ECE382M.20: System-on-Chip (SoC) Design

Lecture 7 – Task Scheduling

With sources from:
Prof. Margarida Jacome, UT Austin
Prof. Peter Marwedel, Univ. of Dortmund

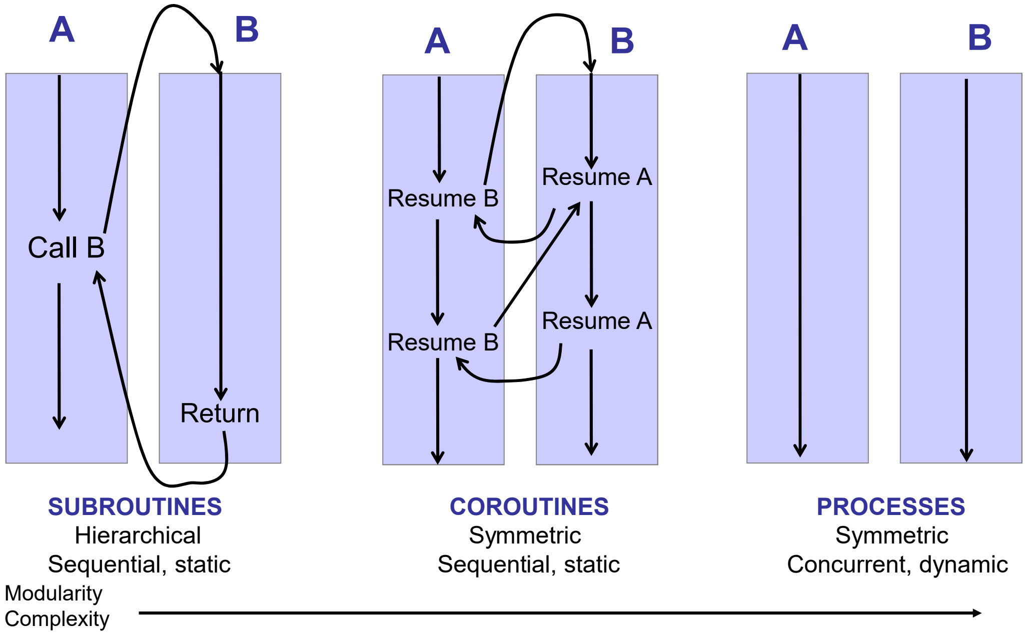
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Lecture 7: Outline

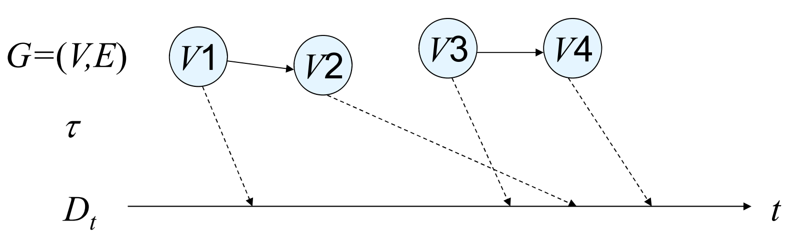
- **Uni-processor scheduling**
  - Aperiodic tasks
  - Periodic tasks
  - Dependent tasks
- **Special considerations**
  - Context-switch times
  - Interrupts
- **Muti-processor scheduling**
  - MPSoC synthesis

Multiplexing Software Modules



Scheduling

- **Assume that we are given a task graph $G=(V,E)$**
- **A *schedule* τ of G is a mapping $V \rightarrow D_t$ of a set of tasks V to start times from domain D_t, such that none overlap**



➢ **Find such a mapping**
  - Optimize throughput (rate of G), latency (makespan of G)
  - Resource, dependency, real-time (deadline) constraints

Task Scheduling Problems

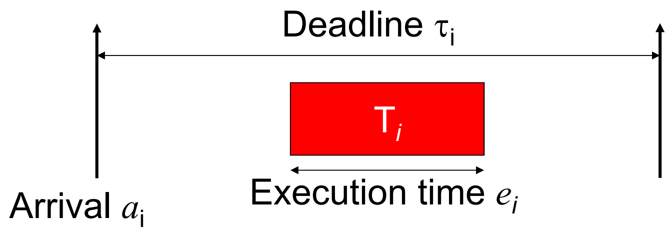
- **Task types**
  - Periodic
    - Set of tasks $\{ T_1, T_2, \ldots \}$
    - Period t_i
    - (Worst-case) execution time e_i
  - Aperiodic/sporadic
    - Arrival/release time a_i
- **Task dependencies**
  - Tasks with precedence constraints
    - Dependencies, task graph
- **Preemptive vs. non-preemptive**
  - Task with higher priority can preempt lower priority one
- **Uni- vs. multi-processor scheduling**
  - Pre-defined vs. joint binding/partitioning
  - Symmetric vs. asymmetric multi-processing (SMP/AMP)
    - Homogeneous vs. heterogeneous

Uni-Processor Scheduling

- **Aperiodic, independent tasks (task set)**
  - Simultaneous (at system start) arrival times
    - Earliest Due Date (EDD) minimizes max. lateness (non-preemptive)
  - Arbitrary arrival times (statically know or dynamic)
    - Earliest Deadline First (EDF) minimizes max. lateness (preemptive)
    - Without preemption optimality only possible if arrival times known
- **Periodic, independent tasks**
  - Schedulability only (preemptive, static or dynamic)
    - Rate Monotonic Scheduling (RMS) is optimal fixed priority scheme
      - » Does not achieve 100% CPU utilization for guaranteed schedulability
    - Earliest Deadline First (EDF) is optimal dynamic priority scheme
      - » 100% utilization, but runtime support/overhead for dynamic priorities
- **Dependent tasks (task graph)**
  - Simultaneous (at system start) arrival times
    - Latest Deadline First (LDF) minimizes max. lateness (non-preempt.)
  - Arbitrary arrival times (statically know or dynamic)
    - Modified EDF* w/ successor-adjusted deadlines

Aperiodic, Independent Task Model

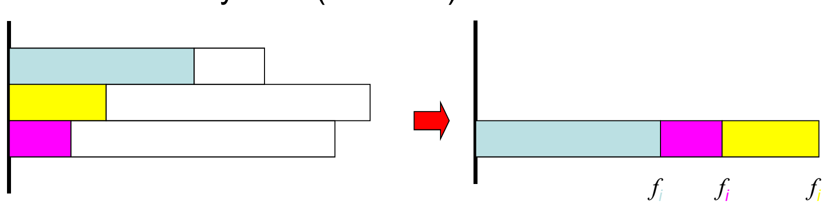
- **e_i is execution time of task i**
- **Arrival time a_i of task i**
- **Deadline τ_i of task i (→ real-time!)**



- **Waiting time**
  - Difference between start time s_i and arrival time a_i
- **Response time**
  - Difference between finish time f_i and arrival time a_i
- **Lateness/slack**
  - Difference between response time r_i and deadline τ_i

Simultaneous Arrival Times

- **Preemption is useless**
  - All tasks arrive at the same time
- **Total schedule makespan/length is fixed to $\sum e_i$**
  - Optimize average waiting time: Shortest Job First (SJF)
  - Optimize lateness under deadlines: real-time scheduling
- **Earliest Due Date (EDD)**
  - Execute task with earliest due date (deadline) first
  - Sort tasks by their (absolute) deadlines



- EDD is optimal to minimize the maximum lateness

Different Arrival Times

- **Requires preemption for optimality**
  - Avg. waiting time: Shortest Remaining Time First (SRTF)
  - Real-time: Earliest Deadline First (EDF)
    - Alternative: Least Laxity (LL) / Least Slack Time First (LSF)

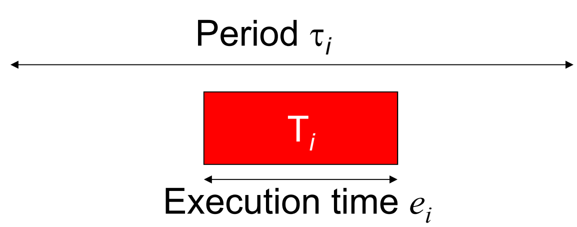
➢ **EDF and LL/LSF are optimal**
  - Minimizing maximum lateness
  - Differences in response times

➢ **If preemption is not allowed**
  - All arrival times need to be know to achieve optimality
  - Problem becomes NP-hard
  - EDF remains best online (unknown arrival times) algorithm

ECE382M.20: SoC Design, Lecture 7 © Peter Marwedel, Dortmund Univ. 9

Periodic Task Model

- **e_i is execution time of task i**
- **Tasks arrive at regular, periodic times $a_i = \{\tau_i, 2\tau_i, 3\tau_i, \ldots\}$**
- **Deadline τ_i is period of task i (always real-time)**



➢ **Waiting time w_i**
➢ **Response time r_i**
➢ **Lateness/slack l_i**

ECE382M.20: SoC Design, Lecture 7 © Margarida Jacome, UT Austin 10

Scheduling Metrics

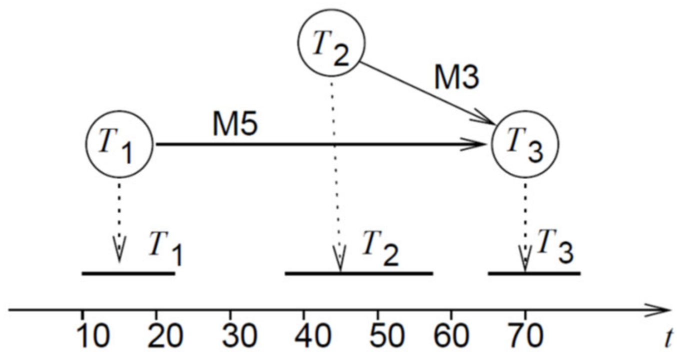
- **How do we evaluate a periodic scheduling policy**
  - Ability to satisfy all deadlines (→ real-time)
  - CPU utilization
    - Percentage of time devoted to useful work
  - Scheduling overhead
    - Time required to make scheduling decision
  - ➢ Schedulability
    - ➢ Find a schedule iff one exists that satisfies all deadlines
    - ➢ Worst-case CPU utilization that guarantees schedulability for any task set
- **Constraints**
  - Set of tasks T with period τ_i each
    - Response time r_i = finish time f_i – arrival time a_i
    - Deadline d_i: $r_i < d_i$, in periodic case often $d_i = \tau_i$
  - Minimize latency
    - Lateness $l_i = r_i - d_i$

Periodic Task Scheduling

- **Scheduling Policies**
  - RMS – Rate Monotonic Scheduling
    - Task Priority = Rate = 1/Period
    - RMS is the optimal preemptive *fixed-priority* scheduling policy
  - EDF – Earliest Deadline First
    - Task Priority = Current Absolute Deadline
    - EDF is the optimal preemptive *dynamic-priority* scheduling policy
- **Scheduling assumptions**
  - Single processor
  - All tasks are periodic
  - Zero context-switch time
  - Worst-case task execution times are known
  - No data dependencies among tasks
- ➢ **RMS and EDF have both been extended to relax these**

Dependent Tasks

- **Task graph**

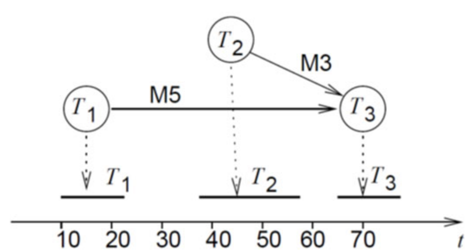


➢ **On uni-processor, periodic derived from aperiodic**
  - All tasks must have same period
  - Period(throughput) = 1 / makespan(latency)

Simultaneous Arrival Times

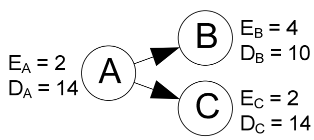
- **Latest Deadline First (LDF)**
  - Process task graph from sinks to sources
  - Among tasks without successors, insert the ones with the latest deadline into the schedule
  - At runtime, process in generated static reverse order

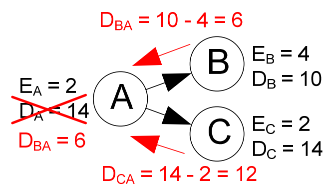
➢ **Optimal for uni-processor**
  - Non-preemptive
  - If no local deadlines, just topological sort



Different Arrival Times

- **Modified EDF***
  - Process graph from sinks to sources
  - Propagate deadlines adjusted for execution times
  - Under global time basis (adjusted for arrival times)
  - At each node, deadline = min(original,propagated)
  - Run regular EDF schedule for independent taks





➢ **Optimal for uni-processor**
  - Preemptive

ECE382M.20: SoC Design, Lecture 7 © 2021 A. Gerstlauer 15

Lecture 7: Outline

✓ **Uni-processor scheduling**
  - ✓ Aperiodic tasks
  - ✓ Periodic tasks
  - ✓ Dependent tasks

- **Special considerations**
  - Context-switch times
  - Interrupts
- **Muti-processor scheduling**
  - MPSoC synthesis

ECE382M.20: SoC Design, Lecture 7 © 2021 A. Gerstlauer 16

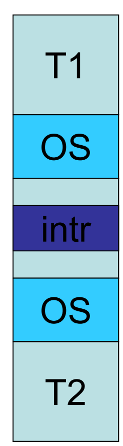
Performance Evaluation

- **Context switch time**
  - Non-zero context switch time can push limits of a tight schedule
  - Hard to calculate effects
    - Depends on order of context switches
  - In practice, OS context switch overhead is small
- **May want to test**
  - Context switch time assumptions on real platform
  - Scheduling policy

ECE382M.20: SoC Design, Lecture 7 © Margarida Jacome, UT Austin 17

What about interrupts?

- **Interrupt overhead**
  - Interrupts take time away from processes
  - Other event processing may be masked during interrupt service routine (ISR)
  - Perform minimum work possible in the interrupt handler
- **Device processing structure**
  - ISR performs minimal I/O.
    - Get register values, put register values
  - Interrupt service process/thread performs most of device function
  - Sporadic tasks
    - Frequent aperiodic tasks
    - Turn into periodic tasks via sporadic task server process



ECE382M.20: SoC Design, Lecture 7 © Margarida Jacome, UT Austin 18

Caches

- **Processes can cause additional caching problems.**
  - Even if individual processes are well-behaved, processes may interfere with each other
  - Worst-case execution time with *bad cache behavior* is usually much worse than execution time with good cache behavior

➢ **Perform schedulability analysis without caches**
  - Take any online performance gains as "free lunch"

Uni-Processor Summary

- **Scheduling**
  - Dynamic, preemptive & priority-based scheduling
    - Real-time operating system (RTOS)
  - Independent periodic tasks
    - Earliest-deadline-first (EDF)
  - Aperiodic or dependent tasks
    - EDF or modified EDF*
  - Mix of periodic/aperiodic/sporadic
    - Split into hierarchy of periodic/independent dynamic schedule with aperiodic/dependent statically scheduled subgraphs
- **What if your set of processes is unschedulable?**
  - Change deadlines in requirements.
  - Reduce execution times of processes.
  - Get a faster CPU
  - ➢ Change the partitioning!

Lecture 7: Outline

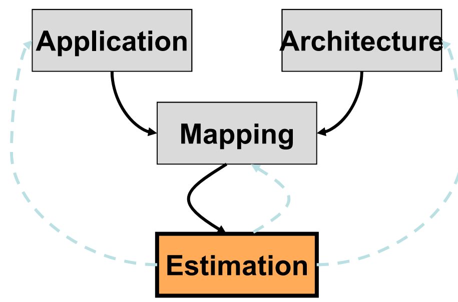
- ✓ **Uni-processor scheduling**
  - ✓ Aperiodic tasks
  - ✓ Periodic tasks
  - ✓ Dependent tasks
- ✓ **Special considerations**
  - ✓ Context-switch times
  - ✓ Interrupts
- **Muti-processor scheduling**
  - MPSoC synthesis

Multi-Processor Scheduling

- **NP-complete in general**
  - Inter-dependency with partitioning/binding
  - Use of heuristics
- ➢ **Independent tasks**
  - Partition and apply uni-processor scheduling
    - Aperiodic tasks: bin packing, longest processing time first (LPT)
  - Uni-processor extensions to homogeneous SMP (multi-core)
    - Global EDF (non-optimal), P-Fair (optimal)
- ➢ **Dependent tasks or heterogeneous processors**
  - For simple cases, partition first and schedule separately
  - In general, solve partitioning & scheduling jointly
  - ➢ Heuristics from compilers & high-level synthesis (Lecture 12)
  - ➢ MPSoC Synthesis

MPSoC Synthesis

- **Design space exploration (DSE)**
  - General application MoCs & architectures
  - Multi-objective, Pareto optimality
  - ➢ Traditional HW/SW co-design approaches not sufficient
- **Iterative process**
  - Determine mapping
    - Partitioning & scheduling
  - Evaluate solutions



➢ **EE382V: Embedded System Design & Modeling**

ECE382M.20: SoC Design, Lecture 7 © Lothar Thiele, ETH Zürich 23