Kernels and Context Switching

- A kernel is responsible of managing tasks in a multitasking system
- Implies some form of context switching

  Two versions:

  - **Non-preemptive kernels/context switching** (Linux 2.4)
    - Cooperative multitasking (tasks will give voluntary control back to the OS)
    - Typically better utilization of the CPU
  - **Preemptive kernels/context switching** (Linux 2.6)
    - Always executes the highest priority task that is ready to run
    - Typically much better response times
    - Most real-time kernels use this

Do not use non-reentrant functions with preemption!

- A reentrant function can be used by multiple tasks without interference (one task will not corrupt data of another task)
- Problem: task preemption may have multiple tasks running the same function (think what this does to your variables)
- Have to use local variables or protected global variables
- Non-reentrant function

```
static int temp;
void swap(int *a, int *b)
{
    temp = *x
    *x = *y;
    *y = temp;
}
```

- Reentrant function

```
void strcpy(char *dest, char *src)
{
    while (*dest++ = *src++)
}
```

Scheduling

- Many scheduling policies aiming for:
  - Good CPU utilisation & performance
  - Good response times (real-time behaviour) (minimising wait time)
  - Good fairness (over all tasks)
  - Also: energy efficiency (battery life), ease of implementation, …
- Many scheduling policies used:
  - Round-robin scheduling (assign time slots per process/task)
  - First come first serve (simple job queue, easy to implement)
  - Multilevel queue scheduling (e.g. multiple queues for different priorities)
  - Priority scheduling, e.g. earliest deadline first scheduling (EDF)
  - …

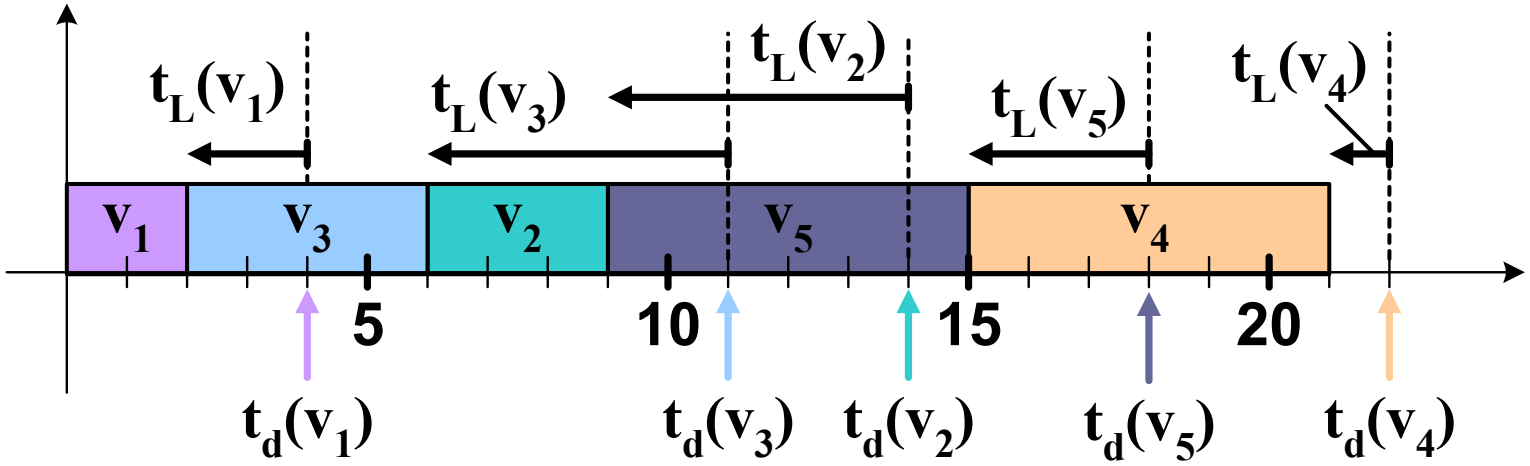
Scheduling with real-time requirements

- Simple round-robin schedulers or job queues cannot guarantee real-time (deterministic execution latencies)
- Different real-time schedulers for different scenarios, for example:

	Same arrival times (non-preemptive)	Different arrival times (preemptive)
Independent tasks	EDD (Jackson) Earliest Due Date	EDF (Horn) Ealiest Deadline First
Dependent tasks	LDF (Lawler) Latest Deadline First	EDF* (Chetto)

Earliest Due Date Scheduling (EDD)

- Start tasks sorted by deadlines



- Maximum lateness: $L_{max} = L(v_4) = -1$
- **Jacksons rule:** given a set of n independent tasks, then sorting the tasks by deadlines minimizes max. lateness.
- Earliest Deadline First (EDF): like EDD, but tasks arrive at runtime and are inserted in a job queue based on deadlines