Scheduling

COMP3231 Operating Systems

Slide 1
- Uniprocessor Scheduling
- Realtime Systems
- Multiprocessor Scheduling

Slide 2

Determination of which process is allowed to run

What are the objectives?

- Maximise:
  - CPU utilisation
  - throughput (number of tasks completed per time unit)
- Minimise:
  - Turnaround time (submission to completion)
  - Waiting time (sum of time spent in Ready-queue)
  - Response time (time from start of request to production of first response)
- Fairness:
  - every task should be handled eventually (no starvation)
  - tasks with similar characteristics should be treated equally
different type of systems have different priorities!

Slide 3

Types of Scheduling

- Long-term scheduling (admission scheduler):
  The decision to admit a process, i.e., add its threads(s) to the pool of threads that can execute (batch systems)

- Medium-term scheduling (memory scheduler):
  The decision to suspend/resume processes, i.e., to control the pool of threads whose process images are fully or partially resident (mainly in the absence of VM)

- Short-term scheduling (CPU scheduler):
  The decision which ready thread will be dispatched next

Slide 4

Types of Scheduling

- Admission Scheduler:
  - Controls the degree of multiprogramming: More threads \(\Rightarrow\) less CPU time

- Memory Scheduler
  - Part of the swapping function, based on the need to manage the degree of multiprogramming

- CPU scheduler
  - Executes most frequently, invoked when an event occurs
CPU SCHEDULER

Scheduling decisions are necessary when a thread

1. switches from running to waiting state
   - e.g., wait for I/O, other thread to terminate, ...

2. switches from running to ready
   - e.g., interrupt

3. switches from waiting to ready
   - e.g., completion of I/O request

4. terminates

PREEMPTIVE VS NONPREEMPTIVE SCHEDULING

Non-preemptive:

- Once a thread is in the running state, it will continue
- Thread can monopolize the CPU
- Co-operative multitasking, thread may yield CPU

Preemptive:

- Currently running thread may be interrupted and moved to the ready state by the operating system
- Requires hardware support (timer)
- Incurs costs (additional context switches, data consistency)
- What about kernel routines?

SCHEDULING CRITERIA

- User-oriented
  - Response Time
    - Elapsed time between the submission of a request until there is output.
  - Waiting time
    - Total time thread has been waiting in ready queue
  - Turnaround time
    - Amount of time to execute a particular thread (from creation to exit)

- System-oriented
  - Effective and efficient utilization of the processor
  - Throughput
    - Number of completed threads per second

- Performance-related
  - Quantitative
    - Measurable such as response time and throughput

- Not performance related
  - Qualitative
  - Predictability
**Scheduling Criteria**

Different priorities for different types of systems:
- **Batch**
  - non-preemptive policies, or preemptive with long quantums are acceptable
  - Throughput, turnaround time, CPU utilisation
- **Interactive**
  - preemption essential
  - response time, proportionality
- **Realtime** (hard & soft)
  - preemption often not necessary for hard realtime systems
  - meeting deadlines, predictability

**CPU-I/O Burst Cycle**

Processes typically consist of alternating
- CPU bursts and
- I/O bursts

Duration and frequency of bursts vary greatly from process to process
- **CPU bound**: few very long CPU bursts
- **I/O bound**: many, short CPU bursts

**Prediction of CPU Burst Length**

- We don’t know length of next CPU burst, can we predict it?

**Assumption**: Next CPU burst will be similar length to previous one.

- $T_i$: actual length of $i$th burst
- $S_i$: estimated length of $i$th burst

- Slide 12
  - **Simple averaging**: Length of next burst is equal to average of previous bursts:
    \[
    S_{n+1} = \frac{1}{n} \times \sum_{i=1}^{n} T_i
    \]
  - or, to avoid recomputing the sum in every step
    \[
    S_{n+1} = \frac{1}{n} \times T_n + \frac{n-1}{n} \times S_n
    \]
**Exponential averaging:** Recent observations are more important than old ones, we want to give them more weight:

\[ S_{n+1} = \alpha \cdot T_n + (1 - \alpha) \cdot S_n \]

for \( 0 < \alpha < 1 \)

The larger \( \alpha \), the less weight is given to older observations

\[ S_{n+1} = \alpha T_n + (1 - \alpha) \alpha T_{n-1} + (1 - \alpha)^2 T_{n-2} + \ldots \]

Fast to compute for \( \alpha = 0.5 \)

\[ S_{n+1} = 0.5 \cdot T_n + 0.5^2 \cdot T_{n-1} + 0.5^3 T_{n-2} + \ldots = 0.5 \cdot (T_n + S_n) \]

**Metrics**

- **Execution time:** \( T_i \)
- **Waiting time:** time a thread waits for execution:
  \[ T_w \]
- **Turnaround time:** time a thread spends in the system (waiting plus execution time):
  \[ T_w + T_i = T_r \]
- **Normalised turnaround time:**
  \[ \frac{T_r}{T_i} \]

(long waiting times can be tolerated for long run times)
First-come-first-served (FCFS) scheduling:

- Nonpreemptive: each thread, once scheduled, runs to completion
- Scheduler selects the oldest thread in the ready queue

Performance:
- Average waiting time: not optimal, since even short threads may have to wait a very long time
- I/O threads have to wait until CPU-bound thread completes, favors CPU-bound threads (convoy effect)
- Not suitable for time sharing systems

Shortest-thread-next scheduling:

- Non-preemptive policy
- Select thread with shortest expected burst length
  - Short thread jumps ahead of longer running threads
- May need to abort thread exceeding its estimate
- Possibility of starvation of longer running threads

Shortest-remaining-time scheduling:

- Preemptive version of shortest-thread-next policy
- Must estimate processing time

Highest-response-ratio-next (HRRN) scheduling:

- Attempt to minimise average normalised turnaround time
- Choose next thread with the highest ratio

\[ \frac{w}{s} \]

- \( w \): waiting time
- \( s \): (expected/past) service time
  - use past behaviour as a predictor for the future
Performance of HRRN:

- Shorted threads are favoured
- Aging without service increases ratio, longer threads can get past shorter jobs