Realtime and Multimedia

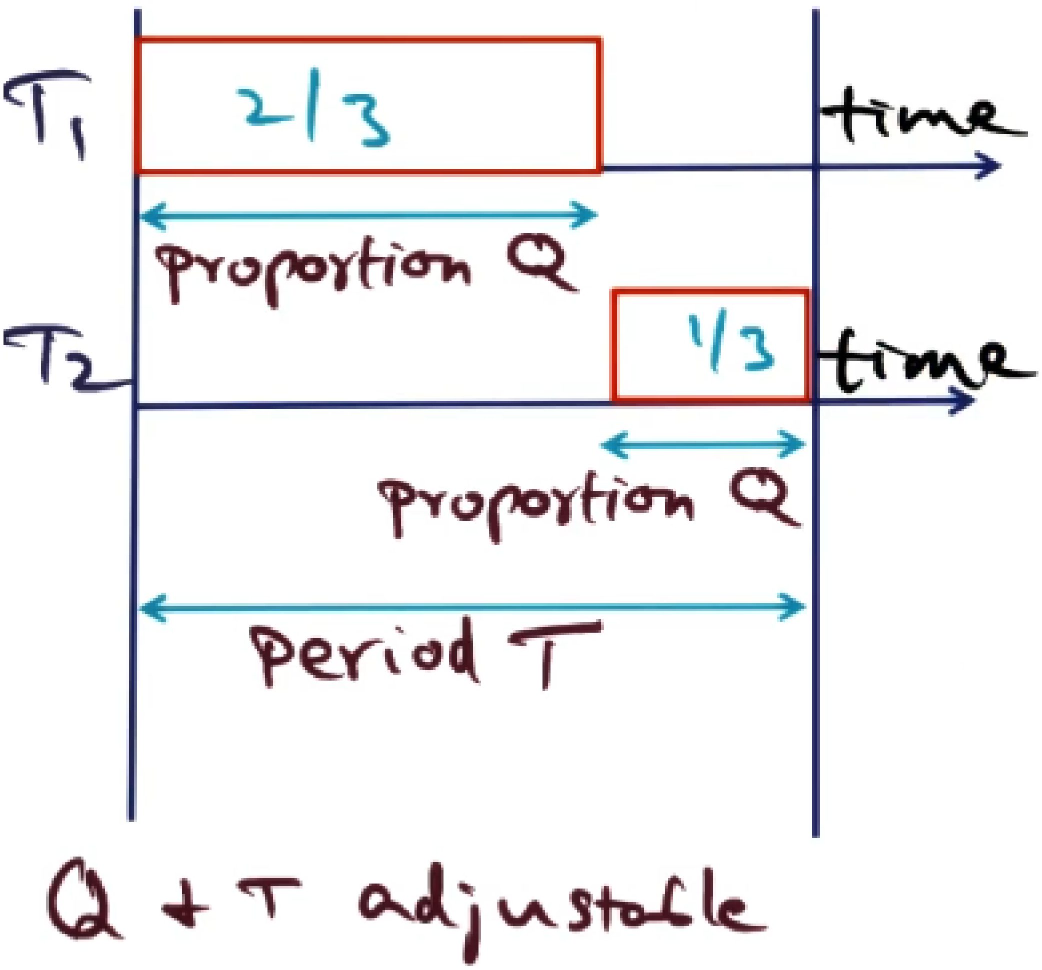
Time Sensitive-Linux

1. Time Sensitive-Linux Introduction
   - Traditional operating systems catered to databases and scientific applications which are throughput oriented
   - Modern workloads support A/V and video games, which have more strict latency requirements and real time guarantees
   - How do you provide guarantees for real time applications in the presence of background throughput-oriented applications such as databases?
   - How do you bound the performance loss of throughput-oriented applications in the presence of latency-sensitive applications?
2. Sources of Latency
   - Time sensitive applications require quickly responding to an event
     - Playing a video game
   - Three sources of latency for time-sensitive events
     - Timer latency: Inaccuracy of timing mechanism; delta between timer event and timer interrupt due to the granularity of the timing mechanism (10ms in Linux)
     - Preemption latency: Interrupt could occur when kernel is in the middle of doing something and cannot be preempted (handler disabled)
     - Scheduler latency: Already a higher priority task in the scheduling queue
   - Latency = Activation of event - Happening of event
     - Latency = Ta - Th
3. Timers Available

Choice of Timer	Pro	Con
Periodic	Periodicity	Event recognition latency
One-shot	Timely	Overhead
Soft	Reduced overhead	Polling overhead, latency
Firm	Combines all	

4. Firm Timer Design
   - Goal: accurate timing with low overhead
   - Combine one-shot and soft timers
   - Overshoot: Knob between hard and soft timers
     - Time between timer expiring and interrupt being triggered
     - If a syscall occurs during this period, we dispatch the expired timers and reprogram the one-shot timers
     - Advantage: The point at which the timer is programmed to interrupt will not actually cause an interrupt
   - Firm timer provides accuracy without additional overhead
5. Firm Timer Implementation
   - Task: Schedulable entity (T1, T2, T3)
   - Timer-q data structure: Queue of tasks sorted by expiry time
   - APIC timer hardware
     - APIC: Advanced Programmable Interrupt Controller
     - Reprogramming a one-shot timer only takes a few cycles
     - Sets a value in a register and decrements it each cycle
     - Allows for the implementation of very fine-grained timers
   - When the APIC timer expires, the interrupt handles traverses timer-q data structure and look for tasks whose timers have expired
     - Callback handler associated with each event are called
     - Tasks are removed from the timer-q

    - If the task is periodic, the handler will re-enqueue the task
  - Soft timers: Eliminates need for fielding one-shot interrupts
  - If there's a long time between one-shot events and many periodic events occurring in between, we can process an upcoming one-shot event at the preceding periodic event
    - Processing an event early is fine, just don't want to be late
    - Periodic events are much more efficient in the kernel (O(1))
    - One-shot events are O(logN) where N is is the number of timers
  - Reduces the timer latency of event handling
6. Reducing Kernel Preemption Latency
  - Approaches
    - Explicit insertion of preemption points in kernel (look for events and take action)
    - Allow preemption anytime the kernel is not manipulating shared data structures (prevents race conditions)
  - Lock-Breaking Preemptive Kernel
    - Combines the above ideas to reduce spin lock holding time
    - Replace long critical section with two shorter critical sections
    - acq() -> rel() vs acq() -> rel() -> reacq() -> rel()
    - Can preempt the kernel after the first release
7. Reducing Scheduling Latency
  - Proportional period scheduler
    - Can reserve a portion of the time quantum for throughput-oriented tasks to guarantee that their requirements are still satisfied in a timely manner
  - T is a time window/quantum (exposed to an application)
    - Task requests Q, a proprtion of T
    - Both are adjustable using a feedback control mechanism
  - Scheduler attempts to satisfy multiple requests within a quantum
    - If one task needs 2/3 T and another needs 1/3 T, both can be satisfied within a single quantum
  - Priority inversion
    - High priority C1 task issues a blocking call to a low priority server
    - Until the server finishes, high priority task cannot proceed
    - While the low priority task is executing, another task C2 can become runnable which will preempt the lower priority (priority C1 > C2)
    - This results in priority inversion for C1
  - Priority-based Scheduling
    - Server assumes the priority of C1 while it is executing its task
    - Therefore, no priority inversion occurs
  - How TS-Linux deals with time-sensitive tasks
    - Firm timer design that increases the accuracy of the timer without additional overhead
    - Using a preemptable kernel to reduce kernel latency
    - Using priority based scheduling to avoid priority and inversion and guaranteeing a portion of the CPU for throughput-oriented tasks



Proportional Period Scheduler

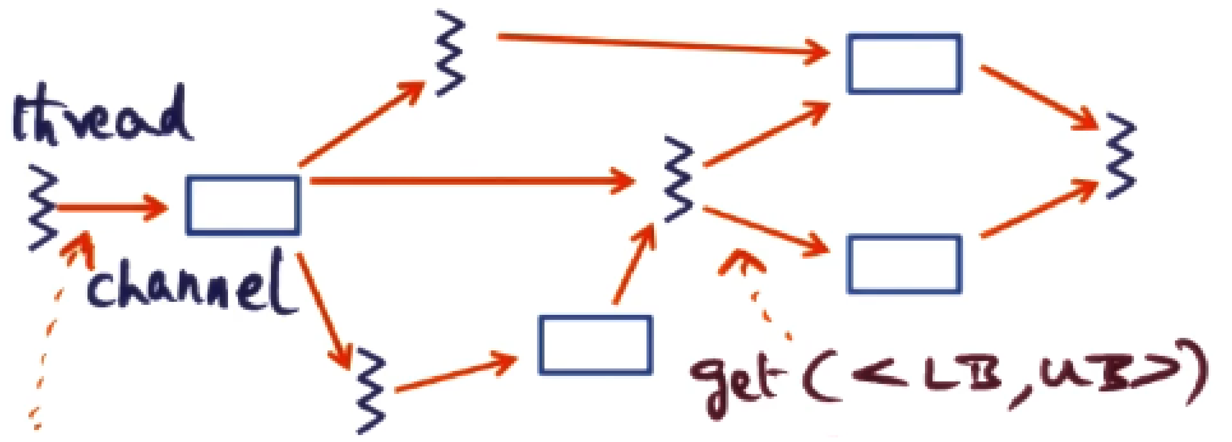
8. Time Sensitive-Linux Conclusion
    - TS-Linux is able to provide quality of service guarantees for realtime applications running on commodity operating systems
    - Admission control using proportional period scheduling ensures that throughput-oriented tasks receive adequate time to execute

Persistent Temporal Streams

1. Persistent Temporal Streams Introduction
    - Focus on middleware between commodity operating systems and realtime, distributed multimedia applications
2. Programming Paradigms
    - Parallel programs: Pthreads
    - Distributed programs: Sockets (NFS, RPC)
3. Novel Multimedia Apps
    - Novel distributed applications tend to be sensor-based
        - Cameras, radars, microphones, temperature sensors, etc.
        - Access sensor data through the Internet
        - Want to do live stream analysis of sensor data
        - Sense -> Prioritize -> Process -> Activate actuators

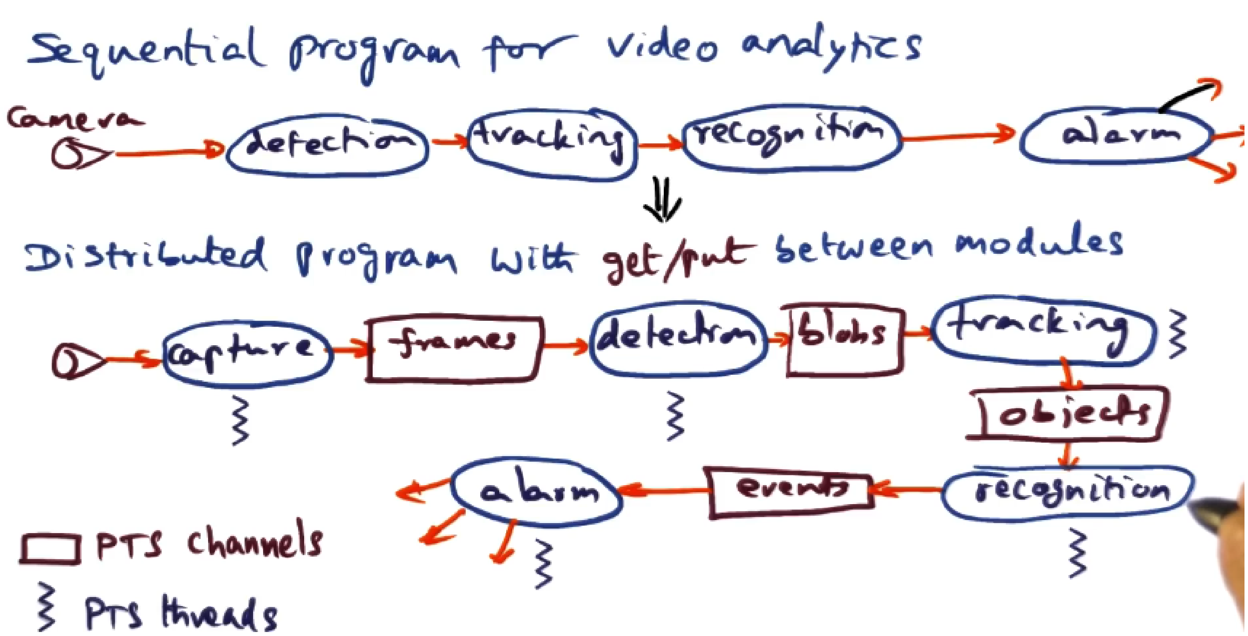
- These computations are intensive and real-time
  - Want to reduce latency as much as possible

4. Example - Large Scale Situational Awareness
   - Monitoring activities in an airport, anything anomalous causes an alert
     - London has over 400,000 cameras to monitor
   - Need to prune the data to deal with this many sensors
     - Can't rely on humans to monitor thousands of cameras
     - Avoid false positives and false negatives
     - Programming infrastructure must facilitate domain expert
   - Pain points
     - Right level of abstraction to promote ease of use
     - Propagate temporal causality
     - Correlate live data with historical data to draw inferences
5. Programming Model for Situational Awareness
   - Sequential program for video analysis
     - Camera -> Detection -> Tracking -> Recognition -> Alarm
   - Objective in situational aware applications: Process streams of data for high level inference (not watching a video)
   - How does this scale to thousands of cameras?
     - PTS is an example of a distributed programming system catering to the needs of situational awareness applications
6. PTS Programming Model
   - Similar to Unix socket API
   - Distributed application with threads and channels
     - Channel holds time-sequenced data objects, allows many-to-many connections
     - Thread is generating time-sequenced data objects and inserting into the channel
     - Can be multiple producers inserting into a channel and multiple consumers receiving from a channel
     - Channel shows temporal evolution of what is being produced by a particular thread (one frame from a camera)
   - put(item,)
   - get(<lowerbound, upperbound>)
     - Can specify time in an abstract way (oldest, newest) in addition to explicit lower/upper bounds
   - Channel ch1 = lookup("VideoChannel");
   - while(true) {
     - response r = ch1.get(<LB, UB>);
     - // process data
     - // produce output
     - ch2.put(item,); }
   - PTS model allows for the ability to associate timestamp with data items produced by a computation
     - Allows for temporal causality
     - Every stream is temporally indexed, so a computation can correlate incoming streams and recognize which data items are temporally related to one another



Proportional Period Scheduler

7. Bundling Streams
    - Might need data from multiple data streams simultaneously
    - Can create a “stream group” which is a bundle of different streams
        - Video, audio, text, gesture
        - One is the anchor stream, the other are dependents
    - groupget: Get corresponding time-stamped items from all the streams in the group
        - Gets temporally related items from all streams in a group at once
8. Power of Simplicity
    - Interpose channels between modules to get/put data
    - Convert from sequential program to distributed using the channel abstraction and get/put primitives

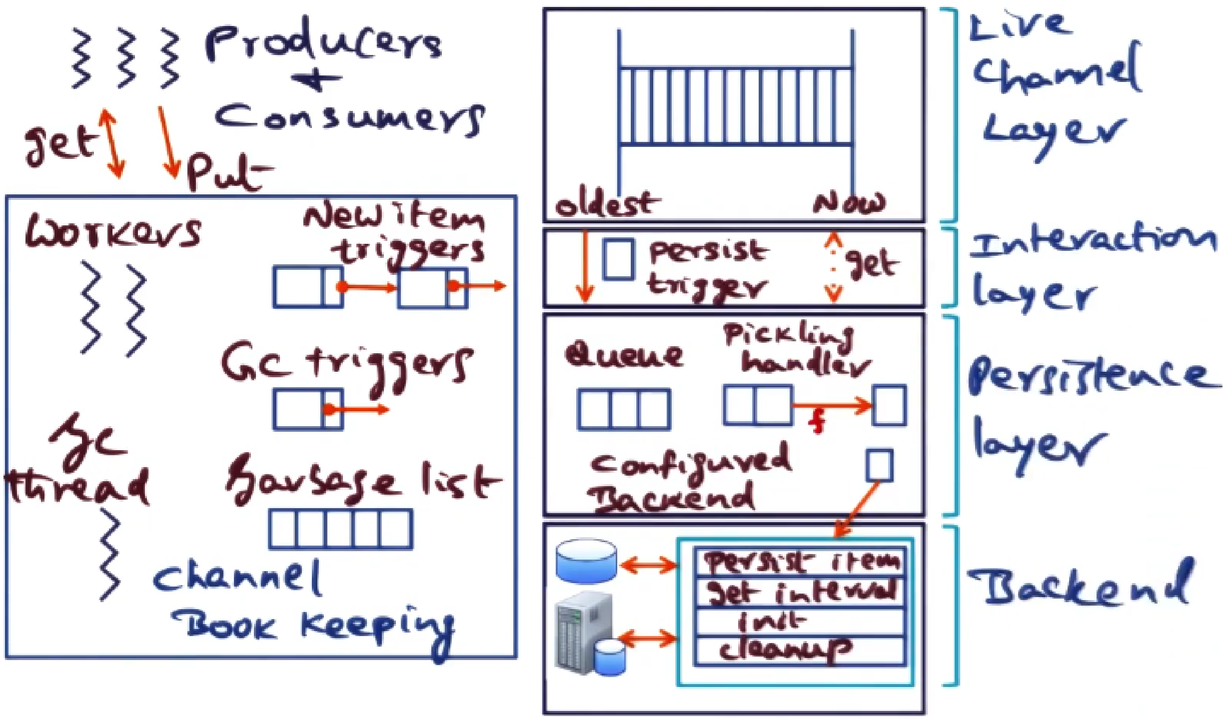


PTS Program Workflow

9. PTS Design Principles
    - PTS provides simple abstractions and interfaces
        - Channel and get/put

- Do the heavy lifting (systems) under the covers
- PTS channels
  - Can be anywhere (similar to Unix sockets)
  - Can be accessed from anywhere (similar to Unix sockets)
  - Network-wide unique (similar to Unix sockets)
  - Time is a first class entity: Time is manipulated by the application in the way it specifies items to the runtime system and queries the runtime system using time as the index into the channel
  - Persistent streams under application control: Data is continuously being produced, can't hold it all in memory. Stored to archival storage
  - Seamlessly handle live and historical data
- PTS is simple to use, but provides the necessary handles to make the life of a domain expert developing a situation awareness application

10. Persistent Channel Architecture
    - How does PTS support this simple get/put programming model?
    - Everything is either a producer or a consumer
    - Threads
      - Workers: Produce new data
      - Garbage collection: Delete data from garbage list
    - Triggers
      - New item triggers occur every time a producer in a worker thread creates a new piece of data
      - Garbage collection triggers: Might specify channel to only keep last 30 seconds of data, throw everything else away. Move data into garbage list (no longer relevant to application)
      - Persist triggers indicate that an item has become old and should be moved from memory to archive
    - Implementation of channel is a three-layer architecture
      - Top: Live channel layer; reacts to new item triggers coming from worker threads
      - Middle: Interaction layer; Go between the live channel and persistence layer
      - Bottom: Persistence layer; receives persistence triggers and moves items from the channel according to the pickling handler. Persisted items must go to some non-volatile storage device
    - Pickling handler: Developer specifies a function to be used every time some of the items are persisted (might condense in some way)
    - Backend layer determines how to keep data in non-volatile storage
      - Application choice as to which backend layer it wants to use
      - Backend layers: MySQL, Unix filesystem, IBM GPFS



PTS Persistent Channel Architecture

11. PTS Conclusion
    - Map reduce is to big data applications as PTS is to live stream analysis applications
    - Unique features
        - Time-based distributed data structures for streams
        - Automatic data management
        - Transparent stream persistence