3 Scheduling

Problems

Kernel Scheduler

User Level Scheduler

Intended Schedule

- Motivation
- Abstract Scheduling Problem
- Scheduling Goals
- Scheduling Policies
- Priority Scheduling and its problems
- Hints to Assignment 1

Schedules & Scheduling?

- Lecturer hands out intended schedule of this course
  - which topic at what date
- Schools/universities etc. need schedules for their various classes, courses, i.e.
  - course
  - time
  - location
- Furthermore, there are schedules for
  - Trains
  - Airlines
  - Ships, fairies
- Travel agency people are experts in scheduling

Example Problem

- Find an appropriate traffic solution for a
  - flight to Sydney via
  - Bahrain and
  - Singapore
- Book a car and a hotel near the conference hall

⇒ *Scheduling* has to be done

Scheduling ~ planning “minor or major events”, e.g. elections, examinations, weddings, recipes, etc.

Abstract Scheduling Problem

Abstract Scheduling Problem*

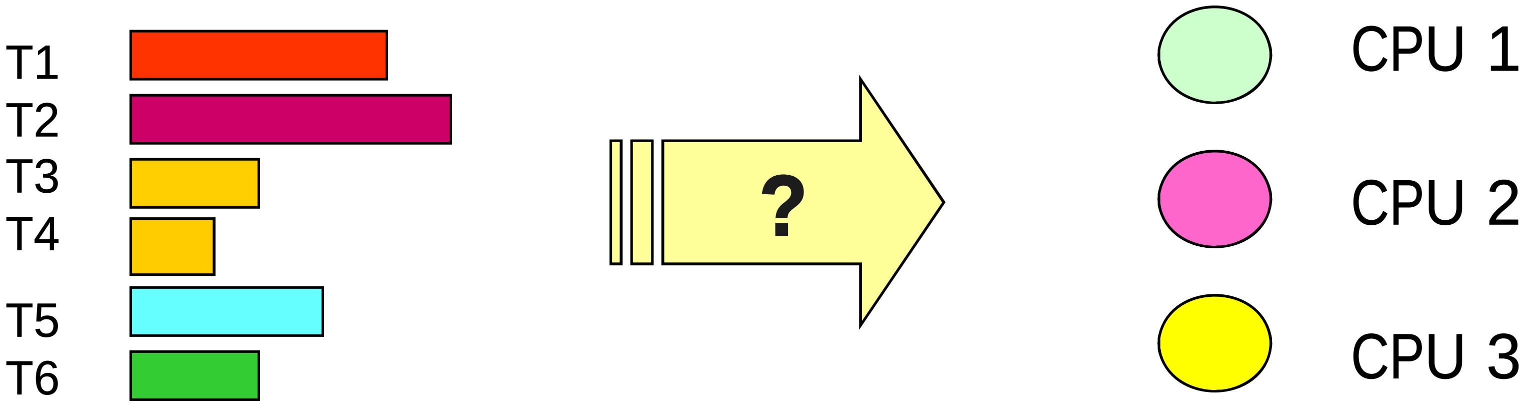
How to map executable units of activity (threads) to executing units (processors)?

Criteria how to schedule?

- overall system goals or
- specific application constraints:
  - Time critical threads should meet their "*deadlines*", neither too early, nor too late
  - Fast response to an interactive input

*Simplification: Only focus on the resource CPU

Abstract Scheduling Problem



How to map these 6 threads to 3 CPUs?
Is there an optimal schedule?

As long as there is no performance measure,
we can neither produce a good, nor a bad schedule

Concrete Scheduling Problems

- In a multi-programming system n > 1 processes (KLTs) can be ready

Which of these processes (KLTs) should run next?

- You're watching a Beatles (...) video
  - *How to manage that*
    - network-software
    - data stream decoding
    - output to screen and
    - audio

    is well done concurrently?
  - Additionally, you have initiated a long running compute-bound job in the background. *When to switch to it?*

- In a multi-threaded application a programmer wants to influence, how her/his threads are scheduled

Concrete Scheduling Problems

- In assignment 1 you must emulate a user-level scheduler
- *What does a scheduler need to know to do its job?*
  - It must know the system state and each process's state, i.e. all relevant scheduling information of each candidate and each resource
  - Related information per KLT/PULT has to be provided at user-level
- You have to install your own TCBs
- *How to find a specific TCB?*
- *What information has to be provided per TCB?*

Scheduling Goals

Quantitative

Qualitative

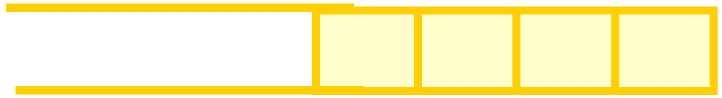
Quantitative Scheduling Goals

- CPU Utilization
  - *When is a CPU unused?*
- Throughput
  - Number of completed jobs per time
- Response Time
- Turnaround Time
- Waiting Time
- Number of Deadline Violations
- Lateness
- Tardiness

(Number of Deadline Violations, Lateness, Tardiness: } Real Time Problems)

What is included in a Waiting Time?

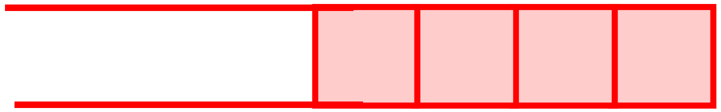
Waiting time?



1. Time a process spends in the ready queue

   influenced by current load & by scheduler

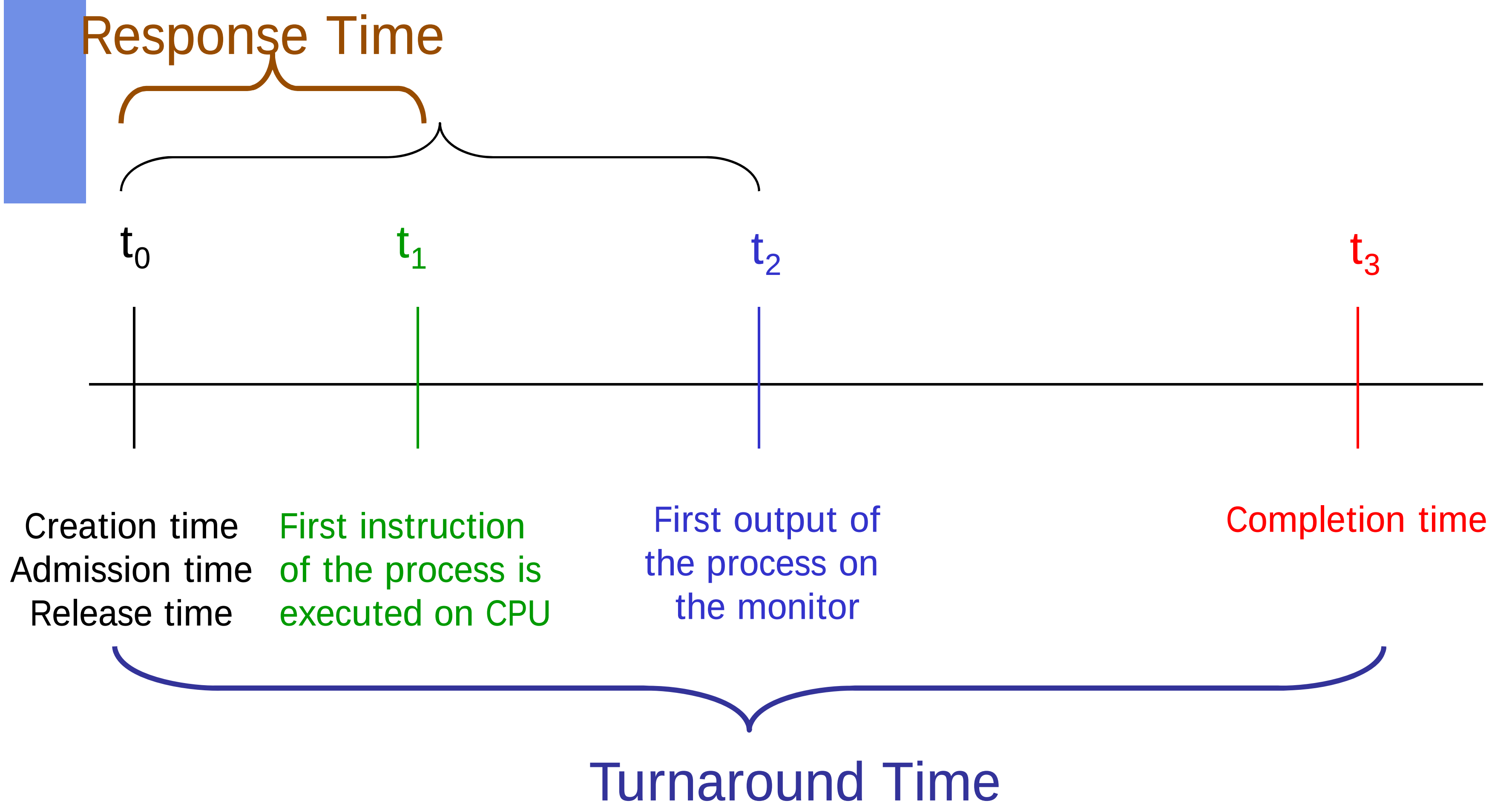
2. Time a process/thread is blocked, i.e. due to
   - missing message
   - missing input
   - missing resource



- Blocked processes/threads should not hold a CPU
- Kernel stores them in a separate data structure, the waiting queue(s)

influenced by process or resource shortage

Response/Turnaround Time



Qualitative Goals

- Predictability
  - Low variance in turnaround times and/or response times of a specific task
  - System guarantees certain quality of service
- Fairness
  - Few starving applications
  - In MULTICS, when shutting down the machine, they found a 10 year old job
- Robustness
  - Few system crashes
  - The simpler the system, the more robust

Scheduling Policies

System Environment

Principle Components of Scheduling

System Environment

Different Systems require different scheduling policies

- Computer server
  - Use budgets (due to contracts) to fulfill requirements of its clients
  - Distinguish between high cost and low cost applications
- Desktop Computer
  - Multiple interactive & batch jobs preferring interactive ones
  - Offer foreground and background jobs
- Soft Real Time
  - Distinguish inside an application mandatory and optional parts, the latter might only improve the quality of a video or audio recording, but are not necessary

Characteristics of a Scheduling Policy

- Scheduling order: where in the ready queue(s) to place a new (or unblocked) thread
- Selection: which ready thread to run next
- Decision mode: when to execute the selection function
  - Non preemptive

    Once a thread is running, it will continue until it
    - terminates
    - yields
    - blocks (e.g. due to I/O or due to a wait())
  - Preemptive

    A running KLT or process is preempted when
    - a more urgent work has to be done or
    - a process or KLT has expired its time slice

Survey on Scheduling Policies

- FCFS = first come first served
- (R)SJF = (remaining) shortest job first

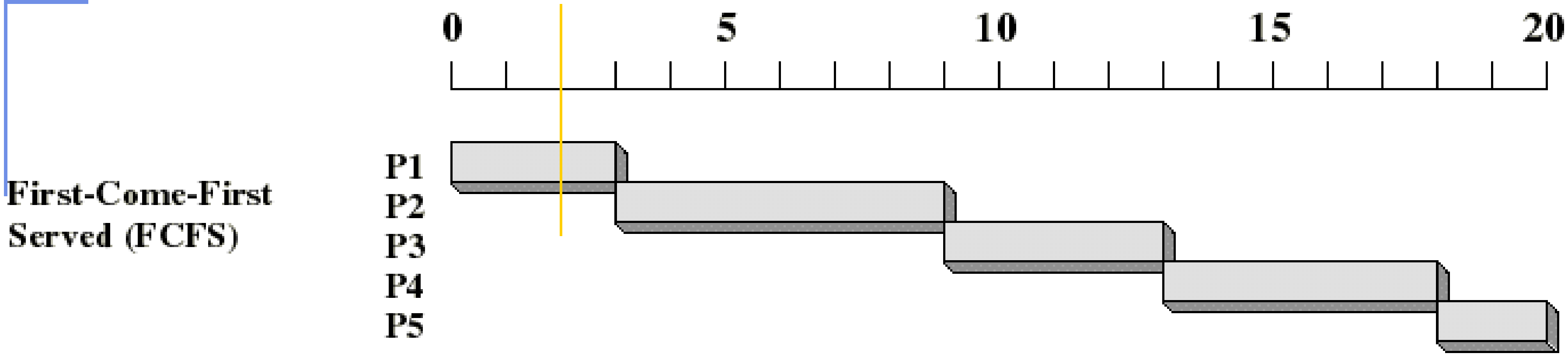
needs at least estimation of execution time

- RR = round robin
  - System wide constant time-slice
  - Job (class) specific time-slice
- MLF = multi-level feedback
- Priority
  - Static priority values
  - Dynamic priority values
- ...

Why ∃ Different Scheduling Policies?

- Different application scenarios
- Different performance measures
  - Response time
  - Turnaround time
  - Throughput
  - ...

First Come First served



- Ready queue: ordered according to *start times*
- Selection function: select the *oldest ready thread*
- Decision mode: non preemptive (or preemptive)
  - *Which one to chose?*

Remark: Many things in daily life are scheduled according to FCFS.
It's quite fair, but not usable under certain circumstances.
Give examples

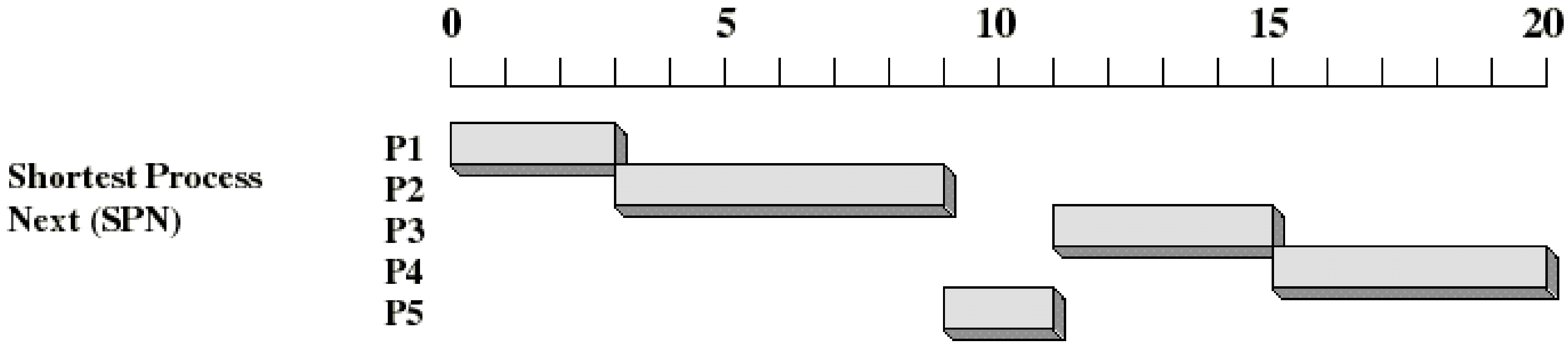
Implementation Remarks

- *What information do you need to implement strict FCFS?*
- *Suppose your process does a blocking I/O. How to deal with this process when its I/O has finished? Do you have to preempt the currently running process?*

Idea:

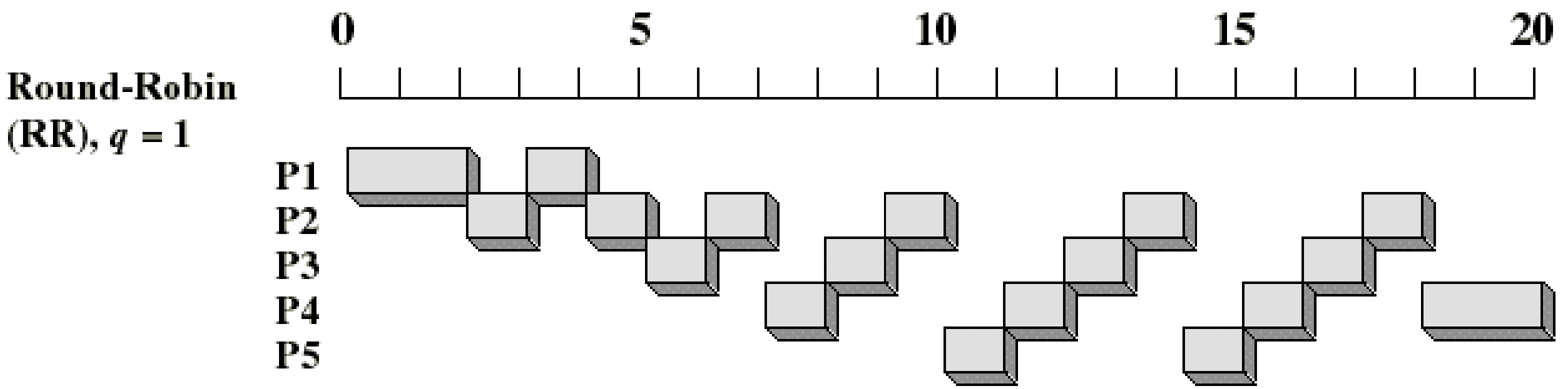
Whenever you have to fill the PCB into a queue, do it according to increasing start times, i.e. the head of the queue must be the senior

Shortest Job First



- Ready queue: *How to order?*
- Selection function: thread with the shortest (expected) execution (burst) time
- Decision mode: non preemptive
- We need to *estimate* the required processing time (CPU burst time) for each thread

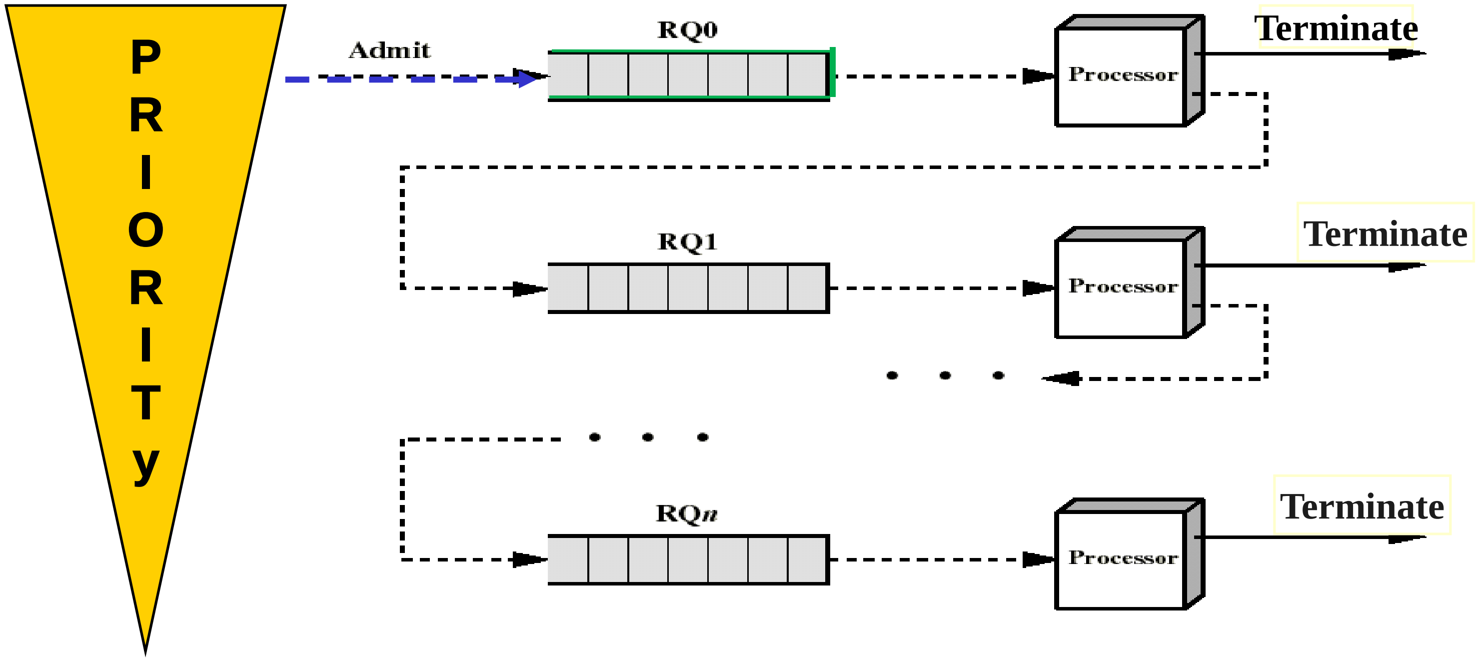
Round Robin



- Ready queue: Append each new ready entry
- Selection function: select first thread in ready queue
- Decision mode: "time" preemptive
  - A non cooperative thread is allowed to run until its time slice TS ends (TS $\in$ [0.1, 100]* ms)
  - When a timer interrupt occurs, the running thread is *appended* to the ready queue

* Depends on the application system & on the CPU speed

Multilevel Feedback in CTTS[1]



Selection: first thread in highest ready queue **RQ_0**

Decision mode: Preemptive (at least due to time slices)
However, you may also add priority preemption

Whenever a thread is unblocked after an I/O it is admitted to **RQ_0**

[1]CTSS started in 1961 at MIT, used until 1973 (reused in MULTICS)

Analysis: Multilevel Feedback Policy

- MLFB approximates SRTF:
  - CPU bound KLTs drop like a rock (they might starve)
  - Short-running I/O bound jobs stay near the top
- Scheduling must be done between the queues
  - Fixed priority scheduling:
    - select a KLT from RQ_i, only if RQ_{i-1} to RQ_0 are empty
  - Time slice:
    - each queue has an individual TS
- Countermeasure = user action foiling the intent of the OS designer
  - Put in a bunch of meaningless I/O to keep KLTs priority high
  - Example of Othello program:
    - insert **`printf`**'s, program ran much faster

Priority Scheduling

Selection function: ready thread with highest priority

Decision mode: non preemptive, i.e. a thread keeps on running until it

- cooperates (e.g. yielding) or
- blocks itself (e.g. initiating an I/O) or
- terminates

Drawbacks: Danger of *starvation* and *priority inversion*

Remark:
Priority based scheduling is often done *with preemption* and *with dynamic priorities*

Problems with Static Priorities

Thread with highest priority runs on CPU

What will happen when this thread is calling **`yield()`***?*

- After a minor delay due to execution time of **`yield()`** the calling thread will run again if ∃ no other ready thread with the same or even a higher priority

Further Problems with Priorities?

- Priority Inversion
  - Mars pathfinder
- Deadlocks
  - Mutual waiting
- Spin Locks
  - Active waiting
- Proper mapping of priority values to KLTs or to processes

Events leading to a Thread Switch

- **`yield()`** works fine if there are other threads with the same priority value
- A thread WT is calling a method of a synchronized class with an internal **`wait()`**
  - WT waits until its partner send a notify
- Partner thread ST does a **`notify()`** within another method of the same synchronized class, whereby thread WT only runs if its priority is higher than the one of thread ST
- A thread **`returns`** or **`exits`** otherwise

Assignment #1 a

- Java Version 1.4 (and later versions)
  - Threads are Kernel Level Threads ⇒
    - scheduling can *hardly* be influenced by the Java VM and
    - it depends heavily on kernel's scheduling policy
    - yielding sets a KLT's state to runnable ⇒ kernel-scheduler may schedule this thread again right after it has yielded
    - What about **`sleep()`**, **`wait()`** & **`notify()`**?