

CS418 — Operating Systems

Lecture 12

Multiprocessor Scheduling

Textbook: Operating Systems
by William Stallings

1. Multiprocessor Types

- Cluster (loosely coupled or distributed multiprocessor)
 - 1. No shared memory.
 - 2. Each processor has its own memory and I/O channels.
 - 3. Each processor can complete tasks almost independently.
- Special Processors — I/O processor.
- Tightly coupled multiprocessing
 - 1. With a shared memory.
 - 2. Must coordinate among the processors to complete tasks.

2. Granularity

- **Granularity:** frequency of synchronization between processes in a system.

3. Issues on Multiprocessor Scheduling

- Assignment of Processes to Processors

- **1.** If all the processors are the same, then we can use either a static or a dynamic policy.
- 1.1. Static assignment: a process is run on a processor until it is finished; each processor also maintains a short-term queue.

Advantage: low overhead.

Disadvantage: unbalanced workload.

Question: Is this policy easy?

- 1.2. Dynamic assignment: a process may be run on different processors during its lifetime.
- **2.** If not all the processors are the same, then we can use master/slave or peer approaches.
- 2.1. Master/slave: Master is responsible for scheduling jobs and slave is responsible for finishing them.

Disadvantage: (a) If master fails, ... (b) Master can be a performance bottleneck.

- 2.2. Peer: OS can run on any processor and each processor does self-scheduling.

Problem?

- Use of Multiprogramming on a Processor?
 - Coarse-grained multiprocessor: yes.
 - Medium- or fine-grained multiprocessor: maybe not. (Think of a job with 6 threads working on shared data.)
- Process Dispatching
 - 1. On uniprocessor scheduling, priority or complicated scheduling will improve performance.
 - 2. On Multiprocessor scheduling, simple scheduling is better. (Thread scheduling is a new issue.)

4. Process Scheduling

- [illegible]

5. Thread Scheduling

- Threads of a process run concurrently within the same address space
- On a uniprocessor, threads can only try to overlap with I/O operation
- On a multiprocessor, threads can obtain great performance gains; of course, it is more difficult to schedule them. The following 3 methods are common.
- **1. Load sharing.**
 - 1.1) A global queue of ready threads is maintained.
 - 1.2) Load is evenly distributed among processors.
 - 1.3) No centralized scheduler is needed.
 - 1.4) Global queue can be maintained using methods on uniprocessor scheduling.
 - **Disadvantage.**
 - 1.5) If many processors are available, ...
 - 1.6) Interrupted threads may not resume execution on the same processor.
 - 1.7) Threads of one process might not be run at the same time.

- **2. Gang scheduling.**

- 2.1) A set of related threads are run (on different processors) at the same time.
- 2.2) Scheduling overhead could be reduced.
- 2.3) Example.

- **3. Dedicated processor assignment.**

- 3.1) Each program is given a set of processors equal to the number of threads it contains.

why it works?

- 3.2) You have many processors, CPU utilization is not that important.
- 3.3) You do not have to do process (context) switching.
- 3.4) Example.