Betriebssysteme

Tutorium 4

Peter Bohner

21. November 2025

0.1 Linking

Static and Dynamic Linking

What is the difference between static and dynamic linking?

- A static library is a collection of object files

⇒ The linked can treat it as normal code

- A dynamic library is loaded and linked *at runtime*

What are advantages of static / dynamic linking?

S+ Unused references can be elided

S+ Library calls just as fast as local ones

S+ No runtime overhead for loading and relocation

S- Library can not be shared ⇒ Memory overhead

D+ Library (Code segment) can be shared

0.1.1 PIC

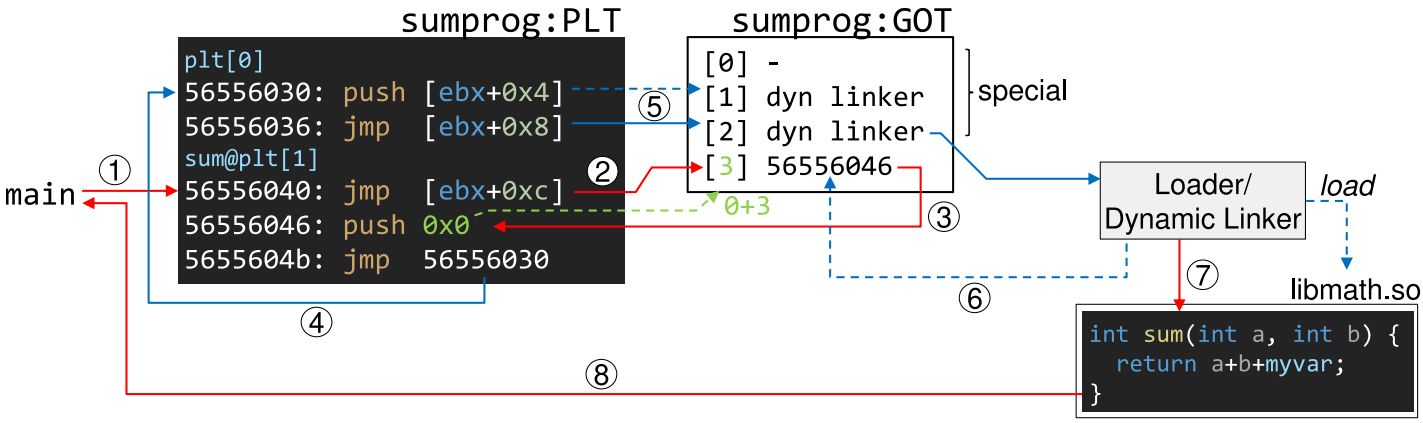
PIC

What does PIC stand for?
Position **I**ndependent **C**ode

Independent to *what*?
It's position in the process' address space.

- Shared libraries are loaded *somewhere* in the address space of a process
- PIC uses *relative* addresses (to what?)(to the instruction pointer)only – and can therefore be loaded anywhere
- How do you find *global* symbols then? ⇒ GOT, the `Global Offset Table`

PIC - Methodenaufrufe



Quelle: Vorlesungsfolien

1 Scheduling basics

Scheduling - About

What is a Scheduler? Why do we even need it?

- Maps processes to resources
- Ideally: Every process gets what it needs some day

What Schedulers do you know?

- CPU-Scheduler: The classic
- Disk-Scheduler: Why have one? Multiplexing but also efficiency!
- Network I/O: When to send packets, which packets to drop, QoL,...

Scheduling - Long- and Short-Term Scheduler

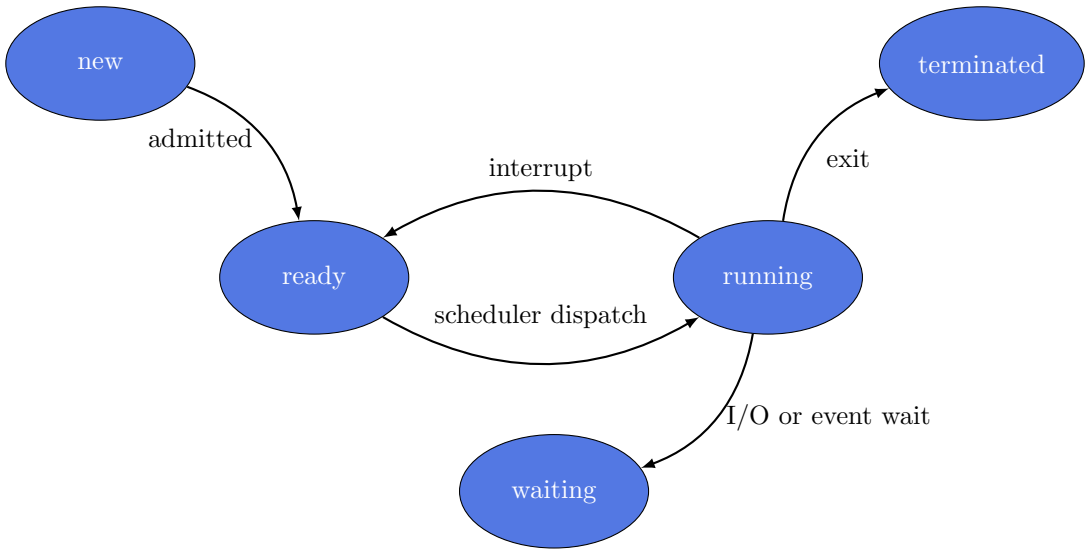
What are the differences? When are they used?

- LTS: Decide which processes to put in the *run queue*
- STS: Decide which process runs on the *CPU*
- MTS: Temporarily removes processes from main memory (and e.g. writes them out to disk)

$\Rightarrow$ Reduce degree of multiprogramming, make room in memory (and a few other reasons)

1.1 Process States

Process states

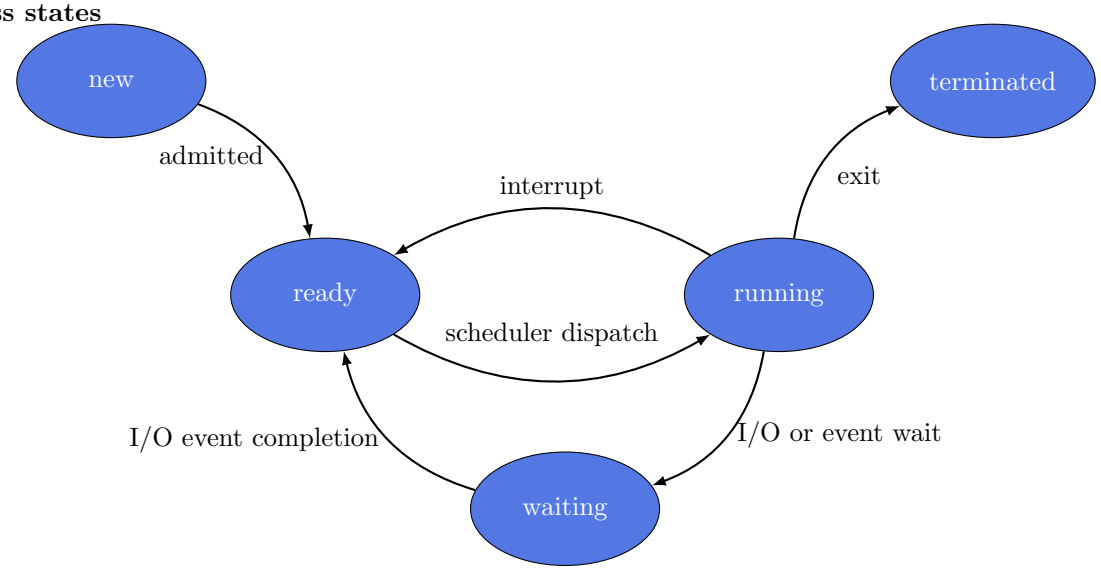


Process States - Waiting

„I/O or event wait“? When does a process move from ready to waiting?

- Network / Disk I/O
- Mutex or other inter-process synchronisation
- Sleepyness

Process states



Scheduling - Scheduler Worldcup

What makes a good Scheduler good?
Let's play scheduler!

Some metrics

- Processor utilization: Percentage of working time
- Throughput: How many jobs do you finish?
- Turnaround time: Wallclock-time from submission to finish
- Waiting time: How long did it spend in the ready queue
- Response time: Time between submission of a request and first response (e.g. key press to echo on screen)

Scheduling - Preemption the third

What does your hardware need to support to allow non-cooperative scheduling?
Timer Interrupts! Waiting for a cosmic ray to hit, a network package to arrive, a system call or any other random interrupt gets old fast :)

Scheduling - When to interrupt

Any guesses for how long a timeslice usually is?
2ms - 200ms

- On windows it depends on the configuration (favor foreground / background processes) Which setting has the longer timeslice? 20ms to 30ms for foreground, 180ms to 200ms for background
- Linux's „Completely Fair Scheduler" adjusts them dynamically based on the priority, number of processes, ...

Benefits of shorter/longer timeslices?

- Short: High interactivity, higher overhead
- Long: Lower interactivity, smaller overhead

1.2 Scheduling Policies

Shortest Job First

Pitfalls - How would you implement this?

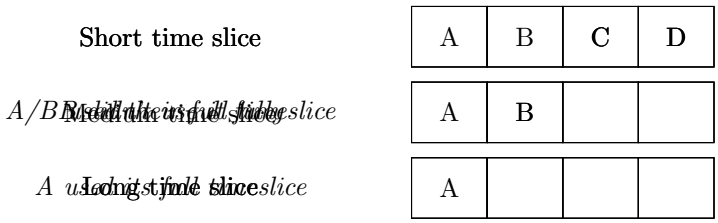
- You would need to have future knowledge to figure out the job length! How do you solve this?
- *Predict* the *future* based on *past* behaviour
- Does this work? *„THERE IS AS YET INSUFFICIENT DATA FOR A MEANINGFUL ANSWER."* ~ Isaac Asimov, „The Last Question" (Comic) You need some balanced initial value. Not *that* big of a deal with preemption though. Why? Interrupt the process after the estimated time is over.

Priorities

What is priority scheduling? Why would you use it?

- Each process is assigned a priority
- The process with the highest priority is chosen

Multi-Level Feedback Queues



How it works

- All processes start in the highest queue
- When they use up their timeslice and are preempted, they descend
- If they block before, they stay in the level (optionally: Are moved up)

⇒ I/O bound processes rise to the top and react quickly, CPU bound processes get longer timeslices but less often

A Flawless Scheduling Algorithm?

Consider the waiting time

- A few I/O-bound jobs could saturate the CPU!

$\Rightarrow$ The lower-level processes starve

How could you fix this?

- E.g. reset the whole thing after a given interval, so all start in the highest level again
- "Boost" processes that waited for a long time

What metrics does it optimize?

- Utilization? Turnaround time? Throughput? Waiting time? Response time?
- Prefer I/O bound, prefer short jobs, group the rest based on their needs