L04e. Scheduling

Introduction:

- How should the scheduler choose the next thread to run on the CPU?
  - First come first serve.
  - Highest static priority.
  - Highest dynamic priority.
  - Thread whose memory contents are in the CPU cache.

Cache Affinity:

- If a thread *T1* is running on a particular CPU *P1*, it's recommended to run the next call of that thread on the same CPU. The reason behind this is that *T1* is likely to find it's working set in the caches of *P1*, which in turn saves time.
  - This can be inefficient if another thread *T2* polluted the cache of *P1* between the two calls of *T1*.

Scheduling Policies:

- First come first serve (FCFS):
  - The CPU scheduling depends on the order of arrival of the threads.
  - This policy ignores affinity in favor of fairness.
- Fixed processor (Thread-centric):
  - For the first run of a thread, the schedular will pick a CPU, and will always attach this thread to that CPU.
  - Selecting the processors might depend on the CPU load.
- Last processor (Thread-centric):
  - Each CPU will pick the same thread that used to run on it in the last operation cycle.
  - This policy favors affinity.
- Minimum Intervening (MI) (Processor-centric):
  - We will save the affinity of each thread with respect to every processor.
  - A thread *T1* affinity will be saved in the form of an affinity index representing the number of threads that ran on the CPU between *T1*'s different calls. The smaller the index the higher the affinity.
  - Whenever a processor is free, it will pick the thread with the highest affinity to its cache.
  - Limited Minimum Intervening: If a lot of processors are running on the system, keep only the affinity of the top few processors.
- Minimum Intervening + queue (Processor-centric):
  - This policy will take into consideration not only the affinity index, but also how many threads are in the queue of the CPU when making the scheduling decision.

Implementation Issues:

- The operating system should maintain a global queue containing all the threads that is available to all the CPUs. This queue will become very huge if the system has a lot of threads.
- To solve this issue, the OS would maintain local policy-based queues for every processor.
- A thread's position in the queue will be determined by its priority.
  Thread priority = Base priority + thread age + affinity
- If a specific processor ran out of threads, it will pull some threads from other processors.

Performance:

- Throughput: How many threads get executed and completed per unit time.
  → System centric.
- Response time: When a thread is started, how long it takes to complete execution.
  → User centric.
- Variance: Does the response time change by time?
  → User centric.
- When picking a scheduling policy, you need to pay attention to:
  - The load on each CPU.
  - In order to boost performance, a CPU might choose to stay idle till the thread with the highest affinity becomes available.

Cache Aware Scheduling:

- In a modern multicore system, we have multiple cores on a single processor, and the cores themselves are HW multi-threaded (switching between the core's threads based on latency).
- We need to make sure that all the threads on a specific core can find their contents on either the core's L1 cache, or at most the L2 cache.
- For each core, the OS schedules some cache frugal threads along with some cache hungry threads. This ensures that the amount of cache needed by all threads is less than the total size of the last level cache of the CPU (e.g. L2).
- Determining if a thread is cache frugal or hungry can be done through system profiling (additional overhead).