Hijack: Taking Control of COTS Systems for Real-Time User-Level Services

Gabriel Parmer and Richard West

Computer Science Deparment
Boston University
Boston, MA 02215

{gabep1, richwest}@cs.bu.edu

April 5, 2007

COTS in RT/Embedded Systems

Commodity Off The Shelf (COTS) general purpose systems provide many advantages for RT/Embedded systems

- Tested and widely deployed code-base
- Established development tools/environments
- Developer familiarity

$\rightarrow$ faster time to market/smaller development costs

COTS in RT/Embedded Systems (2)

General purpose systems have a number of disadvantages

- General-purpose policies are often insufficient/awkward for needs of RT applications
- QoS, predictability, policies absent for satisfying app-specific requirements, i.e. EDF

Semantic gap between the requirements of the application and the functionality/guarantees of the system

Shrinking the *Semantic Gap*

Domain-specific OSs created with a focus on one class of applications (RTOSs)

Extensible systems allow the modification of system policies in an application-specific manner

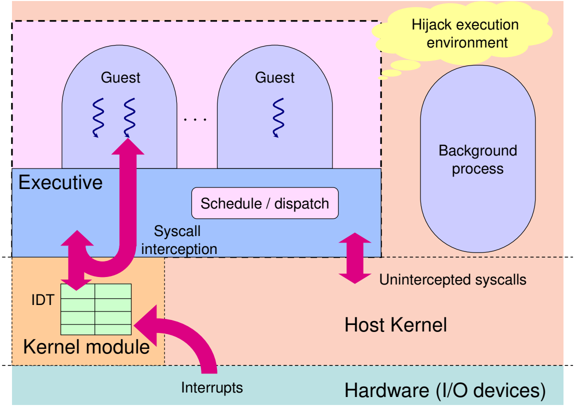
- Generally either not COTS, or not isolation preserving
- Developing extensions requires skill/experience

Goal: provide app-specific policies using a COTS base in a safe and predictable manner

Hijacking your COTS system

Efficient interposition on service requests from specific applications allows the definition at user-level of application-specific policy

Hijack Mechanism



Hijack module receives specific events

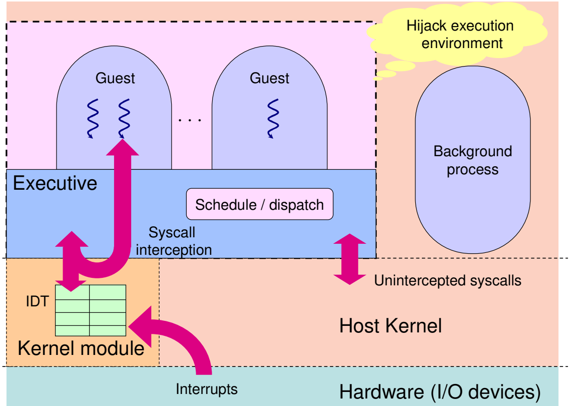
- system calls
- page faults
- possibly device interrupts

Vector *guest* service requests to *executive*

executive controls execution context of *guests*

- create/switch address spaces
- access *guest* registers
- event-triggered executive scheduler

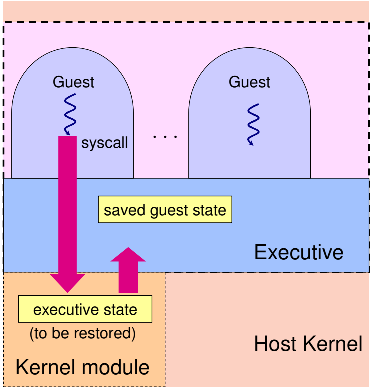
Hijack Mechanism (2)



- *executive* isolated at user-level
- *executive* harnesses base system functionality *where appropriate*

- Does not require changes to the COTS system source-code (no kernel recompilation)
- One (2000 LOC) hijack module enables flexibility in the definition of user-level app-specific services

Case Study: Guest System Call Interposition

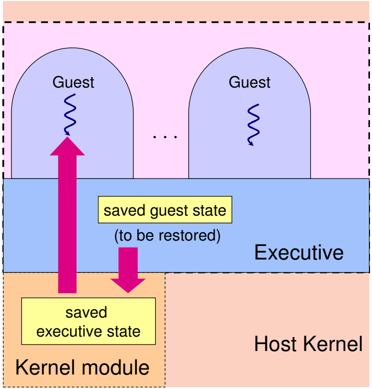


1. *guest* service request intercepted by Hijack module
2. *executive* region *mapped* into current *guest* address space
3. *guest* registers saved into *executive* region
4. *executive* registers restored
5. *executive* executed

executive not present while *guest* is executing – mapped in dynamically

- *executive* isolated from *guests*

Case Study: Guest System Call Return



1. *executive* returns to kernel module
2. *executive* registers saved in module
3. *guest* registers restored from *executive* region
4. *executive* region unmapped from *guest* address space
5. *executive*'s mappings evicted from TLB
6. *guest* executed

Can use *global bits* to avoid flushing *guest* pages from TLB

- set all *guest* pages as global

Experimental Setup

All experiments conducted

- on a 2.4 GHz Pentium 4 processor
- on Linux 2.6.13
- with a clock tick every 10 milliseconds

`nanosleep` Experiments

A goal of Hijack is to offer the ability to enhance default system functionality in an application-specific manner

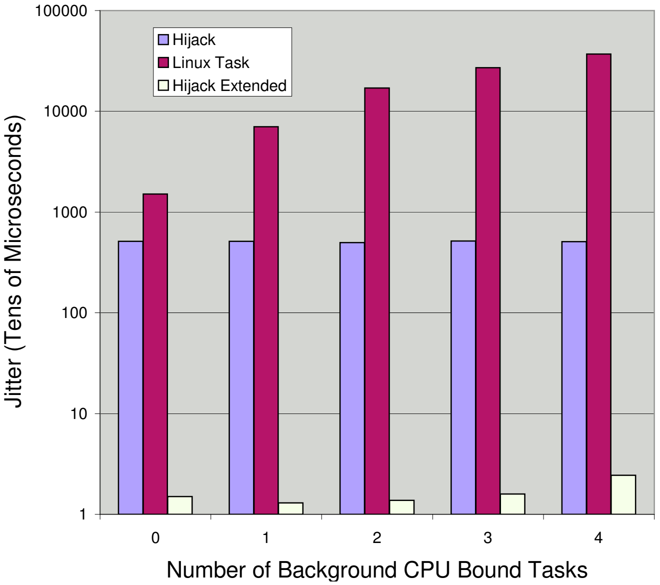
- `nanosleep`: yield for *at least* a specific number of nanoseconds
  - used in multimedia apps such as `mplayer`
- Wake up time variability/unpredictability
  - clock granularity
  - COTS CPU scheduler

nanosleep Experiments (2)

Hijack-provided extensions:

1. Hijack: Executive can give scheduler preference to tasks waking from nanosleep
2. Hijack Extended: Executive can busy wait for periods less than a clock tick

nanosleep Experiments (3)



QoS for Packet Stream Delivery

Scheduling of Tasks dependent on I/O availability with QoS constraints: models traffic shapers, QoS aware stream processing, etc...

- Four streams of 42,000 16 byte packets/second from separate hosts over GigE
- Single host with four tasks, each receiving a stream
- QoS constraints:
  - `Task 0`: 35,000 p/s
  - `Task 1`: 20,000 p/s
  - `Task 2`: 10,000 p/s
  - `Task 3`: best effort

higher QoS

↓

lower QoS

- Start tasks every 5 seconds from `Task 3` to `Task 0`

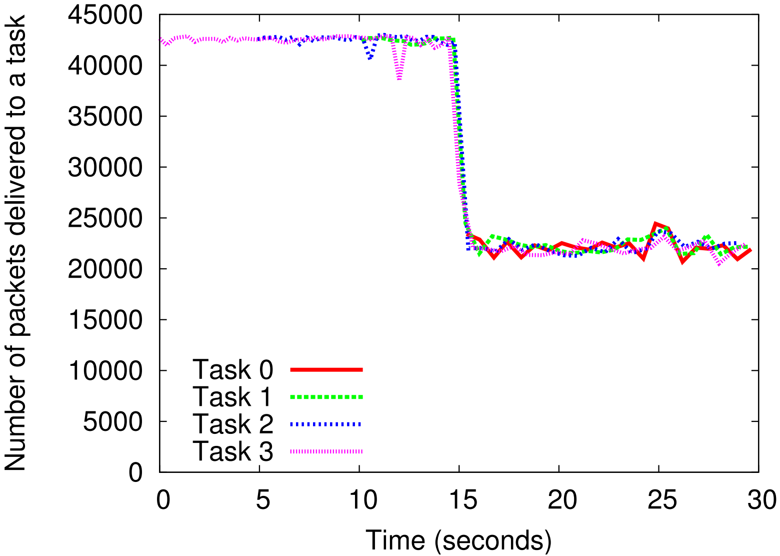
QoS for Packet Stream Delivery (2)

Three scenarios:

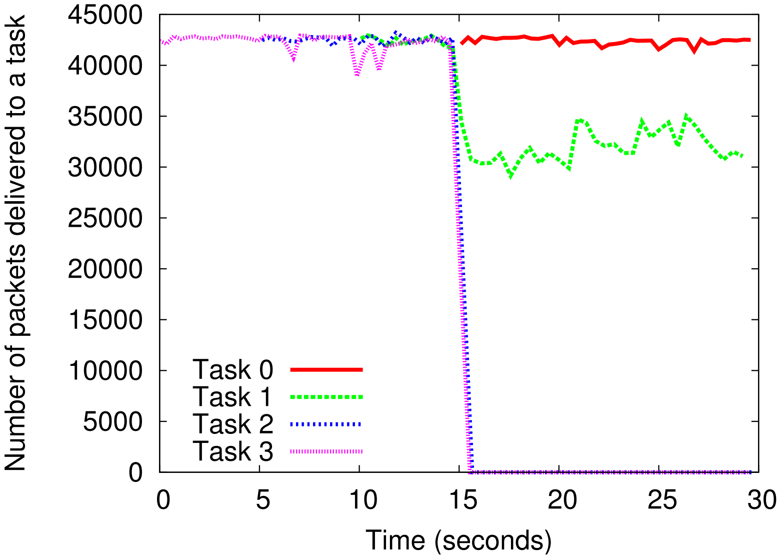
1. Linux, tasks with same priority
2. Linux, tasks with different priority
3. Hijack, Executive using policy similar to proportional-share
   - Tasks assigned tokens proportional to QoS
   - `select` used to probe for I/O activity
   - Task with tokens and available I/O executed
   - Tokens refreshed every given period

   - When guest make system call to read data
   - `read` data into guest buffer until no tokens, or no data

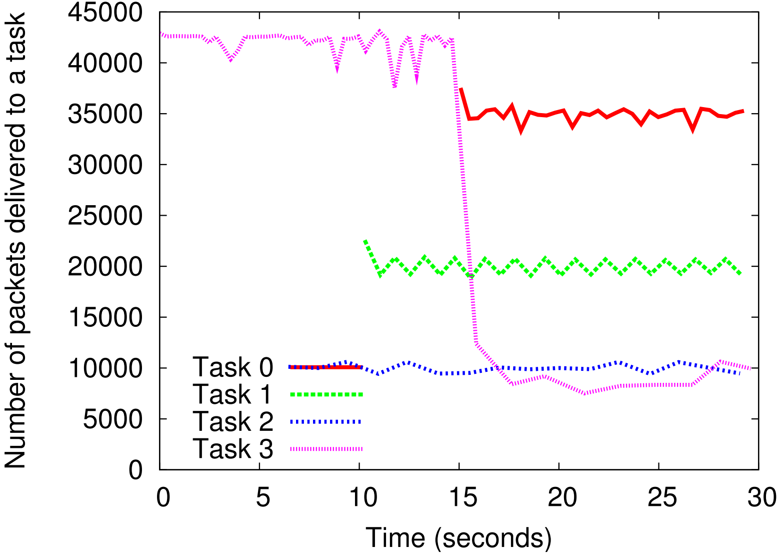
Packet Delivery QoS Results: Linux Same Priority



Packet Delivery QoS Results: Linux Increasing



Packet Delivery QoS Results: Hijacked Linux



Related Work

Related work includes:

RTLinux

- Separate system into two functional domains for Hard-RT predictability
- Focus is on interrupt latency, not app-specific resource management policies

VMs

- Interface provided to guest OSs (*executives*) is identical to the hardware itself
- Focus is on HW virtualization, not on providing app-specific services

Conclusions

Hijack enables app-specific, user-level RT policies using a general purpose computing base

- Use interposition on system service requests to redefine policies
- *executive* defined at user-level can leverage underlying system functionality *where appropriate*
- Demonstrated that complex policies can be introduced

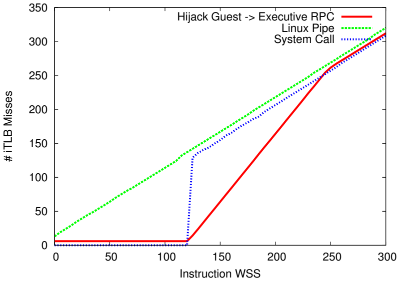
A useful approach towards shrinking the *semantic gap*

Limitations

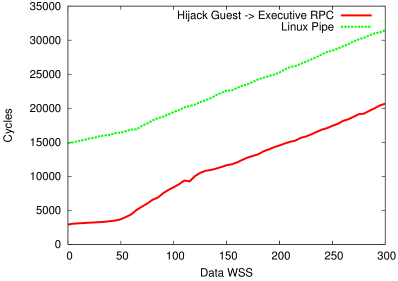
- global bit trick not ideal for all workloads
  - can revert to simply flushing whole TLB or use other techniques
- Certain aspects of the system that cannot be hijacked using these techniques
  - If utilize functionality in base system, generally cannot Hijack that functionality
  - COTS system interrupt handling behavior (prototype limitation)

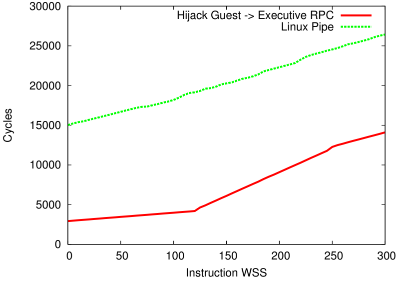
Study the effect of TLB flushes on Executive ↔ Guest communication

- Vary working set size (WSS) of guest by touching data/instruction pages then making system call
- instruction-TLB has 128 entries
- data-TLB has 64 entries
- Global-bit trick avoids TLB flush, thus avoiding misses



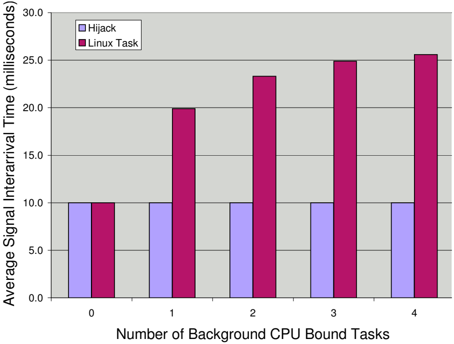
Using the Global-bit Trick to Avoid TLB Flushes



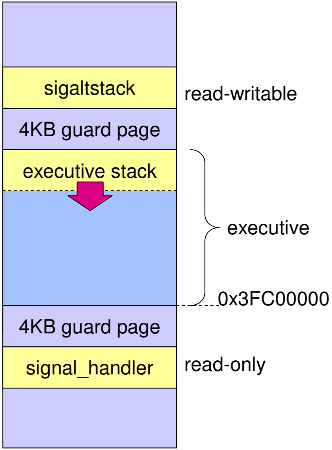


Asynchronous Event Notification Experiments

- Timer interrupts in Executive synthesized with signals
- Predictable notification
- Executive can define customizable policy for scheduling beyond what is present in the COTS system (EDF, PFAIR, DWCS, etc...)



Hijack Execution Environment Address Space



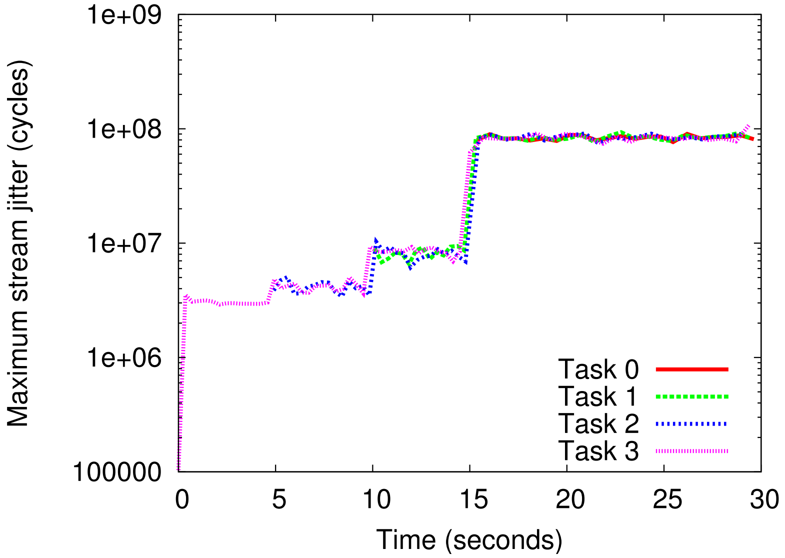
QoS Expts. Executive Algorithm

```
main_event_loop () {
   next = NULL;
   select on the file descriptors for each task;

   if (timing period has expired)
      for (each task in tasks)
           curr_tokens(task) = init_tokens(task);

   for (each task in tasks)
      if (select indicated that task has data &&
          curr_tokens(task) > 0) {
           next = task;
           break;
      }
   if (next == NULL)
      next = best_effort_task;
   execute next;
}
```

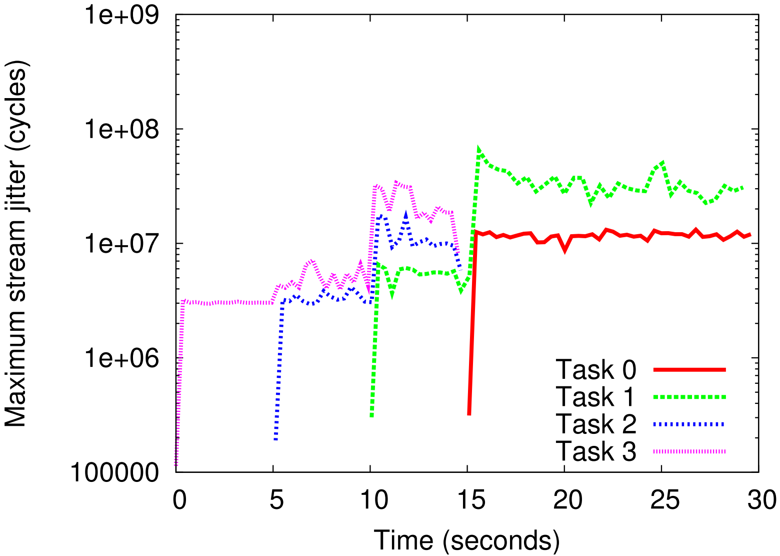
QoS Expts. Executive Algorithm (2)

```
guest_syscall_read(guest_fd, guest_buf, guest_size) {
   fd = translate_to_host_fd(guest_fd);
   loop until (read doesn't return data ||
               curr_tokens(task) == 0) {
      read(fd, guest_buf, guest_size); //nonblocking
      curr_tokens(task)--;
   }
}
```

Max. Jitter QoS Results: Linux Same Priority



Max. Jitter QoS Results: Linux Increasing Priority



Max. Jitter QoS Results: Hijacked Linux