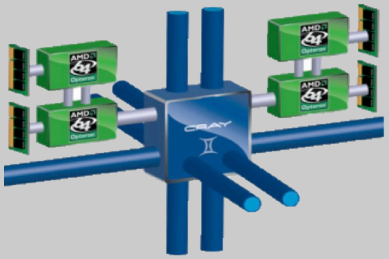
SAND2013-10537P



Kitten Lightweight Kernel Overview

December 11, 2013

Kevin Pedretti
Scalable System Software
Sandia National Laboratories
ktpedre@sandia.gov

Sandia National Laboratories

Exceptional service in the national interest

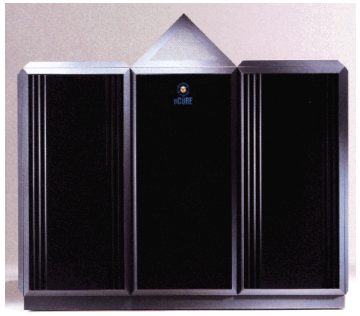
U.S. DEPARTMENT OF ENERGY
NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Sandia Massively Parallel Systems

Sandia National Laboratories

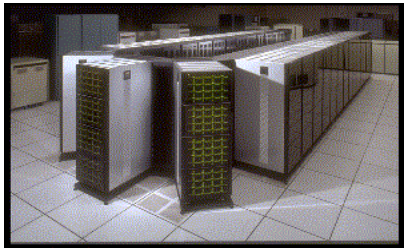
1990



nCUBE2

- Sandia's first large MPP
- Achieved Gflops performance on applications

1993



Paragon

- Tens of users
- First periods processing MPP
- World record performance
- Routine 3D simulations
- SUNMOS lightweight kernel

1997

ASCI Red

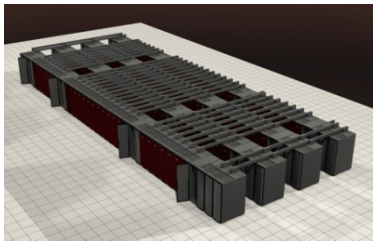
- Production MPP
- Hundreds of users
- Red & Black partitions
- Improved interconnect
- High-fidelity coupled 3-D physics
- Puma/Cougar lightweight kernel

1999

Cplant

- Commodity-based supercomputer
- Hundreds of users
- Enhanced simulation capacity
- Linux-based OS licensed for commercialization
- ~2000 nodes

2004



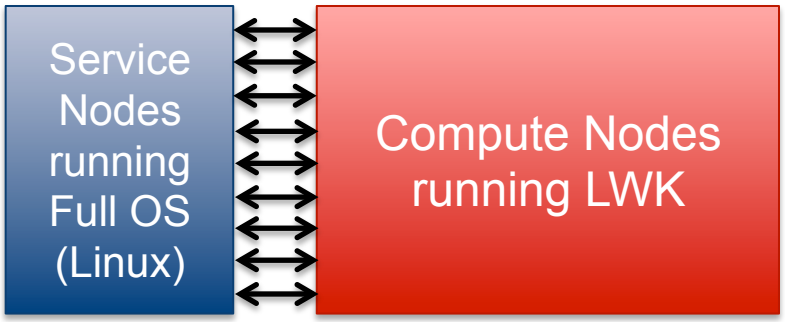
Red Storm

- Prototype Cray XT
- Custom interconnect
- Purpose built RAS
- Highly balanced and scalable
- Catamount lightweight kernel

What is a Lightweight Kernel? (LWK)

Sandia National Laboratories

- It's one component in the overall machine operating system
  - Relies on full-service OS/Linux functionality elsewhere in system
  - Allocates hardware resources to applications, app knows best

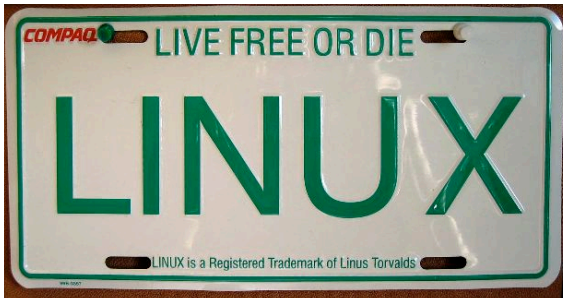


- Sometimes called a "Compute Node OS"
  - A bit of a misnomer, more like an application runtime
  - Derives from partition model: specialize HW/SW for compute nodes, service nodes, login nodes, I/O nodes, etc.
- To a first order, goal is to deliver maximum hardware performance to scalable HPC applications
  - Trade functionality for performance
  - Extends beyond MPI (threads, OpenMP, SHMEM, UPC, …)

What's the Problem with Linux?

Sandia National Laboratories

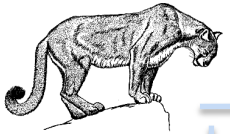
- Non-problems
  - Large developer community, rapid development
  - Supports pretty much all hardware / devices
  - HPC vendors target it, test it, improve it



- Arguably problems
  - HPC is not main focus, mostly afterthought in core community
  - Large codebase, lots to comprehend to make changes
  - Rapid pace of development, difficult to get HPC changes accepted
- Problems
  - Resource management policies designed for overprovisioning, general-purpose functionality
    - Memory pinning, large pages, page cache, swapping, OOM killer, etc.
  - Performance variability, lack of QoS guarantees
  - Poor match for some hardware (BlueGene, embedded hw)
  - Assumes certain hardware characteristics (cache coherency, homogeneous)

Lightweight Kernel Timeline

Sandia National Laboratories



2002
Catamount

2008
Kitten

Hobbes

2002
Virtuoso

2007
Palacios

2004
BG/L CNK

2007
BG/P CNK

2012
BG/Q CNK

FusedOS

2007
Cray Compute Node Linux

Green = Open Source
Red = Closed Source

- Kitten and CNK similar in concept
  - Both support Linux API subset and ABI compatibility
  - Kitten targets x86 (ARM port underway), CNK targets PowerPC
  - Kitten leverages Linux source code, CNK uses no Linux source code

- Palacios and Xen are both hypervisors
  - Palacios designed to be embeddable in a host OS, Kitten or Linux
  - Palacios is designed for HPC, low overhead, predictable performance
  - Palacios targets x86, Xen targets x86 + other archs

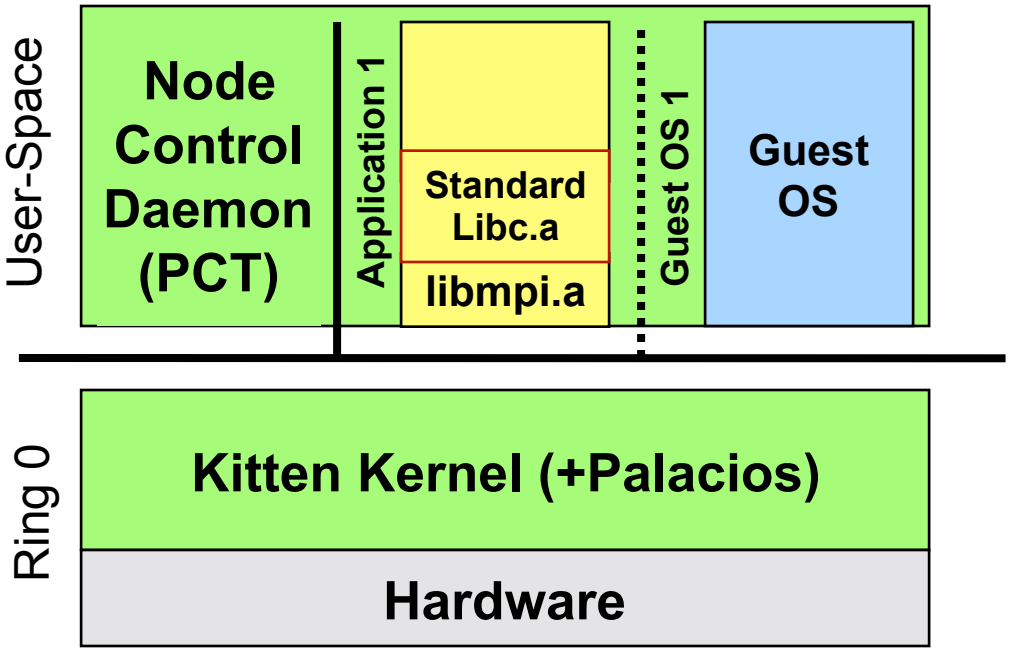
Kitten LWK Goals

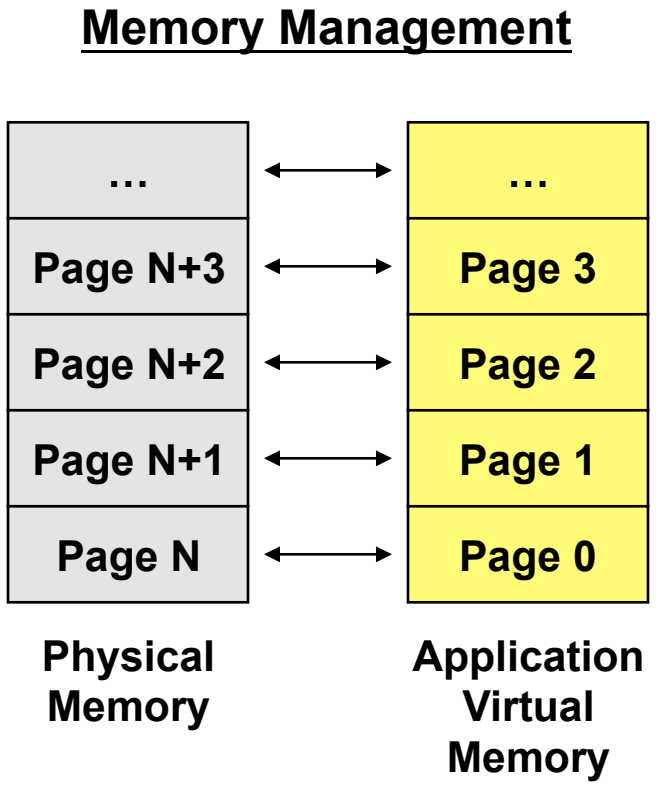
Sandia National Laboratories

- Maintain goodness of Catamount, but modernize
  - Inverted resource management: application in control, not OS kernel
  - Predictable performance behavior
- Provide baseline Linux ABI compatibility (like CNK)
- Add support for multi-threading, POSIX threads, OpenMP, …
- Leverage virtualization to support full-OS functionality
- Enforce security, prevent app from crashing node or other nodes
- Be good platform for HPC OS R&D
- Be suitable for deployment

- Non-goals
  - Be a general-purpose OS
  - Breadth of driver support

Kitten Basic Architecture





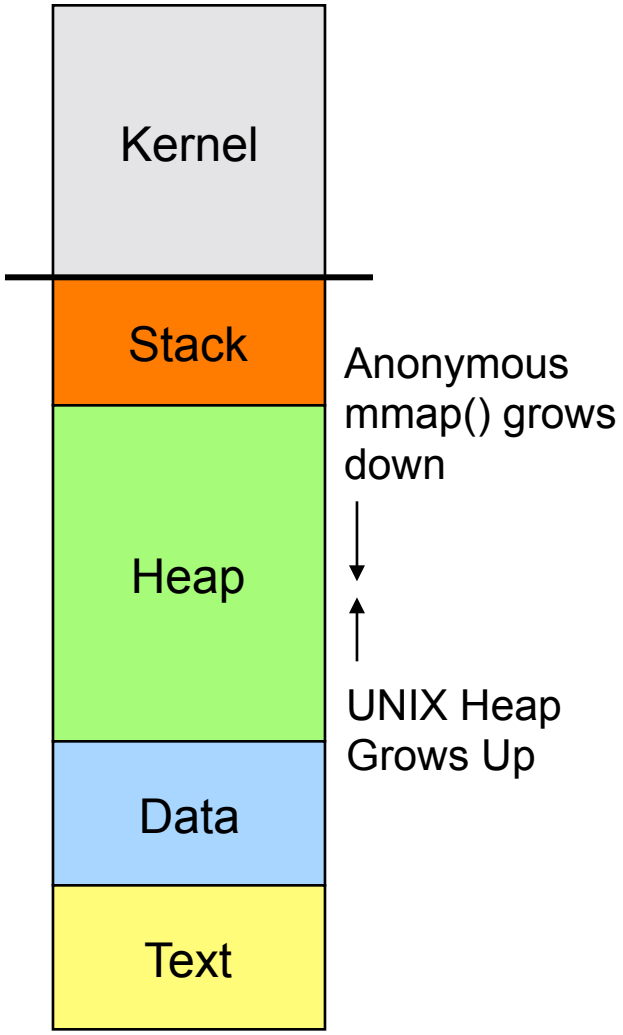
- Kitten kernel provides mostly mechanisms + interfaces
  - Bootstraps HW, figures out what resources are available
  - Controls access to privileged hardware
- PCT in user-space directs kernel how to allocate resources
  - Create address spaces, virtual memory regions
  - Map physical memory to address spaces
  - Create threads/processes, bind to CPUs

Kitten LWK Implementation

Sandia National Laboratories

- Monolithic, C code, GNU toolchain, Kbuild configuration
  - Core Kernel 12K SLOC, x86_arch 12K SLOC, include 22K SLOC
- Supports x86-64 architecture only, porting to ARM
  - Boots on standard PC architecture, Cray XT, and in virtual machines
  - Boots identically to Linux (Kitten bzImage and init_task)
- Repurposes basic functionality from Linux
  - Hardware bootstrap
  - Basic OS kernel primitives (lists, locks, wait queues, etc.)
  - Directory structure similar to Linux, arch dependent/independent dirs
- Custom address space management and task management interfaces
  - User-level API for managing physical memory, building virtual address spaces
  - User-level API for creating tasks, which run in virtual address spaces

LWK Virtual Memory Regions

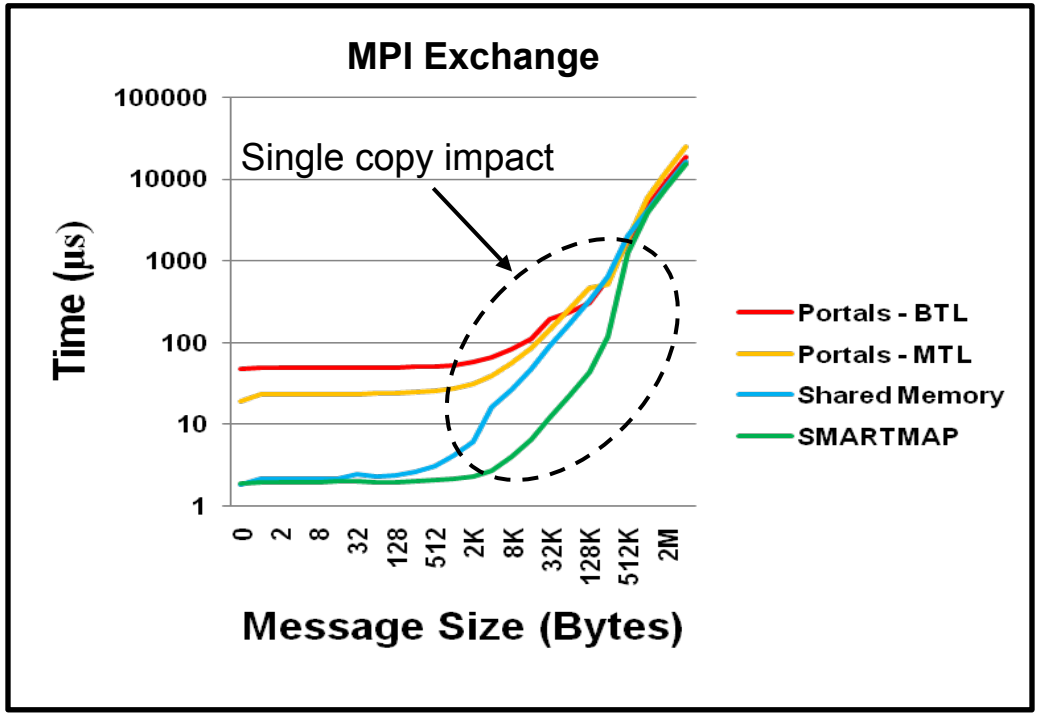


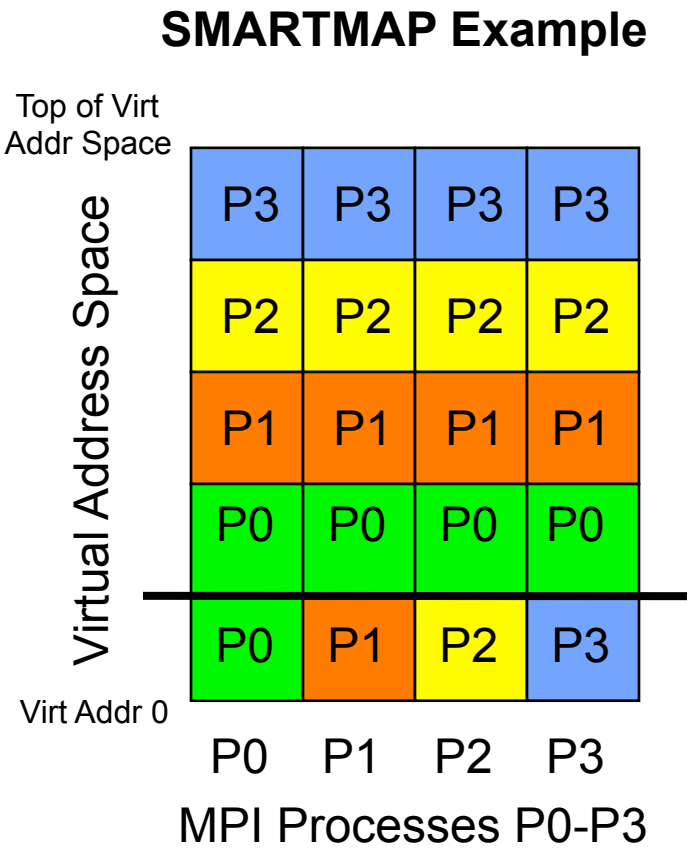
- User address space divided into virtual memory regions:
  - Text
  - Data
  - Heap
  - Stack
- Each region is mapped to a contiguous region of physical memory
  - Straightforward to use large pages
  - PCT in user-space sets up the mapping
- All virtual<->physical mapping occurs before application starts
  - No demand paging
  - No memory oversubscription

SMARTMAP Intra-node Optimization Eliminates Unnecessary Memory Copies

Sandia National Laboratories

- **Basic Idea: Each process on a node maps the memory of all other processes on the same node into its virtual address space**
- **Enables single copy process to process message passing (vs. multiple copies in traditional approaches)**





Task Migration Optimization

Sandia National Laboratories

Operating System	Round-trip Task Migration Time (task on core A migrates to core B, then back to A)
Linux 2.6.35.7	4435 ns
Kitten 1.3	2630 ns

Core-switching performance between two cores in the same Intel X5570 2.93 GHz processor. Kitten achieves a speedup of 1.7 compared to Linux, due to simpler implementation.

Getting Started

Sandia National Laboratories

hg clone https://code.google.com/p/kitten

make menuconfig (chose all defaults)

make isoimage

- Then boot the isoimage wherever you'd like
- You should see a bunch of boostrap messages detailing the hardware detected
- Once boostrap is done, the "hello world" init task will be started
- You can replace the "hello world" init task with an ELF executable of your choosing (e.g., an OpenMP application)
- All binaries must be statically linked
- By default, init_task limited to 64 MB. To increase, either edit kernel/init_task.c to increase defaults or use kernel command line options:
  - init_heap_size=1073741824 init_stack_size=4194304

Future Directions

Sandia National Laboratories

- Define “Node Virtualization Layer” interfaces (DOE/ASCR Hobbes)
  - Example: Run science app in Kitten partition, analysis app in Linux partition, share memory regions (or snapshots) between the two
  - Doesn’t necessarily have to use virtualization hardware, both OS stacks could run natively if the cooperate
- Define runtime system interfaces (DOE/ASCR XPRESS)
- Better networking and I/O support
- Explore capabilities of other research OSes / kernels
  - Physical memory management interfaces in L4
  - Partition management interfaces in Tesselation (Berkeley)
  - Message passing interfaces of Barrelfish

Conclusion

Sandia National Laboratories

- Kitten LWK is a special purpose “Compute Node OS” for HPC
  - Component of overall system
  - Relies on an external general-purpose OS, nominally Linux
  - Simple resource management, gives app control
- Kitten supports a basic Linux user-level environment
  - Standard Glibc, POSIX threads support, OpenMP
  - Subset of Linux system calls supported
- Kitten leverages virtualization for full OS support
- Kitten being leveraged by several current research projects

Acknowledgements

- Ron Brightwell (SNL)
- David DeBonis (SNL, HP)
- Michael Levenhagen (SNL)
- Patrick Bridges (U. New Mexico)
- Peter Dinda (Northwestern)
- John Lange (U. Pittsburgh)

Sandia National Laboratories

Additional Slides

Physical Memory Management

Sandia National Laboratories

- Region based physical memory management
- Broadly separated into two partitions
  - Kmem (Kernel Memory)
  - Umem (User Memory)
- Kmem pool is fixed at boot time, doesn't grow
  - Size configurable using kmem_size boot parameter, 64 MB by default
  - Kernel uses kmem API to allocate kmem
- Umem pool managed by user-space
  - PCT uses pmem syscall API to allocate physical memory
  - PCT uses aspace syscall API to bind physical memory to address spaces

Pmem Region Data Structure

(include/lwk/pmem.h and kernel/mm/pmem.c)

Sandia National Laboratories

```
/**
 * Defines a physical memory region.
 */
struct pmem_region {
        paddr_t         start;              /* region occupies: [start, end) */
        paddr_t         end;

        bool            type_is_set;        /* type field is set? */
        pmem_type_t     type;               /* physical memory type */

        bool            numa_node_is_set;   /* numa_node field is set? */
        numa_node_t     numa_node;          /* locality group region is in */

        bool            allocated_is_set;   /* allocated field set? */
        bool            allocated;          /* region is allocated? */

        bool            name_is_set;        /* name field is set? */
        char            name[32];           /* human-readable name of region */
};
```

Pmem Core API

(include/lwk/pmem.h and kernel/mm/pmem.c)

Sandia National Laboratories

```
/* Add a region of physical memory to the pmem pool */
int pmem_add(const struct pmem_region *rgn);

/* Update a region of physical memory's meta-data */
int pmem_update(const struct pmem_region *update);

/* Find a region of physical memory meeting given criteria */
int pmem_query(const struct pmem_region *query,
               struct pmem_region *result);

/* Atomically query and mark result as allocated */
int pmem_alloc(size_t size, size_t alignment,
               const struct pmem_region *constraint,
               struct pmem_region *result);
```

Example Pmem Layout after Boot

Sandia National Laboratories

- VMware guest configured for 4 GB memory:

```
Physical Memory Map:
    [0000000000000000, 0x00000000083000) BOOTMEM    numa_node=0  (Bootstrap allocs)
    [0x00000000083000, 0x0000000009f000) KMEM       numa_node=0
    [0x0000000009f000, 0x00000000100000) BOOTMEM    numa_node=0  (BIOS reserved)
    [0x00000000100000, 0x00000000200000) KMEM       numa_node=0
    [0x00000000200000, 0x00000000413000) BOOTMEM    numa_node=0
    [0x00000000413000, 0x00000004000000) KMEM       numa_node=0
    [0x00000004000000, 0x00000004119000) INITRD     numa_node=0
    [0x00000004119000, 0x00000006162000) INIT_TASK  numa_node=0
    [0x00000006162000, 0x000000bfee0000) UMEM       numa_node=0
    [0x000000bfee0000, 0x000000bff00000) BOOTMEM    numa_node=0  (ACPI stuff)
    [0x000000bff00000, 0x000000c0000000) UMEM       numa_node=0
    [0x000000c0000000, 0x00000100000000) BOOTMEM    numa_node=0  (GPU, APIC, ...)
    [0x00000100000000, 0x00000140000000) UMEM       numa_node=0

Total User-Level Managed Memory: 4192722944 bytes
```

Kmem Management API

(include/lwk/kmem.h and kernel/mm/kmem.c)

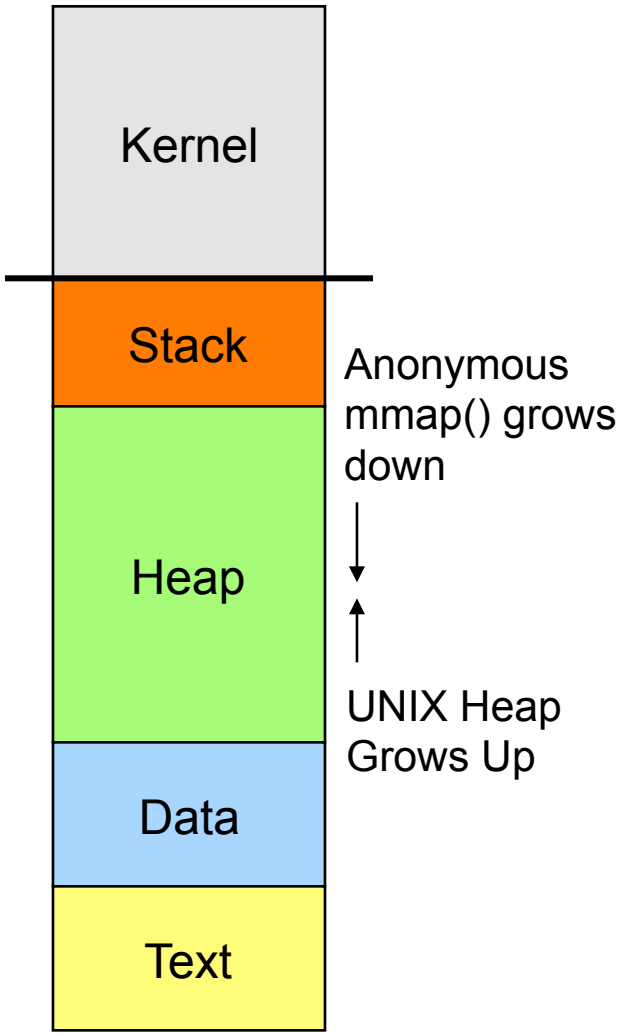
Sandia National Laboratories

- All Kmem managed by buddy allocator (kernel/mm/buddy.c)
- Two ways to allocate:
  - malloc() style give me some memory
  - Page-based give me a contiguous set of pages

```
/* malloc-style, implementation tracks block size internally */
extern void *kmem_alloc(size_t size);
extern void kmem_free( const void *addr);

/* page-based, caller must remember order of the block */
extern void *kmem_get_pages(unsigned long order);
extern void kmem_free_pages(const void *addr,
                            unsigned long order);
```

LWK Virtual Memory Regions



- User address space divided into virtual memory regions:
  - Text
  - Data
  - Heap
  - Stack
- Each region is mapped to a contiguous region of physical memory
  - Straightforward to use large pages
  - PCT in user-space sets up the mapping
- All virtual<->physical mapping occurs before application starts
  - No demand paging
  - No memory oversubscription

Aspace Management

Sandia National Laboratories

- Every execution context must execute in the context of a virtual address space, represented by an **aspace structure**
- After bootstrap, all address spaces have the kernel mapped into them above PAGE_OFFSET (matches Linux design)
  - Avoids context switch to enter kernel
  - Enables kernel threads to run without context switch
- Address space consists of non-overlapping virtual memory regions, each mapped to physical memory or hardware

- Currently no support for handing page faults
- In future may allow dynamic binding of virtual memory region to a physical memory pool for NUMA first-touch support

Aspace Core API

(include/lwk/aspace.h and kernel/mm/aspace.c)

Sandia National Laboratories

```
/* Create a new aspace, possibly with a specific ID */
int aspace_create(id_t id_request, const char * name,
                  id_t *id);

/* Create a virtual memory region */
int aspace_add_region(id_t id, vaddr_t start, size_t extent,
                      vmflags_t flags, vmpagesize_t pagesz,
                      const char * name);

/* Map physical memory to a virtual memory region */
int aspace_map_pmem(id_t id, paddr_t pmem,
                    vaddr_t start, size_t extent);

/* Map one aspace into another at a given virtual address */
int aspace_smartmap(id_t src, id_t dst,
                    vaddr_t start, size_t extent);
```

Task Management

Sandia National Laboratories

- Every context of execution represented by a task
  - Each task is associated with an aspace
  - Threads implemented as multiple tasks associated with the same aspace
- Each task represented by a kernel-level task_struct
  - Contiguous block of memory including TCB and kernel stack (on x86)
  - Includes the task's permissions (uid/gid), fdtable, signal table, etc.
- Each CPU maintains its own task queue
  - Runnable tasks schedule round-robin
  - Blocked tasks are idle until they are woken up

Task Core API

(include/lwk/task.h and kernel/task.c)

```
/* Specifies the initial conditions to use when spawning a new task */
typedef struct {
    id_t        task_id;
    char        task_name[32];

    id_t        user_id;        // User ID the task executes as
    id_t        group_id;       // Group ID the task executes as
    id_t        aspace_id;      // Address space the task executes in
    id_t        cpu_id;         // CPU ID the task starts executing on

    vaddr_t     stack_ptr;      // Ignored for kernel tasks
    vaddr_t     entry_point;    // Instruction address to start executing at

    int         use_args;       // If true, pass args to entry_point()
    uintptr_t   arg[4];         // Args to pass to entry_point()
} start_state_t;

/* Spawn a new task with the requested start_state */
int task_create(const start_state_t *start_state, id_t *task_id);

int task_switch_cpus(id_t cpu_id);  /* allow task to migrate itself */
```

Thread Support

Sandia National Laboratories

- Kitten user-applications link with standard GNU C library (Glibc) and other system libraries installed on the Linux build host
- Functionality added to Kitten to support Glibc NPTL POSIX threads implementation
  - Futex() system call (fast user-level locking)
  - Basic support for signals
  - Match Linux implementation of thread local storage
  - Support for multiple threads per CPU core, preemptively scheduled
- Kitten supports runtimes that work on top of POSIX threads
  - GOMP OpenMP implementation
  - Qthreads
  - Probably others with a little effort

System Calls

Sandia National Laboratories

- Kitten syscall calling conventions identical to Linux
- Syscall linkage defined in include/arch/unistd.h
- Syscall implementations
  - Linux syscall implementations kernel/linux_syscalls/
  - LWK specific syscalls kernel/lwk_syscalls
- General approach is to implement a Linux syscall when we find it is needed, only implement as much as is needed
- Current Linux syscall list, some are –ENOSYS stubs:
  - brk, clock_gettime, clone, close, dup2, dup, exit, exit_group, fcntl, fork, fstat, futex, getcpu, getdents64, getdents, getgid, getgroups, getpid, getrlimit, getrusage, gettid, gettimeofday, getuid, ioctl, kill, lseek, madvise, mkdir, mknod, mmap, mprotect, mremap, munmap, nanosleep, open, pipe, poll, read, readlink, readv, rmdir, rt_sigaction, rt_sigpending, rt_sigprocmask, sched_getaffinity, sched_yield, sethostname, set_robust_list, set_tid_address, settimeofday, stat, time, uname, unlink, wait4, write, writev

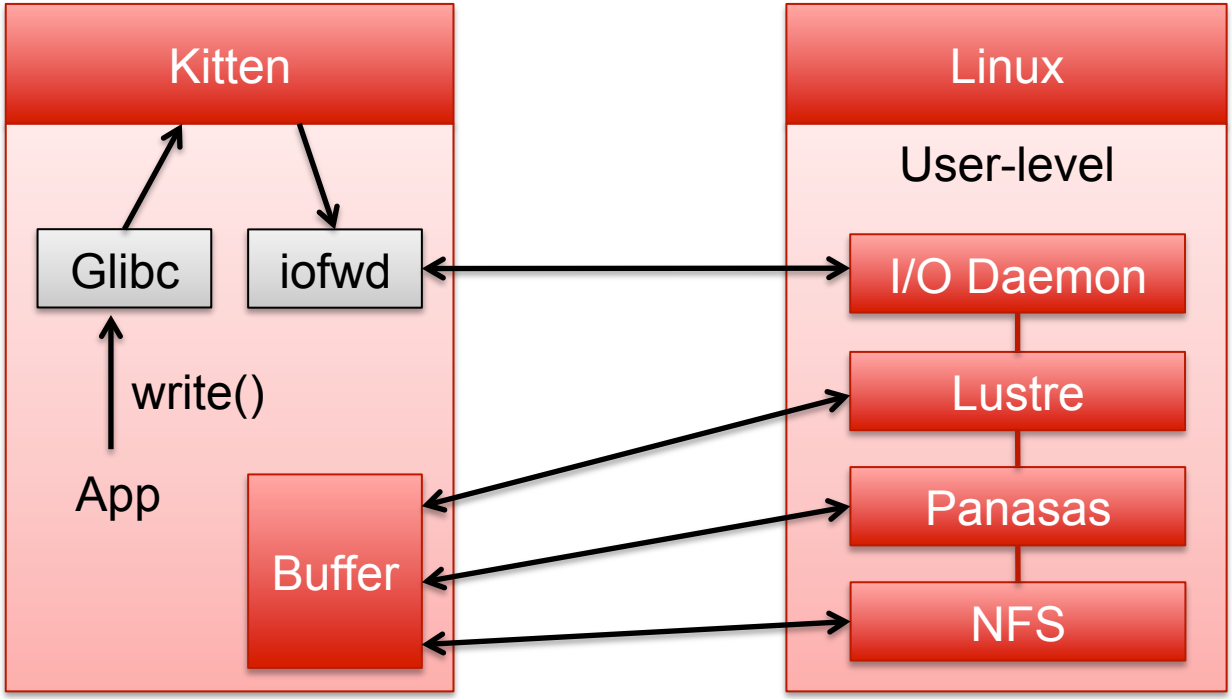
Kitten Networking

Sandia National Laboratories

- Ported OFA Infiniband stack to Kitten a couple years ago
  - Implemented Linux compatibility layer to support OFA stack mostly unmodified
  - Turned out to be a lot of work
  - Difficult to make work on new IB clusters different than ours
- Recently started focusing on Portals4
  - Target Portals4 as lowest-level communication API
  - For development purposes, create implementations over Ethernet and (possibly) Infiniband
  - Portals4 reference implementation currently running in VMware virtual machine over VMware's virtual e1000 Ethernet device
    - Enables Kitten virtual cluster development environment

Kitten I/O Forwarding

- Prototype implementation developed over summer
- Influenced by IOFSL, wanted to use SMARTMAP and Portals
- Supports local files for drivers, forwards all else off node
- Kitten reflects off-node I/O calls to user-space
  - Avoids need for custom Glibc port
  - Only control reflected, no extra buffer copies



Other Bits

Sandia National Laboratories

- Platform independent subsystems, rely on arch code to implement
  - ELF loader (mostly in user-space liblwk)
  - PCI enumeration, reads/writes to config space
  - Driver infrastructure
  - Interrupt registration and dispatch
  - Cross-calls
  - Timekeeping and timers
  - Console subsystem
  - KGDB support
- Job launch tool in progress
  - Similar to yod, aprun, mpirun, etc… Linux tool for launching Kitten apps
  - Uses Portals4 for all communication
  - Implements PMI over Portals4
  - I/O forwarding layer over Portals4