Norwegian University of Science and Technology

Operating Systems

Lecture overview and Q&A Session 5 – 14.2.2022

Michael Engel

Lectures 7 and 8

Concurrency: Deadlocks and Starvation

- Deadlock definition, necessary and additional conditions
- Resource allocation graphs
- Dining philosophers problem
- Preventing deadlocks, detection and resolution

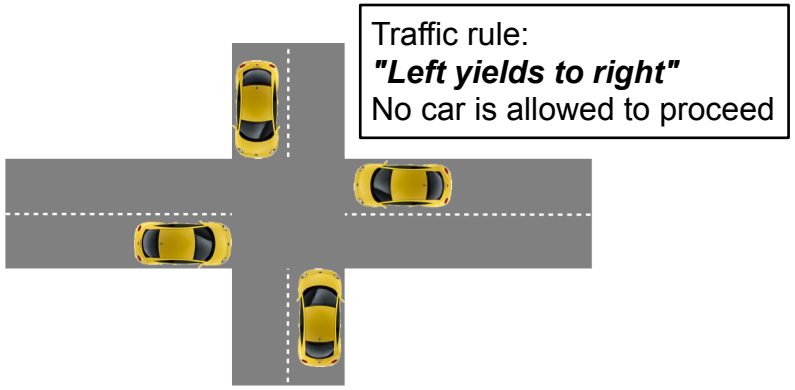
From source code to process

- Compilation process
- ELF file format and contents
- Linking and symbols
- Introduction to virtual memory and process memory layout
- Fork and exec system calls in detail and program startup

Deadlock definition

„[…] a situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.“

[Stallings]



- **Deadlock**: passive waiting, process state is BLOCKED
- **Livelock**: Active waiting (busy waiting/“lazy” busy waiting)
  - Arbitrary process state (including RUNNING), but none of the involved processes is able to proceed
- Deadlocks are the “lesser evil”
  - This state is uniquely discoverable
    → Basis to *”resolve”* deadlocks is available
- Active waiting results in an extremely high system load

Deadlocks: necessary and additional conditions

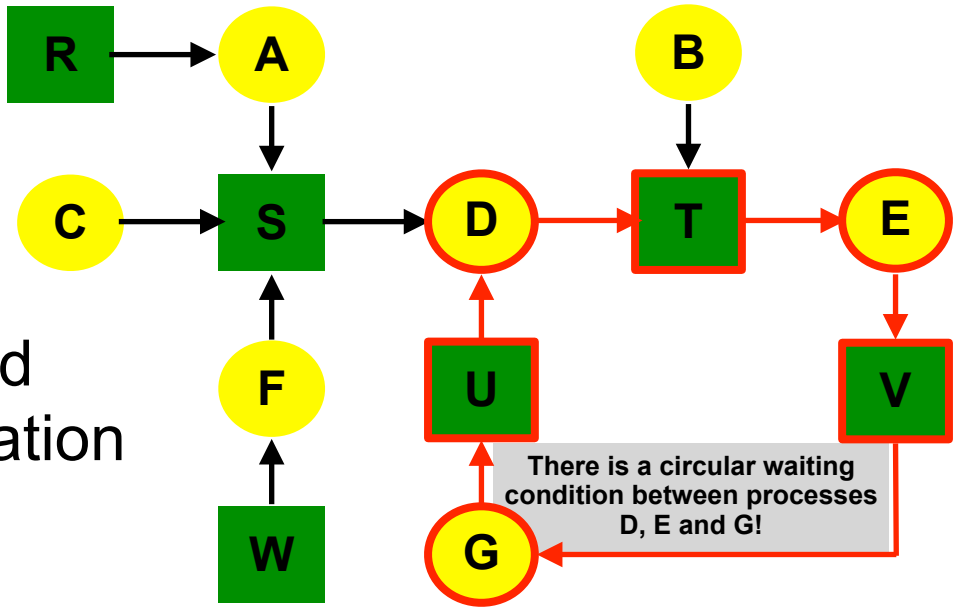
All of the following three conditions must be fulfilled for a deadlock to occur (***"necessary conditions"***):

1. Exclusive allocation of resources (***"mutual exclusion"***)
    - Only one process may use a resource at a time. No process may access a resource unit that has been allocated to another process
2. Allocation of additional resources (***"hold and wait"***)
    - A process may hold allocated resources while awaiting assignment of other resources
3. No removing of resources (***"no preemption"***)
    - The OS is unable to forcibly remove a resource from a process once it is allocated
4. Only if ***an additional condition occurs*** at runtime, we really have a deadlock:
    - ***"circular wait"***
    - A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain

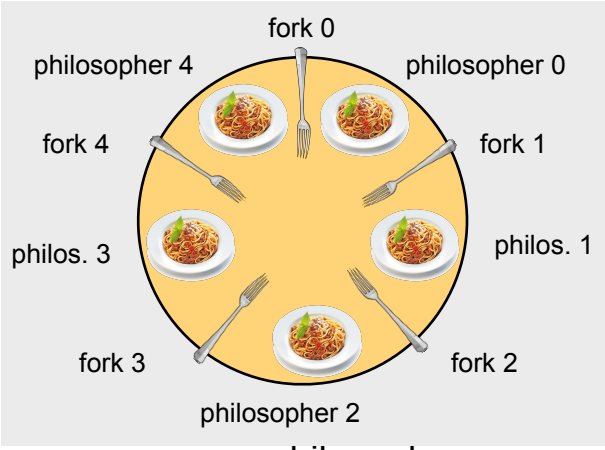
Resource allocation graphs

- Visualize and also automatically detect deadlock situations
  - describe the current system state
  - nodes are processes and resources
  - edges show an allocation or a request
- A ***circle*** in the graph indicated a deadlock condition
  - graph has to be updated for each resource allocation and deallocation

A *allocates* R and *requests* S.
B *allocates* nothing but *requests* T.
C *allocates* nothing but *requests* S.
D *allocates* U and S and *requests* T.
E *allocates* T and *requests* V.
F *allocates* W and *requests* S.
G *allocates* V and *requests* U.



Dining philosophers problem



process → philosopher
resource → fork (indivisible)

- Philosophers are either thinking or eating spaghetti
  - Two forks required for eating
  - Philosophers can only take one fork after another
- ***necessary conditions are fulfilled***
  - *mutual exclusion:* need both forks in order to eat
  - *hold and wait:* neither take both forks at the same time nor have the idea to put back a single fork
  - *no preemption:* not appropriate to take another philosopher's fork while it is in use
- ***Does this necessarily lead to a deadlock?***
- We discussed ***different solutions*** (incorrect, inefficient correct, efficient correct) ➙ check these out!

Preventing deadlocks

- **Indirect methods** invalidate one of the conditions 1–3
  1. use non blocking approaches
  2. only allow atomic resource allocations
  3. enable the preemption of resources using virtualization
     - virtual memory, virtual devices, virtual processors
- **Direct methods** invalidate condition 4
  4. introduce a linear/total order of resource classes:
     - Resource R_i can only be successfully allocated before R_j if i is ordered linear before j (i.e. $i < j$)
- Rules that prevent deadlocks
  - Methods at design or implementation time
- Discussion of ***safe/unsafe states*** ➙ check these out!

Deadlock detection and resolution

- **Common implementation:**
  - Deadlocks are (silently) accepted („**ostrich algorithm**“)
- Alternatives:
  - create **resource graph** and search for cycles → O(n)
  - tradeoffs between high overhead and waste of resources

- Resolution approaches (after detection):
  - **Terminate processes** to release resources
  - **Preempt resources**, start with the “most effective victim”
  - **Balance** between damage and effort
- Little practical relevance in the context of operating systems

Compilation proces

Preprocessor

- Expands `#includes` and macros

Compiler

- Generates assembler source code from C source code

Assembler

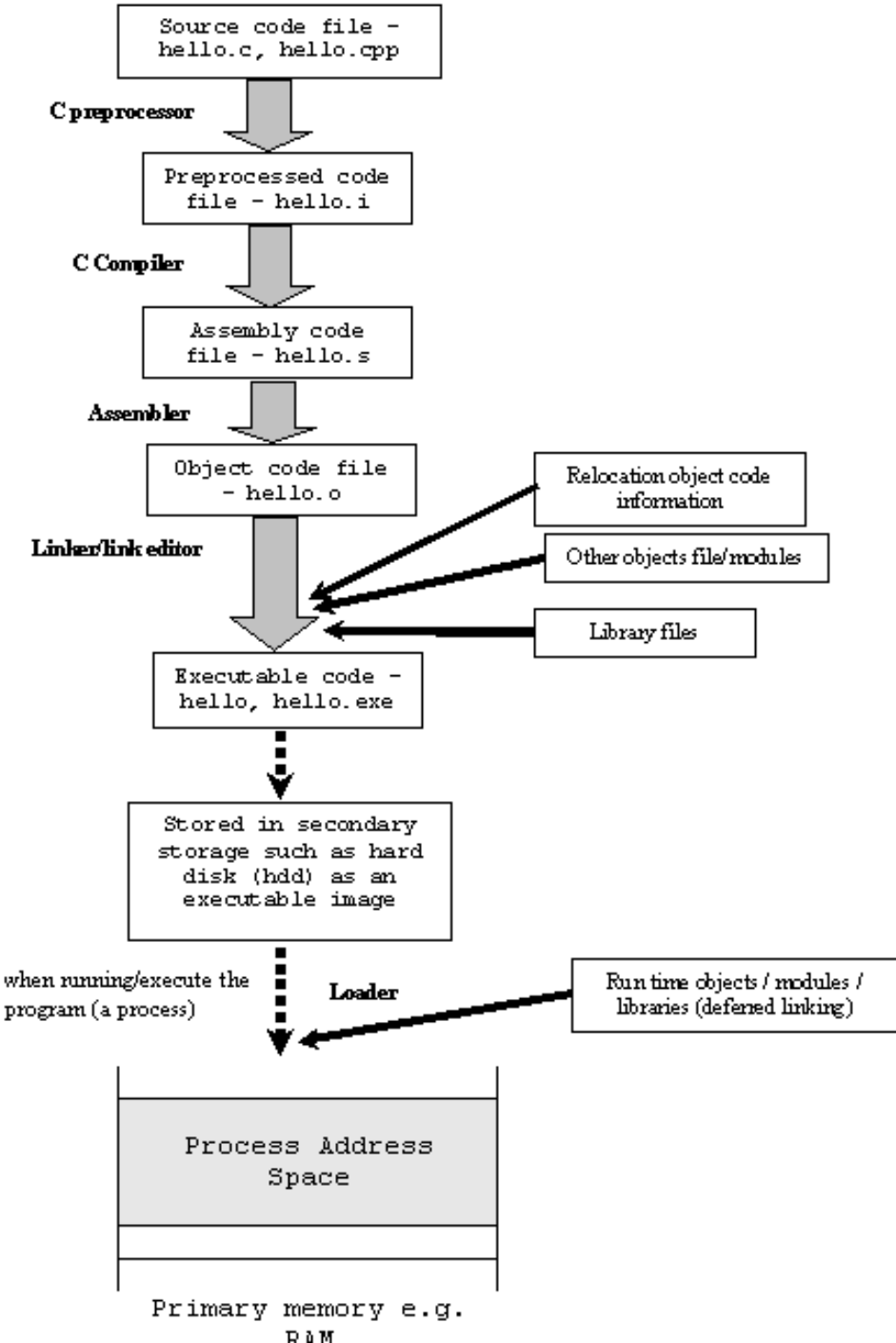
- Generates *object code* from assembler source code

Linker

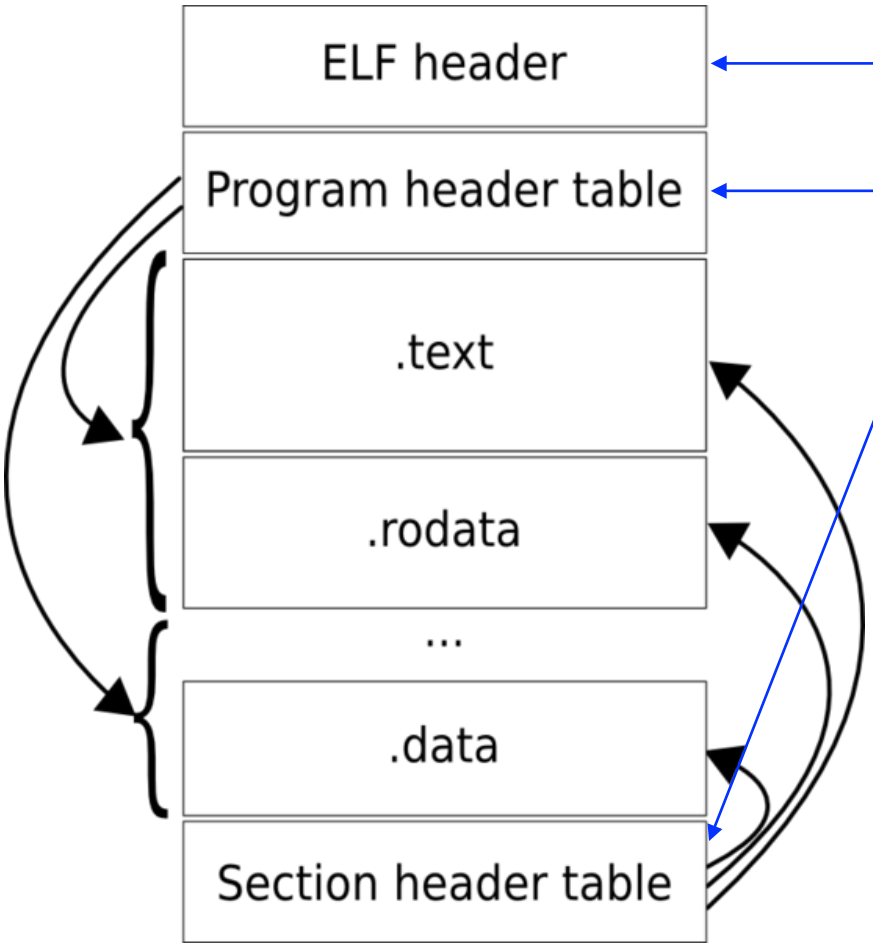
- Combines (one or) multiple object files (+ libraries) to an executable file

Loader

- Loads executable file into main memory



ELF file format and contents



Header with general information (see previous slides)

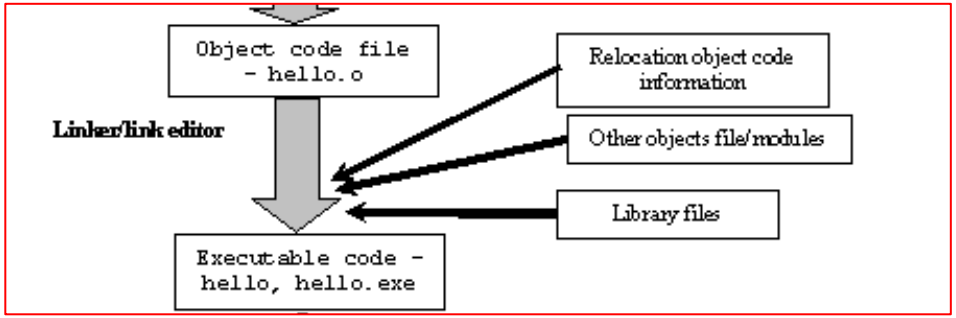
Information on the ELF file segments

Table of section headers with detailed information on each segment

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          644 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         11
  Section header string table index: 8
```

Linking and symbols

.o object files cannot be executed directly!



- Important parts are missing:
  - ***crt0*** – startup code
    - ***initialization*** – variables in .bss are initialized (to 0), C++ constructors
    - ***jump to "main"*** function and parameter passing (argc, argv, envp)
  - ***libraries***, e.g. libc (C standard library), have to be added
  - ***Linker*** adds these and builds executable
- **Addresses of variables and functions are not resolved**
  - One of the main tasks of the linker

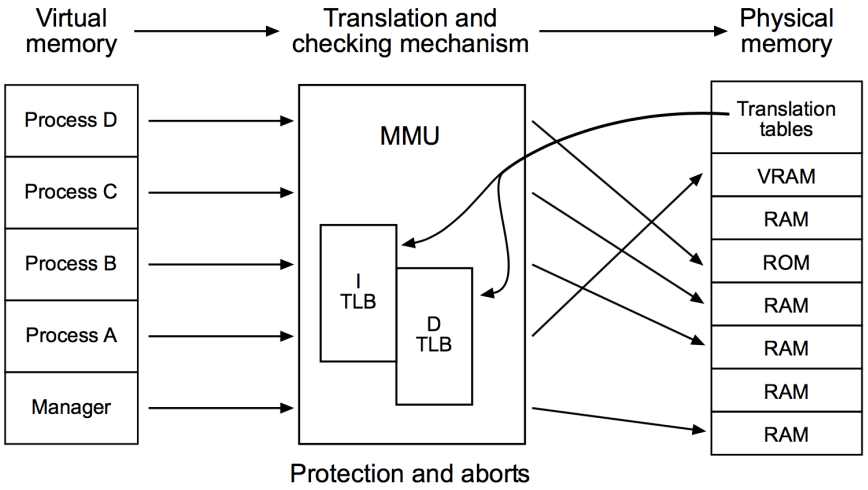
```
$ readelf -s foo.o

Symbol table '.symtab' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS foo.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    6
     8: 00000000     4 OBJECT  GLOBAL DEFAULT    5 a
     9: 00000000     4 OBJECT  GLOBAL DEFAULT    3 b
    10: 00000000    44 FUNC    GLOBAL DEFAULT    1 main
    11: 00000004     4 OBJECT  GLOBAL DEFAULT  COM c
```

ELF Section	Function
Symbols (.symtab)	Addresses for symbolic names

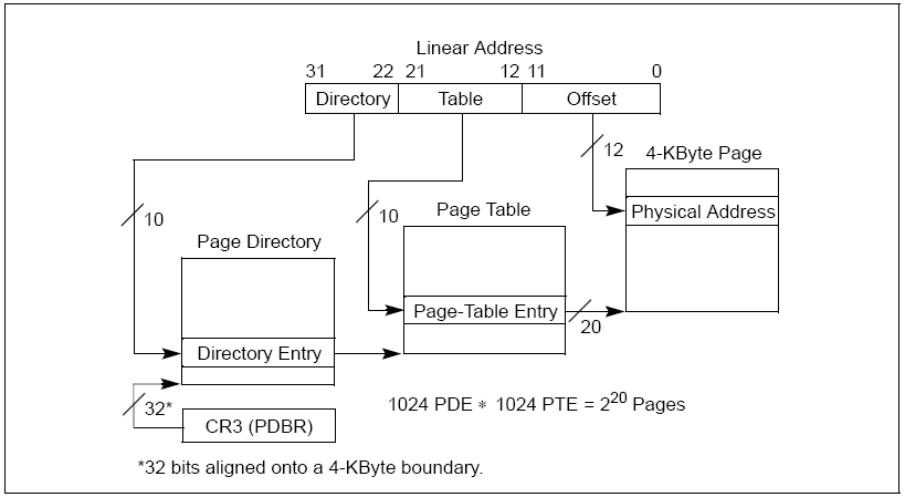
Intro to virtual memory and process memory layout

Linux requires a memory management unit (MMU)

- Translates virtual to physical addresses using **page table**
  - **Illusion:** every process has the complete address space for its own use
  - Protection of (physical) memory from unwanted accesses
  - Granularity: "page" (e.g. 4096 bytes)
  - **TLB:** cache for page table entries



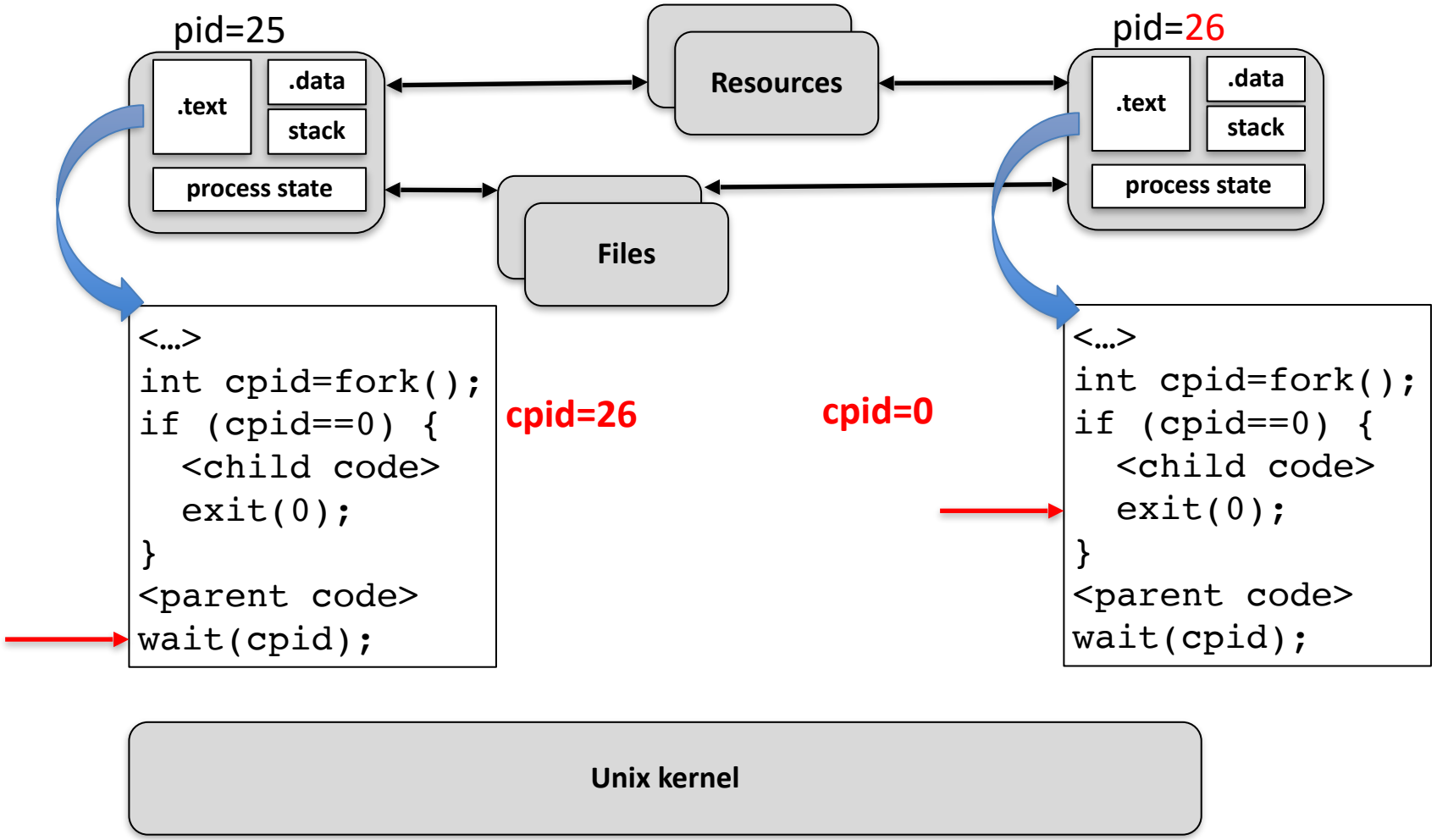
Assignment of process virtual address spaces to physical memory



Page table structure (here: x86 architecture)

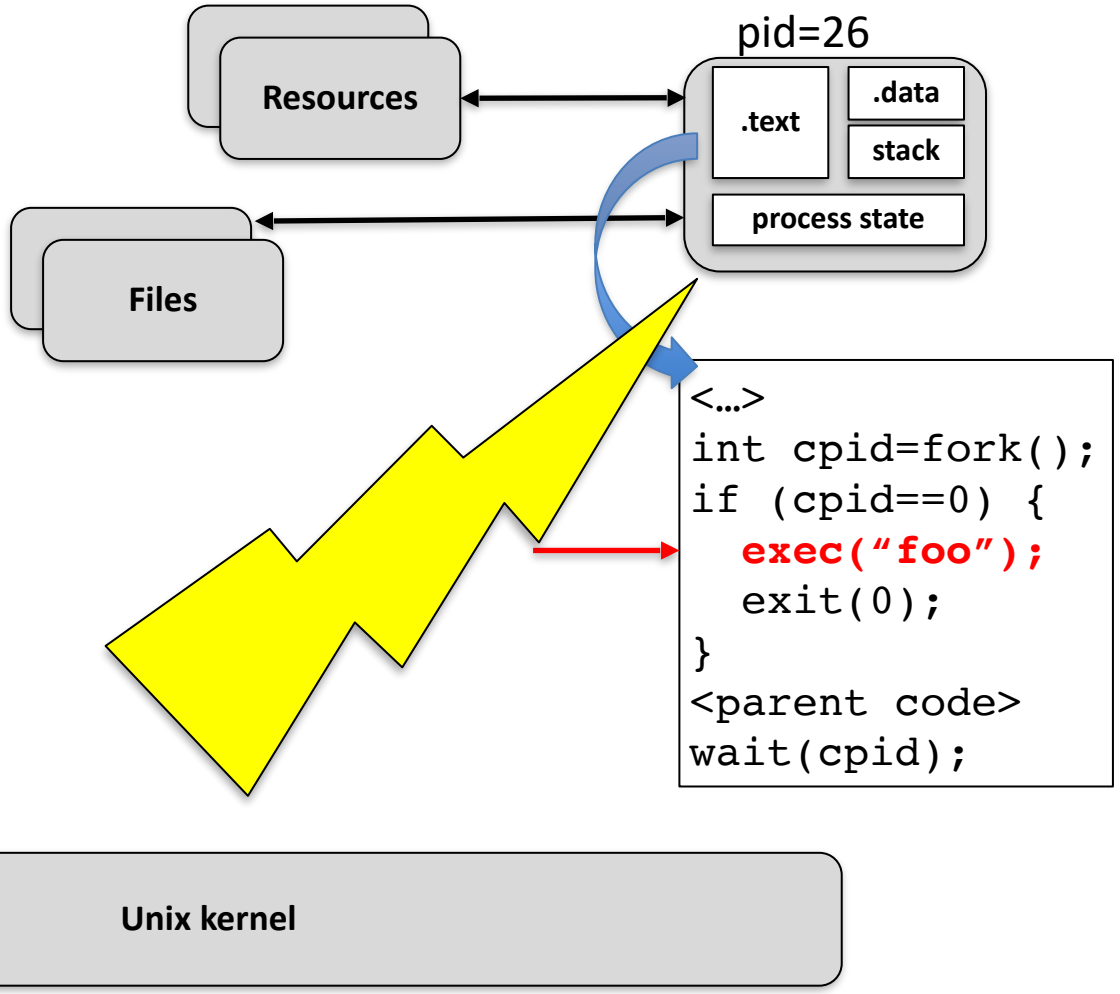
fork syscall in detail

pid25 waits for termination of pid26,
pid26 executes exit(0) and terminates



exec syscall

Kernel "removes" memory content of pid26



Overview Theoretical Exercise 3

Deadlocks and the software development process

Why?

- Deadlocks are an important problem that is hard to reproduce (e.g., race conditions that might cause a deadlock be rare) and difficult to debug
- The software development process is often hidden behind complex UIs today (Eclipse…) and seems "magical"
  - We want to give you a bit of an insight to gain back control over what you compile and execute

The forum, once more

- We are currently ~~discussing~~ setting up a Discourse server
  - open source solution (https://github.com/discourse/discourse)
  - IDI has provided us with a VM (thanks!)
    - Currently struggling with the Discourse system itself and it's TLS certificate requirements