An Object Oriented Operating System

Amit Gupta
amit.gupta@colorado.edu

CSCI 5448
Spring 2011
Introduction

✧ We’re taking a closer look at HAiku
✧ A free and open source operating system
✧ Object Oriented design
✧ Implemented in C++
✧ We’ll look at selected classes
✧ ... and how they are used
History

✧ Inspired by BeOS
  ✧ A closed source, object oriented OS developed by Be Inc.
  ✧ Started in the early 90’s. Targeted at desktop users specifically

✧ Envisioned as a multimedia platform
  ✧ To compete with OS-X and Windows
  ✧ Failed to capture market share
  ✧ But did enjoy a small, loyal following

✧ Be Inc. acquired by competitor Palm Inc.
   (BeOS development ceased as a result of it. Last release was R5, named “Dano” in 2001)
History (2)

✲ Haiku started in 2001

✲ Main Goal: Pick up where BeOS left off

✲ Not a clone project \textit{(But a full reimplementation)}

✲ Implemented largely in C++
  ✷ Parts of the kernel written in C with C++ wrappers

✲ Open Sourced under MIT license

✲ Still an active project today …
Overview

✧ Divided into subsystems (called “Kits”)
✧ Provides source code & binary backward compatibility with BeOS
  ✧ Apps written for BeOS can run on HAIKU
  ✧ A clever design choice: *Haiku comes into being with a wide range of apps ready*
✧ Modular design of BeOS
  ✧ Allowed “Kits” to be developed in relative isolation
  ✧ Written as replacements to the BeOS subsystem prior to the barebones Haiku being up and running
Overview (2)

✧ Optimized for desktop responsiveness
✧ Hybrid Kernel design
  ✧ Somewhat in between a monolithic & microkernel
✧ Kits: Cleanly divided Modular Subsystems
  ✧ Application
  ✧ Device
  ✧ Kernel
  ✧ Networking
  ✧ Media
  ✧ Printing
  ✧ ... many others
Some Methodologies seen

✧ One way to tackle each problem
 ✧ Easier to test and debug
 ✧ i.e One Desktop, One GUI
    ✧ Contrast with Linux, where there are different desktops bundles available, each with its own unique set of bugs.

✧ Rich APIs built into the OS (rather than as libraries)
 ✧ Avoids excessive lib dependencies
    ✧ Libraries are still supported, so it's not all built-in

✧ Both of the above promote code reuse

✧ Modular and extensible design
  ✧ Examples we’ll see: Replicants, System wide Messaging service
What does it look like?...

Similar to a GNU Desktop System
What does it look like?

Similar to a GNU Desktop System
The Application Kit

✧ Starting point for all applications
✧ Collection of classes that an application uses to make itself an identifiable entity
  ✧ Implies it can communicate and interact with the rest of the apps and the system
✧ All except the very simplest of apps will make use of classes contained in this kit
✧ Classes related to messaging (like IPC), scripting support & application monitoring
The Application Kit

A closer look at its classes

✧ BApplication
  ✧ Connects to the App Server (daemon), runs the applications main message loop

✧ Messaging Kit
  ✧ Provides classes that implement the messaging service threads can use to talk to each other
  ✧ Service used for both, inter and intra process thread communication
  ✧ Also used by System to send messages to applications

✧ Scripting Kit
  ✧ This helps provide scripting support to any application
  ✧ i.e. Allows the application written to be controlled by commands issued by other applications
The Application Kit

A closer look at its classes...

✧ BRoster
  ✧ Provides Application/Task monitoring services
  ✧ Can help identify applications, launch them, setup communications with them

✧ BClipboard
  ✧ Provides an interface to clipboard (where “cut” and “paste” items are stored)

✧ BCursor
  ✧ Represents a cursor in an application
  ✧ Other classes lets you assign cursors to all or part of your application
The Application Kit

**Messaging Kit Classes**

- **BLooper**
  - Receives/sends and processes messages between applications
  - Does this in a separate thread
  - Passes received message to a BHandler

- **BHandler**
  - Handles messages passed on by an apps BLooper
  - BLooper is actually a subclass of Handler
  - Can also be used as a state machine to keep track of state of the application

- **BMessage**
  - A message container/unit of communication
  - Sent between process/threads using the messaging system
The Application Kit

Messaging Kit Classes...

✧ BMessageFilter
  ✧ Can describe BMessage properties that an incoming message should have to be processed by the BLooper/BHandler
  ✧ Is also extensible by using its `filter_hook` method
  ✧ Filtering based on 2 other criteria besides `filter_hook`
    ✧ Its `what` attribute
    ✧ Its `message_source` attribute

✧ BMessageQueue
  ✧ Used by applications, specifically their BLooper class, to maintain a queue of messages to be processed
  ✧ Also has methods for locking/unlocking the message queue besides the typical queue operations
  ✧ Also has a method to search the message queue
The Application Kit

Class Inheritances

- The BHandler and the Blooper are in fact subclasses of BArchivable
- BArchivable class is what provides an interface for putting and extracting objects into message archives
- This provides the core of the functionality of sending objects between apps
- Remember that every application typically has a BLooper (which sends/receives messages) and several BHandler’s (which processes messages).
- Here we see that a BLooper in fact inherits from BHandler
  - Implies it can itself serve as a message handler if no specific handlers are required by the app OR
  - An app could chain together several handlers for its messages instead
  - This clever use of inheritance in their design lets the application writer leverage code reuse and modularity
The Application Kit

What a simple application looks like

- We see here a part of a simple “Hello World” application
- It includes a BApplication class (HelloApplication being a derivation of BApplication)
- Calling the Run() method of a BApplication object starts processing of any messages it receives
- This simple application doesn’t leverage the messaging system as it doesn’t need to talk to another app.
- It does give us an idea of how application code looks on Haiku (by leveraging system classes to access system services)
- Link to more example code in the references section

```c
#ifndef HELLO_WINDOW_H
#include "HelloWindow.h"
#endif

#ifndef HELLO_WORLD_H
#include "HelloWorld.h"
#endif

int main(int, char**)
{
    HelloApplication myApplication;
    myApplication.Run();
    return(0);
}

HelloApplication::HelloApplication() :
    BApplication("application/x-vnd.Be-HelloWorldSample")
{
    HelloWindow *aWindow;
    BRect aRect;

    // set up a rectangle and instantiate a new window
    aRect.Set(100, 80, 260, 120);
    aWindow = new HelloWindow(aRect);

    // make window visible
    aWindow->Show();
}
```
Scripting

✧ Every application (*BApplication*) can define its own scripted commands

✧ A set of these commands forms the applications interface to other applications who may wish to “script” the application

✧ Script commands are sent to an application using the infrastructure provided by the Messaging Kit

✧ Centers around the notion of
  ✧ Commands: *What action to perform?*
  ✧ Property: *What the command should act on within the app?*
  ✧ Specifiers: *Used to target (specify) which instance of Property has to be targeted*

✧ Mechanism can be leveraged by apps written in any supported language. *Not easy to do on other Operating Systems.*
The Support Kit

✧ Provides a useful set of classes for the rest of the system API to use (*All other Kits and applications*)

✧ Can be considered as the “base” of what supports the rest of the system

✧ Provides classes towards the following functionality
  - Thread Safety
  - Thread Local Storage
  - Archiving and I/O
  - Memory Accounting and Management
  - Common Datatypes and Objects
  - Most of the error codes for the system
The Support Kit
A closer look at its classes

✧ BPositionIO
✧ Actually inherits from BDataIO: Models a generic IO Stream
✧ Provides advanced read, write and seek on top of BDataIO
✧ It’s the primary operand for most of the functionality some of the other kits (ex. Translation kit which we’ll see next)

✧ BFile
✧ This represents a file, actually inherits from BPositionIO
✧ This makes the kit naturally extensible for dealing with files as sources and sinks of data

✧ BMemoryIO
✧ Inherits from BPositionIO
✧ Provides I/O functionality on existing memory

✧ BMallocIO
✧ Also inherits from BPositionIO
✧ Used to allocate a memory chunk based on a block size
✧ Provides the position I/O interface to newly assigned memory

✧ BBufferIO
✧ Provides a buffer “adapter” to a stream
✧ Does NOT provide an entity to read/written to but a “front-end” to a IO Stream
The Support Kit
I/O Classes: Inheritances

✧ We see the use of inheritance in providing different behaviors
✧ The generic functionality is provided by BDataIO
✧ A more sophisticated mechanism is provided by BPositionIO
✧ Further derivations are used for similar objects that have slightly different semantics
✧ Care taken not to have a deep inheritance hierarchy
The Support Kit
Some more classes...

✧ **BLocker**
✧ Provides a semaphore like mechanism for locking
✧ Allows *recursive locks*
 ✧ i.e A function that calls a lock() on a semaphore, calls another function that also calls lock() on the same semaphore
 ✧ Possible because this object *keeps track* of number of locks from the same thread
✧ Has a *faster* mechanism than conventional semaphores
 ✧ Has checks to determine if acquiring lock is necessary

✧ **BAutolock**
✧ Makes the use of a lock easier and less prone to programmatic errors
 ✧ Provides a convenient mechanism to protect sections of code
✧ Used in combination with a BLocker object to *acquire lock*
✧ Created & used locally within a function: *created on the stack*
✧ Therefore destroyed when leaving the function
✧ As part of the object destruction, *lock gets released*
The Support Kit

Some more classes...

✧ BBlocKCache
  ✧ Creates and maintains a pool of memory blocks
  ✧ Can be used in performance critical app sections
    ✧ Memory allocation/release are expensive operations
    ✧ Instead have many blocks at our disposal and release them all later

✧ BStopWatch
  ✧ A timer class, provides methods to time events
  ✧ Designed to behave like a physical stop watch
  ✧ Also useful in debugging code, like an inexpensive profiler

✧ Data Structures
  ✧ Bstring
    ✧ Class with extensive support for string operations
  ✧ Blist
    ✧ Generic ordered container to hold any kind of object type
    ✧ Provides common access, modification, comparison methods
    ✧ Grows and shrinks automatically depending on contents
    ✧ No Memory Management Required by the app
  ✧ ... Several others including basic data types and constants
The Translation Kit

✧ The Translation Kit provides a framework for converting data streams between different formats

✧ Picture a word processor application importing and exporting documents in a variety of formats (HTML, ASCII, PostScript)

✧ Provides classes for applications to leverage this functionality (and also extend it with add-ons)

✧ Abstraction at the design level: Abstracts out format details for the application writer

✧ Inherently provides good opportunities for code reuse
  ✧ Example: A file format converter and file viewer can leverage the same code
The Translation Kit
A closer look at its classes

✧ BTranslator
  ✧ These is the superclass to extensible “add-ons” referred to earlier
  ✧ Derived from this are the classes that does the conversions
  ✧ Can be native or provided by a 3rd party

✧ BTranslatorRoster
  ✧ Provides apps dependency resolution of translation modules (add-ons)
  ✧ Services Provided
    ✧ Provides Initialization (ex: load this translator)
    ✧ Information (ex: which translators are loaded)
    ✧ Translation (choosing an appropriate translator and performing the translation)
    ✧ Configurations services (changing behavior of a translator)

✧ BTranslationUtils
  ✧ Contains functions to load bitmap images from an IO Stream/File
Interesting Features

Replicants

✧ Stepping back in granularity from classes
✧ Lets take a look at an interesting features in the Haiku resulting from it being Object Oriented
✧ Replicants are essentially an application instance that is launched within another application
✧ Its trick lies in the BView UI class *(from the Layout Kit)*
  ✧ *BView represents the rectangular area of a Window*
✧ It has a function that allows it to Archive itself
  ✧ Archive is an object that can be exchanged with other processes/threads using the messaging infrastructure
✧ It also has a link to a BDragger class *(used as a handle to move the window)*
Interesting Features

Replicants...

- BShelf is a second class that enables replicants
- It’s a specialized BView that can receive dragged and dropped BView objects (*Implemented as specialized messages*)
- Application is launched by one app calling the instantiate method of the archive of the other
- Helps integrating apps with one another
- Shows the power object oriented design for GUIs
Interesting Features

File System (OpenBFS)

✧ The journaling filesystem used to support Haiku
✧ Started as remnants of the BeOS File System (BFS)
✧ Has had several improvements since
✧ Upto x10 times faster than the original BFS
  ✧ Uses a file cache in addition to a block cache
✧ Entirely written in C++
✧ OpenBFS developers claim it helped their development process
  ✧ It resulted in cleaner code which was easier to understand
  ✧ Easier to maintain the codebase
  ✧ Made it easier for them to implement it a module at a time
  ✧ It was also clear to them at the very start as to how they would integrate
    each developmental step into the project
Interesting Features

File System (OpenBFS)...

✧ The most unique feature of this file system is that it has a Transaction API: Database like file system
  ✧ Supports queries run on the file system similar to a database, in addition to the typical file operations
  ✧ You can also group multiple queries together

✧ Has an internal indexing mechanism on the metadata that enables very fast queries *(very fast file searches)*

✧ Allows users to extend attributes of any file *(like “artist” for an mp3 file)*

✧ Allows users to leverage indexing on any attribute

✧ All of this combined makes for very powerful search/query semantics

✧ User is able to do all this without having to write code. *Impressive!*
HAiku
Where it's headed

✧ It's far from complete. Several things planned for the near future.
✧ System Calls are much slower when compared to Linux/Unix
✧ Wide range of devices still not supported: Drivers
✧ Support for other file systems is still lacking
✧ Bugs exist in several kits.
   ✧ System and Applications crashes are still being resolved.
✧ More Apps need to be ported to HAiku
✧ Size of the effort is not yet at the scale of Linux Kernel or comparable projects.
   ✧ Some of the developers just haven’t gotten around to these issues
Interested in learning more? Head over to...

-

- Overview
  - http://en.wikipedia.org/wiki/BeOS

- Project Website: http://haiku-os.org/
  - Documentation section
  - Articles section
  - Plenty of other blogs (ex. *How to run Haiku on a VM and so on*)

- BeOS sample code:
  - Has code examples from every Kit (subsystem)

- BeBook:
  - Excellent documentation of BeOS, most of which is still relevant to Haiku

- Lastly the HAIKU mailing list would be the best place to ask for anything you can’t find in the above resources
Questions?

Thank You