Operating Systems

02.02 Context switch

Process Scheduling Queues

- Processes are organized in queues (whose element is PCB or a pointer to it) depending on their status.
- The process currently in the CPU is identified through a special name (such as "current")
- Ready queue (one or more)
  - Ready for execution (can be or not in CPU)
  - Linked list
- I/O queues
  - Processes waiting for some I/O service
- Internal queues (synchronization and communication)
  - Processes waiting for other processes (synchronization queues)
- Processes moves from one queue to another depending on their activity and status
Ready Queue And Various I/O Device Queues

Process Lifetime

- An OS component called **short term scheduler** picks-up process from the ready queue and assign it to the CPU
- The process stays in CPU until
  - It performs an OS request (syscall) -> I/O, synchronization queue
  - An interrupt arises (time sharing) -> ready queue
- Processes also be swapped out: they are unloaded from the memory to a secondary storage device (i.e. HDD)
  - Out from ready queue (not ready since not in memory)
  - Can be in other queues
- Another OS component (long term scheduler) picks-up process from the memory and assign it to ready queue
Scheduling Diagram

Schedulers

- *Long-term scheduler* (or job scheduler) – selects which processes should be brought into the ready queue
- *Short-term scheduler* (or CPU scheduler) – selects which process should be executed next and allocates CPU
Schedulers

• Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
• Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
• The long-term scheduler controls the degree of multiprogramming
• Processes can be described as either:
  – I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
  – CPU-bound process – spends more time doing computations; few very long CPU bursts

Scheduling

• Scheduling policies
  – Do not forget any process (fairness)
  – Execute more “important” first (priority)
  – Take care of deadlines (real-time)
  – Optimize something (processor utilization, response time, completion time, etc…)
• Scheduling points
  – Points in time where a scheduling decision is taken by the OS
  – These are the points in which OS takes control
Scheduling Points

- Traps: events generated by the current process (the one in execution)
  - system calls
  - errors (undefined instructions, data abort)
  - page faults (concerns virtual memory – we will see)
- Interrupts: external events (to the process)
  - I/O interrupt
  - timer interrupt (preemption points)
- From a process viewpoint:
  - explicit: the process leaves the control to another process (yield, sleep syscalls)
  - implicit: the process asks for a OS service
- At a scheduling point, the current process can be replaced by another process depending on the scheduling policy
  - Context switch

Context Switch

- The process currently in CPU is replaced by a different one
  - Save the state of old process (current)
  - Load (or resume) the state of new process (next)
- What must be saved? Architectural dependent:
  - General-purpose & floating point registers, co-processor and MMU registers, other register (Alpha, Sparc machines)
- How to save the status without corrupting current registers?
  - CISC and some RISC (e.g. ARM): a single instruction to push register into a memory area.
  - Pure RISC: dedicated software techniques that save a single register at a time
  - It is in any case an assembly routine (complete control of register usage)
CPU Switch From Process to Process

Context Switch Overhead

- Overhead of a context switch
  - Direct: Save/restore
  - Indirect: caches and TLB (Translation Lookaside Buffer) flush

- The whole overhead of scheduling point comprehends:
  - Context switch overhead
  - Scheduling decision
  - Syscall/interrupt management
Preemption

- Interruption of the execution of a process during normal execution
- Two types of preemption
  - User-level preemption: the process is preempted while in user mode (normal execution within its address space)
  - Kernel-level preemption: the process is preempted while in privileged mode (execution of a OS service routine)
- We identify “kernel” with OS at the moment. We will distinguish between the two terms later in this course
- In the second case the kernel must be preemptive

Preemptive Kernel

- A process can be preempted when executing an OS routine (syscall)
  - Another process can replace it in CPU
- If also the other process performs a syscall, eventually both of them alternate their syscall execution
  - Each syscall execution represents a single kernel control path (KCP)
- In this case we define the kernel as preemptive and reentrant
- A kernel can be reentrant even without preemption
Reentrant Kernels

- Reentrancy: support for multiple kernel control paths
- Multiple processes can executed in kernel mode at a time
  - Each one of them has a reserved memory area inside the kernel to store partial data but the also access shared OS data
  - Some of them may get blocked (e.g. waiting for a locked resource), so that another kernel control path can be executed
  - What if an interrupt happens when in kernel mode?

**Non preemptive kernels**
- Multiple kernel control path are allowed but they cannot be preempted
- This prevents synchronization problems and complex save/restore mechanisms inside the OS code
- Alternation between KCP happens at controlled points, where OS data structures are left consistent

**Preemptive kernels**
- KCP can be interrupted, its current status is saved and another KCP is executed. KCPs can alternate in CPU with an fine granularity
- More complex routine are required to perform save and restore.
- Synchronization mechanism must be implemented to control access to global OS data

Reentrant Kernels (II)

- Each process has its own private space in OS memory space (*kernel stack*)
- Processes share code and kernel data

**Synchronization issues**
- Prevent race conditions
- Define atomic operations
- Define critical regions of code
- Implement mutual exclusive data access

- Same problems arise for process synchronization in user-space
- Synchronization and mutual exclusion mechanisms will be studied later in this course
Reentrancy & Preemption

• Reentrancy
  – More than one process in kernel mode
  – Only one at a time is executed but others can be blocked
  – Reentrant functions: they access only to private process kernel stack

• Non-Preemptive
  – A process in kernel mode cannot be preempted
  – It may release voluntarily the CPU