Window-Constrained Process Scheduling for Linux Systems

Richard West

Ivan Ganev

Karsten Schwan

Talk Outline

Computer Science

- Goals of this research
- DWCS background
- DWCS implementation details
- Design of the experiments
- Experimental results
- Conclusions

Goals

Computer Science

- Explore the performance limits of a general purpose Linux kernel equipped with the DWCS scheduler
- Collect performance data with respect to different loads
- Analyze and interpret the data

Process Scheduling Using DWCS

- “Guarantee” **minimum** quantum of service to processes (i.e. tasks) every fixed window of service time

- NOTE: DWCS originally designed for packet scheduling:
  - “Guarantee” at most **x** late / lost packets every window of **y** packets
  - Now extended to service processes, so that no more than **x** out of **y** periodic processes (or process timeslices) are serviced late

DWCS Process Scheduling

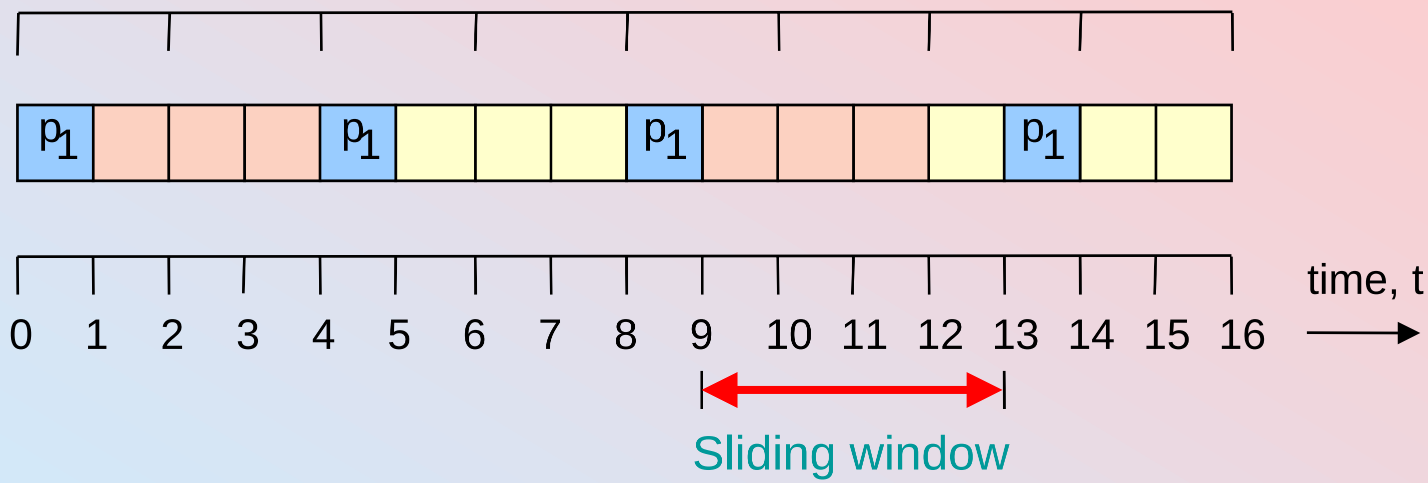
Computer Science

- Three attributes per process, $\mathbf{P_i}$:
  - Request period, $\mathbf{T_i}$
    - Defines interval between deadlines of consecutive invocations of a (potentially periodic) process $\mathbf{P_i}$
  - Window-constraint, $\mathbf{W_i} = \mathbf{x_i/y_i}$
    - Constrains number of missed deadlines $\mathbf{x_i}$ over window of $\mathbf{y_i}$ deadlines
  - Request length, $\mathbf{C_i}$
    - Specifies the requested service length per period

"x out of y" Guarantees

Computer Science

- e.g., Process P_1 with C_1=1, T_1=2 and W_1=1/2



- Example feasible schedule if "x out of y" guarantees are met.

DWCS Algorithm Outline

Computer Science

- Find process $\mathbf{P_i}$ with highest priority **(see Table)**
- Service $\mathbf{P_i}$ for its time quantum or until it blocks
- Adjust $\mathbf{W_i'}$ accordingly
- $\mathbf{Deadline_i}$ = $\mathbf{Deadline_i}$ + $\mathbf{T_i}$
- For each process $\mathbf{P_j}$ missing its deadline:
  - While deadline is missed:
    - Adjust $\mathbf{W_j'}$ accordingly
    - $\mathbf{Deadline_j}$ = $\mathbf{Deadline_j}$ + $\mathbf{T_j}$

(x,y)-Hard DWCS: Pairwise Process Ordering Table

Computer Science

Precedence amongst pairs of processes
• Earliest deadline first (EDF)
• Same deadlines, order lowest window-constraint first
• Equal deadlines and zero window-constraints, order highest window-denominator first
• Equal deadlines and equal non-zero window-constraints, order lowest window-numerator first
• All other cases: first-come-first-serve

Bandwidth Utilization

Computer Science

- Minimum utilization factor of process $\mathbf{P_i}$ is:

$$U_i = \frac{(y_i - x_i)C_i}{y_i T_i}$$

i.e., min required fraction of CPU time over interval $\mathbf{y_i T_i}$.

Scheduling Test

- If:

$$\sum_{i=1}^{n} \frac{(1-\frac{x_i}{y_i}).C_i}{T_i} \leq 1.0$$

  and C_i=**K**, T_i=**qK** for all **i**, where **q** is 1,2,…etc, then a feasible schedule is possible.
- For processes with variable execution time:
  - Can preempt at fixed intervals (e.g., 10mS) if preemptible.

Linux DWCS Implementation

Computer Science

- Modular DWCS implementation
- Design with a scheduler plug-in architecture
- Scheduler info interface: **/proc/dwcs**
- Implementations exist for kernels 2.2.7 and 2.2.13

Plug-in Architecture

- Plug-in architecture:
  - 3 new system calls for linkage:
    - **load_scheduler()**
    - **unload_scheduler()**
    - **DWCS_schedule()**

- Also changed: **struct sched_param**, hence **sched_getscheduler()** / **sched_setscheduler()**

Info Interface: /proc/dwcs

Computer Science

- Normally provides instantaneous snapshots of RT scheduled processes and their parameters & deadlines

- Behavior modified for experimental purposes as follows:
  - Select statistics accumulated in a memory buffer
  - Info interface changed to provide convenient means of extracting the buffer’s contents out of kernel space
  - Collecting data done only after experiment finish to avoid performance disturbances

Experiment Design

Computer Science

- Experimental setup: run a variety of loads recording performance metrics

- Experiment Space: The discrete scheduling parameters define too many dimensions to explore so we combine them in one – CPU utilization:

$$U = \sum_{i=1}^{n} \frac{\left(1 - \frac{x_i}{y_i}\right).C_i}{T_i} \leq 1.0$$

- Metric: Number of deadline violations per process

Experiment Loads

Computer Science

- Two classes of loads:
  - CPU-bound: FFT on a matrix of 4 million floating point numbers (completely in-core)
  - I/O-bound: read 1000 raw bitmaps from disk

- Load codes calibrated to run for about a minute wall-time each on a quiescent system

Experimental Testbed

- **CPU**: 400 MHz Pentium II (Deschutes) w/ 512KB L2 Cache
- **RAM**: 1 GB PC100 SDRAM
- **HD**:
  - Adaptec AIC-7860 Ultra SCSI controller
  - SEAGATE ST39102LC SCSI disk (8GB)
- **Kernel**: Linux 2.2.13 with DWCS

Experiment Engine

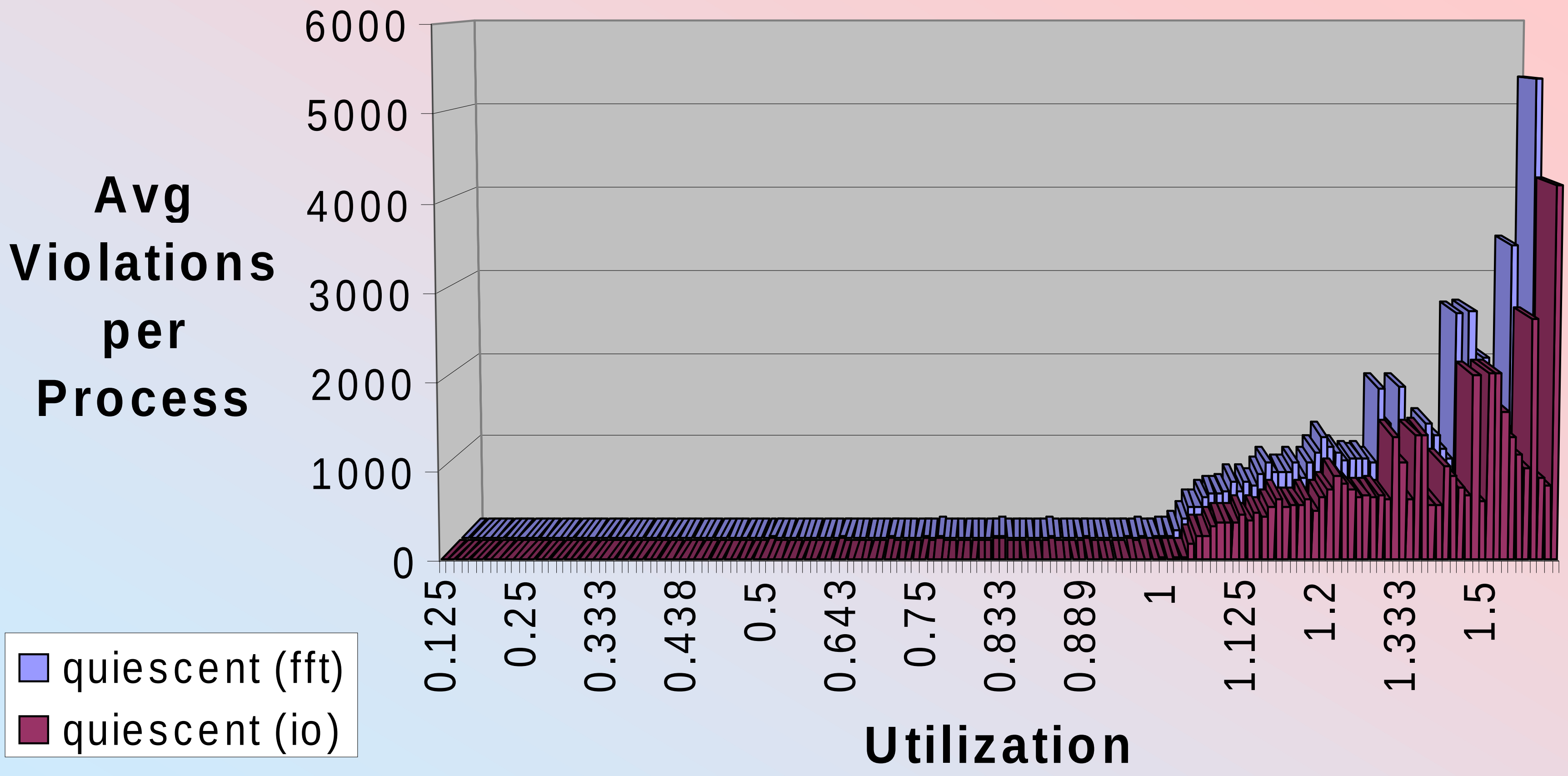
Computer Science

- A parent process reads experiment descriptions from a file
- It forks the needed number of load processes which block
- It collects initial statistics from /proc/stat
- Atomically (by means of a kernel driver) the parent:
  - Resets all load processes' sched. constraints
  - Sends a signal to each load process
- The parent collects exit statistics from /proc/stat and /proc/dwcs
- Each set of parameters is repeated 30 times for statistical significance

Results!

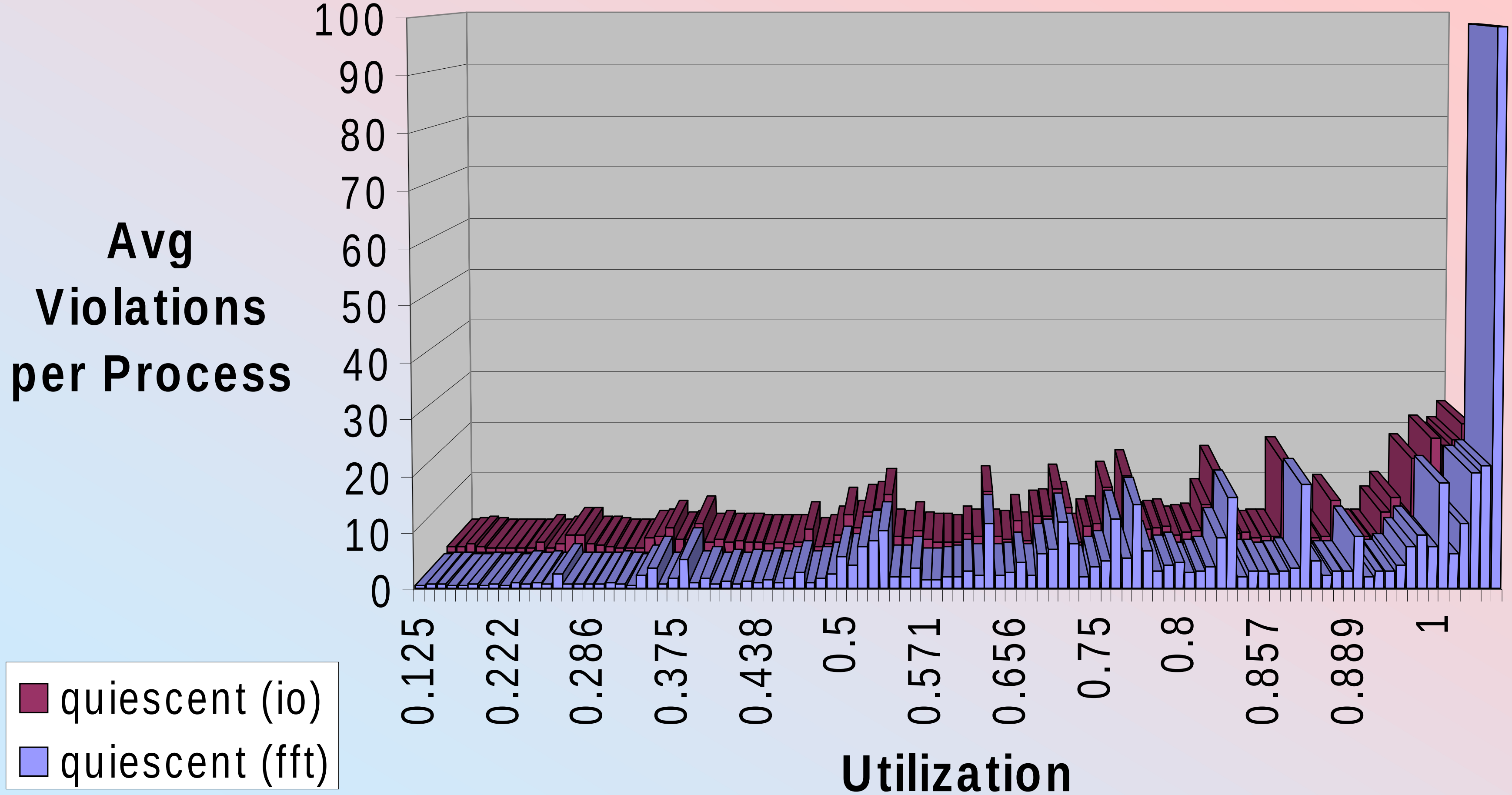
Quiescent System: Average Violations per Process

Computer Science



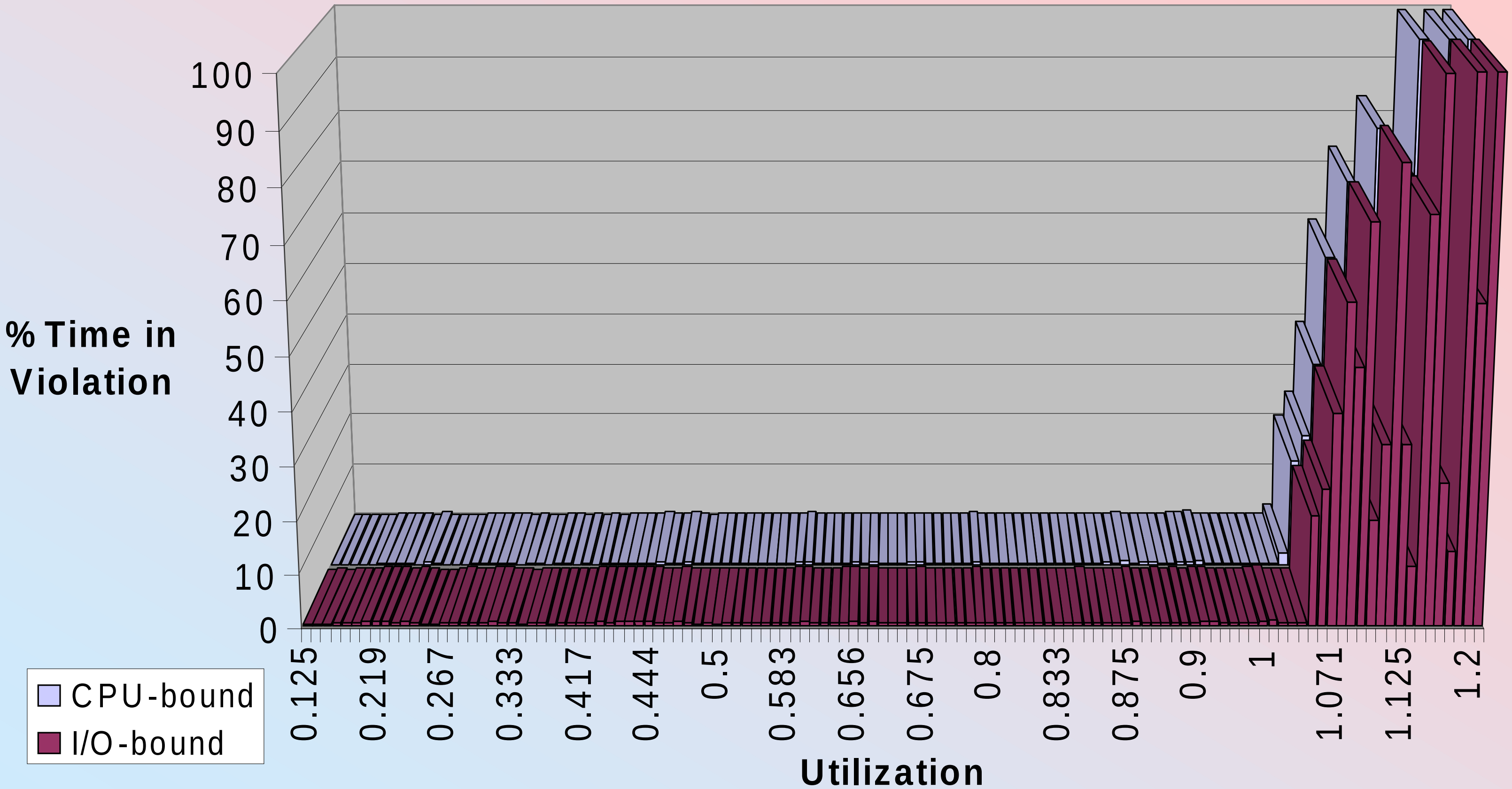
Flood-Pinged System: Average Violations per Process

Computer Science



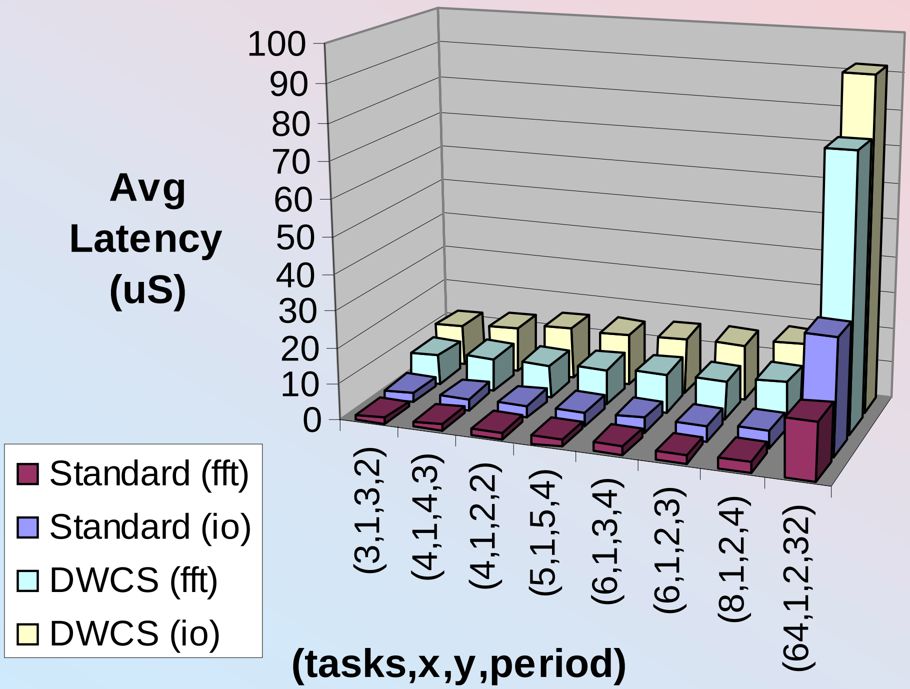
% Execution Time in Violation

Computer Science



Scheduling Latency

Computer Science



- **NOTE**: Measurements w/ DWCS shown when there is about 20 times more context-switching than normal (makes overheads looks worse than they really are)
- There is an I/O latency anomaly, due to servicing bottom halves immediately after interrupts

Providing Better Service Guarantees

- Initial results look encouraging, however violations are still present even when theoretically possible to eliminate them

- Interrupt service time charged to the interrupted process by the scheduler so even though DWCS <u>starts</u> servicing tasks on time, a full quantum cannot always be guaranteed

- Accounting for ISR and bottom halves’ runtime can be done, but we conjecture this will still not be enough…

Remaining Problems

Computer Science

- Lack of fixed preemption points in Linux (variability in scheduler invocation)
  - Due to kernel code calling schedule() directly (i.e. not from the regular timer ISR)
  - Due to nested kernel control paths
- We attempt to control against too frequent invocations (the first case) in software
  - Using a flag for scheduling in the same jiffy
- **In**frequent invocations cannot be helped so simply

Remaining Problems (2)

- As in all other general purpose OSes – unpredictable resource management
  - Memory allocation
  - Paging
  - Semaphores
  - Locks
  - File systems
  - Etc…

Current & Future Work

Computer Science

- Promotion of bottom halves to schedulable threads for better predictability.
  - Must limit bottom half delays due to limited time before function is invalid e.g., if tty device closes.
- Hopefully can achieve better proportional share guarantees for processes.
- So far, DWCS *begins* execution of processes such that 99% deadlines are met, but actual time spent by a process may be affected by time lost to e.g. servicing interrupts.
  - Need to account for lost time to ensure processes make correct progress wrt service constraints.