Multitasking & Scheduling

Klaus-Peter Zauner

COMP2215: Computer Systems II
Operating System (OS)

Uniform Interface
- ... for users
- ... for programs

Management of Resources
- CPU time
- Memory
- Access to devices

Management of Interactions
- Desired: networking, comms. among users/processes
- Undesired: protection, security
# Operating System (OS)

## Uniform Interface
- ... for users
- ... for programs

## Management of Resources
- CPU time
- Memory
- Access to devices

## Management of Interactions
- Desired: networking, comms. among users/processes
- Undesired: protection, security
OS: Multiprogramming

- Multiple programs available for execution on CPU
- If one process needs to wait for I/O, the CPU can work on another process
  - increased throughput
  - increased CPU utilization
- Key issue: protection

⟶ Pseudo-parallelism
Microcontroller: Multitasking

- Typically all programs in memory
- CPU can go to sleep
- More CPUs is often an option
- Key issues:
  - deadlines
  - priorities
  - starvation
Processes

Process/Task/Thread

▶ ... an abstraction of a program in execution

A process has an entry in the process table that typically contains:

▶ Instruction counter and stack pointer
▶ Assigned address space
▶ Associated resources
▶ State

Depending on the complexity of switching between and the level of isolation among processes one also talks about tasks and threads.
Process States

- **Running** → currently using the CPU
- **Ready** → runnable, but stopped to let another process run
- **Blocked** → unable to run until an external event happens
There is one running process, but a ready set and blocked set.
State Transitions of Processes

Processes typically alternate bursts of computing with I/O requests. While waiting for I/O they are blocked from CPU access.

- When does a process become ready?
- When does a process become blocked?
- What can happen if more than one process is blocked?
- What should happen if more than one process is ready?
State Transitions of Processes

Processes typically alternate bursts of computing with I/O requests. While waiting for I/O they are blocked from CPU access.

- When does a process become \textit{ready}? \\
- When does a process become \textit{blocked}? \\
- What can happen if more than one process is \textit{blocked}? \\
- What should happen if more than one process is \textit{ready}?
State Transitions of Processes

Processes typically alternate bursts of computing with I/O requests. While waiting for I/O they are blocked from CPU access.

- When does a process become *ready*?
- When does a process become *blocked*?
- What can happen if more than one process is *blocked*?
- What should happen if more than one process is *ready*?
State Transitions of Processes

Processes typically alternate bursts of computing with I/O requests. While waiting for I/O they are blocked from CPU access.

- When does a process become \textit{ready}?
- When does a process become \textit{blocked}?
- What can happen if more than one process is \textit{blocked}?
- What should happen if more than one process is \textit{ready}?
State Transitions of Processes

Processes typically alternate bursts of computing with I/O requests. While waiting for I/O they are blocked from CPU access.

▶ When does a process become ready?
▶ When does a process become blocked?
▶ What can happen if more than one process is blocked?
▶ What should happen if more than one process is ready?
Scheduling

Scheduler decides which process from the set of ready processes will get the CPU next.

The requirements for the scheduler differ according to the nature of the processes and the computer system:

- Fairness
- Response time
- Throughput
- Turnaround
- CPU utilization
- Deadlines (hard/soft)
- Predictability
- Adherence to policy
Scheduling Algorithms
According to what criteria can/should the scheduler select?

- First-come first-served
- Shortest job first
- Round robin
  - quanta, preemptive, context switch
- Shortest remaining time
- Priority Scheduling
- Lottery Scheduling
  - fraction of CPU, no guarantees
Scheduling Algorithms

- Properties of processes (e.g., length of job) may need to be estimated.
- Often combinations of the algorithms are used, e.g.:
  First-come first-serve from different priority queues with demotion to a lower queue after each consumed time quantum and round robin in the lowest priority queue.
Context Switching

- Preemptive OS → OS can switch among processes

Hardware Interrupts or Software Interrupts ("Traps") are used to switch among processes.
Context Switching

- Preemptive OS → OS can switch among processes
- Hardware Interrupts or Software Interrupts ("Traps") are used to switch among processes
Context Switching

- Switch from user process to kernel process
- Switch from kernel process to user process

Context switching requires CPU time—this time is wasted.
Context Switching

- Switch from user process to kernel process
- Switch from kernel process to user process

← possibly the same

Context switching requires CPU time—this time is wasted.

... time for what?
Round robin

- Every process gets a time slice and is served in a fixed order

- **Short Quantum:**
  - overhead of context switching is high

- **Long Quantum:**
  - long response time
  - good efficiency → processes block for I/O before quantum expires

- Typically 20–50 ms
Scheduling Scenarios

- Interactive Applications (maybe malicious)
  - User on terminal
  - Client on server
- Real-time tasks (cooperative)
- Batch processing (typically cooperative)
Real-time Scheduling
Constraints on Computation

1. Correct

2. within Deadline
   - soft real-time systems
   - hard real-time systems

3. within Power budget
Real-time Systems

- Interaction with physical environment
  - vehicle, power plant, robot, assembly line

- Timing constraints
  - A correct reaction too late is not useful

Properties of Interest

- Fine-grained priority control
- Determinism
- Responsiveness
- Reliability
- Stability
Real-time: Deadlines

- Tasks have deadlines
- Tasks can be periodic or aperiodic
- Task duration can be constant or not

**Deadline**
The latest time by which a task has to be completed.

- Often require predictable behaviour ➔ guarantees
CPU Utilization $U$

\[ U = c_{\text{total}} - c_{\text{idle}} \leq 1 \]

$c_{\text{total}} \rightarrow$ Total CPU time available (100%)
$c_{\text{idle}} \rightarrow$ Fraction of CPU time spent in idle task or sleeping
Assumptions for Analysis

1. Tasks are periodic  
   ▶ convert aperiodic tasks by polling
2. The deadline for a task is its next invocation
3. Context switches take no time  
   ▶ leave some margin in duration and deadline
Load from Set of Periodic Tasks

\[ U = c_{\text{total}} - c_{\text{idle}} \leq 1 \]

Given a set of tasks \( T_1 \cdots T_n \) with periodicity \( p_i \) and fixed CPU time \( c_i \) for \( T_i \) the utilization is:

\[
U = \sum_{i=1}^{n} \frac{c_i}{p_i}
\]
A Real-time System is schedulable if

\[ U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1 \]

From this we know for a system in which the CPU requirements and periods for all tasks are known, whether it is possible to meet the deadlines for all tasks.
How to schedule it?