Operating Systems

Timo Hönig
Bochum Operating Systems and System Software (BOSS)
Ruhr University Bochum (RUB)

X. Scheduling
June 21, 2023 (Summer Term 2023)

RUB Faculty of Computer Science

RUHR UNIVERSITÄT BOCHUM RUB

www.informatik.rub.de
Chair of Operating Systems and System Software

Agenda

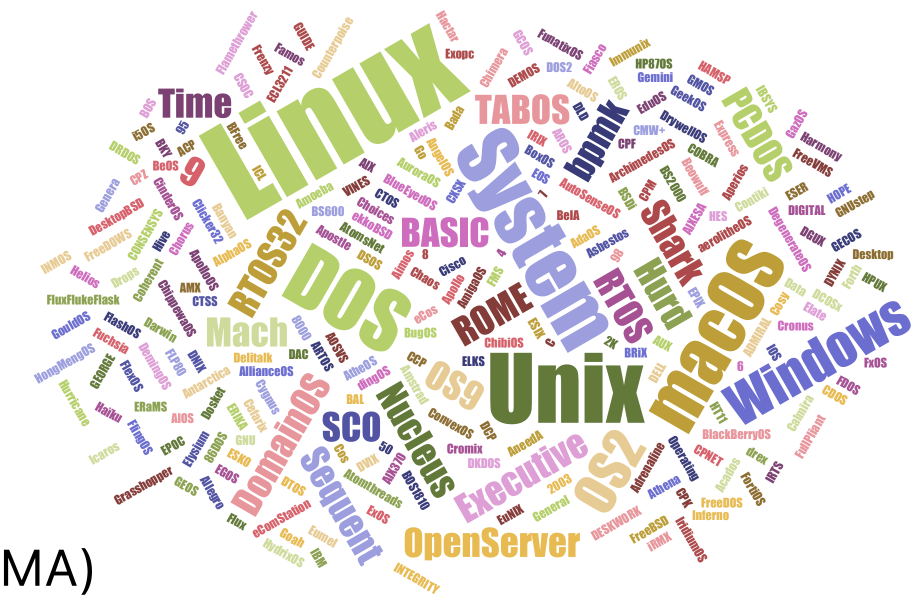
- Recap
- Organizational Matters
- Scheduling, State Transitions
- CPU Scheduling Strategies
  - Basic (First-Come First-Served)
  - Time-slice Driven (Round Robin, Virtual Round Robin)
  - Prediction Driven (Shortest Process Next, Highest Response Ratio Next)
  - Priority Driven (Multilevel Feedback Queues)
- Discussion: Reflection and Comparison
- Scheduling in Practice: The Linux O(1) Scheduler
- Summary and Outlook

Literature References

Silberschatz, Chapters 3.2, 5

Tanenbaum, Chapters 2.4

⏮ Recap

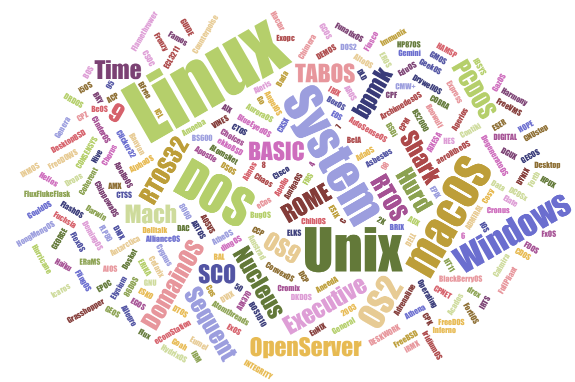


- **input/output** with **hardware devices** in operating systems
  - classification (character-based, block-based, and others)
  - operating principals (interrupts, DMA)
- device programming
  - address space models
  - operating modes: polling, interrupt- and DMA-driven
- I/O in practice: UNIX system services
  - UNIX **system calls** (i.e., open/read/write/close)
  - improve efficiency with buffers

Agenda

- Recap
- **Organizational Matters**
- Scheduling, State Transitions
- CPU Scheduling Strategies
  - Basic (First Come First Serve)
  - Time-slice Driven (Round Robin, Virtual Round Robin)
  - Prediction Driven (Shortest Process Next, Highest Response Ratio Next)
  - Priority Driven (Multilevel Feedback Queues)
- Discussion: Reflection and Comparison
- Scheduling in Practice: The Linux O(1) Scheduler
- Summary and Outlook

Organizational Matters



- lecture
  - Wednesday, 10:15 – 11:45
  - format: synchronous, **hybrid**
    - → in presence (Room HID, Building ID)
    - → online lecture (Zoom)
  - **exam:** August 7, 2023 (first appointment)
    September 25, 2023 (retest appointment)

  - **evaluation is live!**

    `https://tinyurl.com/25twb8qr`

- manage course material, asynchronous communication: Moodle
- `https://moodle.ruhr-uni-bochum.de/course/view.php?id=50698`

Agenda

- Recap
- Organizational Matters
- **Scheduling, State Transitions**
- CPU Scheduling Strategies
  - Basic (First Come First Serve)
  - Time-slice Driven (Round Robin, Virtual Round Robin)
  - Prediction Driven (Shortest Process Next, Highest Response Ratio Next)
  - Priority Driven (Multilevel Queue)
- Discussion: Reflection and Comparison
- Scheduling in Practice:
  The Linux O(1) Scheduler
- Summary and Outlook

Scheduling

- definition: „scheduling concerns the allocation of limited resources to processes over time"
  → **maximize CPU utilisation**

- depending on the **scheduling level**, each process is assigned a logical state that specifies its processing state at a point in time:
- **short-term scheduling**
- **medium-term scheduling**
- **long-term scheduling**

Rule of thumb for the frequency of **scheduler decisions** or **process state changes**:

- **short term**: µs ... ms
- **medium term**: ms ... min
- **long term**: min ... hrs

Long-Term Scheduling

Controls the degree of multiprogramming. Dynamically at runtime or in extreme cases statically within the operating system.

- **NEW** (dt. erzeugt): ready for program processing via `fork(2)`
  - the process is instantiated, a program has been assigned to it
  - however, the allocation of necessary operating resources (e.g., memory, devices) may still be pending

- **EXIT** (dt. beendet): termination via `exit(2)/wait(2)` is expected
  - the process is terminated, its resources are (to be) released
  - if necessary, another process must complete the "cleanup"

Medium-Term Scheduling

A process is completely **swapped out**: the contents of its entire address space have been moved to background memory (swap-out) and the foreground memory occupied by the process has been freed. The swap-in of the address space must be awaited.

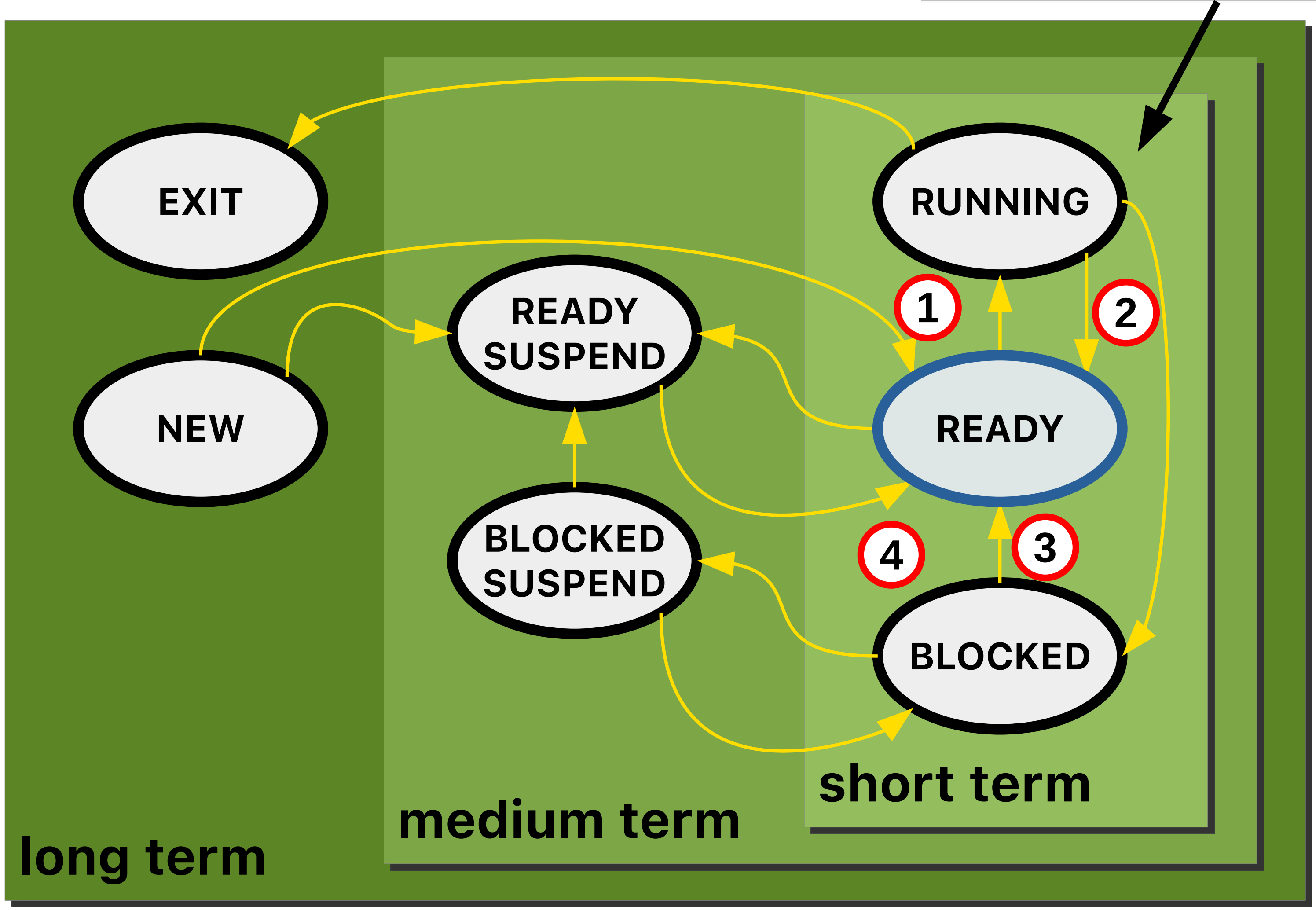
- **READY SUSPEND** (dt. ausgelagert bereit)
  - the CPU allocation ("ready") is disabled
  - the process is on the waiting list for memory allocation
- **BLOCKED SUSPEND** (dt. ausgelagert blockiert)
  - the process still expects an event (BLOCKED)
  - once the event occurs, the process is READY SUSPEND

Short-Term Scheduling

- **READY** (dt. bereit) for execution by the processor (the CPU)
  - the process is on the waiting list for CPU allocation (**ready list**)
  - list position is determined by the scheduling procedure
- **RUNNING** (dt. laufend) the resource CPU has been allocated
  - the process performs **calculations**, it completes its CPU burst
  - for each processor there is only one running process at a time
- **BLOCKED** (dt. blockiert) blocks on a specific event
  - the process performs **input/output operations**, it performs its I/O burst
  - it expects the fulfillment of at least one condition

State Transitions

focus:
short-term scheduling



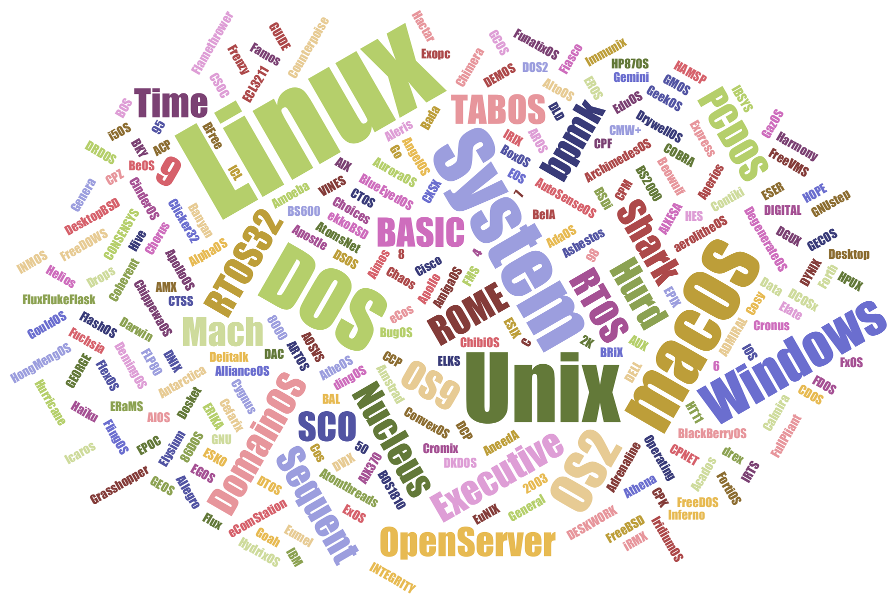
Process Dispatching: Time and Selection

- transitions to the ready state (READY) <u>update</u> the CPU ready list
  - a decision regarding the **placement** of the **process control block** is made
  - the result depends on the system's **CPU scheduling strategy**
- scheduling or rescheduling (dt. Einplanung/Umplanung) occurs, ...
  - 1 after a process has been created
  - 2 when a process yields control of the CPU
  - 3 if the event which is expected by a process has occurred
  - 4 as soon as an swapped-out process is resumed

A process can be **forced** to release the CPU → **preemptive scheduling**

- for example, by a timer interrupt 2

Agenda



- Recap
- Organizational Matters
- Scheduling, State Transitions
- CPU Scheduling Strategies
  - Basic: First-Come First-Served
  - Time-slice Driven: Round Robin, Virtual Round Robin
  - Prediction Driven: Shortest Process Next, Highest Response Ratio Next
  - Priority Driven: Multilevel Feedback Queues
- Discussion: Reflection and Comparison
- Scheduling in Practice:
  The Linux O(1) Scheduler
- Summary and Outlook

Basic: First-Come First-Served (FCFS)

- First-Come First-Served: a simple and fair procedure?
  - queuing criterion is the **arrival time** of a process
  - **non-preemptive**, thus requires cooperative processes
- Example:

Process	Times					
	Arrival	Service T_s	Start	End	Turnaround T_r	T_r/T_s
A	0	1	0	1	1	1.00
B	1	100	1	101	100	1.00
C	2	1	101	102	100	**100.00**
D	3	100	102	202	199	1.99
Average					100	26.00

  - discussion: **service time** (dt. Bedienzeit) vs. **turnaround time** (dt. Durchlaufzeit)
  - wanted: proportionality between service time and turnaround time
  - example: the normalized execution time (T_r/T_s) of process C is very poorly in comparison to its service time T_s

Discussion: FCFS – The Convoy Effect

- the problem is faced by short-running **I/O-heavy processes** that follow long-running **CPU-heavy processes**
  - processes with **long CPU bursts** are **favored**
  - processes with **short CPU bursts** are **penalized**
- FCFS **minimizes** the number of **context switches**. However, the convoy effect causes the following problems:
  - high response times
  - low I/O throughput
- FCFS is therefore unsuitable for a mixed workloads (CPU- and I/O-heavy processes)
- typically, FCFS is only in used in batch processing systems

Time-slice Driven: Round Robin (RR)

- reduces the **penalization of processes with short CPU bursts** that occurs with FCFS:
  - the processor time is divided into **time slices** (dt. Zeitscheiben)
- when the time slice expires, a process change <u>may</u> take place
  - the interrupted process is pushed to the end of the ready list
  - the next process is taken from the ready list according to FCFS
- basis for CPU protection: timer interrupt enforces end of time slice
- the **time slice length** determines the effectiveness of this scheduling method
  - time slice
    - ➔ too long: degeneration of RR to FCFS
    - ➔ too short: high scheduling overhead
  - rule of thumb: slightly longer than the duration of a "typical interaction"

Discussion: Round Robin – Performance Issues

- **I/O-heavy** processes finish CPU burst within their time slice
  - they block and return to the ready list at the end of their I/O burst
  - it is likely that their time slice has not yet been exhausted
- **CPU-heavy** processes, on the other hand, make full use of their time slice
  - they are preempted and immediately return to the ready list
- result: **CPU time is unevenly distributed** in favor of CPU-heavy processes
  - causal link: I/O-heavy processes are poorly operated and thus devices poorly utilized
  - response time of I/O-heavy processes increases

Time-slice Driven: Virtual Round Robin (VRR)

- VRR addresses the uneven distribution of CPU times that is possible with RR
  - processes are added to a *preferred list* at the end of their I/O bursts
  - scheduler processes *preferred list* **before** the *ready list*
- the method works with **time slices of variable lengths**
  - processes on the *preferred list* are **not** allocated a full time slice
  - instead, they are granted the remaining term of their previously not fully used time
  - if their CPU burst takes longer, they are preempted and put back to the ready list
- process handling more complex compared to RR → overhead

Agenda

- Recap
- Organizational Matters
- Scheduling, State Transitions
- CPU Scheduling Strategies
  - Basic: First-Come First-Served
  - Time-slice Driven: Round Robin, Virtual Round Robin
  - Prediction Driven: Shortest Process Next, Highest Response Ratio Next
  - Priority Driven: Multilevel Feedback Queues
- Discussion: Reflection and Comparison
- Scheduling in Practice:
  The Linux O(1) Scheduler
- Summary and Outlook

Prediction Driven: Shortest Process Next (SPN)

- reduces the disadvantage of short CPU bursts that occurs with FCFS: "the little ones to the front"
  - necessary basis: knowledge of the process execution times
  - preemption does **not** happen
- the main problem is the **estimation** of the CPU burst time
  - batch operation: programmers **specify** the required **time limit***
  - interactive operation: **estimation** based of the **previous bursts** lengths of the process
- response times are significantly reduced and overall performance increases
  - but: risk of **starvation** of CPU-heavy processes

* the execution time within which the job will (probably/hopefully) be completed before it is canceled

Discussion: SPN – Estimate CPU Burst Time (I)

- estimate next CPU burst (S_{n+1}): calculate average over all previous CPU bursts lengths (T_i) of a process

$$S_{n+1} = \frac{1}{n} \cdot \sum_{i=1}^{n} T_i$$

- **challenge** of this calculation is the equal **weighting** of all CPU bursts
  - however, recent CPU bursts are more important than older ones
  - hence, greater weighting for more recent process behavior
- cause is the **principle of locality**

Discussion: SPN – Estimate CPU Burst Time (II)

- the CPU bursts further back are to be given less weight:

$$S_{n+1} = \alpha \cdot T_n + (1-\alpha) \cdot S_n$$

  - for the constant **weighting factor α** holds: $0 < \alpha < 1$
  - it reflects the **relative weighting** of the of individual CPU bursts of the time series

Exponential Smoothing

- recursive insertion leads to...

$$S_{n+1} = \alpha T_n + (1-\alpha) \alpha T_{n-1} + \ldots + (1-\alpha)^i \alpha T_{n-i} + \ldots + (1-\alpha)^n S_1$$

$$S_{n+1} = \alpha \cdot \sum_{i=0}^{n-1} (1-\alpha)^i T_{n-i} + (1-\alpha)^n S_1$$

- for α = 0.8:

$$S_{n+1} = 0.8 T_n + 0.16 T_{n-1} + 0.032 T_{n-2} + 0.0064 T_{n-3} + \ldots$$

Problem: Starvation!

Prediction Driven: Highest Response Ratio Next (HRRN)

- **prevents** the **starvation** of CPU-heavy processes
  - scheduler considers **aging (i.e., the waiting time)** of processes

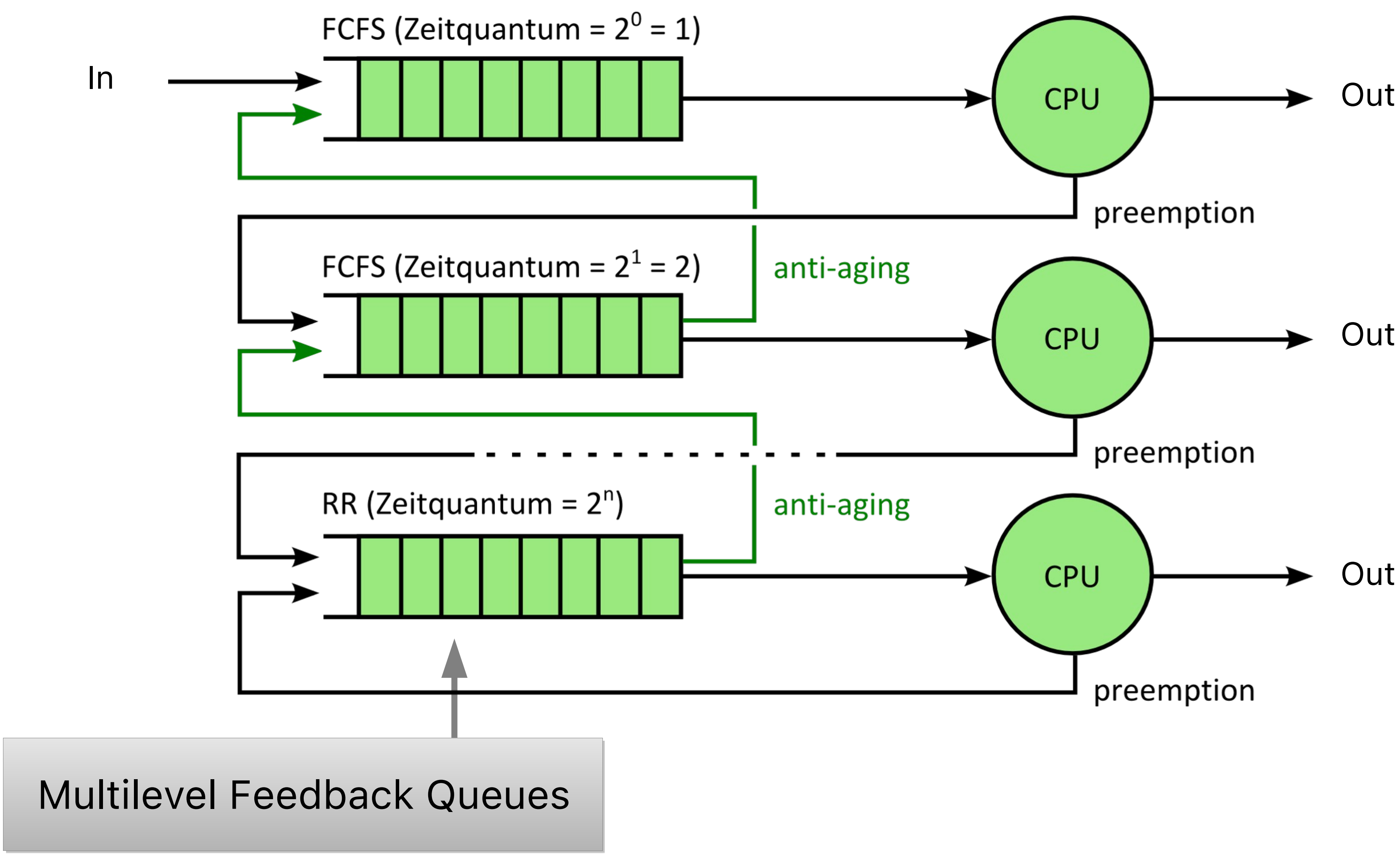
$$R = \frac{w+s}{s}$$

  - w is the "waiting time of the process" so far
  - s is the "expected service time"
  - R is the **response ratio**
- the process with the **highest response ratio is chosen**
- based again on the estimate of the CPU burst time

Priority Driven: Multilevel Feedback Queues

- favors short processes **without estimating** the CPU burst time of the processes
  - principle: "punishment" (dt. Bestrafung) of processes with long execution time
  - processes are subject to preemption
- **multiple** ready lists, depending on the number of **priority levels**
  - when a process first arrives, it runs at the highest level
  - with the expiration of its time slice, he comes to the next lower level
  - the lowest level operates according to RR
- processes with short execution time complete relatively quickly
- processes with long execution time are prone to starvation
  - the waiting time can be considered to regain higher levels (**anti-aging**)

Priority Driven: Multilevel Feedback Queues



Discussion: Priority Driven

- process priority := a process "precedence" that significantly influences scheduling decisions by the operating system

- **static priorities** are set at the time of process generation
  - the priority level is **not** changed during execution
  - this method enforces a deterministic order between processes

- **dynamic priorities** are calculated and updated during the process execution
  - updates takes place in the operating system, but also from the user
  - SPN, HRRN, and MFQ are special cases of this method

Discussion: Priority Inversion

- priority inversion: scheduler dispatches a **process with lower priority** while another process with **higher priority is blocked** (on a resource held by a process with low priority)
- cross-cutting concerns of OS scheduler and OS process synchronisation primitives and protocols
- possible solutions:
  - appropriate synchronisation methods to protect critical sections
    - → non-blocking synchronisation
  - priority ceiling protocols

Agenda

- Recap
- Organizational Matters
- Scheduling, State Transitions
- CPU Scheduling Strategies
  - Basic (First-Come First-Served)
  - Time-slice Driven (Round Robin, Virtual Round Robin)
  - Prediction Driven (Shortest Process Next, Highest Response Ratio Next)
  - Priority Driven (Multilevel Feedback Queues)
- **Discussion: Reflection and Comparison**
- Scheduling in Practice:
  The Linux O(1) Scheduler
- Summary and Outlook

Scheduler Evaluation Criteria and Goals

- goals from perspective of a **user**:
  - **processing time** – time between arrival and completion of a process including waiting time(s) → **batch processing**
  - **response time** – time between user input and system response → **interactive systems**
  - **deadline adherence** – deadlines *should* be met for the interaction with external physical processes → **real-time systems**
  - **predictability** – processes are always handled the same regardless of the load → **hard real-time systems**
- goals from the perspective of the **system**:
  - **throughput** – process as many processes as possible per time unit
  - **CPU utilization** – CPU should always be busy, if possible. Overhead (scheduling decisions, context switches) should be minimised.
  - **fairness** – no process should be penalized (avoid starvation)
  - **load balancing** – I/O devices should also be evenly utilized

Comparison – Quantitative Evaluation

	Process	A	B	C	D	E	
	Arrival	0	2	4	6	8	**Average**
	Service T_s	3	6	4	5	2	
FCFS	Finish	3	9	13	18	20	
	Turnaround T_r	3	7	9	12	12	8.60
	T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR	Finish	4	18	17	20	15	
q=1	Turnaround T_r	4	16	13	14	7	10.80
	T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
SPN	Finish	3	9	15	20	11	
	Turnaround T_r	3	7	11	14	3	**7.60**
	T_r/T_s	1.00	1.17	2.75	2.80	1.50	**1.84**
HRRN	Finish	3	9	13	20	15	
	Turnaround T_r	3	7	9	14	7	8.00
	T_r/T_s	1.00	1.17	2.25	2.80	3.50	2.14
MFQ	Finish	4	20	16	19	11	
q=1	Turnaround T_r	4	18	12	13	3	10.00
	T_r/T_s	1.33	3.00	3.00	2.60	1.50	2.29

Based on William Stallings, „Operating Systems – Internals and Design Principles"

Service T_s: Bedienzeit
Turnaround T_r: Durchlaufzeit
T_r/T_s: Größenverhältnis von Bedien- und Durchlaufzeit (kleiner ist besser)

Comparison – Qualitative Evaluation

scheduling strategy	preemptive/ cooperative	prediction necessary?	implementation effort	prone to starvation?	impact on processes
FCFS	cooperative	no	minimal	no	convoi effect
RR	preemptive (timer)	no	small	no	fair, but puts I/O-heavy processes at a disadvantage
SPN	cooperative	yes	big	yes	puts CPU-heavy processes at a disadvantage
HRRN	cooperative	yes	big	no	good load balancing
MFQ	preemptive (timer)	no	huge	yes	may favor I/O-heavy processes

Based on William Stallings, „Operating Systems – Internals and Design Principles“

Agenda

- Recap
- Organizational Matters
- Scheduling, State Transitions
- CPU Scheduling Strategies
  - Basic (First-Come First-Served)
  - Time-slice Driven (Round Robin, Virtual Round Robin)
  - Prediction Driven (Shortest Process Next, Highest Response Ratio Next)
  - Priority Driven (Multilevel Feedback Queues)
- Discussion: Reflection and Comparison
- **Scheduling in Practice: The Linux O(1) Scheduler**
- Summary and Outlook

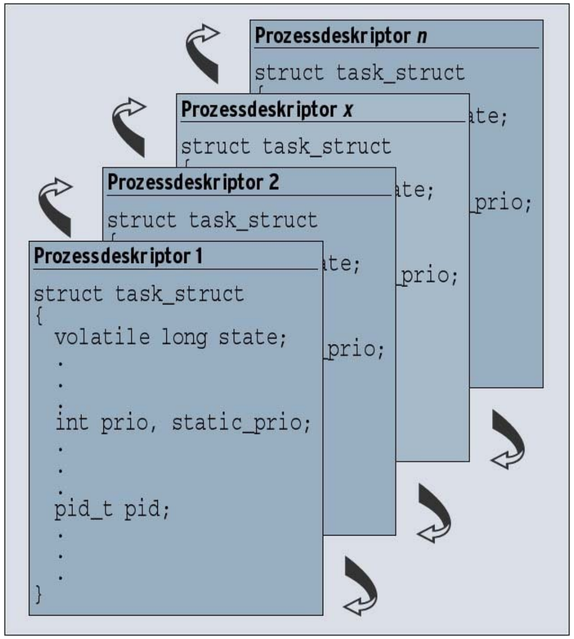
The Linux O(1) Scheduler

- **motivation**
  - runtime complexity of Linux 2.4 scheduler (until 2002): $O(n)$
  - thus, poor performance with increasing number of processes
  - issue: **overhead scales linear** with **number of processes**

- ***O*(1) process scheduler**
  - Linux 2.6 (2002-2007): improved scheduler which reduces process **scheduling efforts** to a **constant** amount of work ($O(1)$) – independent of the number of processes
  - **goals:** reduce scheduling overhead, improve system responsiveness for interactive processes

The Linux O(1) Scheduler

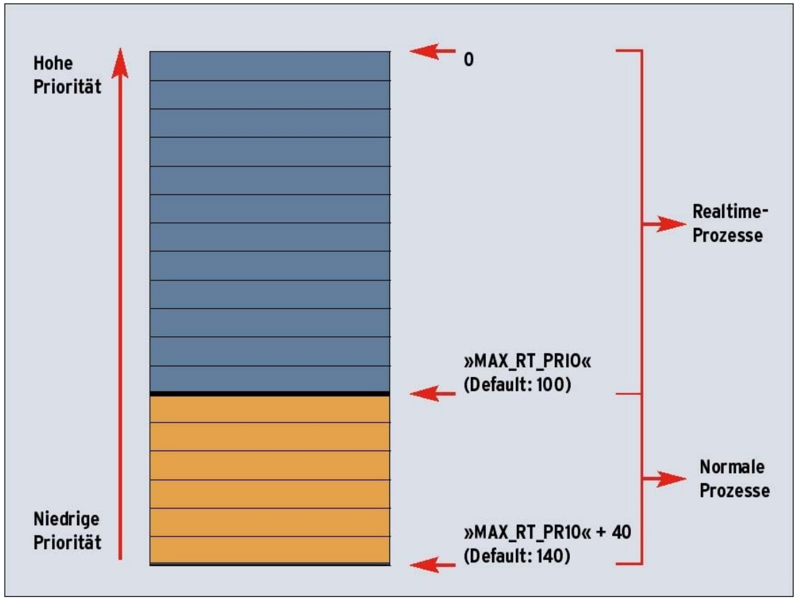
- operating principle and implementation
  - each CPU: individual ready lists, avoiding concurrent access by different CPUs
  - each run queue consists of two lists:
    active list (ready processes, time available)
    expired list (ready processes, expired time)
  - preemptive scheduling, process priorities



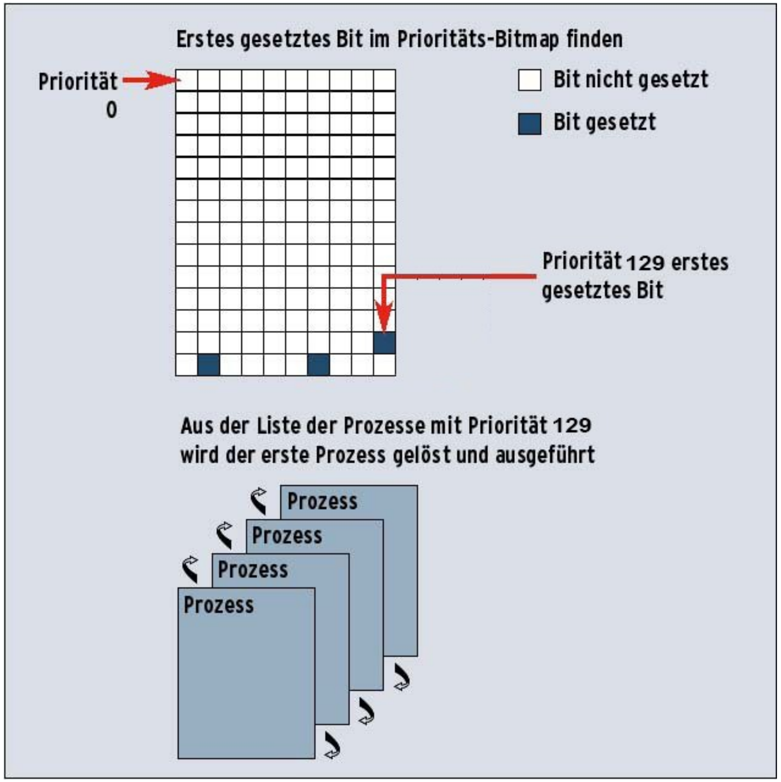
- reduced overhead, improve performance
  - decreased memory pressure on the kernel stack
  - bitmaps for implementing process priorities
  - variable time slices (10 ... 200 ms)

The Linux O(1) Scheduler

- process priorities
  - highest priority: 0
    lowest priority: 139
  - priority is based on **nice** value of each process (100 … 139)
  - additional priorities of **real-time** processes (0 … 99)
- tracking processes and priorities
  - bitmap tracks for which **priority** at least one **ready process exists**, references **double linked** list of corresponding processes
  - scheduler handles each priority level with **RR**

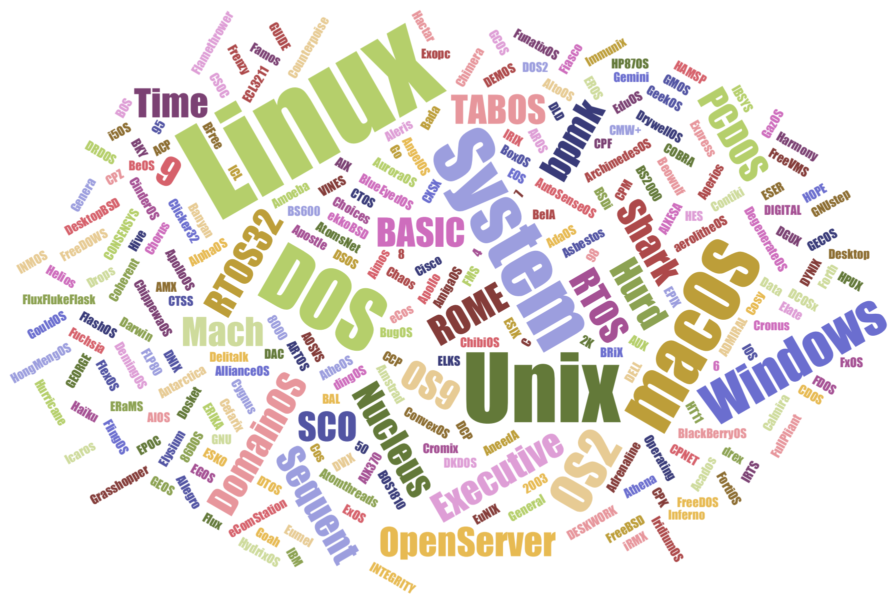


The Linux O(1) Scheduler



- when the last **process** of a given priority **terminates**, the corresponding **field in the bitmap** is set to **null**
- scheduler moves to **next** (lower) **priority level** and continues
- when **active** run queue is empty:
  - exchange **active** and **expired** run queue
  - repeat until processes exhaust their time slices, again
- O(1): scheduling effort is constant (proportional to the size of the bitmap)

Agenda



- Recap
- Organizational Matters
- Scheduling, State Transitions
- CPU Scheduling Strategies
  - Basic (First-Come First-Served)
  - Time-slice Driven (Round Robin, Virtual Round Robin)
  - Prediction Driven (Shortest Process Next, Highest Response Ratio Next)
  - Priority Driven (Multilevel Feedback Queues)
- Discussion: Reflection and Comparison
- Scheduling in Practice: The Linux O(1) Scheduler
- **Summary and Outlook**

⏭ Summary and Outlook

- **summary**
  - operating systems make **CPU scheduling** decisions at three scheduling levels:
    - **long-term** scheduling: allowance of processes to the system
    - **medium-term** scheduling: swap-out and swap-in of processes
    - **short-term** scheduling: allocation of CPU time for the execution of processes
  - **short-term scheduling methods**
    - **basic**
    - **advanced** algorithms based on time slices, prediction, priorities
  - the Linux O(1) scheduler – **practical process scheduler** to improve system responsiveness
- **outlook: inter-process communication**
  - **communication between process** to exchange data (e.g., for cooperation)
  - methods: shared memory and messages

References and Acknowledgments

Lecture

- Systemnahe Programmierung in C (SPiC), Betriebssysteme (Jürgen Kleinöder, Wolfgang Schröder-Preikschat)
- Betriebssysteme und Rechnernetze (Olaf Spinczyk, Embedded Software Systems Group, Universität Osnabrück)

Teaching Books and Reference Book

[1] Avi Silberschatz, Peter Baer Galvin, Greg Gagne: *Operating System Concepts*, John Wiley & Sons, 2018.

[2] Andrew Tanenbaum, Herbert Bos: *Modern Operating Systems*, Pearson, 2015.

[3] Wolfgang Schröder-Preikschat: *Grundlage von Betriebssystemen – Sachwortverzeichnis*, 2023.
https://www4.cs.fau.de/~wosch/glossar.pdf