Scheduling

OS Scheduling choices

- What job to run next
- How long to let the process run
- What order to handle a set of block requests
- If multiple messages to be sent over the network, what order should they be sent

Scheduling Goals

- Design a scheduling algorithm to achieve goals
- Different algorithms try to optimize different metrics
- CHanging algorithm changes system behaviors
- Policy decision

Process Queue

- OS keeps a queue of processes that are ready to run
    - Ordered by the task to run next

- Processes that aren't ready to run:
    - Aren't in the queue
    - At the end of the queue
    - Ignored by the scheduler

Potential Goals

- Maximize throughput
    - Maximize user work done
    - Running OS code doesn't count

- Minimize the average waiting time
    - Avoid delaying too many tasks

- Ensure fairness
    - Minimize worst case waiting time
    - Attempt to distribute the work being done among all tasks

- Meet priority goals
    - Certain tasks have a higher priority than others and should be done first

- Real time scheduling
    - Scheduled items tagged with a deadline that must be met

Different Kinds of Systems

- Time Sharing
    - Fair response time to interactive programs

    - Each user gets and equal share of the CPU
    - Ex. Many users running programs on a server

- Batch
    - Maximize total system throughput
    - Delays of individual processes is unimportant
    - Ex. Supercomputers, we want to maximize the amount of work we finish efficiently

- Real Time
    - Critical operations must happen on time
    - Ex. playing music, streaming

- Service Level Agreement
    - To share resources between multiple customers
    - Agreement which promises a certain amount of resources to the customer
    - Cloud providers must try to rent out all of their services while maintaining the promises, efficient usage is done by scheduling

Scheduling: Policy and Mechanism

- Dispatching: Process in which the scheduler moves jobs onto and off the processor core
- Dispatching is irrelevant of the scheduling algorithm
- Dispatching shouldn't depend on the policy to decide who to dispatch
- Separate policy and the dispatching mechanism

Preemptive v.s Non preemptive Scheduling

- When we schedule a piece of work:
    - We could let it use the resource until it finishes
    - Or interrupt it to work on another process

- Non preemptive: scheduled work always runs to completion

- Preemptive: Scheduler halts a task to run another task

Non Preemptive Scheduling

- Advantage
    - Low scheduling overhead since we alway run the process until it's finished
    - Produces high throughput
    - Conceptually simple and easy to implement

- Disadvantage
    - Poor response time

- Bugs can freeze the machine or cause infinite loop, since OS doesn't interrupt the process
- Poor fairness, since a long process can keep running
- Make realtime and priority scheduling difficult

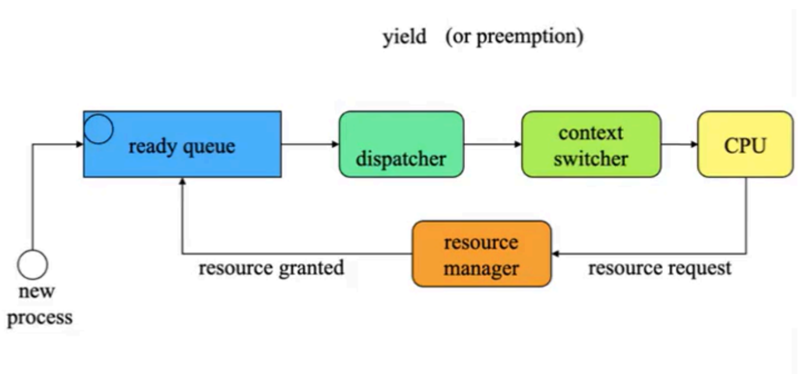
Preemptive Scheduling

- Advantages
  - Good response time
  - Very fair usage
  - Good for real time and priority scheduling

- Disadvantages
  - More complex
  - Requires ability to halt process and save state
  - Throughput suffers
  - Higher overhead

Scheduling

- Yield: When a process gives up its spot and allows another process to run on the CPU



Quantifying metrics

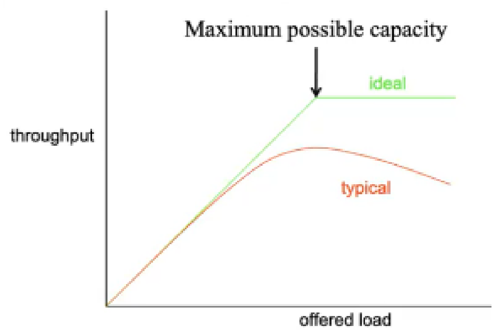
- Throughput
  - (finished process / unit of time)
  - Different processes need different run times
  - Completion time isn't controlled by scheduler

- Delay
  - Not clear hat delays measure (time to finish job, time to get response)

No Ideal Throughput

- Scheduling is costly
    - It takes time to dispatch a process
    - More dispatches means more overhead
    - Less time available to run processes

- Minimizing the performance gap
    - Reduce the overhead per dispatch
    - Minimize the number of dispatches (per second)

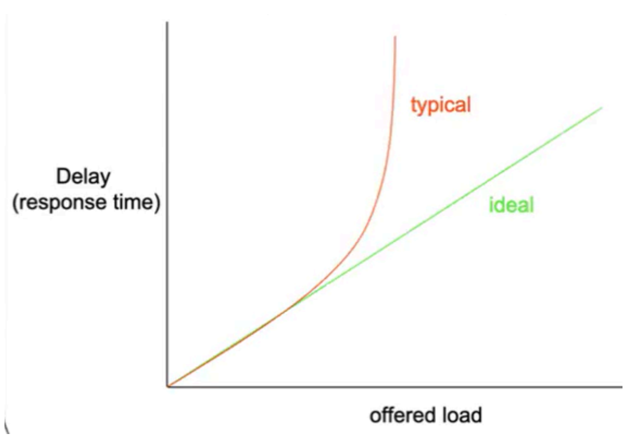


No Ideal Response Time

- Real systems have finite limits
    - Such as queue size

- When limits exceed, request are typically dropped
    - Results in infinite response time
    - May be automatic retries, which could also get dropped

- Too much load
    - Lots of dropped items
    - During heavy load, overheads will increase drastically



Graceful Degradation

- System overloaded
    - When it can no longer meet service goals
    - Continued service results in degraded performance
    - Maintains performance by rejecting work
    - Resume normal service when load returns to normal

- Things NOT to do when overloaded
    - Allow throughput to drop to zero
    - Allow response time to grow without limit

Non Preemptive Scheduling

- Scheduled process continues running until it yields CPU
- Works well for simple systems
    - Small number of processes
    - With natural producer consumer relationships
- Good for maximizing throughput
- Spends on each process to voluntarily yield
    - A buggy process which never yields will freeze the whole system

First Come First Serve

- Run first process in the ready queue until it completes/yields
- Continue the next process in the queue
- Results high variable delays and poor response times, since a long process will run to completion

    Use Cases
    - Response time not important
    - Minimizing overhead is more important than any single job's completion time
    - Ex. Embedded systems when computations are brief

Real Time Schedulers

- Some things must happen in particular time
- In a video, one frame must come after another
- Systems must schedule based on real time deadlines

Hard Real Time Schedulers

- Hard deadlines: system is configured to absolutely meet the deadline (controlling nuclear power plant)
- Schedule is worked out ahead of time

    Ensuring hard deadlines
    - Deep understanding of the code and how long it takes to run
    - Avoid non deterministic timings (every action is predictable)

- Turn off interrupts (which is non deterministic)
- Scheduler is non preemptive (everything is planned beforehand)
- Runs on a predefined schedule (no real time decisions are made)

Soft Real Time Schedulers

- Goal of scheduler is to avoid missing deadlines (although some can be missed)
- Not as vital to run tasks to completion
- Does not need as much analysis
- Missing a deadline could cause system to fall behind or drop future job

  Algorithms
    - Earliest Deadline First
    - Job has a deadline and queue is sorted based on earliest deadline
    - Always pick the job with the earliest deadline
    - Goal: minimize total lateness

Preemptive Scheduling

- A running process can continue running, yields, or OS can interpret it

- An interrupt allows another process to run and the interpreted process can restart alter

- A process can be forced to yield
    - If there is a more important process (such as result of an I/O completion)
    - If the running processes important decreased, it can be replaced by a higher priority one

- Interrupted process might not be in a clean state
    - Complicated saving and restoring its state

- Fluid context switches needed to avoid resource sharing problems
    - More context switches = higher overhead

Implementing Preemption

1. Need OS to regain control from the process
    - Syscall or clock interrupt

2. Consult scheduler
    - Has any process increased its priority
    - Has any process been awakened
    - Has current process had its priority lowered

3. Scheduler finds highest priority ready process
    - If current process is highest, then continue running
    - If not, replace it with a higher priority process

Clock interrupts

- Can generate an interrupt at a fixed time interval to temporarily halt a running process
- Prevents runaway infinite process so it doesn't keep control forever
- Important for preemptive

Round Robin Algorithm

- Fair share scheduling
    - All processes given equal amounts of time to run
    - All processes have the same delays in the queue

- All processes re assigned a normal time slice
    - Usually the same sized slice for all
    - Each process is scheduled in these time slices, then placed placed back into the queue

Advantages

- All processes get relatively quick chance to so some computation
    - Good for interactive processes (allows for scheduling tasks between waiting for I/O)
    - More responsive: All processes have a chance to execute relatively quickly
- Runaway processes do relatively little harm since they are stopped by interrupts

Disadvantages

- No process is quickly finished
- Far more context switching, extra overhead

Choosing Time Slice

- Performance of a preemptive scheduler depends heavily on how long the time slices
- Long time slices avoid context switches
    - Better throughput
- Short time slices and more frequent switches
    - Better response time to process

Costs of Context Switch

- Entering the OS requires saving registers and calling scheduler
- Cycles to choose who to run
- Moving OS context to the new process (switching stack and process descriptor)
- Switching process address spaces
- Losing instructions and data caches results in slower next instructions
    - Losing process caches reduces performance

MLFQ: Priority Scheduling

- Starvation: low priority algorithms might not be able to run
- Solution adjust priority over time
    - Processes that have run for a long time have priority temporarily lowered
    - Processes that haven't run in a long time have increased priority

1. All processes star tin the highest priority queue
2. Move it to a lower priority when it uses its allocated resources or time
3. Periodically move all processes to a higher priority queue

    Advantages

    - Acceptable response time for interactive jobs
        - Jobs don't have to wait for long before being scheduled

    - Efficient but fair CPu use for non interactive job
        - Only run for an allocated time slice

    - Automatic adjustment ot schedules is done based on the behavior of jobs