux
1 inc64:
2 movl 4(%esp), %eax
3 addl $1, (%eax)
4 adcl $0, 4(%eax)
5 ret

GCC 4.2.1, MacOSX
6 _inc64:
7 movl 4(%esp), %eax
8 movl (%eax), %ecx
9 movl 4(%eax), %edx
10 addl $1, %ecx
11 adcl $0, %edx
12 movl %edx, 4(%eax)
13 movl %ecx, (%eax)
14 ret

pseudo-parallel actions (case 4.2.1)
- (UNIX-) signal
- asynchronous program interrupt

real parallel actions: (multi-core) multiprocessor
- the actions in lines 3–4 are critical as well: divisible read-modify-write

Consistency Coordination of Interacting Processes

prevention of race conditions by the protection of critical sections
- transfer a non-sequential process into a temporary sequential process
  - strictly: the shorter the sequential time span, the better the solution
- or, if applicable, rewrite conflict-prone program sequences as a transaction...
prevention of race conditions by the protection of critical sections
- transfer a non-sequential process into a temporary sequential process
  - strictly: the shorter the sequential time span, the better the solution
- or, if applicable, rewrite conflict-prone program sequences as a transaction

Lookahead: prevent overlapping by means of mutual exclusion

blocking of interacting processes

```c
void mutex_inc64(int64_t *i, pthread_mutex_t *lock) {
    pthread_mutex_lock(lock); /* indivisible, now */
    inc64(i); /* reuse code @ p.7 */
    pthread_mutex_unlock(lock); /* divisible, again */
}
```

Consistency Coordination of Interacting Processes

prevention of race conditions by the protection of critical sections
- transfer a non-sequential process into a temporary sequential process
  - strictly: the shorter the sequential time span, the better the solution
- or, if applicable, rewrite conflict-prone program sequences as a transaction

Lookahead: prevent overlapping by means of mutual exclusion

blocking of interacting processes: comparatively long time span

```c
void mutex_inc64(int64_t *i, pthread_mutex_t *lock) {
    pthread_mutex_lock(lock); /* indivisible, now */
    inc64(i); /* reuse code @ p.7 */
    pthread_mutex_unlock(lock); /* divisible, again */
}
```

Consistency Coordination of Interacting Processes

- anchoring of processes can be different within a computing system

Localisation Operating-System v. Application Context

anchoring of processes can be different within a computing system
**Localisation**

- **anchoring** of processes can be different within a computing system
- namely inside or outside the operating-system machine level:
  - **inside** – originally, within the operating system or its kernel
    - incarnation of the process is root of possibly other processes
    - partial virtualisation of the CPU as the real processor (core)
    - "kernel thread", in computer science folklore
  - **outside** – optional, within run-time or even application system
    - incarnation of the process as leaf or inner node (of a graph)
    - partial virtualisation of the root process as an abstract processor
    - "user thread", in computer science folklore
- usually, a processor (core) is entirely unaware of being multiplexed
  - a kernel thread may serve as an abstract processor for user threads
  - no nowadays known (real) processor is aware of what it is processing
  - particularly, a kernel thread does not know about potential user threads
  - when it gets switched or delayed, all of its user threads will as well evolve from time sharing their underlying processor (core)
Weight Category Interruption and Resumption Overhead

- * feather−, ** lightweight
  - partial virtualization

* user threads
* user level
* kernel level

- kernel threads

- **

- modes of process switches as to partial processor virtualisation:
  * inside the same (user/kernel) address space, \textit{ibidem}^{9} continuing
  ** inside kernel address space, same user address space sharing
  *** inside kernel address space, at other user address space landing

\textsuperscript{9}(Lat.), “at the same place”

© wosch CS (WS 2016, LEC 3) Characteristics–Physical
25
Sequencing of Processes Scheduling v. Synchronisation

- **scheduling** (Ger. *Ablaufplanung*) the **dispatching** (Ger. *Einlastung*) of processes or, to be precise, process incarnations
  - a big theoretical/mathematical side of operating systems [2, 1, 6, 7]
  - but enforcing the scheduling policies faces several practical challenges
  - unpredictable dynamic system behaviour at run-time dashes hopes
  - on the one hand interrupts, on the other hand resource sharing
  - breeds **asynchronism** and, as a result, foregrounds **heuristic**

Sequencing of Processes Scheduling v. Synchronisation

- **scheduling** (Ger. *Ablaufplanung*) the **dispatching** (Ger. *Einlastung*) of processes or, to be precise, process incarnations
  - a big theoretical/mathematical side of operating systems [2, 1, 6, 7]
  - but enforcing the scheduling policies faces several practical challenges
  - unpredictable dynamic system behaviour at run-time dashes hopes
  - on the one hand interrupts, on the other hand resource sharing
  - breeds **asynchronism** and, as a result, foregrounds **heuristic**

  - process **synchronisation** is notorious for producing interference
  - once it comes to contention resolution, which **implies sequencing**
    - blocking – in matters of allocating consumable and/or reusable resources
    - non-blocking – pertaining to indivisible machine (CPU) instructions
  - especially susceptible for inducing interference is **blocking synchronisation**
Sequencing of Processes  Scheduling v. Synchronisation

- **scheduling** (Ger. *Ablaufplanung*) the **dispatching** (Ger. *Einlastung*) of processes or, to be precise, process incarnations
- a big theoretical/mathematical side of operating systems [2, 1, 6, 7]
- but enforcing the scheduling policies faces several practical challenges
- unpredictable dynamic system behaviour at run-time dashes hopes
- on the one hand interrupts, on the other hand resource sharing
- breeds **asynchronism** and, as a result, foregrounds **heuristic**
- process **synchronisation** is notorious for producing interference
- once it comes to **contention resolution**, which **implies sequencing**
  - blocking – in matters of allocating consumable and/or reusable resources
  - non-blocking – pertaining to indivisible machine (CPU) instructions
- especially susceptible for inducing interference is blocking synchronisation
- to **control resource usage**, processes pass through logical states
- whereby synchronisation emerges jointly responsible for state transitions
- taken together, scheduling and synchronisation are **cross-cutting concerns**

Process States and State Transitions

- **typical life time cycle** of processes:
  - ready
  - running
  - blocked

  - ready to run, but still waiting for a processor (core)
  - executing on a processor (core), performing a CPU burst
  - waiting for an event (being in sync), performing an I/O burst
Process States and State Transitions

- **typical life time cycle** of processes:
  - **ready**
    - ready to run, but still waiting for a processor (core)
  - **running**
    - executing on a processor (core), performing a CPU burst
  - **blocked**
    - waiting for an event (being in sync), performing an I/O burst

- relevant resources:
  - processor
    - start
    - seize
    - yield
    - cease
    - await
    - cause

- waitlists involved:
  - ready list of runnable processes
  - blocked list of processes unable to run

Outline

- Preface
- Fundamentals
  - Program
  - Process
- Characteristics
  - Physical
  - Logical
- Summary

Résumé

- a process is **predetermined by a program** that is to be executed
  - the process inherits the static characteristics of its program
  - when being existent, the process adds dynamic characteristics
    - as a function of data processing and interaction with the environment
- a process may be **sequential or non-sequential** (as to its program)
  - that is to say, composed of non-overlapping or overlapping actions
  - whereby overlapping is caused by multiprocessing in a wider sense
    - real parallelism, but also pseudo-parallelism in its various forms
- processes are **parallel, concurrent, simultaneous, or interacting**
  - simultaneous processes comprise concurrent and interacting periods
  - each of these can be parallel on their part, i.e., if their actions overlap
    - by either multiplexing or multiplication of the necessary processing units
- as to implementation, processes may be **kernel or user threads**
  - regardless of which, logical states report on the life time cycle of a process
    - whereby synchronisation emerges jointly responsible for state transitions
    - taken together, scheduling and synchronisation need to be complementary
a process is **predetermined by a program** that is to be executed
- the process inherits the static characteristics of its program
- when being existent, the process adds dynamic characteristics
  - as a function of data processing and interaction with the environment

a process may be **sequential or non-sequential** (as to its program)
- that is to say, composed of non-overlapping or overlapping actions
- whereby overlapping is caused by multiprocesssing in a wider sense
  - real parallelism, but also pseudo-parallelism in its various forms

processes are **parallel, concurrent, simultaneous, or interacting**
- simultaneous processes comprise concurrent and interacting periods
  - each of these can be parallel on their part, i.e., if their actions overlap
    - by either multiplexing or multiplication of the necessary processing units

as to implementation, processes may be **kernel or user threads**
- regardless of which, logical states report on the life time cycle of a process
  - whereby synchronisation emerges jointly responsible for state transitions
    - taken together, scheduling and synchronisation need to be complementary

---

a process is **predetermined by a program** that is to be executed
- the process inherits the static characteristics of its program
- when being existent, the process adds dynamic characteristics
  - as a function of data processing and interaction with the environment

a process may be **sequential or non-sequential** (as to its program)
- that is to say, composed of non-overlapping or overlapping actions
- whereby overlapping is caused by multiprocesssing in a wider sense
  - real parallelism, but also pseudo-parallelism in its various forms

processes are **parallel, concurrent, simultaneous, or interacting**
- simultaneous processes comprise concurrent and interacting periods
  - each of these can be parallel on their part, i.e., if their actions overlap
    - by either multiplexing or multiplication of the necessary processing units

as to implementation, processes may be **kernel or user threads**
- regardless of which, logical states report on the life time cycle of a process
  - whereby synchronisation emerges jointly responsible for state transitions
    - taken together, scheduling and synchronisation need to be complementary

---

**Résumé**

- a process is **predetermined by a program** that is to be executed
  - the process inherits the static characteristics of its program
  - when being existent, the process adds dynamic characteristics
    - as a function of data processing and interaction with the environment

- a process may be **sequential or non-sequential** (as to its program)
  - that is to say, composed of non-overlapping or overlapping actions
  - whereby overlapping is caused by multiprocesssing in a wider sense
    - real parallelism, but also pseudo-parallelism in its various forms

- processes are **parallel, concurrent, simultaneous, or interacting**
  - simultaneous processes comprise concurrent and interacting periods
    - each of these can be parallel on their part, i.e., if their actions overlap
      - by either multiplexing or multiplication of the necessary processing units

- as to implementation, processes may be **kernel or user threads**
  - regardless of which, logical states report on the life time cycle of a process
    - whereby synchronisation emerges jointly responsible for state transitions
      - taken together, scheduling and synchronisation need to be complementary

---

**Résumé**

- a process is **predetermined by a program** that is to be executed
  - the process inherits the static characteristics of its program
  - when being existent, the process adds dynamic characteristics
    - as a function of data processing and interaction with the environment

- a process may be **sequential or non-sequential** (as to its program)
  - that is to say, composed of non-overlapping or overlapping actions
  - whereby overlapping is caused by multiprocesssing in a wider sense
    - real parallelism, but also pseudo-parallelism in its various forms

- processes are **parallel, concurrent, simultaneous, or interacting**
  - simultaneous processes comprise concurrent and interacting periods
    - each of these can be parallel on their part, i.e., if their actions overlap
      - by either multiplexing or multiplication of the necessary processing units

- as to implementation, processes may be **kernel or user threads**
  - regardless of which, logical states report on the life time cycle of a process
    - whereby synchronisation emerges jointly responsible for state transitions
      - taken together, scheduling and synchronisation need to be complementary

---

**Résumé**

- a process is **predetermined by a program** that is to be executed
  - the process inherits the static characteristics of its program
  - when being existent, the process adds dynamic characteristics
    - as a function of data processing and interaction with the environment

- a process may be **sequential or non-sequential** (as to its program)
  - that is to say, composed of non-overlapping or overlapping actions
  - whereby overlapping is caused by multiprocesssing in a wider sense
    - real parallelism, but also pseudo-parallelism in its various forms

- processes are **parallel, concurrent, simultaneous, or interacting**
  - simultaneous processes comprise concurrent and interacting periods
    - each of these can be parallel on their part, i.e., if their actions overlap
      - by either multiplexing or multiplication of the necessary processing units

- as to implementation, processes may be **kernel or user threads**
  - regardless of which, logical states report on the life time cycle of a process
    - whereby synchronisation emerges jointly responsible for state transitions
      - taken together, scheduling and synchronisation need to be complementary
Reference List I

[1] **Coffman, E. G.; Denning, P. J.**:
*Operating System Theory.*
Prentice Hall, Inc., 1973

[2] **Conway, R. W.; Maxwell, L. W.; Millner, L. W.**:
*Theory of Scheduling.*
Addison-Wesley, 1967

[3] **Hansen, P. B.**:
Concurrent Processes.
In: *Operating System Principles.*

[4] **IEEE**:
POSIX.1c Threads Extensions / Institute of Electrical and Electronics Engineers.
Standarddokument

[5] **Kleinöder, J.; Schröder-Preikschat, W.**:
Rechnerorganisation.
In: *Lehrstuhl Informatik 4* (Hrsg.):
*Systemprogrammierung.*
FAU Erlangen-Nürnberg, 2014 (Vorlesungsfolien), Kapitel 5

Reference List II

[6] **Kleinrock, L.**:
John Wiley & Sons, 1975

[7] **Liu, J. W. S.**:
*Real-Time Systems.*
ISBN 0–13–099651–3

[8] **Löhr, K.-P.**:
Nichtsequentielle Programmierung.
In: *Institut für Informatik* (Hrsg.):
*Algorithmen und Programmierung IV.*
Freie Universität Berlin, 2006 (Vorlesungsfolien)

[9] **Neufeldt, V. (Hrsg.); Guralnik, D. A. (Hrsg.)**:
*Webster’s New World Dictionary.*
Simon & Schuster, Inc., 1988

[10] **Schröder-Preikschat, W.**:
Concurrency.
In: *Lehrstuhl Informatik 4* (Hrsg.):
*Concurrent Systems.*
FAU Erlangen-Nürnberg, 2014 (Lecture Slides), Kapitel 2

Reference List III

Multilevel Machines.
In: *Structured Computer Organization.*

[12] **Wikipedia**:
*Process.*

Originally as a Concept of Law acc. [12, Legal process]

Process “particularly, describes the formal notice or writ used by a court to exercise jurisdiction over a person or property”

- analogy in computer science or operating-system concepts, resp.:
  - *writ* order to abandon rivalry\(^{10}\) in the claiming of resources
  - direction to resolve competition of resource contenders

- *court* incarnation of the function of scheduling or coordination
  - point of synchronisation in a program

- *jurisdiction* sphere of authority of contention resolution

- *property* occupancy/ownership of resources, ability to proceed
  - functional or non-functional attribute

- generally, the action or trial, resp., follows a hierarchical jurisdiction

  - thereby, the process step related to a certain level is denoted as *instance*
    - in informatics, translation to (Ger.) “Instanz” however was rather unexpect\(\text{!!!}\)

- operating systems often command a multi-level processing of processes

\(^{10}\text{Lat. rivalis “in the use of a watercourse co-authored by a neighbour”}\)
refinement of [11, p. 5]: levels present on todays computers
right, the method and (bracketed) program that supports each level