Open Portability Abstraction Layer (OPAL)

Brian Barrett

Support Library

- Utilities for making your life easier
- Utilities for portably interacting with the Operating System
  - Memory management issues on Wednesday
- C-based object management system
- Rich set of container classes
  - Lists
  - Free Lists
  - Hash Tables

Initialization

- opal_init to initialize library
  - Few functions can be used before opal_init
  - Completely local operation - no communication required
- opal_finalize to free library resources
  - Most functionality unavailable after call

Utility Code

- Actual, real documentation!
- opal/util/*.[h,c]
- Lots of compatibility code
  - asprintf, qsort, basename, strncpy
- Useful “add-on” code
  - Get listing of all network devices (if.h)
  - Manipulate argv arrays (argv.h)
  - printf debugging code (output.h)
  - Error reporting (show_help.h)

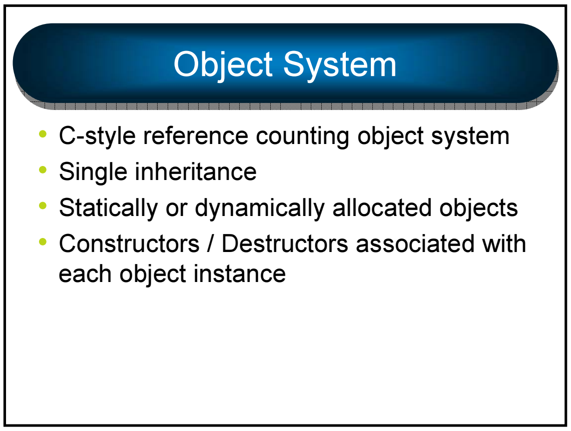
opal_output Debugging Code

- Function to emit debugging / error messages to stderr, stdout, file, syslog, …
- Versions to simplify debugging output
- Printf-like arguments

```
opal_output(0, “hello, world”);
opal_output_verbose(0, 10, “debugging…”);
OPAL_OUTPUT(0, “--enable-debug only”);
OPAL_OUTPUT_VERBOSE(….);
```

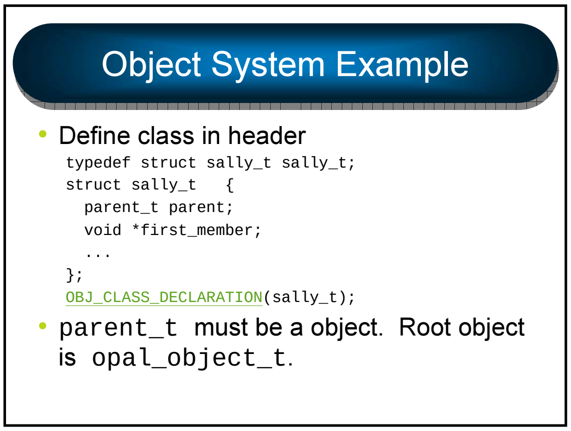
Nice Error Messages

- opal/util/opal_show_help.[h,c]
- Print detailed error messages for common user errors
- Message in text file rather than in source code
- Could (maybe) one day allow for minimal internationalization support
- Example….



Object System

- C-style reference counting object system
- Single inheritance
- Statically or dynamically allocated objects
- Constructors / Destructors associated with each object instance

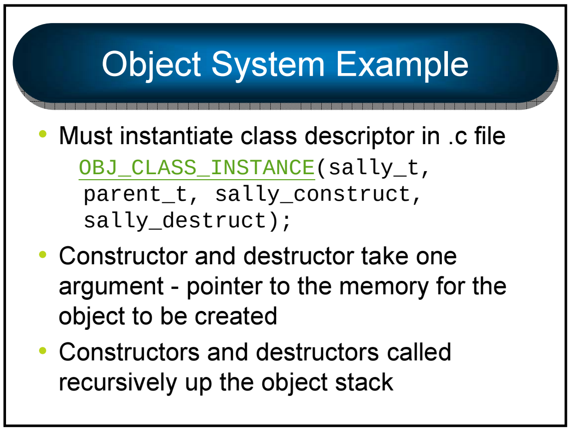


Object System Example

- Define class in header

```
typedef struct sally_t sally_t;
struct sally_t   {
  parent_t parent;
  void *first_member;
  ...
};
OBJ_CLASS_DECLARATION(sally_t);
```

- `parent_t` must be a object. Root object is `opal_object_t`.

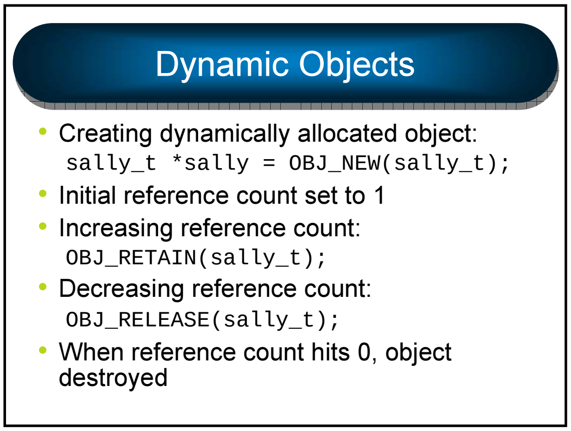


Object System Example

- Must instantiate class descriptor in .c file

```
OBJ_CLASS_INSTANCE(sally_t,
 parent_t, sally_construct,
 sally_destruct);
```

- Constructor and destructor take one argument - pointer to the memory for the object to be created
- Constructors and destructors called recursively up the object stack



Dynamic Objects

- Creating dynamically allocated object:

```
sally_t *sally = OBJ_NEW(sally_t);
```

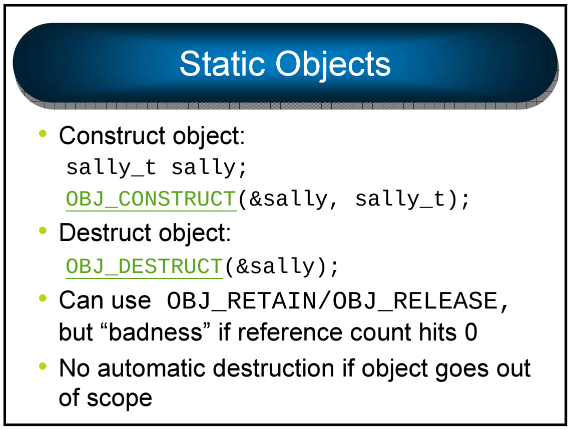
- Initial reference count set to 1
- Increasing reference count:

```
OBJ_RETAIN(sally_t);
```

- Decreasing reference count:

```
OBJ_RELEASE(sally_t);
```

- When reference count hits 0, object destroyed



Static Objects

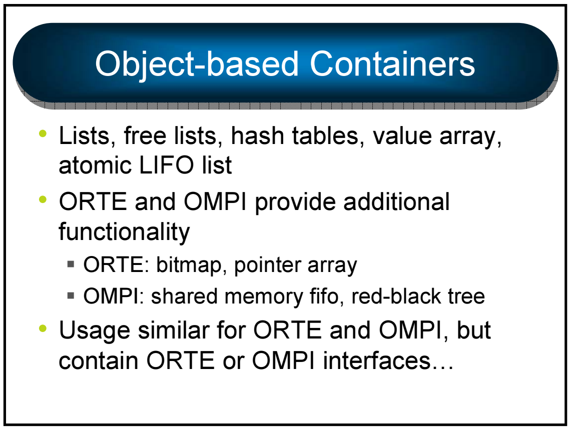
- Construct object:

```
sally_t sally;
OBJ_CONSTRUCT(&sally, sally_t);
```

- Destruct object:

```
OBJ_DESTRUCT(&sally);
```

- Can use `OBJ_RETAIN/OBJ_RELEASE`, but “badness” if reference count hits 0
- No automatic destruction if object goes out of scope



Object-based Containers

- Lists, free lists, hash tables, value array, atomic LIFO list
- ORTE and OMPI provide additional functionality
  - ORTE: bitmap, pointer array
  - OMPI: shared memory fifo, red-black tree
- Usage similar for ORTE and OMPI, but contain ORTE or OMPI interfaces…

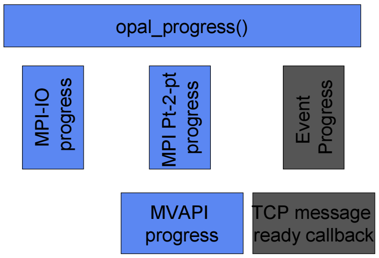
Linked List

- opal_list_t is a doubly-linked list
- Item ownership transferred
  - No copies like in STL
  - Item only belong to one list
- Pointers to items never invalidated by opal_list functions
- O(1) insert, delete, join, get size
- Splice and sort routines
- Large debugging performance impact

More objects…

- Free Lists
  - Bulk object allocator
  - objects must have parent class opal_free_list_item_t
  - Objects can always be put in linked lists
- Hash table
  - Keys either 32 or 64 bit integers (pick one at creation and stick with it)
  - WARNING: performance O(N), not O(log(N))

Progress Engine



opal_progress()

MPI-IO progress

MPI Pt-2-pt progress

Event Progress

MVAPI progress

TCP message ready callback

Progress Engine (continued)

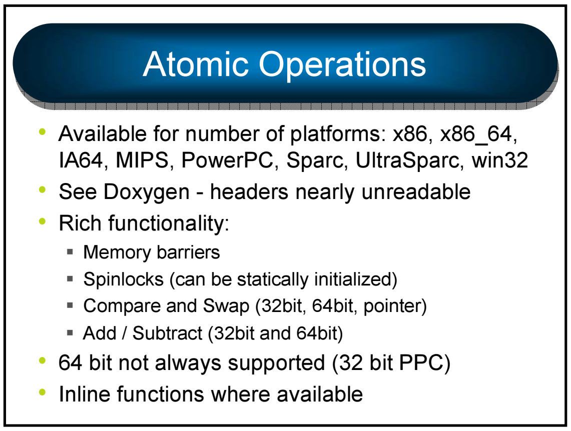
- opal_progress() triggers callbacks to registered functions
- Event library for complicated progression
  - triggers for file descriptors (like select, but with callbacks)
  - Timer callbacks
  - Signal callbacks (not in signal handler context!)
  - Event library can run in own thread

Threads

- Generic interface for PTHREADS, Solaris and Windows native
- Support for:
  - Thread manipulation
  - Mutexes
  - Condition variables
- Mutexes support either OS locks or atomic locks
  - Pick one and stick with it
- No static initializers

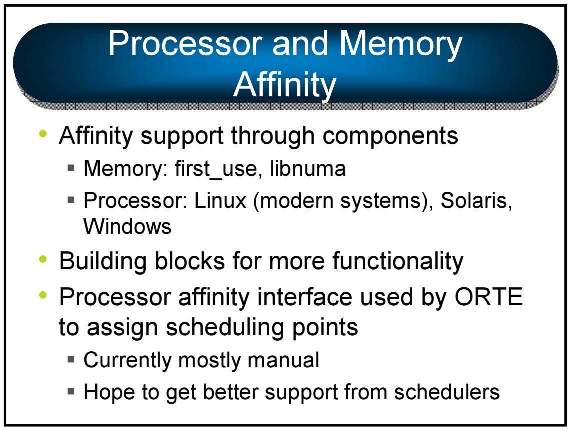
Condition Variables

- Semantics as usual for condition variables
- If progress threads enabled:
  - Call underlying system condition variable
- Otherwise:
  - Call opal_progress until signalled
- Currently, always use software implementation for Solaris or Windows threads



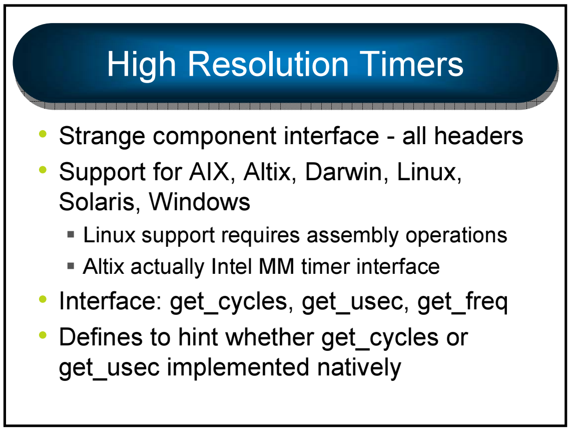
Atomic Operations

- Available for number of platforms: x86, x86_64, IA64, MIPS, PowerPC, Sparc, UltraSparc, win32
- See Doxygen - headers nearly unreadable
- Rich functionality:
  - Memory barriers
  - Spinlocks (can be statically initialized)
  - Compare and Swap (32bit, 64bit, pointer)
  - Add / Subtract (32bit and 64bit)
- 64 bit not always supported (32 bit PPC)
- Inline functions where available



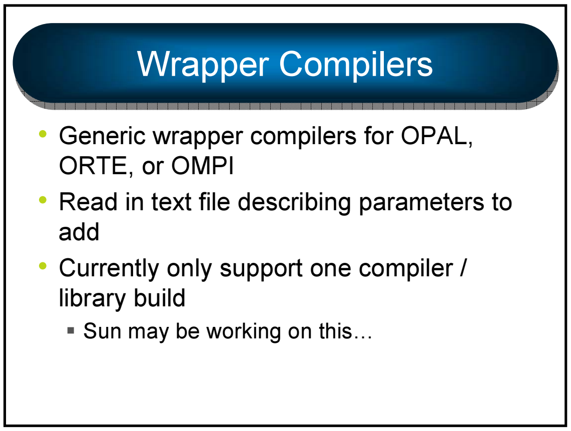
Processor and Memory Affinity

- Affinity support through components
  - Memory: first_use, libnuma
  - Processor: Linux (modern systems), Solaris, Windows
- Building blocks for more functionality
- Processor affinity interface used by ORTE to assign scheduling points
  - Currently mostly manual
  - Hope to get better support from schedulers



High Resolution Timers

- Strange component interface - all headers
- Support for AIX, Altix, Darwin, Linux, Solaris, Windows
  - Linux support requires assembly operations
  - Altix actually Intel MM timer interface
- Interface: get_cycles, get_usec, get_freq
- Defines to hint whether get_cycles or get_usec implemented natively



Wrapper Compilers

- Generic wrapper compilers for OPAL, ORTE, or OMPI
- Read in text file describing parameters to add
- Currently only support one compiler / library build
  - Sun may be working on this…