Today's plan

- scheduling
    - round-robin scheduling
    - priority scheduling
    - modified priority scheduling
    - real-time scheduling
    - seL4 and Minix scheduling
- Minix message passing

Scheduling

- the scheduler must determine which process to run next
- some goals for a scheduler include:
  - fairness – important for multi-user systems
  - full utilization of the CPU – important for expensive systems
  - fast response time for processes – important for interactive systems
  - fast execution – so the OS takes only a small fraction of the time
  - meeting deadlines – important for real-time systems
  - giving priority to some processes – important if some processes need better throughput or response time
- most of these goals conflict in some way or another (e.g. fairness and priority), so a scheduler attempts to make tradeoffs among these goals
  - e.g. even low-priority processes should make at least some progress

Cost of Scheduling

- the scheduler may take significant time to execute
  - e.g. Linux before 2.5 was not O(1)
- pre-emption requires context switching, which takes time
- context switching is especially expensive if the processes run in different address spaces
- context switching is extremely expensive if the process to be executed is swapped out (on the disk) rather than in main memory
- to minimize these costs:
  - optimize the context switch code
  - switch as infrequently as possible, but no less frequently
  - minimize the scheduler cost (e.g. O(1) Linux scheduler in 2.5)
  - switch between threads within a process if possible
  - switch between in-memory processes if possible (and maybe prefetch swapped processes)
  - only switch to a process where the currently executing code and data are already in memory

Round-Robin Scheduling

- this simply keeps a queue of processes
- as processes become ready, they are placed in the tail of the queue
- the scheduler always executes the process at the head of the queue,
- gives good fairness, good utilization (if the **quantum** / **timeslice** is long) or fast response (if the quantum is short), no priority or deadlines

Priority Scheduling

- each process has a priority
- processes with the highest priority are handled in round-robin order
- strict priority scheduling: lower-priority processes are only executed once all the higher-priority processes have completed or blocked
- non-strict priority scheduling: lower-priority processes get less time than higher priority processes
- gives fairness among equal-priority processes, utilization/fast response tradeoff, no deadlines

Ways of Implementing Non-Strict Priority Scheduling

- single queue, but timeslice is longer for higher-priority processes
- multiple queues, but each process only gets to go through the queue once (or a few times, or for a maximum period of time) before lower-priority processes get to run
- priority can be changed dynamically if a process has been ready (waiting in the queue) for too long

Real-Time Scheduling

- **periodic** processes must execute every n ms
- **aperiodic** (reactive) processes must complete by a certain deadline
- **hard real time** systems must meet their deadlines or fail
  - e.g. aircraft or vehicle control system
- **soft real time** systems can miss an occasional deadline, though that is undesirable
  - e.g. video/audio playback
- information about how long each process will take to complete may be inaccurate or unavailable

Real-Time Scheduling Algorithms

- assume that the tasks are **schedulable**, that is, there is at least one way to schedule the tasks that satisfies the real-time requirements
  - otherwise, no algorithm can schedule the given tasks
- Earliest Deadline First: the runnable process with the earliest deadline should be scheduled now
  - always works if schedulable
- Rate monotonic: the process that executes most frequently should have the highest priority (strict), i.e. should always execute first

Priority Inversion

- if a low-priority process holds a resource (memory, lock, file) needed by a high-priority process
- the high-priority process must block and wait for the low-priority process to complete
- the low-priority process should now be scheduled with high priority: **priority inversion**
- but a medium-level process may execute first

User Control over Scheduling

- Unix: `nice(2)`: only superuser can decrement/improve priority, anyone can increment/worsen their own priority
- user (or kernel configurator) may be able to set, e.g. quantum, HZ value, default process priority, priority for specific processes
- any others?

seL4 Scheduling

- 256 priority levels
- strict priority scheduling
- thread is re-queued behind other same-priority threads once it uses up its quantum
- if the kernel is compiled to support domains, each domain gets a fixed schedule
  - each domain has its own idle thread
  - code in one domain can neither tell nor guess what is going on in another domain

Minix Scheduling

- four priority levels: task, driver, server, and user process
- strict priority scheduling, as long as a process doesn't use up its quantum
- tasks, drivers, and servers are expected to complete within finite time, so normally they are not pre-empted
- user processes may be pre-empted once their quantum has expired
- scheduler is called on every interrupt or system call (caused by a software interrupt), may reschedule same process
- careful design must handle re-entrant behavior: interrupt handler and scheduler may themselves be interrupted
- interrupt handler postpones interrupt when scheduler is already active

Minix Message Passing

- sender has a message in a buffer that it wants to give to a specific receiver
- receiver has a message buffer that it wants to fill from a given or from any sender
- first one to send or receive blocks
- blocked sender is queued on receiver's process table entry
- all "regular" (Posix) system calls are implemented by sending a message to the File System or the Memory Management server (or the INET server)
- all system calls send, then receive (sendrec), otherwise a caller could block a server forever
- message passing also used among tasks and the kernel