

# Tornado Quick-Start Workshop

**Wind River Systems**



# Objectives

---

- Define the different parts of the Tornado Architecture.
- State essential characteristics of the VxWorks operating system, and the purposes of important VxWorks libraries.
- Boot a VxWorks target; cross-compile code and download it to the target; start and make simple use of Tornado tools.
- Describe the services provided by optional development tools and OS components.
- State where additional help and documentation is available.

# Contents

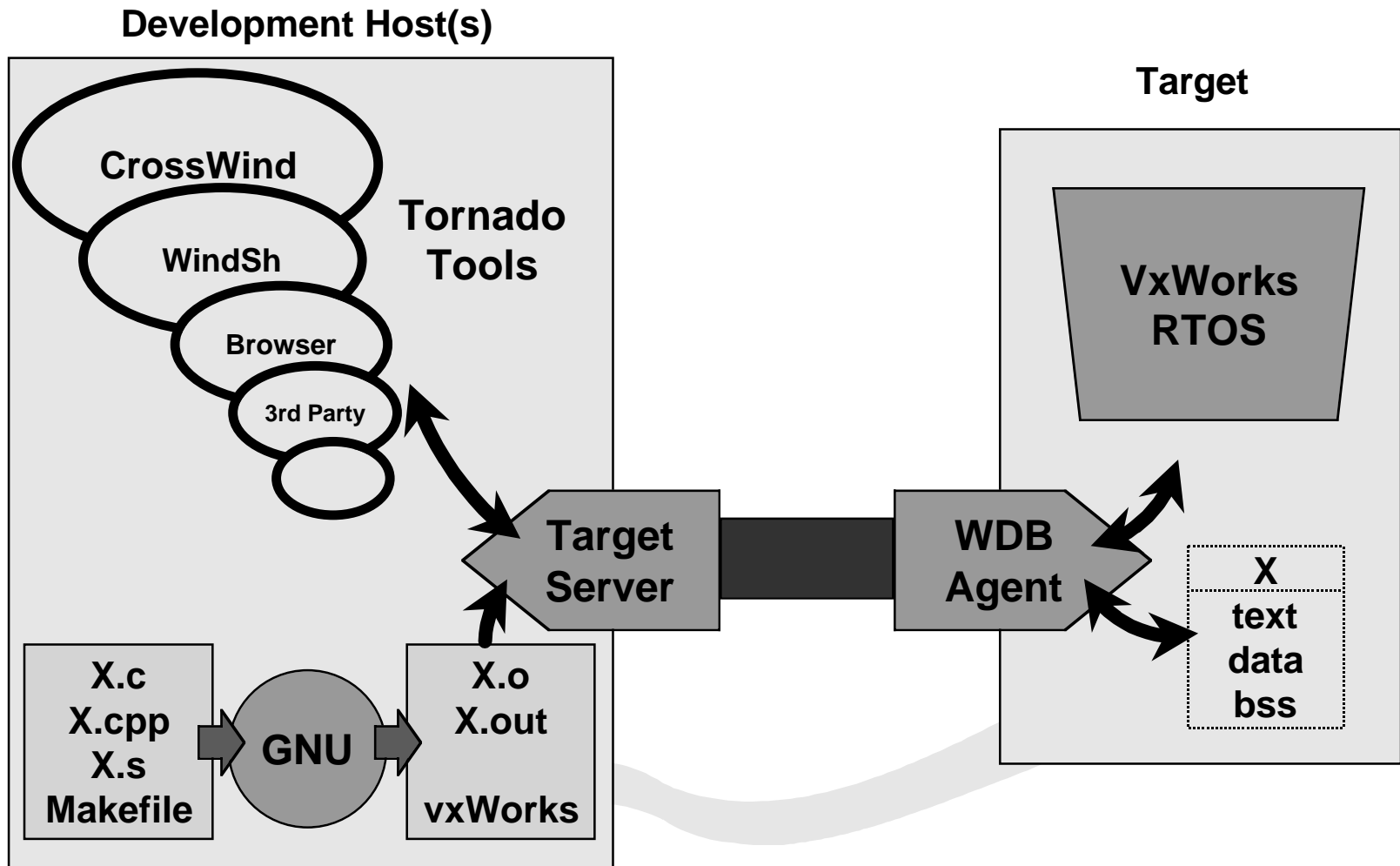
---

- Tornado Architecture
- Reconfiguring VxWorks
- VxWorks Basics
- VxWorks Libraries and Facilities
- VxWorks OS Extensions
- Using Tornado
- Optional Development Tools
- Help and Documentation

# Tornado Quick-Start Workshop

## Tornado Architecture

# Subdividing Tornado

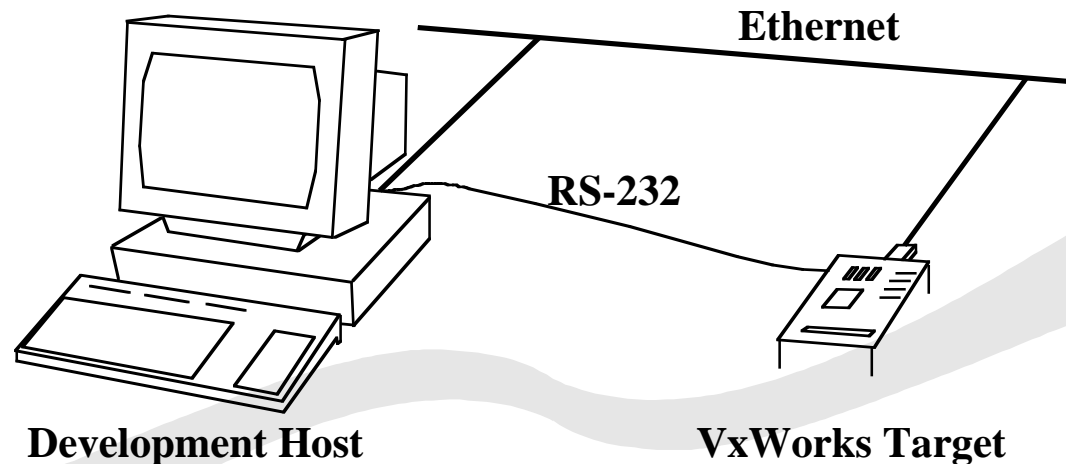


# Cross-Development

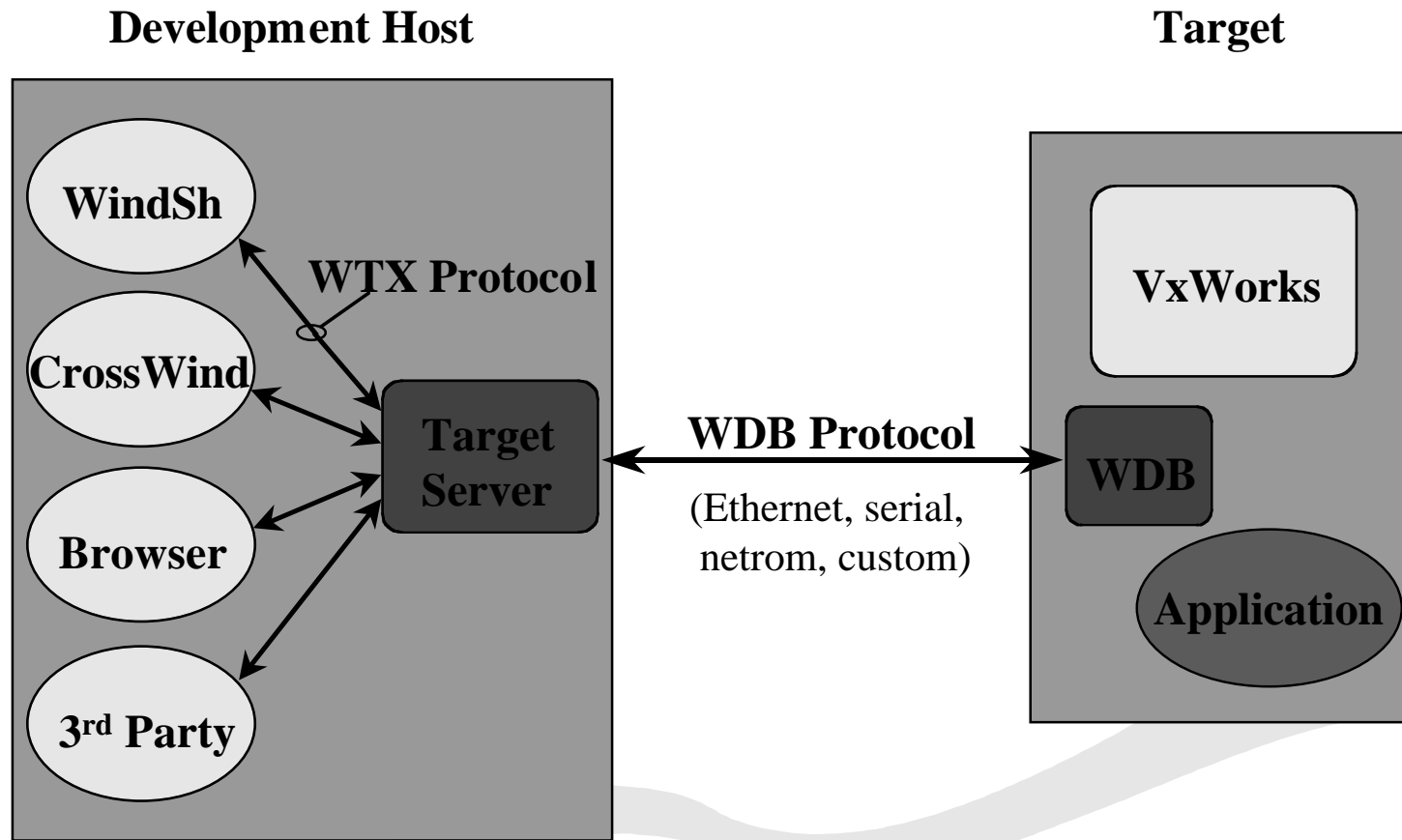
---

- Typical scenario:

1. Boot target.
2. Attach target server.
3. Edit & compile.
4. Download object module.
5. Test & Debug.
6. Return to 3 or 1 as necessary!



# Target Server / WDB Agent



WTX = Wind River Tool eXchange

WDB = Wind DeBug  
 **WindRiver**  
SYSTEMS

# Auxiliary Host Processes

- The Registry

- Manages a list of target servers. Must be running before you launch a target server.
- Provides information tools need to contact a target server.
- Prevents two target servers from attaching to the same target.



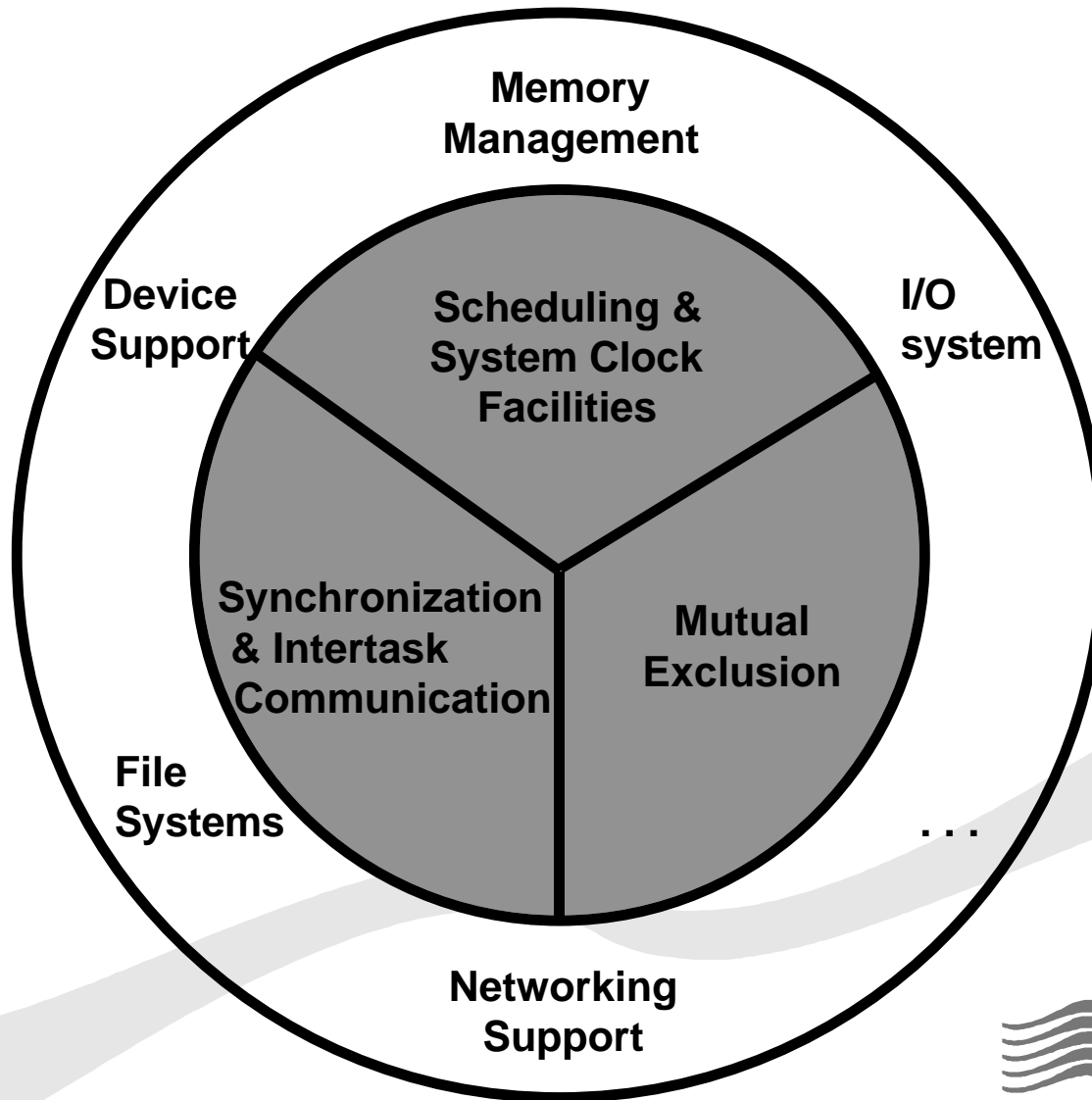
- Wind License Manager Daemon

- Floating license manager daemon allocates a finite number of *seats* to users of target servers.
- Each seat allows one user on one networked host to create any number of target servers.



# VxWorks RTOS

---



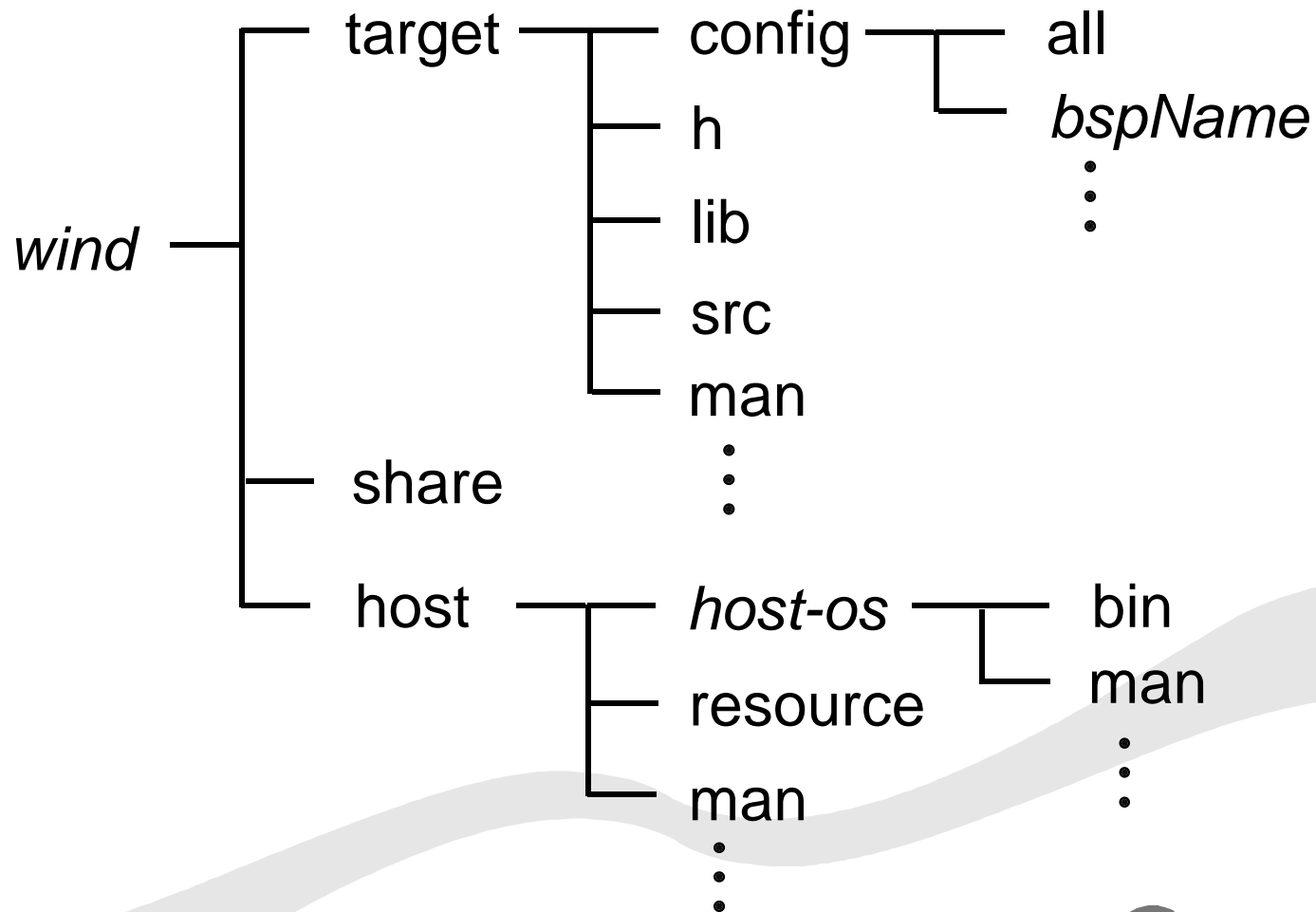
# Board Support Package

---

- Most of VxWorks is independent of the particular target board used.
- The board-specific code for initializing and managing a board's hardware is called the BSP. The BSP provides VxWorks with standard hardware interface functions which allow it to run on a board.
- The primary BSP source file, `sysLib.c`, lives in the BSP directory `target/config/bspName`. This is also the directory in which VxWorks may be reconfigured and rebuilt.

# Tornado Directory Structure

---



# Tornado Quick-Start Workshop

## Reconfiguring VxWorks

# Scaling & Configuration

---

- Configuring VxWorks involves
  - ñ Specifying which VxWorks facilities will be included (*scaling* VxWorks).
    - May be done with the WindConfig tool, or by editing files.
  - ñ Specifying additional constants needed by some modules.
    - Requires editing files.
- The configuration of the VxWorks images you build is governed by information from three sources:
  - ñ `target/config/all/configAll.h`      default base configuration
  - ñ `target/config/bsp/config.h`      board specific configuration
  - ñ The WindConfig tool, if used.

# configAll.h and config.h

---

- target/config/all/configAll.h specifies a default, or base configuration which applies to all boards. WRS discourages modifying this file.
- target/config/*bsp*/config.h extends and/or overrides the defaults specified in configAll.h for the particular board *bsp*. Modify this file to change the VxWorks image for this board.

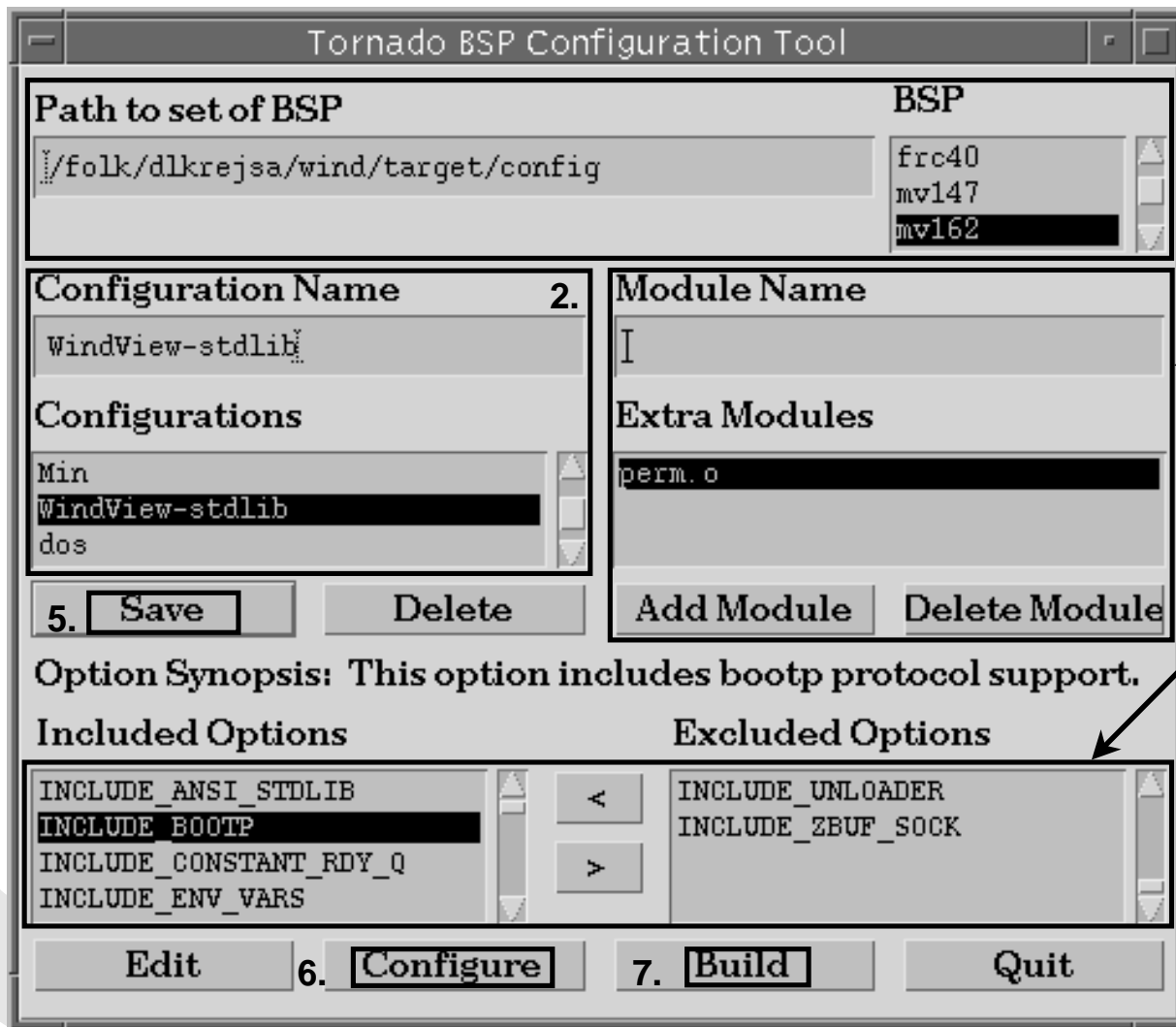
```
...
#include configAll.h
#include bsp.h

/* Override defaults here */
#undef NUM_FILES
#define NUM_FILES (80)
...
/* Added by WindConfig */
#include configdb.h
```

config.h

# WindConfig

Overrides *scaling* choices made in configAll.h and config.h.



1. Specify BSP.

2. Choose an existing configuration, or name a new one.

3. Specify additional modules to be linked with VxWorks.

4. Edit configuration.

5. Save configuration.

6. Construct configdb.h and modify config.h.

7. Rebuild vxWorks.

# Standard Images

Types of Images	VxWorks Tornado	VxWorks Standalone	Boot Program
ROMable compressed	–	vxWorks.st_rom	bootrom
ROMable uncompressed	vxWorks_rom	–	bootrom_uncmp
ROM resident	vxWorks.res_rom_nosym	vxWorks.res_rom	bootrom_res
Downloadable uncompressed	vxWorks	vxWorks.st	–

- i Not all BSPs will support all of these images. Some BSPs support additional images.
- i Standalone VxWorks has a target shell and built-in symbol table. Network support is included but not initialized.
- i The file target/h/make/rules.bsp has the *make* rules for building these images.



# Tornado Quick-Start Workshop

**VxWorks Basics**

# VxWorks

---

- VxWorks is a multitasking operating system optimized for real-time and embedded applications.
  - ñ Low interrupt and context switch latency.
  - ñ Low system call overhead.
  - ñ Scalable.
  - ñ Portable: well defined BSP.
- VxWorks consists of core kernel facilities and peripheral facilities which depend on the kernel.
  - ñ Loosely, kernel functions are those which can *directly* change the states of tasks.

# What is a VxWorks Task?

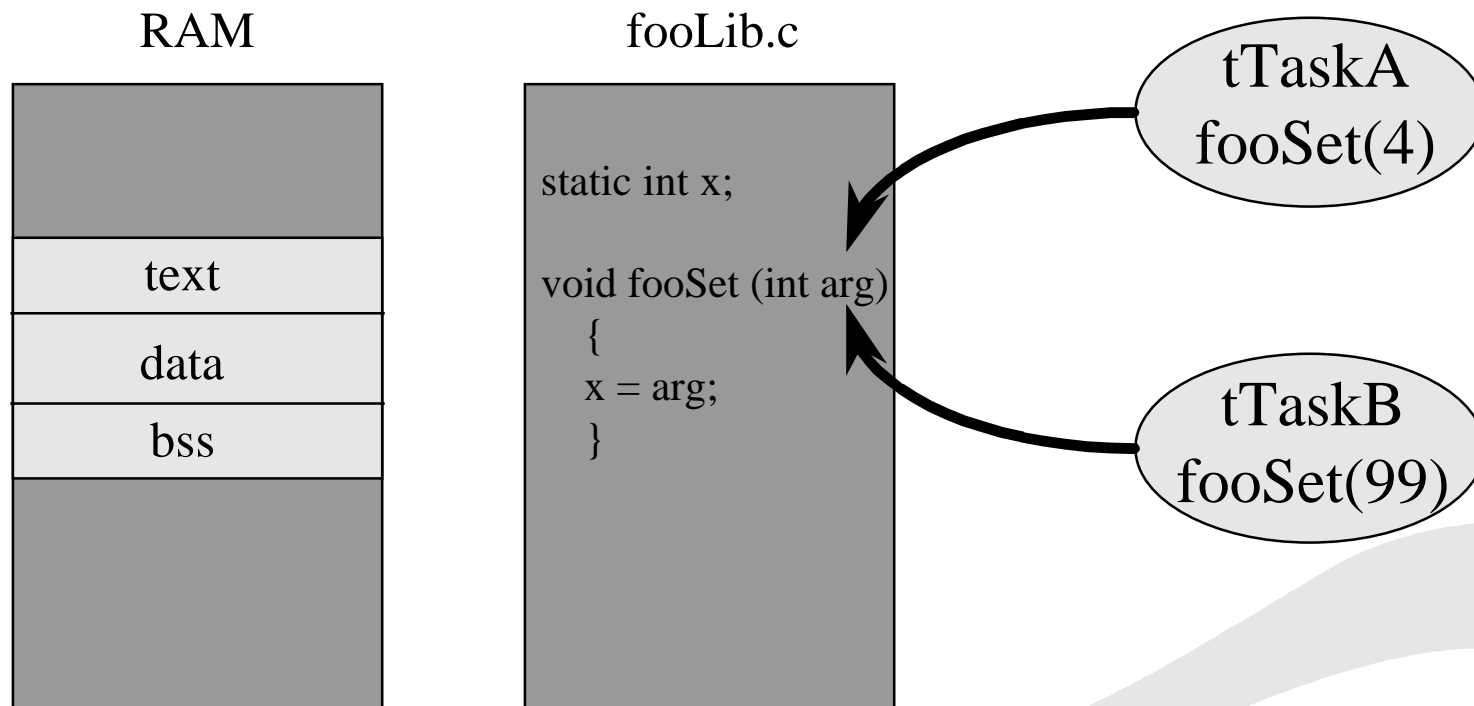
---

- A task is a *context of execution*. It has:
  - ñ a program counter (current location of execution).
  - ñ private copies of CPU registers.
  - ñ a stack for local variables, function arguments, and function call chain information.
- As VxWorks is a uniprocessor system, only one task is actually executing at any given time.
  - ñ When a task is not executing, its context is stored in its *Task Control Block* (TCB) and stack. The TCB is the data structure which the kernel uses to represent and control the task.

# Performance Optimizations

---

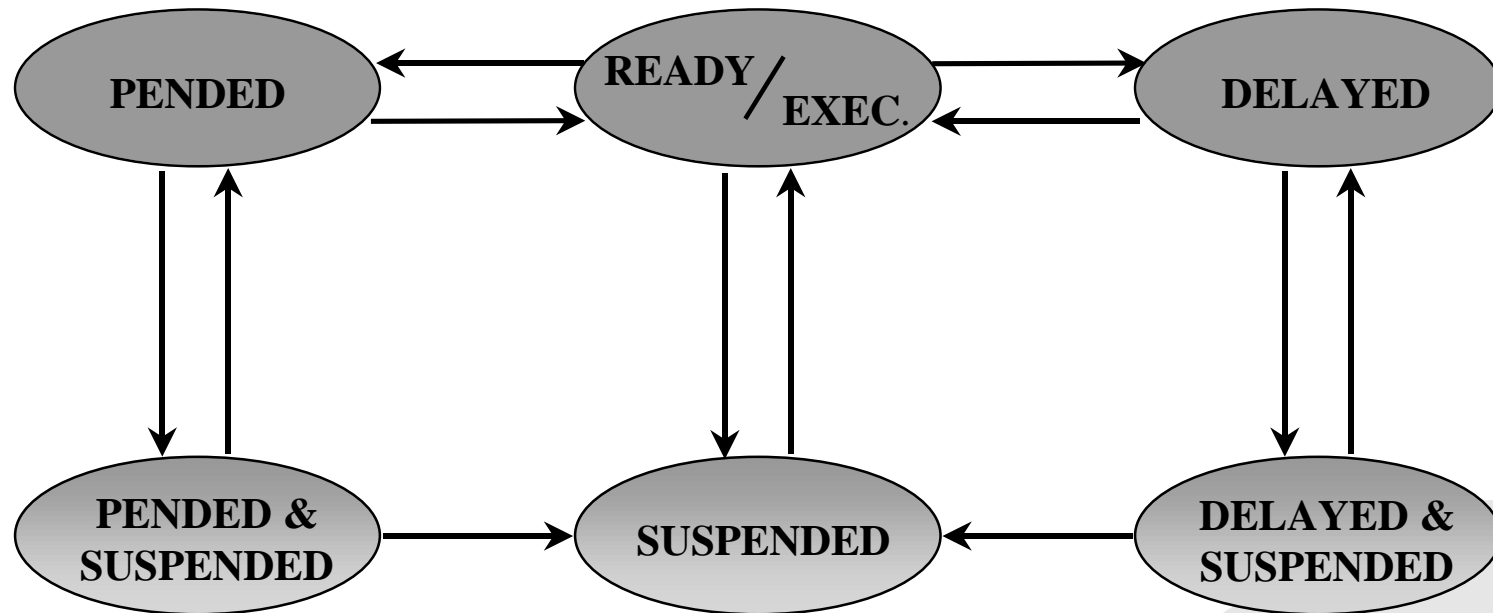
- All tasks execute in the same address space:



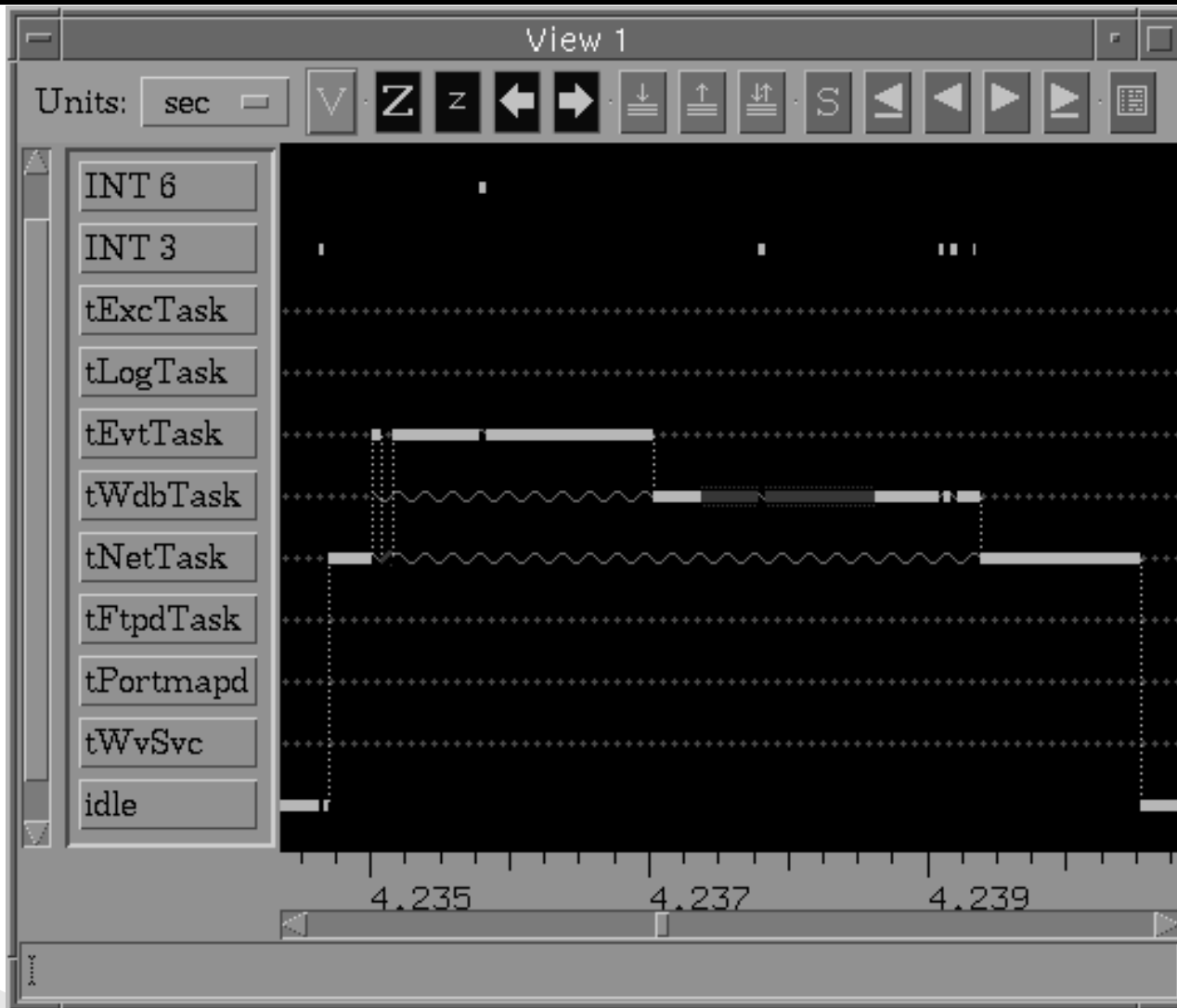
- All tasks execute at highest CPU privilege level.

# Task State Transition Diagram

---



# Preemptive Priority Scheduling



**At any time, the highest priority task *ready* to run, runs!**

**A higher priority task which is made ready preempts the executing task.**

**Interrupts preempt any task.**

(On this board, INT 6 is the system clock, INT 3 is the network interrupt.)

# Context Switches

---

- When one task stops executing and another task starts, a *context switch* has occurred.
- Context switches occur:
  - ñ If a higher priority task becomes ready to run
    - made ready by executing task.
    - made ready in an interrupt, or times out on a blocking call.
  - ñ If the executing task makes a blocking kernel call (moving into a *pending*, *delayed*, or *suspended* state).
- What happens:
  - ñ CPU registers for outgoing task stored in its TCB.
  - ñ CPU registers for incoming task retrieved from its TCB.

# Interrupt Service Routines

---

- An ISR is a piece of code which is *connected* to a particular hardware interrupt. When the hardware interrupt occurs, the ISR runs.
- ISRs have an effective priority higher than any task.
- Whether and how ISRs can preempt each other is board dependent.
- An ISR has no permanent context; it is **NOT** a task.
  - ñ An ISR may call only a limited set of VxWorks functions.
  - ñ Basic rule of thumb: If a routine could block, an ISR cannot call it.



# Tornado Quick-Start Workshop

**VxWorks Libraries and  
Facilities**

# VxWorks Libraries / Modules

---

- VxWorks routines are grouped into *libraries* (modules containing code and data).
- Many of these libraries are optional; VxWorks may be built with or without them.
- Each library has one or more header files. Examples:

<u>Module</u>	<u>Routine</u>	<u>Header files</u>
taskLib	taskSpawn()	taskLib.h
semLib	semTake()	semLib.h
memPartLib	malloc()	stdlib.h
sockLib	sendto()	sys/types.h, sys/socket.h, sockLib.h

- Reference manual lists header files for each library.

# taskLib

---

- taskLib contains the kernel functions for creating, destroying, starting and stopping tasks.
- Example routines:

- ñ To create and start a new task:

```
int taskSpawn (name, priority, options, stackSize,  
              entryPt, arg0, Ö , arg9)
```

- ñ To delay the executing task for a certain number of system clock ticks:

```
STATUS taskDelay (ticks)
```

# Semaphores

---

- Semaphores are VxWorks kernel objects which allow blocking and unblocking of tasks, to coordinate tasks' actions with those of other tasks and with external events.
- VxWorks provides three varieties of semaphores:
  - ñ Binary (synchronization) semaphores.
  - ñ Counting semaphores.
  - ñ Mutex (mutual exclusion) semaphores.
- Each type of semaphore is intended primarily for a particular kind of programming problem.

# Binary Semaphores

---

- Binary semaphores allow tasks to wait for an event without taking up CPU time polling.
- The event might be an interrupt, or the action of another task.
- Usage:
  - ñ Create the binary semaphore using `semBCreate()`
  - ñ A task which wishes to wait for the event calls `semTake()` to block until the event occurs (or a specified time-out expires).
  - ñ Whichever ISR or task detects or creates the event calls `semGive()` to allow a waiting task to proceed.

# Mutex Semaphores

---

- Mutex semaphores are used when multiple tasks share a resource (data structure, file, hardware).
- When used correctly, mutex semaphores prevent multiple tasks from accessing the resource at the same time, and so corrupting it.
- Usage:
  - ñ Create mutex for the resource with `semMCreate()`.
  - ñ A task wanting to use the resource first calls `semTake()` to block until the resource is available (or time-out expires).
  - ñ When done with the resource, a task calls `semGive()` to allow other tasks to use the resource.

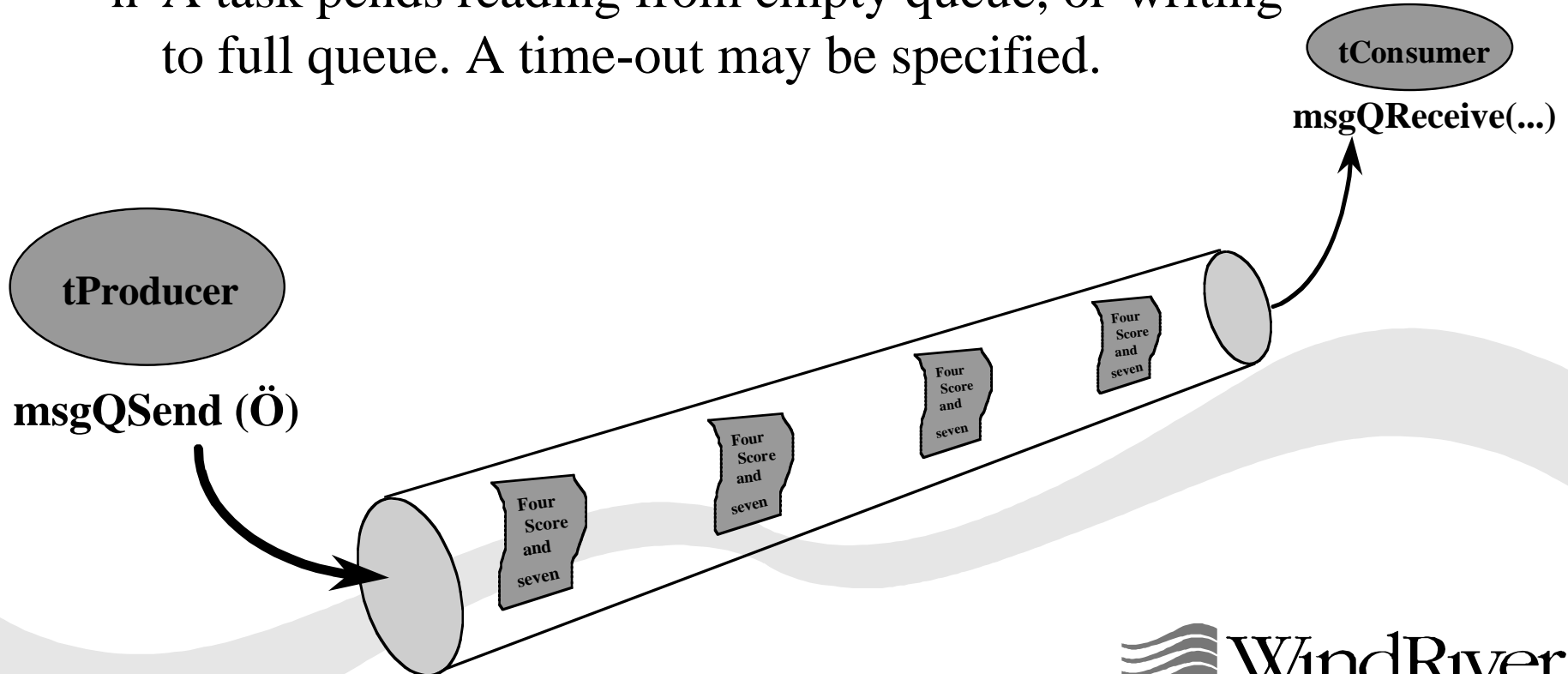
# Counting Semaphores

---

- Binary semaphores keep track of whether or not an event has occurred, but not *how many times* the event has occurred (since the last time the event was serviced).
- Counting semaphores keep a count of how many times the event has occurred, but not been serviced.
- May be used to ensure that the event is serviced as many times as it occurs.
- May also be used to maintain an atomic count of multiple equivalent available resources.

# Message Queues

- FIFO buffers of bounded length messages.
- Synchronization and mutual exclusion built in.
  - ñ A task pends reading from empty queue, or writing to full queue. A time-out may be specified.





# Watchdogs

---

- Watchdogs allow a routine to run after a specified delay, as part of the system clock tick ISR.
- Used for high-reliability periodic polling, and deadline-miss detection.

## Periodic Execution

```
wd = wdCreate();
wdStart (wd, period,
         myWdISR, arg);
...
void myWdISR (int param)
{
    wdStart (wd, period,
            myWdISR, param);
    doIt (param);
}
```

## Deadline-miss Recovery

```
wd = wdCreate();
FOREVER
{
    wdStart (wd, deadline,
            panicWd, arg);
    doWork();
}
...
void panicWd (int param)
{ /* Handle deadline miss */
}
```

# Signals

---

- VxWorks implements reliable POSIX signals.
- Used in two ways:
  - ñ To notify a task asynchronously of some event.
  - ñ In exception handling.
- If a task commits an exception, it is suspended unless it has installed a signal handler for the corresponding signal.
  - ñ If it has installed the handler, the handler may attempt to recover from the exception by restarting the task, or by using `longjmp()` to return to a state previously saved by calling `setjmp()`.

# VxWorks I/O System

---

- Provides a consistent, familiar interface to various I/O devices and file systems.
  - ñ Open a file/device by name with `open()`.
  - ñ Use the integer *file descriptor* returned from `open()` in later calls to `read()`, `write()`, and `ioctl()`.
  - ñ File descriptors may be shared among multiple tasks.
  - ñ When all tasks are done with a file descriptor, have one task call `close()` so that the file descriptor may be reused.
- Global or task-specific redirection of *standard input*, *standard output*, and *standard error* is possible.

# Additional I/O Support

---

- VxWorks supports the Standard C buffered I/O library (ansiStdio, header file stdio.h):
  - ñ fopen(), fclose(), fread(), fwrite(), getc(), putc(), etc.
- Built on top of VxWorks Basic I/O system.
- Since the basic I/O system calls reach driver level very quickly (no protection boundary crossing overhead), buffered I/O may be less important.
- VxWorks implements printf() in a separate library fioLib as an unbuffered call. This allows formatted output without support for buffering.
- C++ Iostreams library provided.

# Local File Systems

---

- VxWorks provides an implementation of the DOS file system, dosFsLib.
- Compatible with MS-DOS up to release 6.2.
- File system may be used with local block devices such as SCSI, ATA, or floppy disk drives; or RAM disks.
- To access the raw block device via the I/O system, rawFsLib may be used.
- The module tapeFsLib may be used to access a SCSI sequential device (tape drive).

# VxWorks Network Support

---

- Network interfaces:

- ñ Ethernet

- ñ PPP/SLIP/CSLIP

- ñ Shared Memory Network (VME backplane)

- ñ Custom

- Network Programming APIs:

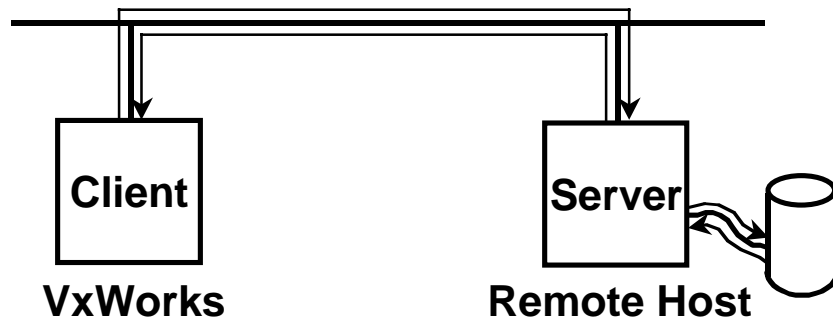
- ñ 4.3 BSD sockets.

- ñ *zbuf* (Zero-copy TCP/UDP) sockets.

- ñ Sun RPC.

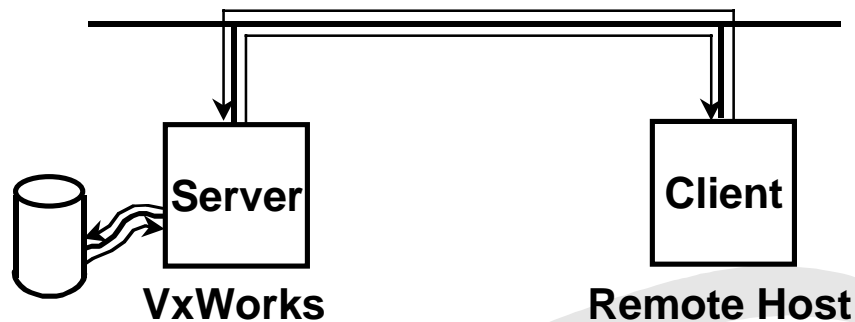
# Network File Access

- VxWorks can act as client or server for remote file access:



**Client:** VxWorks target accesses files on remote host via NFS, FTP or RSH.

- NFS                      nfsDrv
- FTP or RSH            netDrv, ftpLib



**Server:** Remote host accesses files local to VxWorks target via NFS or FTP.

- NFS                      nfsdLib, mountLib
- FTP                      ftpdLib

# Tornado Quick-Start Workshop

**VxWorks OS Extensions**



# Extending VxWorks

---

- These optional products provide target-side capabilities not present in the core VxWorks product.

ñ VxVMI

Virtual Memory Support

ñ VxMP

Multiprocessing across VMEbus

ñ Wind Foundation  
Classes

C++ libraries for VxWorks

# VxVMI

---

- Provides an architecture-independent interface to the Memory Management Unit (MMU).
- Automatic write-protection of program text segments and the interrupt vector table may be enabled.
- Private virtual memory contexts may be established.
  - ñ Context may be accessed by a single task, by a group of tasks, or only via particular interface functions.
  - ñ User is responsible for managing and switching between contexts. Example code available in *Programmer's Guide*.
  - ñ Intended primarily for data protection; does not completely isolate or protect tasks from each other.

# VxMP

---

- VxMP (also called *shared memory objects*) provides MultiProcessing support for VxWorks targets on a VME bus. It allows one to synchronize actions or send messages between different CPUs on the bus.
- Shared memory objects provided:
  - ñ semaphores (binary & counting)
  - ñ message queues
  - ñ memory partitions
- Name database available for publishing and looking up individual shared memory objects by name.

# Wind Foundation Classes

---

- Provides three libraries to aid C++ development in VxWorks:
  - VxWorks Wrapper Class Library
    - ñ Thin C++ interface to VxWorks objects and facilities
  - Tools.h++ library from Rogue Wave Software
    - ñ Collection classes, template based classes, persistent store facility, B-trees, strings, etc.
  - Booch Components Library
    - ñ Common Data structures (graphs, rings, queues, stacks, etc.)
    - ñ Common algorithms (date/time, searching, sorting, etc.)

# Tornado Quick-Start Workshop

**Using Tornado**

# Booting a Target

---

- Hardware must first be configured:
  - ñ VxWorks Boot ROMs replace board manufacturer's ROMs.
  - ñ Jumpers (etc.) set as described in target/config/bsp/target.txt.
- VxWorks Boot ROMs enable:
  - ñ Setting boot parameters via a serial connection.
  - ñ Downloading & executing VxWorks image.
- Booting scenarios:
  - ethernet
  - serial (PPP/SLIP)
  - BOOTP / TFTP
  - shared memory network
  - local disk

# Boot Parameters

---

Press any key to stop auto-boot...

3

[VxWorks Boot]: **c**

'.' = clear field; '-' = go to previous field; ^D = quit

```
boot device           : ei
processor number      : 0
host name             : sylvan
file name             : /usr/wind/target/config/mv162/vxWorks
inet on ethernet (e) : 167.21.32.168
inet on backplane (b):
host inet (h)         : 167.21.32.17
gateway inet (g)      :
user (u)              : robinh
ftp password (pw) (blank = use rsh): .
flags (f)             : 0x8
target name (tn)      : ash
startup script (s)    :
other (o)             :
```

# A Successful Boot!

---

[VxWorks Boot]: @

< *boot parameters printed* >

Attaching network interface ei0... done.

Attaching network interface lo0... done.

Loading... 335336 + 28936 + 33948 ← *text + data + bss*

Starting at 0x20000...

← *downloaded image runs now*

Attaching network interface ei0... done.

Attaching network interface lo0... done.

NFS client support not included.

VxWorks

Copyright 1984-1996 Wind River Systems, Inc.

CPU: Motorola MVME162

VxWorks: 5.3.1

BSP version: 1.1/4

Creation date: Sep 28 1997

WDB: Ready.






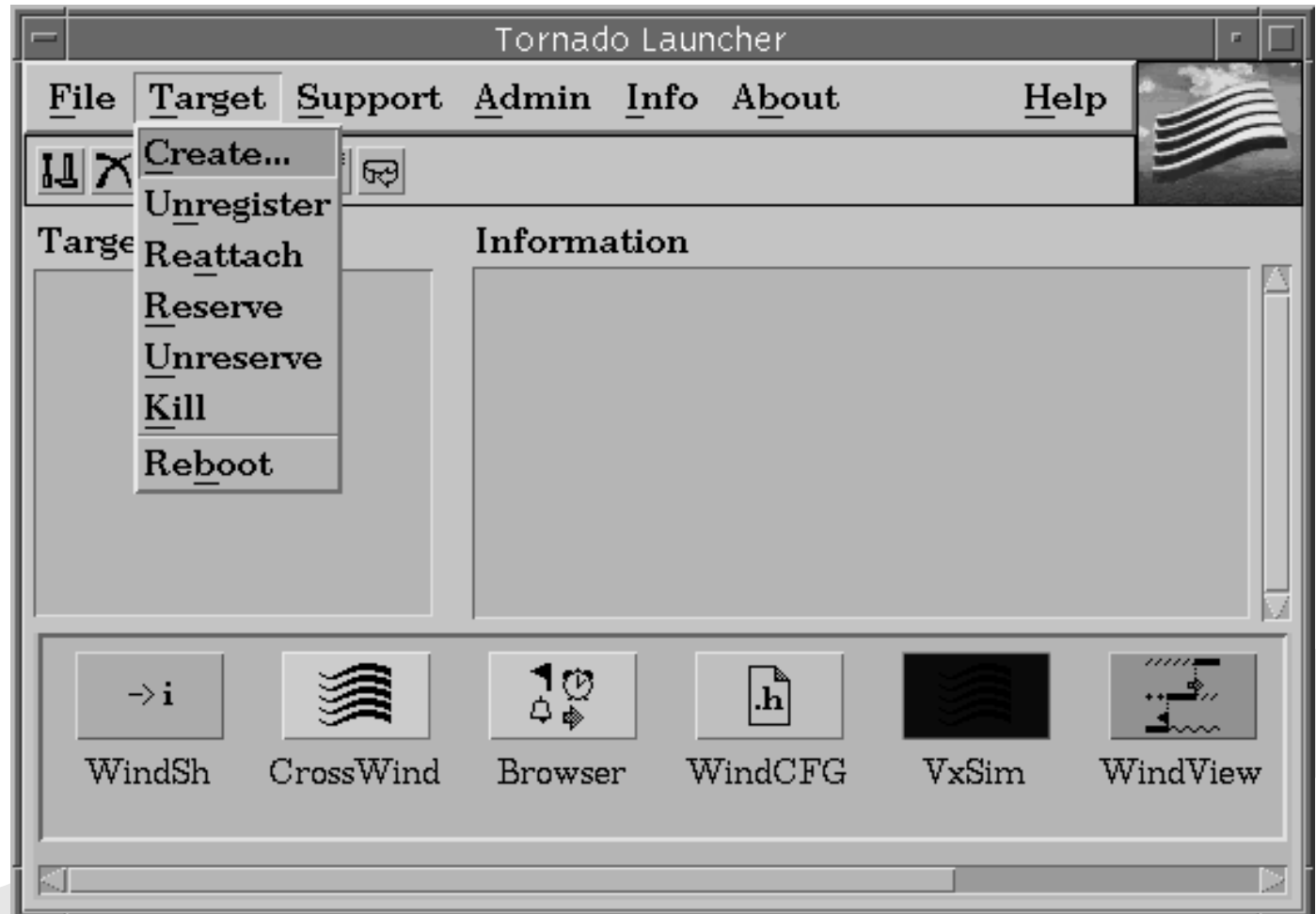
# Launching a Target Server

Start the Tornado Launcher from the UNIX command line as follows:

```
% launch &
```

Use *Target => Create* to configure and launch a new target server. The configure button  is a shortcut.

With either method, the *Create Target Server* dialog will appear.



# Configuring a Target Server

Name this target server configuration.

Enter target's IP address or equivalent host name.

igor  
igorLog  
igorSer

Target name or IP address  
t12-168

Options:  
Authorized users file  
Core file

Target server name  
igor

Object module format  
a.out  
coff

Console display  
Memory cache size  
default

Virtual console

You may check *Virtual Console* to create a window to which target I/O may be directed. Useful when developing remotely.

server uses to access the object module for the VxWorks image on the target. The default is the path specified in the boot parameters.

*Core file* is the path the target

# Configuring a Target Server

The *default* back end (wdbrpc) uses a shared network connection between the target server and WDB agent.

Specify *Verbose* to log diagnostic output.

No symbols     All symbols     Verbose     Locked

Target/Host symbol tables synchronization

Backend list	Serial line speed	Serial line device
default	1200	
loopback	2400	

Backend timeout	Backend resend	Backend Log file

Target server launch command

```
tgtsvr t12-168 -L -V -n igor
```

Help...    Delete    Launch    Quit

Clicking *Locked* keeps other users from accessing the target server.

As you fill in the dialog options, the launch command line is automatically constructed.

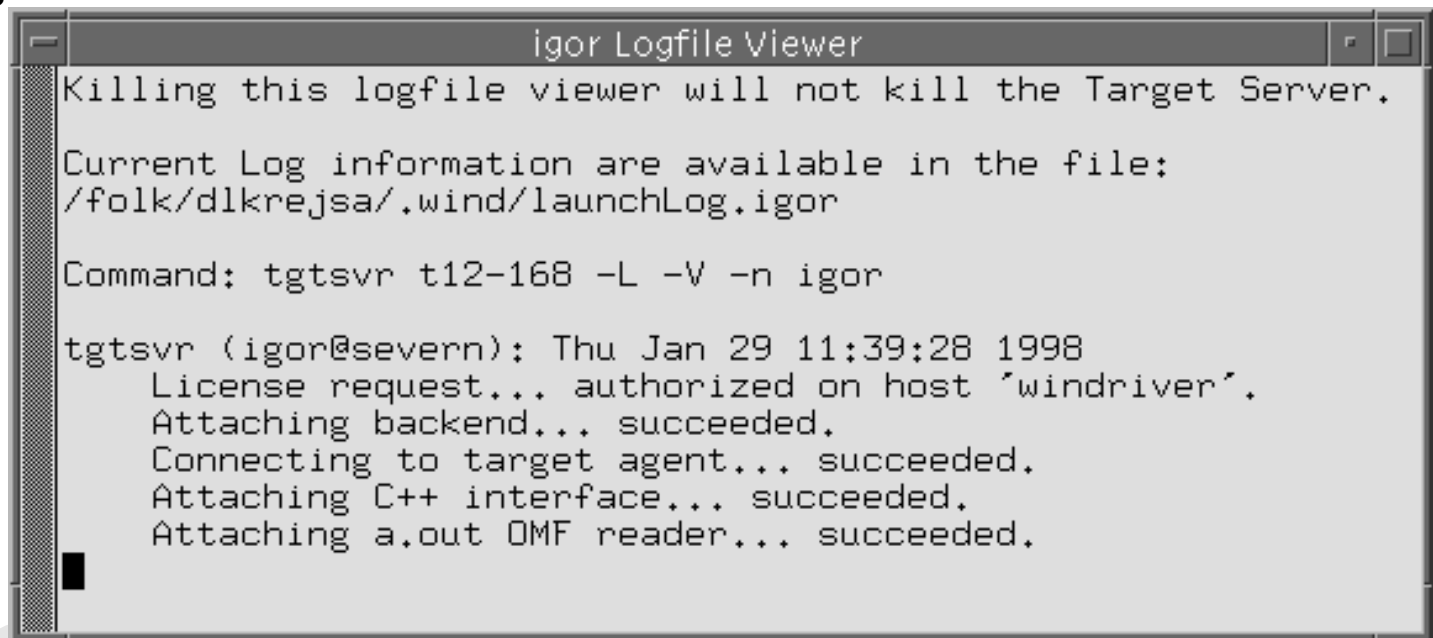
*Launch* starts the target server and saves its configuration.

# A Successful Launch!

---

The Logfile Viewer displays diagnostic information about:

- ï Acquiring a license
- ï Connecting to the target
- ï WTX protocol errors
- ï Lost connection to target
- ï Target reboots



```
igor Logfile Viewer
Killing this logfile viewer will not kill the Target Server.

Current Log information are available in the file:
/folk/dlkrejsa/.wind/launchLog.igor

Command: tgtsvr t12-168 -L -V -n igor

tgtsvr (igor@severn): Thu Jan 29 11:39:28 1998
License request... authorized on host 'windriver'.
Attaching backend... succeeded.
Connecting to target agent... succeeded.
Attaching C++ interface... succeeded.
Attaching a.out DMF reader... succeeded.
```

# Starting Tornado Tools



1. Choose a target server from the registry list.

2. Information on that target server and the attached target is displayed.

3. Click on a tool icon.

# CrossWind

```
File Targets Source Tcl Windows About Help
[Icons]
/severn1/dlkrejsa/prog/perm.c
44     FOREVER
45     {
46     ▼     if (semTake (mutex, WAIT_FOREVER) == EF
47           return;
48
49           i1 = rand() % length;
50           i2 = rand() % length;
51     ▶     temp = string[i1];
52           string[i1] = string[i2];
53           string[i2] = temp;
54
55           printf ("%s) '%s'\r",taskName(0), stri
56           semGive (mutex);
57
58
59
Breakpoint
46     if (semTake (mutex, WAIT_FOREVER) == EF
(gdb) ]
```

A graphical, source-level debugger built on GDB from GNU.

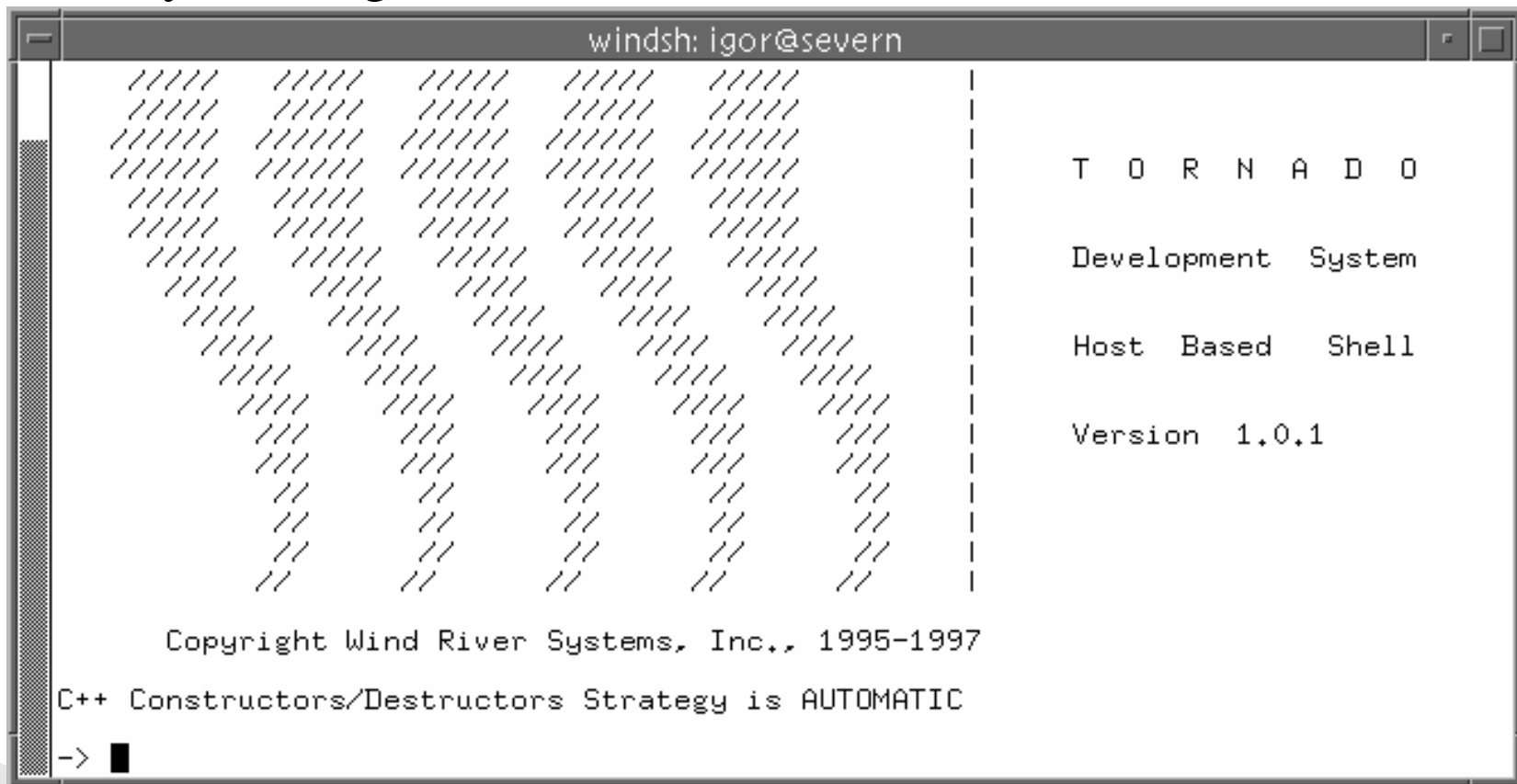
Provides task level and system level debugging.

Use either the graphical or the command line interface.



# WindSh - The Tornado Shell

- A C expression interpreter and an associated Tcl interpreter.
- Download code to the target; spawn tasks to execute functions; modify existing variables, or create new ones.



```
windsh: igor@severn  
  
T O R N A D O  
Development System  
Host Based Shell  
Version 1.0.1  
  
Copyright Wind River Systems, Inc., 1995-1997  
C++ Constructors/Destructors Strategy is AUTOMATIC  
-> █
```

# The Browser



The screenshot shows the 'Browser igor@severn' window. It has a menu bar with 'File', 'About', and 'Help'. Below the menu is a toolbar with icons for help, refresh, search, and other functions. The main area is titled 'Numerical / Cumulative' and contains two expandable folders: 'WRS Tasks' and 'User Tasks'. The 'WRS Tasks' folder is expanded, showing a list of tasks with their addresses, names, and states.

Address	Task Name	State
0x3e8a70	tExcTask	PEND
0x3e615c	tLogTask	PEND
0x3e1d28	tNetTask	READY
0x3c7204	tPortmapd	PEND
0x3b91f4	tWdbTask	READY

Below the task list is a 'Show' button. At the bottom of the window, there are two input fields: 'Tools' with the value '0' and 'Application' with the value '280136'. Below these fields is a table showing memory usage for the 'vxWorks' application.

ID	NAME	.text	.data	.bss
0x98520	vxWorks	343140	28684	33896
	Total:	343140	28684	33896

Graphical tool which lets you monitor the state of the target, and display information on particular VxWorks system objects.

Information displayed may be updated on demand  or periodically .

The screenshot shows a window titled 'igor@severn: SEM 0x3fec48'. It contains a list of attributes for a task:

- Attributes
  - type = MUTEX
  - queue = FIFO
  - pending = 0
  - state = NotOwned
- Blocked Tasks



# Compiling Code

---

- Compile individual source files using the appropriate cross-compiler and flags. Example:
  - ñ `cc68k -c -I${WIND_BASE}/target/h -fno-builtin -nostdinc -DCPU=MC68040 -Wall -O myProg.c`
  - ñ See the *Programmer's Guide* appendix for your architecture.
- For more complicated builds, create your own *Makefile*.
  - ñ You may imitate the structure of VxWorks BSP makefiles, which include make definitions and rules from the `target/h/make` directory. You may even extend the BSP makefiles to build your own application.

# Downloading Object Modules

---

- Object modules may be downloaded dynamically and linked with modules already present on the target.
  - ñ You will be notified of any unresolved symbols in the downloaded module.
  - ñ Unresolved symbols on the target are *not* resolved upon later loading other modules. Resolving mutual dependencies among modules requires host-side incremental linking.
- To download from the Wind shell:
  - **ld < myProg.o**
- File => Download also loads debugging information used by CrossWind.

# Using the Tornado Shell

---

- Evaluate C expressions including Target functions:

```
→ len = msgQReceive (msgQId, buf, size, 0)
new symbol "len" added to symbol table
len = 0xb7f1c: value = 10 = 0xa
→ printf ("%s\n", buf)
value = 9 = 0x9
```

- Spawn new tasks to execute repetitive code:

```
→ sp (myServerTask, msgQId, "Whoops!\n")
task spawned: id = 3c2d6c, name = u0
value = 3943788 = 0x3c2d6c
```

- Other things you can do from the shell:

Download object code  
Assembly level debugging  
More!

Display system objects  
Automate testing with Tcl



# WindSh Limitations

---

- Does not understand structures and arrays.
- Assumes expressions are of type *int* or *double* unless typecasts used.
- C interpreter cannot handle looping or branching constructs (if, while, for, switch, etc.).
- Function call argument limitations:
  - ñ 10 four-byte arguments (*int*, *float*, or *pointer*) passed to entry point function for all tasks spawned.
  - ñ Different sized arguments (e.g. *doubles*) may often be passed for architectures which expect arguments on the stack; this is not always supported for architectures expecting such arguments in registers.

# Symbolic Debugging

---

- For source-level debugging with CrossWind, modules must be compiled with additional debug information:
  - ñ Specify the `-g` compiler flag. If using VxWorks makefiles, you may do this by defining the make variable `ADDED_CFLAGS = -g` or `ADDED_C++FLAGS = -g`.
- When started, CrossWind looks for debug information for all the modules already on the target. It searches for the object modules, and the corresponding source code, in an ordered set of directories called the *source path*.
- Configure the source path with the GDB *directory* command:



(gdb) **directory** *dirname* ...

# Debugging Tasks






---

- To download additional modules:  
File => Download
- To create a new task to run a loaded function  
(gdb) **run *myFunc arg1*** **Ö**  
ñ Arguments should be separated with spaces.
- To debug an already running task:  
Targets => Attach Task
- To debug multiple tasks independently, you may start multiple CrossWind sessions.

# CrossWind GUI

Set regular  or temporary  breakpoints. Right-click on a line to set a regular breakpoint.

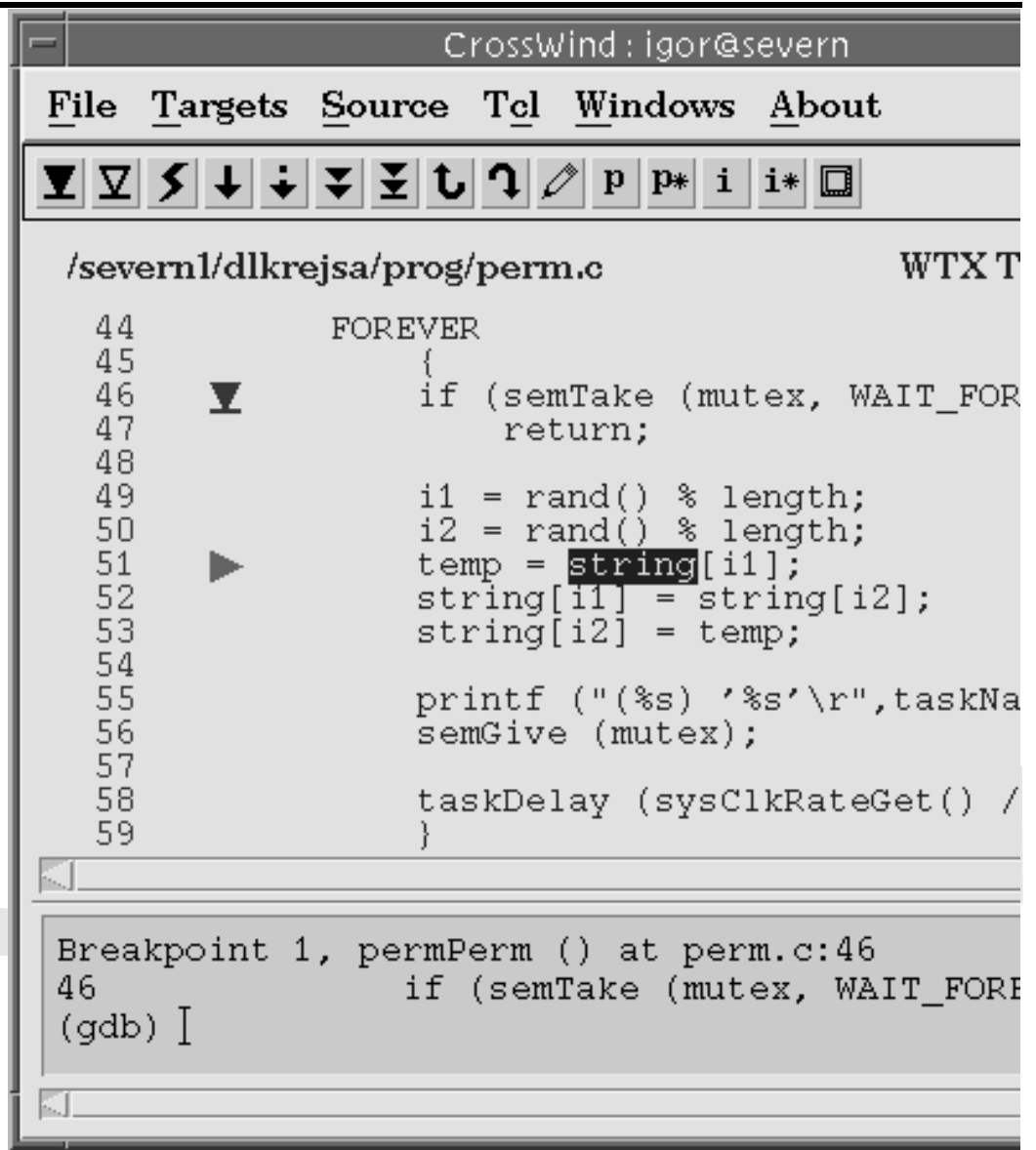
Suspend execution .

    Step, Next (step over function call), Continue, Finish current subroutine, or just middle-drag the execution point  to a new location.

Move Up  or Down  the stack to view variables in other stack frames.

Display expressions in the command panel  .

or separate windows  .



The screenshot shows the CrossWind GUI window titled "CrossWind: igor@severn". The menu bar includes "File", "Targets", "Source", "Tel", "Windows", and "About". The toolbar contains icons for setting regular and temporary breakpoints, suspend execution, step over, step next, continue, finish, move up/down stack, panel, panel\*, window, and window\*.

The source code window displays the following code from `/severn1/dlkrejsa/prog/perm.c`:

```
44         FOREVER
45         {
46             if (semTake (mutex, WAIT_FOR
47                 return;
48
49             i1 = rand() % length;
50             i2 = rand() % length;
51             temp = string[i1];
52             string[i1] = string[i2];
53             string[i2] = temp;
54
55             printf ("%s) '%s'\r", taskNa
56                 semGive (mutex);
57
58             taskDelay (sysClkRateGet() /
59                 }
```

The command panel at the bottom shows:

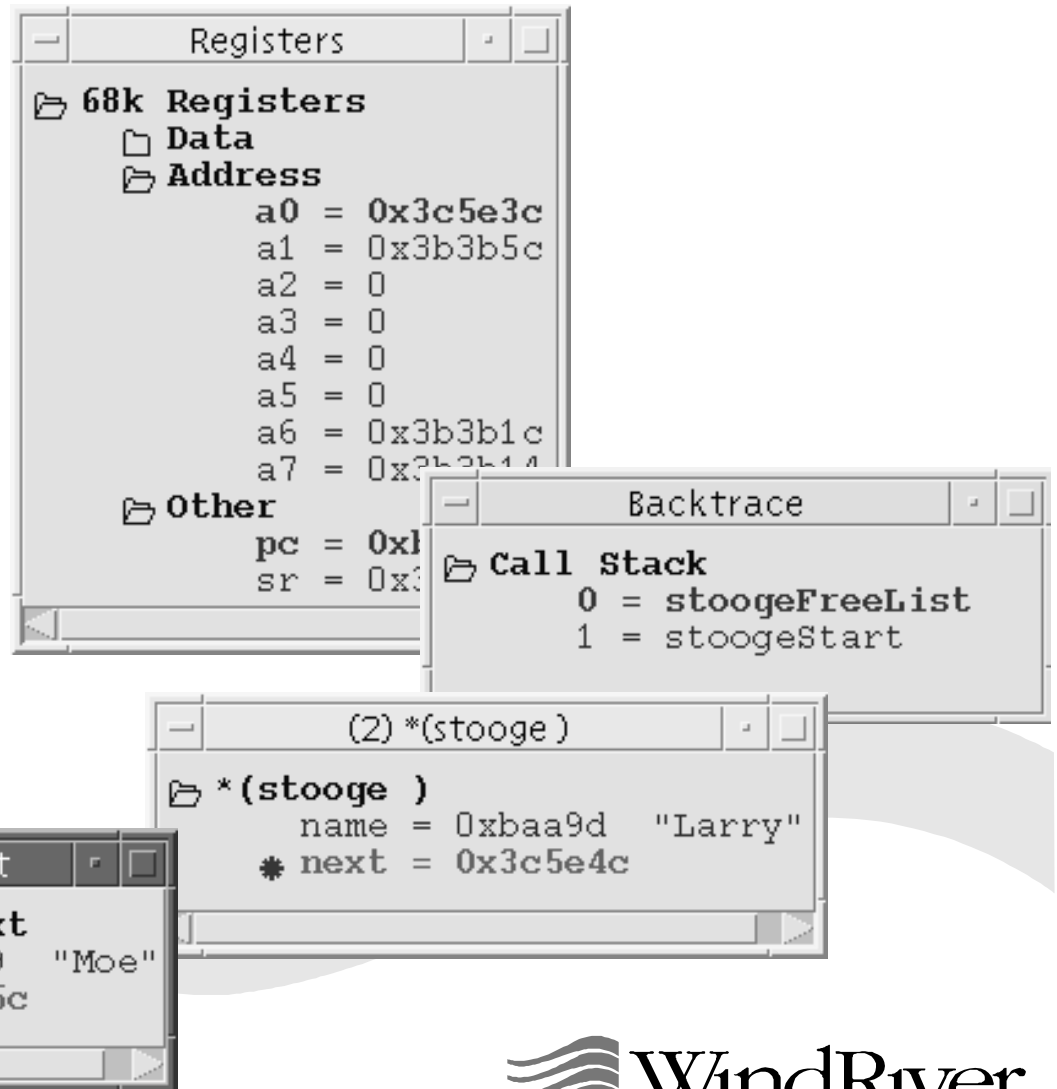
```
Breakpoint 1, permPerm () at perm.c:46
46             if (semTake (mutex, WAIT_FOR
(gdb) ]
```

# CrossWind Displays

- Application expressions
- CPU registers
- Stack call chain

These graphical displays are updated whenever control returns to the debugger.

Click on a pointer in a display to display the object pointed to.







# Task vs. System Debugging

---

- When debugging at task level, if the attached task hits a breakpoint, only that task stops. The rest of VxWorks continues to run.
- System level debugging allows debugging of ISRs, or debugging before VxWorks has started. When any task or ISR hits a breakpoint, VxWorks and the user application stop cold, interrupts are locked out, and control transfers to the external WDB agent.
- System level debugging requires an external WDB agent configured for a polled-mode back end (e.g. *wdbserial* or *netrom*).

# Browser Information

---

- Target Information
- Memory Usage (tools / application)
- Module Information (symbols)
- Object Information (particular system objects)
- Spy Chart (CPU utilization) 
- Stack Check 
- Tasks



Update Now



Update Periodically



Configure Browser

# Tcl - Tool Command Language

---

- Tcl is an interpreted scripting language, providing
  - ñ functions with arguments
  - ñ iteration and branching constructs
  - ñ variables of many types, represented as strings
- The Tornado tools may be customized using Tcl:
  - ñ Modify or extend the Graphical User Interface.
  - ñ Automate frequently used or repetitive procedures.
  - ñ Access the WTX protocol directly.
- Tcl source code for Tornado tools is provided in `host/resource/tcl`.

# Tornado Quick-Start Workshop

**Optional Development Tools**

# WindPower Tools

---

- WindView

- ñ Provides detailed graphical displays of the timing of system and user events.
- ñ Aids in detecting and diagnosing real-time programming errors and system crashes.

- Stethoscope

- ñ Collects and graphs time histories of user or system variables.
- ñ Additional utilities include an execution profiler and memory pool monitoring.

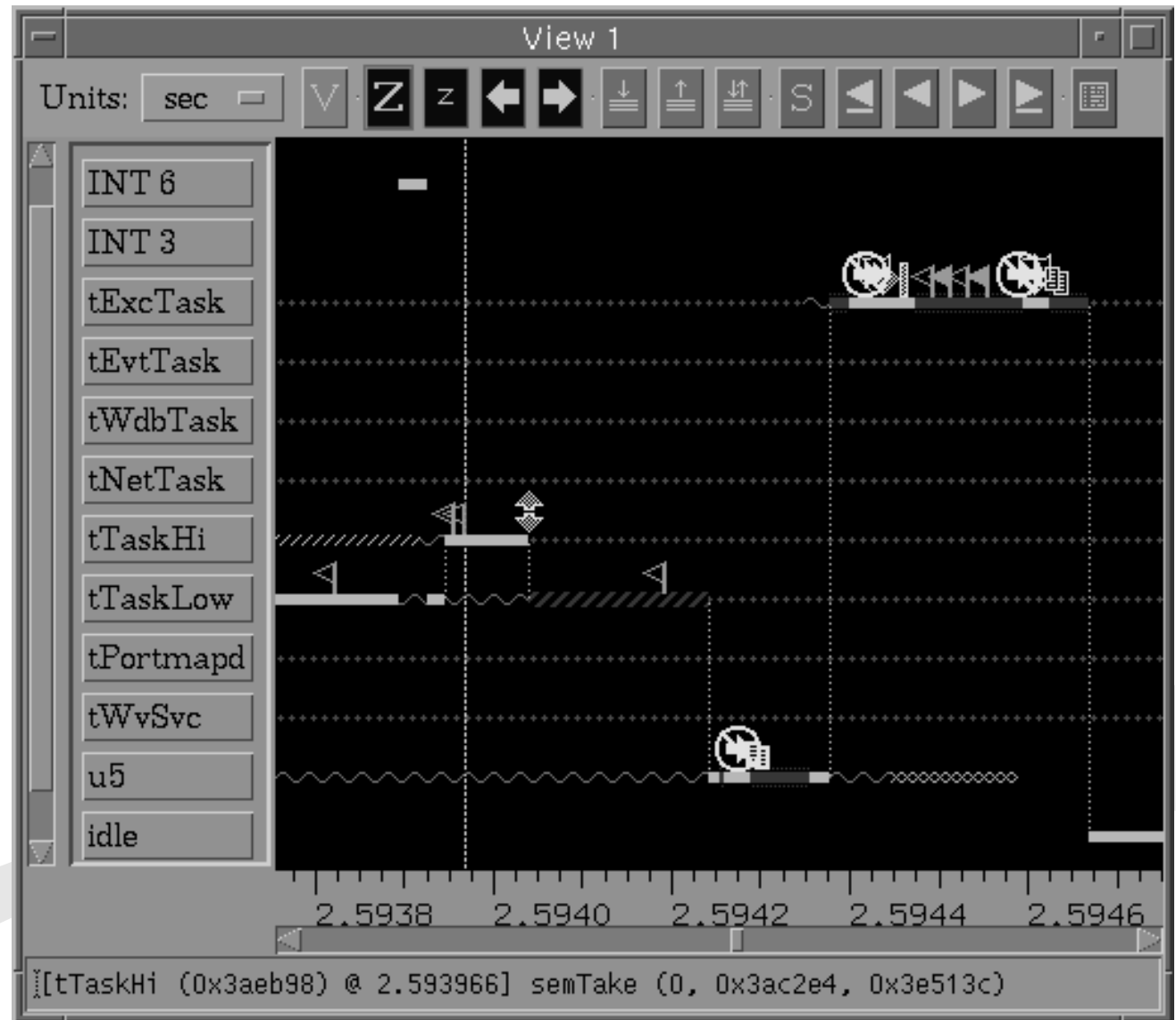
- VxSim

- ñ Lets you start VxWorks development before hardware is ready.

# WindView

WindView instruments the VxWorks kernel to record information on system (or user) events as they occur. If a high resolution timer ( $\approx 1\mu\text{s}$ ) is available, events are assigned a timestamp.

This WindView graph illustrates a deadlock between *tTaskHi* and *tTaskLow*, and also shows task *u5* exiting (with help from the task *tExcTask*).



# WindView Event Logging

---

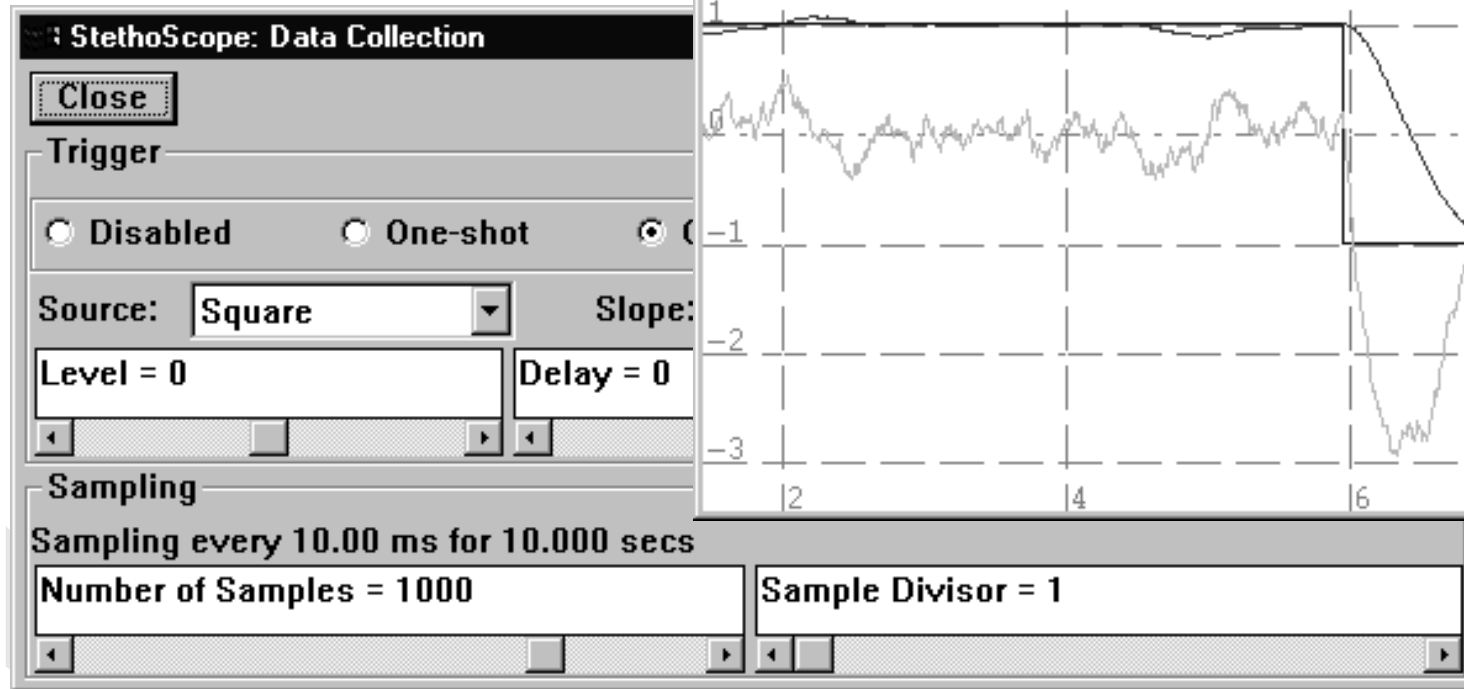
- There are three levels of event logging:
  - ñ 1. Log only context switches.
  - ñ 2. Log task state changes also.
  - ñ 3. Log instrumented object events also.
- Higher logging levels provide more information, but impose more overhead.
- Object instrumentation is flexible:
  - ñ Log information on specific objects, all objects of a class (e.g. all semaphores), or objects created during a particular time interval.
- Log user events by calling `wvEvent( )`.

# StethoScope

• Graphically monitor system or application variables.

• Collect signal time-histories flexibly and with low overhead.

• Use utilities for execution profiling and memory leak detection.





# StethoScope

---

- Target module *ScopeProbe* collects time histories of variables (*signals*) and uploads them to the host.
  - ñ Synchronous or asynchronous collection supported.
  - ñ Triggering on a signal by value/slope with +/- delay.
  - ñ Collection buffer uploaded to host by low priority task *tScopeLink*.
- StethoScope host tools display histories graphically or numerically for analysis. Also print, save to file, or export buffers to analysis tools such as MATLAB.
- Control the data collection process from the host tools. Specify at run time which signals to collect or display.

# VxSim

---

- VxSim is a port of VxWorks to run as a Unix process.
  - ñ True VxWorks preemptive multitasking implemented.
  - ñ No emulation of instructions; code compiled for host's own architecture.
- Allows development using Tornado to proceed before actual hardware is available.
  - ñ Standard VxWorks facilities (Networking, I/O system, file systems, etc.) are available.
  - ñ Tornado tools and WindView all work with simulated target.
- Most appropriate for portions of application not closely tied to the hardware.

# Tornado Quick-Start Workshop

**Help and Documentation**

# Documentation Resources

---

- Tornado provides documentation on
  - ñ Product installation
  - ñ Customer Support
  - ñ VxWorks Programming
  - ñ Using Tornado
  - ñ GNU compiler toolkit & debugger
  - ñ Tornado API (WTX & WDB protocols, etc.)
- Documentation is provided in hard copy and online.
- Online documentation provided as hypertext and *man* pages.

# Primary References

---

- Tornado User's Guide (WindSurf)
  - ñ Booting; using Tornado tools; cross-development.
- VxWorks Programmer's Guide (WindSurf)
  - ñ Programming under VxWorks, including optional products.
- VxWorks Reference Manual (Online)
  - ñ Detailed reference on VxWorks libraries and functions.
- GNU Toolkit User's Guide
  - ñ GNU compiler, assembler, linker, make, binary utilities...
- Debugging with GDB
  - ñ GNU documentation on the debugger underlying CrossWind.

# Additional References

---

- Wind River Products Installation Guide, (WindSurf)  
Tornado 1.0.1
  - ñ Installing Tornado; license manager configuration.
- Customer Support Userís Guide (WindSurf)
  - ñ Procedures and recommendations for making effective use of Wind River Customer Support.
- Tornado API Guide (Mostly online)
  - ñ WTX / WDB Protocols; customizing or writing Tornado Tools; writing a custom Target Server back-end.
- WindView for Tornado Userís Guide
  - ñ Using the WindView optional product.

# Customer Support

---

- Requires a maintenance contract.
- Provides help on:
  - ñ Installation problems.
  - ñ WRS software, documentation, and service errors.
  - ñ Understanding WRS product features & functionality.
- WindSurf web pages provide:
  1. Interactive search engine.
  2. Known problems list.
  3. FAQ
  4. Patches.
  5. Document updates.
- Designate a primary and secondary technical contact at your site to interact with WRS Customer Support.

# Wind River Users Group

---

- An active group for users of VxWorks and other Wind River products.
  - ñ News group: `comp.os.vxworks`
  - ñ Email exploder
  - ñ Software archive
  - ñ Meetings
- Email to *[inquiries@wrs.com](mailto:inquiries@wrs.com)* for information on becoming a member.
- See also the WRS web page, <http://www.wrs.com>



# Wind River Training

---

- Want more information? Wind River Systems Customer Training provides the following courses:
  - ñ Tornado Training Workshop (VxWorks)
    - More in-depth information on Tornado Tools and VxWorks. Recommended solutions to common real-time programming problems. Hands-on laboratory practice applying techniques learned in lectures.
  - ñ Device Driver Workshop (VxWorks)
    - VME bus access and interrupt handling. Writing standard and non-standard VxWorks device drivers. VxWorks I/O system internals. Extensive laboratory practice.
  - ñ Tornado BSP Porting Workshop (VxWorks)
    - Porting VxWorks to custom target hardware.