computer science house

# ShagOS Internals

Past, Present, and Future


Frank Barrus
shaggy@csh.rit.edu
Computer Science House


computer science house

## Seminar Overview *ShagOS Internals*

- Overview and History
- ShagOS Classic (0.01-0.38)
  - General Architecture
  - Object Structure
  - Device Drivers
  - Demos
- Current ShagOS (0.54)
  - Object Paradigm and Implementation
  - General Architecture
  - System Features
- Future of ShagOS (1.00+)



computer science house

## Audience Survey

- O.S. Experience?
- O.O. Experience?
- Systems Programming?
- What is ShagOS?
  - What have you heard about it?
  - What makes it unique?



computer science house

## What Makes ShagOS Unique?

- Object Oriented
  JavaOS, Chorus, etc...
- MicroKernel
  QNX, Mach, etc...
- Modular
  Mach, JavaOS, Linux (to some extent), etc...
- Posix compatibility
  nearly every one...
- Can run distributed objects
  Plan9, JavaOS, etc...
- ‘Hot-Swappable’ Device Drivers
  Linux, etc...

## ShagOS Unique? *(continued)*

- Object Forwarders
  - some other OO systems have these...
- Completeness
  - Can thoroughly inspect all objects, including their types/classes, attributes, and methods, within security limitations.
- Small
  - Plan9, Inferno, QNX, etc...



computer science house

## ShagOS Unique? *(continued)*

- The Name
  - The name ShagOS (previously Shag/OS) evolved out of ShagTerm, a terminal emulator written for the Kodak KIMS Workstations at CSH. (try ‘S 20077002’ if you ever come across one -- it’s in the EPROMs of two of them). ShagTerm was derived from my CSH nickname, Shaggy, but I’ve recently been informed that ‘Shag’ as a verb has a different slang meaning in England. Oh well.



computer science house

## What Makes ShagOS Unique?

- Dedication to Ideals
  - Core Set of Guiding Principles
    - The ShagOS Philosophy
  - No ‘Time to Market’ Pressures
    - May be some short-term deadlines, such as this seminar, and the parts of ShagOS that comprise my Thesis, but for the most part sound design, according to the guiding principles, takes precendence over short-term hurried designs.
    - -- of course, this also means that we may all be waiting a very long time.



computer science house

## Guiding Principles

- Efficient
  - Best Balance of Small/Fast
    - No huge sacrifices in one category for small gains in another
  - ‘One Instance’ Rule
- Flexible
- Modular

## Guiding Principles *Small*

- Memory is NOT Cheap (OS vs. Apps)
  - If the OS wastes memory it impacts all applications that run under it.
- An OS Should Not Consume 8MB
  - Memory upgrades should be required for large applications, not for the operating system.
- Reclaim Unused Storage
  - Free Initialization Code
  - Nearly All Kernel Code Swapable



computer science house

## Origins *Summer, 1992*

- Culmination of Multiple Projects
  - JakeDev (1991)
  - Multitasking additions to ShagTerm
    - on Kodak KIMS workstations (uVAX) -- included 'SWindows'
  - Object-Oriented Environment Experiments
- Independent Study for Ken Reek at RIT -- Fall 1992
- Demonstrated in early 1993
- Lost final version



computer science house

## Shag/OS v0.34

- Object-Oriented MicroKernel-based Operating System
- Written in C, with custom OO extensions
- Small
  - Microkernel: 43K
  - Drivers: 13K
    - init, hard-drive, virtual console terminal, serial, keyboard
  - 17,000 lines of source
    - including all utilities, drivers, shell, demos, etc...



computer science house

## Rewrite *Fall, 1995*

- Independent Study for Andrew Kitchen at RIT
- Complete redesign and rewrite -- Now in C++
  - Object System
  - Language Support
  - VM System
  - Module Handling

# Thesis

*Spring, 1996*

- **Dynamic Encapsulation of C++ Objects for Object Oriented Distributed Systems**
  - Dynamic Object Modelling System
  - Core Component of ShagOS
  - DECO

    Dynamic Encapsulator of C++ Objects
    Dynamic Extensions to C++ Objects



computer science house

# Rewrite

*Still going...*

- **Never Quite Finished...**
  - Old ideas always seemed inadequate
  - Some parts rewritten many times
  - Constantly being more unified

    Adding code to fix problems tends to make programs bloated. Instead, ShagOS just keeps getting parts of it rewritten, thus the end product keeps getting smaller, but more functional.



computer science house

# Some Features

*0.34*

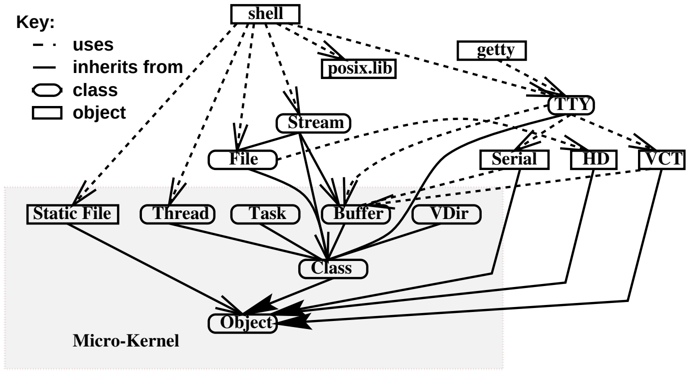
- **Object-oriented**
- **Only 3 System Calls**
  - Label
  - ObjString
  - ObjCall
- **Can change HZ Rate at any time**



computer science house

# System Organization

*Shag/OS 0.34*



## Basic System Entities

- Objects
  - Classes
  - Tasks
  - Threads
  - Directories
  - Buffers
  - System



computer science house

## Objects

- Basic Unit of Organization and Communication
- Attributes

  the features of an object that you can examine or change the value of
- Methods

  the operations you can perform on an object
- Class

  defines the implementation of an object



computer science house

## Everything is an Object

- Classes
  - Meta-Objects
  - Define implementation of an object
- Types
  - Abstract Meta-Objects
  - Abstract Classes
- Define the interface to an object



computer science house

## Object *Methods*

- init

  initialize an object
- new

  allocate a new object of the same type
- free

  free the storage for the given object
- next

  get the OID of the superobject
- eval

  returns the intrinsic value of an object, which is, by default, the OID. Some objects may override this to execute a function.

## Class

*Methods*

- **init**
- **new**
- **class**
  get the class OID of an object



computer science house

## Task

- Largest Unit of Memory Organization
- Kernel Task always mapped
- One other (current) Task is mapped
- Threads can wander between Tasks
- Multiple programs can be loaded into one Task



computer science house

## Task

*Methods*

- **init/free**
  create or remove a virtual address space
- **mem_alloc/mem_free**
  allocate and free from the task's heap
- **vm_map**
  map a region of physical memory
- **vm_alloc**
  allocate a block of memory (actual allocation is demand-paged)
- **vm_reserve**
  reserve a region of memory (usually used to catch illegal memory references)
- **exec**
  load, link, and execute an a.out-format binary



computer science house

## Threads

- Basic Unit of Execution
- CPU Register Context
- Stack
- Current Task

## Thread *Methods*

- init
- free
- get
  - get attributes
- msleep
  - millisecond sleep
- signal
  - signal an event
- wait
  - wait for an event
- suspend
  - temporary sleep (not removed from run-queue)





## Thread *Attributes*

- parent
- name
- cwd, root
- quantum, max_quantum
- rtime, stime
- stdin, stdout, stderr
- w_event, w_param
- data, ltime, itime, dvmem, dpmem, dsmem, tvmem, tpmem, tsmem, svmem, spmem, ssmem





## Directories

- Store Associations of Keys and Values
- Any Object can be a Key
  - Strings, numbers, labels, classes, etc...
- Can Associate any Value
  - Files, numbers, addresses, classes, etc...
- Form a Hierarchy of Objects





## Dir *Methods*

- init
- get
  - get the value associated with an entry
- set
  - set the value associate with an entry
- create
  - add a new entry, and give it a value
- remove
  - delete an entry
- getattr
  - get an attribute key by enumeration

# Buffer

*Methods*

- init
- free
- write

  put data into a buffer
- read

  get data out of a buffer





# Buffer

*Attributes*

- rnoblock
- wnoblock
- ctrl
- param
- maxsize
- size





# Sys

*Methods*

- init
- write

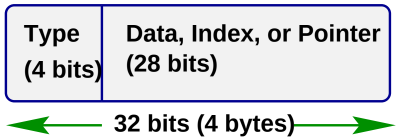
  writes to the system console
- halt
- reboot
- nuke_fs

  delete the static filesystem, and reclaim the memory
- vm_status, vm_list, zone_status, mem_status, mem_list, sys_stat, load_average





# Simple Object Descriptor




- **16 Possible Types (10 supported)**

  for direct short form. Indirectly, any number of 'types' can be supported.
- **28 Bits of Data**

  can be used to store immediate data, in an "inline" format, or for a pointer to access external data that is either too large to fit, or passed by reference.

# Inline Descriptor Types

| Type (4 bits) | Data, Index, or Pointer (28 bits) |
|---|---|

← 32 bits (4 bytes) →

- INT (0), NINT (15), CHAR (1), SSTR (2)
  
  All of these store the data inline. Integers can range from -268435456 to 268435455 (29 bits), and short strings can have up to four 7-bit characters.





# Object Descriptor Type

| Type (4 bits) | Data, Index, or Pointer (28 bits) |
|---|---|

← 32 bits (4 bytes) →

- OBJ (8)
  
  Index into the per-Task object table. From the object table, a more complex object descriptor is used to get the class of the object and its data pointer and size.





# Pointer Descriptor Types

| Type (4 bits) | Data, Index, or Pointer (28 bits) |
|---|---|

← 32 bits (4 bytes) →

- 16 Possible Types (10 supported)
  - TPTR (9), SPTR (10), KPTR (11)
    
    Pointer into task, stack, or kernel space (28 bits) -- this limits the addressable size per task or stack to 256MB.
  - LABEL (13)
    
    System-wide label (unique ID), used for method selectors, etc.





# Error Descriptor Type

| Type (4 bits) | Data, Index, or Pointer (28 bits) |
|---|---|

← 32 bits (4 bytes) →

- ERR (14)
  
  Unique error codes are predefined in the implementation, but a dynamic error hierarchy was being developed. Any object call can return an error object instead of a valid one. Calls that don't check for an error will cause a thread to abort if one occurs.

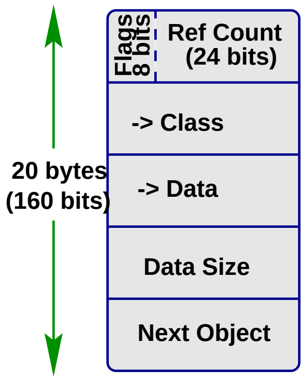
## Special Objects

- OBJ_NULL
- OBJ_TRUE, OBJ_FALSE
- OBJ_SYS
  - System level functions (reboot, etc)
- OBJ_KERNEL, OBJ_TASK, OBJ_THREAD
  - Kernel task, current task, and current thread
- OBJ_ROOT, OBJ_CWD
  - Quick directory access
- OBJ_STDIN, OBJ_STDOUT, OBJ_STDERR
  - Quick standard I/O access





## Object Descriptor *complex*




**Flag Bits:**
**0: HASVEC**
**1: VECTOR**
**2: FORWARDER**
**3-4: <RESERVED>**
**5: NOREAD**
**6: NOWRITE**
**7: KERNEL**

**"Next Object" link is an object ID, not a simple pointer**





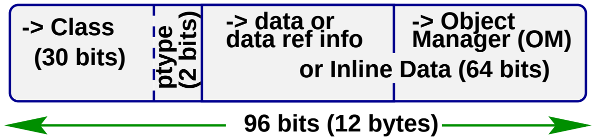
## Some Weaknesses of old Shag/OS

- No persistent storage
- Not easily distributable
- Label database keeps growing
- Cannot look up methods
- Kernel invocation for every method
- Tasks maintain objects database
- Security hole in method call





## Object Descriptor *Class*




- Class

# Object Descriptor *PtrType*

- Pointer Type (2 bits)
  - 0= Inline
    Up to 64 bits of data are contained directly in the descriptor.
  - 1= DataPtr
    The data is pointed to, and an Object Manager is also provided.
  - 2= Missing
    The data is not present, but some reference to it is here instead, to be used by the Object Manager.
  - 3= Reserved
    For future use.





# Object Descriptor *DataPtr (1)*

| -> Class (30 bits) | ptype (2 bits) | -> data or data ref info | -> Object Manager (OM) |
|---|---|---|---|
| | | or Inline Data (64 bits) | |

96 bits (12 bytes)

- Direct pointer to anywhere
  - Could be on the stack
  - Could be dynamically allocated
  - Could be static
  - Not necessarily created by OO system
    A descriptor can be used to turn any data into an object, without duplicating the data.





# Object Descriptor *Inline (0)*

| -> Class (30 bits) | ptype (2 bits) | -> data or data ref info | -> Object Manager (OM) |
|---|---|---|---|
| | | or Inline Data (64 bits) | |

96 bits (12 bytes)

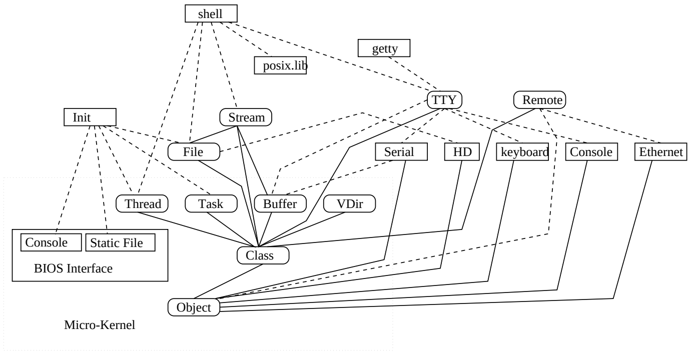
- 64 bits (8 bytes) of Inline storage
  - Sufficient for most simple objects
  - Int, Char, and most Floats will fit
  - String, Array Descriptors
  - Any 2 pointer or Ptr/Len tuple
  - Short Strings (up to 12 characters)





# System Organization

an early proposed system object layout for 1.00



## Current Architecture





## New Features





## DECO

- Automatic Creation of Types/Classes
- Reuse Existing C++ Code
- Minimal Number of New Keywords





## DECO

*Class Syntax*

```
dynamic class class_int {
type_num:
    obj pack(obj);
    obj operator== ();
};
```

## DECO

*Type Syntax*

```
dynamic virtual class type_num : type_obj {
    obj operator == ();
};
```





## DECO

*shortcuts*

**Or, for those who prefer a shorter form:**

```
#define dtype dynamic virtual class
#define dclass   dynamic class

dtype t_window {
    ...
};

dclass c_Window {
    ...
t_window:
    ...
};
```





## References

- **Shag/OS 0.38 sources**
  **http://www.csh.rit.edu/~shaggy/software.html**
- **ShagOS 0.54 sources**
  **(available Spring '97?)**
- **Dynamic Encapsulation of C++ Objects for Object Oriented Distributed Systems**
  **(my Masters Thesis)**
- **The Design and Implementation of ShagOS**
  **(to be published in 1999?)**





## Misc Info

**These slides were produced entirely with free software, including 'groff', which is available from any GNU software archive (such as `ftp://prep.ai.mit.edu`), as well as many other places, and 'msl', my slide macro package for groff, which I will be making available, along with the slides, on my webpage, at `http://www.csh.rit.edu/~shaggy`**