IT5002

Computer Systems and Applications

Lecture 13

Process Scheduling

colinta… du.sg

NUS National University of Singapore

School of Computing

What does your Computer Spend its Time Doing?

- **A mix of jobs:**
  - Computations + reading/writing memory.
  - Input/Output
    - ✓**Reading from the keyboard**
    - ✓**Writing to the screen**
    - ✓**Reading from the mouse**
    - ✓**Sending/receiving data over the network.**
    - ✓**Reading/writing the disk**
    - ✓**…**
  - How do we manage all these varied jobs?

NUS National University of Singapore

School of Computing

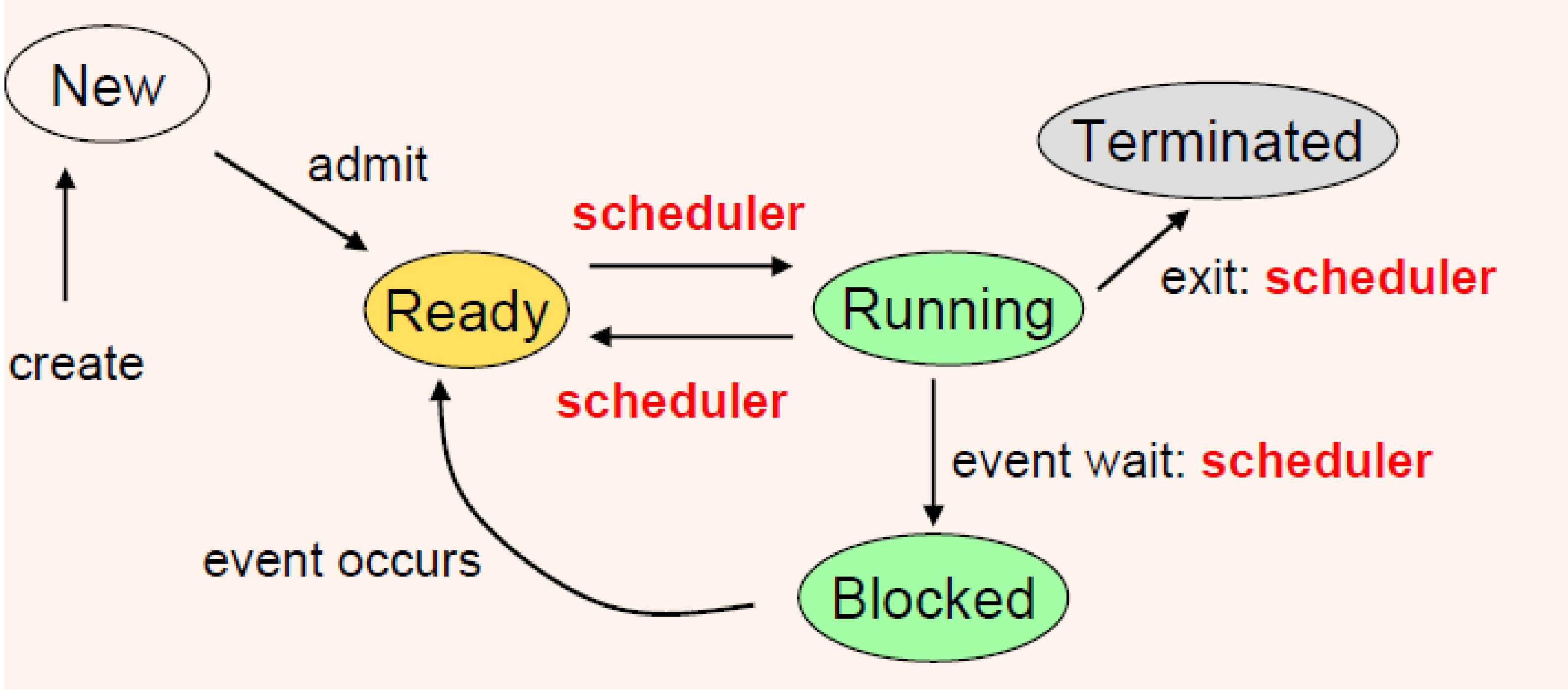
The Scheduling Environment

- **Processes can be:**
  - CPU bound
    - ✓**Most of the time spent on processing on CPU**
    - ✓**Graphics-intensive applications are considered to be "CPU" bound.**
    - ✓**Multitasking opportunities come from having to wait for processing results.**
  - I/O bound
    - ✓**Most of the time is spent on communicating with I/O devices**
    - ✓**Multitasking opportunities come from having to wait for data from I/O devices.**

NUS National University of Singapore
School of Computing

Process States

- **Processes switch between a fixed set of states depending on events that take place.**
  - Scheduler is invoked at various points as shown below.



Generic Scheduler Algorithm

```
schedule() {
  while (queue not empty) {
    task = pick task from ready queue; // policy dependent
    delete task from queue;
    switch to task; // how is it this done? architecture dependent
  }
}
```

Question: How do we determine policies to pick the next task?

Types of Multitaskers

- **Policies are determined by the kind of multitasking environment.**
  - Batch Processing
    - ✓**Not actually multitasking since only one process runs at a time to completion.**
  - Co-operative Multitasking
    - ✓**Currently running processes cannot be suspended by the scheduler.**
    - ✓**Processes must volunteer to give up CPU time.**
  - Pre-emptive Multitasking
    - ✓**Currently running processes can be forcefully suspended by the scheduler.**

Types of Multitaskers

- Real-Time Multitasking
  - ✓**Processes have fixed deadlines that must be met.**
- What if we don’t meet the deadlines?
  - ✓**Hard Real Time Systems: Disaster strikes! System fails, possibly catastrophically!**
  - ✓**Soft Real Time Systems: Mostly just an inconvenience. Performance of system is degraded.**

NUS National University of Singapore School of Computing

Scheduling Policies for Multitaskers

- **Scheduling Policies enforce a priority ordering over processes.**
  - As mentioned earlier, determined by multitasking type.
- **Example Policies**
  - Simplest Policy (Great for all types of multitaskers)
    - ✓ **Fixed Priority**
  - Policies for Batch Processing
    - ✓ **First-come First Served (FCFS)**
    - ✓ **Shortest Job First (SJF)**
  - Policies for Co-operative Multitaskers
    - ✓ **Round Robin with Voluntary Scheduling (VS)**

Scheduling Policies for Multitaskers

- **Example Policies**
  - Policies for Pre-emptive Multitaskers
    - ✓ **Round Robin with Timer (RR)**
    - ✓ **Shortest Remaining Time (SRT)**
  - Policies for Real-Time Multitaskers (Not covered)
    - ✓ **Rate Monotonic Scheduling (RMS)**
    - ✓ **Earliest Deadline First Scheduling (EDF)**

Fixed Priority Policy

- **This is a simple policy that can be used across any type of multitasker.**
  - Each task is assigned a priority by the programmer.
    - ✓**Usually priority number 0 has the highest priority.**
  - Tasks are queued according to priority number.
  - Batch, Co-operative:
    - ✓**Task with highest priority is picked to be run next.**
  - Pre-emptive, Real-Time:
    - ✓**When a higher priority task becomes ready, current task is suspended and higher priority task is run.**

Batch Scheduling Policies

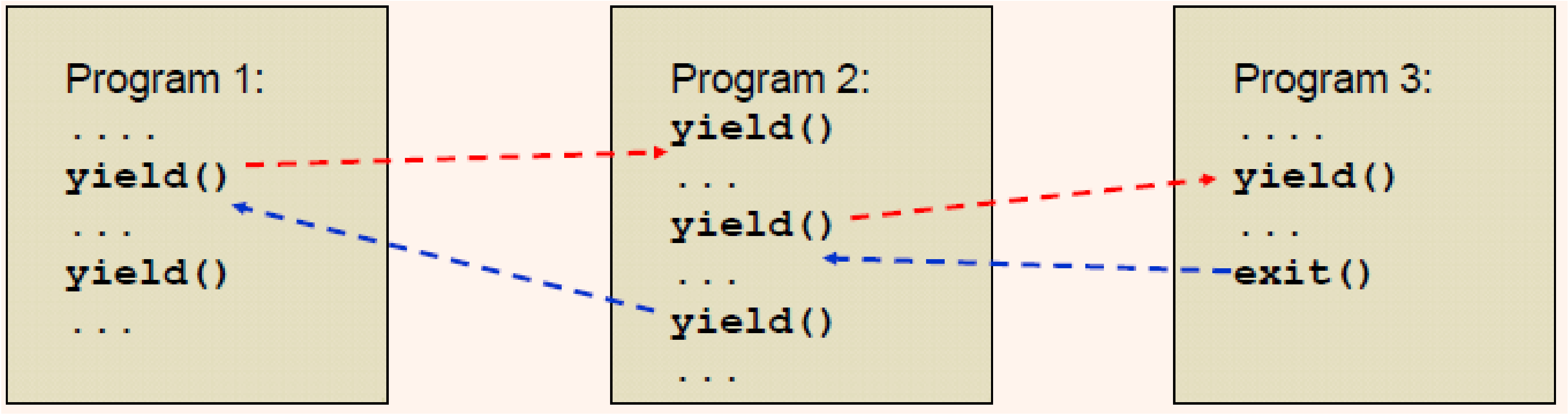
- **First Come First Served**
  - Arriving jobs are stored in a queue.
  - Jobs are removed in turn and run.
  - Particularly suited for batch systems.
  - Extension for interactive systems:
    - ✓ **Jobs removed for running are put back into the back of the queue.**
    - ✓ **This is also known as "round-robin scheduling"**
  - Starvation free as long as earlier jobs are bounded.

Batch Scheduling Policies

- **Shortest Job First**
  - Processes are ordered by total CPU time used.
  - Jobs that run for less time will run first.
  - Reduces average waiting time if number of processes is fixed.
  - Potential for starvation.

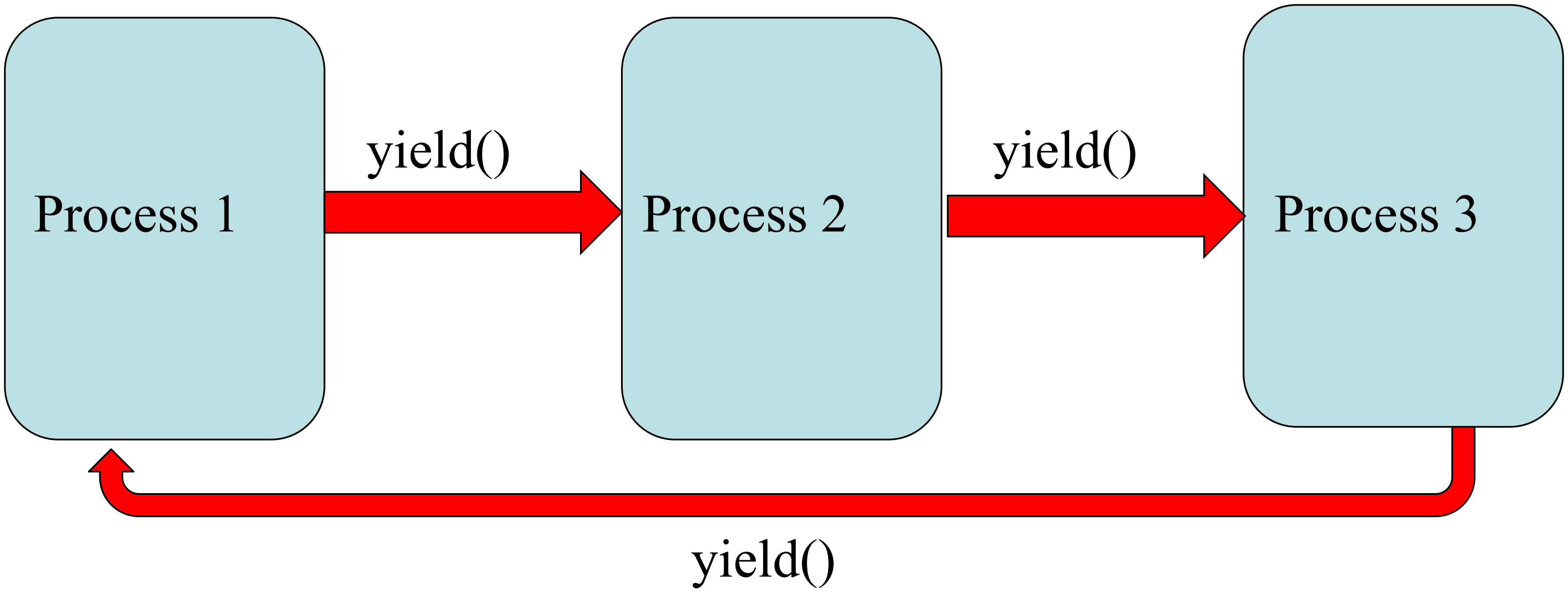
Co-operative Scheduling Policies

- **Voluntary Scheduling.**
  - Processes call a special "yield" function.
    - ✓ **This invokes the scheduler.**
    - ✓ **Causes the process to be suspended and another process started up.**



Co-operative Scheduling Policies

- **In many systems VS is used with a round-robin arrangement.**



Pre-emptive Scheduling Policies

- **Shortest Remaining Time**
  - Pre-emptive form of SJF.
  - Processes are ordered according to remaining CPU time left.
- **Round-robin with Timer**
  - Each process is given a fixed time slot c_i.
  - After time c_i, scheduler is invoked and next task is selected on a round-robin basis.

Managing Multiple Policies

- **Multiple policies can be implemented on the same machine using multiple queues:**
  - Each queue can have its own policy.
  - This scheme is used in Linux, as we will see shortly.

high priority:	P1, P3	(RR policy)
medium priority:	P2	(RR policy)
low priority:	P4, P5	(batch queue, FCFS policy)

Scheduling in Linux

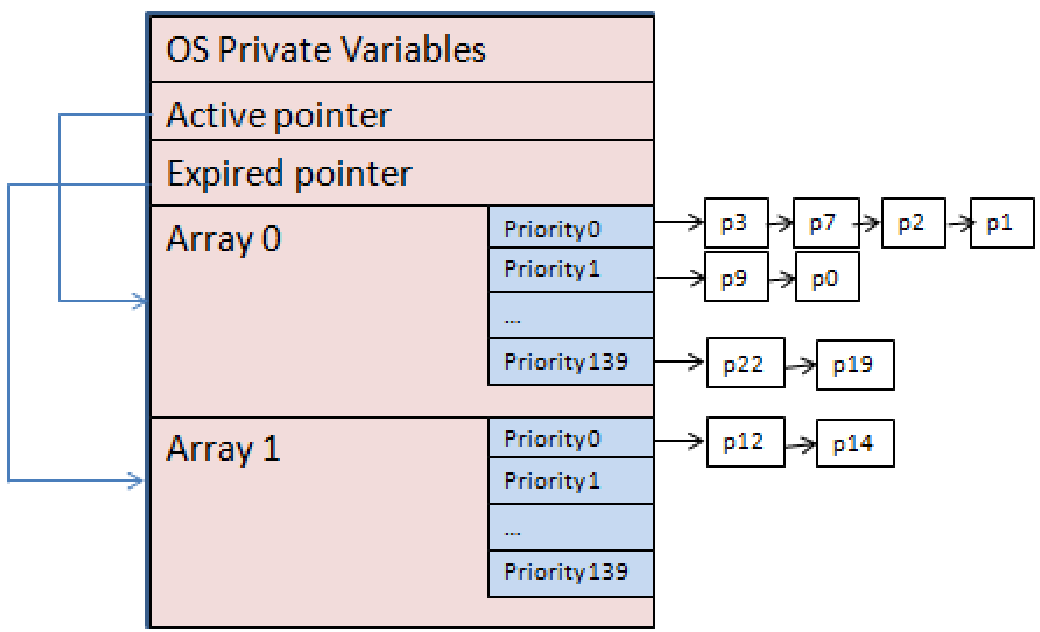
- **Processes in Linux are dynamic:**
  - New processes can be created with fork()
  - Existing processes can exit.
- **Priorities are also dynamic:**
  - Users and superusers can change priorities using "nice" values.
  - nice –n 19 tar cvzf archive.tgz *
    - ✓ **Allows tar to run with a priority lowered by 19 to reduce CPU load.**
    - ✓ **Normal users can only $0 \leq n \leq 19$**
    - ✓ **Superusers can specify $-20 \leq n \leq 19$. Negative nice increases priority.**

Scheduling in Linux

- **Linux maintains three types of processes:**
  - Real-time FIFO:
    - ✓**RT-FIFO processes cannot be pre-empted except by a higher priority RT-FIFO process.**
  - Real-time Round-Robin:
    - ✓**Like RT-FIFO but processes are pre-empted after a time slice.**
  - Linux only has "soft real-time" scheduling.
    - ✓**Cannot guarantee deadlines, unlike RMS and EDF we saw earlier.**
    - ✓**Priority levels 0 to 99**
  - Non-real time processes
    - ✓**Priority levels 100 to 139**

Scheduling in Linux

- **Linux maintains 280 queues in two sets of 140:**
  - An active set.
  - An expired set.



NUS National University of Singapore School of Computing

Scheduling in Linux

- **The scheduler is called at a rate of 1000 Hz.**
  - E.g. time tick is 1 ms, called a “jiffy”.
  - RT-FIFO processes are always run if any are available.
  - Otherwise:
    - ✓**Scheduler picks highest priority process in active set to run.**
    - ✓**When its “time quantum” is expired, it is moved to the expired set. Next highest priority process is picked.**
    - ✓**When active set is empty, active and expired pointers are swapped. Active set becomes expired set and vice versa.**
    - ✓**Scheme ensures no starvation of lowest priority processes.**

Scheduling in Linux

- **What happens if a process becomes blocked? (e.g. on I/O)**
  - CPU time used so far is recorded. Process is moved to a queue of blocked processes.
  - When process becomes runnable again, it continues running until its time quantum is expired.
  - It is then moved to the expired set.
- **When a process becomes blocked its priority is often upgraded (see later).**

Scheduling in Linux

- **Time quantums for RR processes:**
  - Varies by priority. For example:
    - ✓**Priority level 100 – 800 ms**
    - ✓**Priority level 139 – 5 ms**
    - ✓**System load.**
- **How process priorities are calculated:**
  - Priority = base + f(nice)+g(cpu usage estimate)
    - ✓**f(.) = priority adjustment from nice value.**
    - ✓**g(.) = Decay function. Processes that have already consumed a lot of CPU time are downgraded.**

Scheduling in Linux

- Other heuristics are used:
  - ✓**Age of process.**
  - ✓**More priority for processes waiting for I/O – I/O boost.**
  - ✓**Bias towards foreground tasks.**
- **I/O Boost:**
  - Rationale:
    - ✓**Tasks doing read() has been waiting for a long time. May need quick response when ready.**
    - ✓**Blocked/waiting processes have not run much.**
    - ✓**Applies also to interactive processes – blocked on keyboard/mouse input.**

Scheduling in Linux

- Implementation: We can -
  - ✓**Boost time quantum.**
  - ✓**Boost priotiy.**
  - ✓**Do both.**
- How long does this boost last?
  - ✓**Temporary boost for sporadic I/O**
  - ✓**Permanent boost for the chronically I/O bound?**
  - ✓**E.g. Linux gives -5 boost for interactive processes.**