The JAviator Project

Christoph Kirsch
Universität Salzburg

CHESS Seminar, February 2006
Vision

High-performance real-time applications entirely written in Java using standard JDKs
Challenges

1. application: flight control (inspired by the Starmac project at Stanford)
2. memory model: multiple heaps
3. concurrency model: exotasks
4. write-once-run-anywhere in the temporal domain
Team

- Salzburg:
  - 2 PhD students (Harald Roeck, Rainer Trummer), 1 masters student (Werner Gitschthaler)
- Timisoara:
  - 1 PhD student (Daniel Iercan)
- IBM T.J. Watson:
  - 1 staff researcher (David Bacon), possibly more
Platform
It’s a ‘Bicycle Wheel’
Design

The hardware design including all blueprints will be made available at:

javiator.cs.uni-salzburg.at
Ouch: Carbon Fiber Blades
Weightless

gear transmission ratio: 6:1
max. rotor speed: 1850 rpm

© C. Kirsch 2006
Brushless Motors

Power: 100 W
Weight: 26g
Thrust: 600g
3 Gyros, 3 Accelerometers, and 3 Magnetometers

Microstrain 3DM-GX1
Dynamic orientation: gyros
Static orientation: accs, mags
Fusion: onboard programmable filter
I/O: RS-232, RS-485, analog output
10 Ultrasonic Sensors

Devantech SRF10 Sonar Ranger
Frequency: 40KHz
Range: 3cm-6m
I/O: I2C Bus

...but what about lasers?
Board: Gumstix
CPU: XScale 400MHz
RAM: 64MB
Flash: 16MB
Network: Bluetooth
OS: Linux 2.6
I/O Board

Board: Robostix
Bus: I2C
I/O: 6 PWM,
  8 A/D,
  25 GPIO,
  2 UART (Atmega)
Rate Requirements

- gyros, accs, mags: up to 350Hz
- ultrasonic sensors: ~12Hz
- motors: ~100Hz
Current Capabilities

- IBM’s real-time GC (Metronome) has a worst-case latency of 700us

- “eventrons” may run at up to 20KHz with a worst-case jitter of 5us (on 2.6 Linux kernel with preemption patches)

- ...but on a 2.4GHz Pentium, though with a 100MB heap...
Next Steps

- port Metronome to ARM (done at IBM)
- integrate GC and exotasks (IBM, Salzburg)
- enable logical execution times (Salzburg)

➡ write-once-run-anywhere in the temporal domain
RT Programming Tradition

Environment

Release 1 2 3 4 5 6 7 Deadline
Input Output

System

© C. Kirsch 2006
Logical Execution Time (LET)

Environment

Input

Programming as if there is enough CPU time

Compiler checks if there is enough CPU time

If not, program is not time-safe:

compiler error or runtime exception

System

Output
LET Programming

Environment

System
Single CPU, EDF Scheduler

Environment

System

Input

Output

Input

Output

© C. Kirsch 2006
Implementation

- JVMs often map Java threads 1:1 to POSIX threads (IBM’s J9 does this)
- POSIX threads invoke system calls to do I/O
  ➞ we have implemented a POSIX-compliant threading library that schedules system calls with respect to a given queueing policy
System Call Scheduling

- system calls are seen as ‘network packets’
- threads ‘send’ system calls
- system calls are enqueued and dequeued according to a given policy
- multiple queues: disk, network, cpu, memory, real-time I/O

➡ part of the TAP project: tap.cs.uni-salzburg.at
Logical Execution Time Policy

- Environment
- System

Input → 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → Output
Traffic Shaping System Calls

- queueing discipline: prioritized FIFO
- thread behavior is the classification scheme:
  - e.g., "short-running" threads may have higher "queueing priority" than "long-running" threads
  ➡ improves latency of interactive threads
Example: Web Server Latency
Thank you