

Scheduling

Why Make an Elevator?

- It demonstrates the producer / consumer analogy
 - People are produced
 - Through system calls
 - The elevator consumes them
 - By taking them to the appropriate floor

Examples

- File system
 - Users produce read/write requests
 - Disk consumes the requests by returning the file data
 - Prioritizes based on disk head position and rotational delay

Examples

- Web
 - Clients request to view web pages
 - Server consumes requests by returning the page data
 - Prioritizes based on decreasing the amount of time each client has to wait

Simple Design

- While elevator is on
 - Pickup as many people as you can on a floor
 - Loop until empty
 - Go to the destination of the first person
 - Drop off as many people as you can
 - Optionally, pick people up from that floor as well
- Start with something like this
- Optimize time permitting

Scheduler Design Choices

- Different tasks have different needs
 - There is no one size-fits-all model
- Some metrics that can be optimized (not mutually exclusive)
 - Throughput
 - Maximize total number of requests serviced
 - Burst throughput
 - Be able to handle bursts of requests
 - Latency
 - Minimize time it takes to service each request
 - Priorities
 - Service requests with higher priorities first
 - Energy
 - Minimize processing time
 - Different than the time to service the requests

Scheduler Design Choices

- Elevator scheduler should be optimized for throughput
 - You don't need to worry about
 - How long it takes for each person
 - Priorities among different types of people
 - Priorities for which floor to service
 - Amount of effort spent loading/moving

Scheduler Algorithms

FIFO

- Method
 - Service requests in the order they arrive
- Pros:
 - No computational time needed
 - Easy implementation
 - Every request gets a turn
- Cons:
 - Bad at random requests
 - Elevator needs to go back and forth several times in processing requests
 - Low throughput potential

Scheduler Algorithms

Shortest Seek Time First

- Method
 - Service requests near elevator first
- Pros
 - Very low computational time expended
 - Implementation is fairly easy
 - Fairly high throughput by minimizing seek time
- Cons
 - Unfair to distant requests
 - Note, you're trying to maximize throughput, you're not worrying about the individual requests
 - May ignore distant clusters of requests

Scheduler Algorithms

SCAN

- Method
 - Service requests in one direction, then go backwards
- Pros
 - Very low computational time expended
 - Implementation is easy (up, down, up, down, ...)
 - Throughput is somewhat high
 - Fair due to no starvation
- Cons
 - May take up to two trips to collect a set of requests (up,down)

Scheduler Algorithms

LOOK

- Method
 - Improvement on SCAN
 - Uses information of request locations to determine where to stop going in a certain direction
- Pros
 - Low computational time expended
 - Moderate implementation difficulty
 - Set upper bound, go up, set lower bound, go down
 - Throughput is somewhat high
 - Fair due to no starvation
- Cons
 - May miss a new request at outer boundary just as direction change occurs

Scheduler Algorithms

Hybrid

- Combine methods and come up with something new
- Up to your creativity
- Example
 - Shortest Seek Time First w/ Scan
 - Do shortest seek time on a window size of k
 - Move the window up, down, up, down