Secure Operating System Design and Implementation
Memory

Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago

February 20, 2011

Part I

Memory Background

User space and kernel space

Virtual address space includes a single process and the kernel

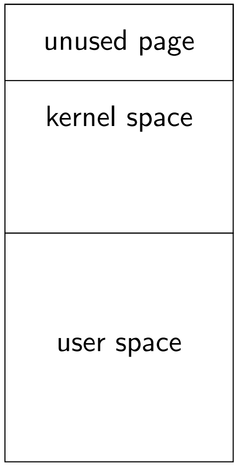
user space contains a process's code and data
- cannot access kernel space
- executes only unprivileged instructions
- described by regions

kernel space contains kernel code, data structures, plus some information for each process.
- executes unprivileged and privileged instructions
- can access user space

Memory protection

- To ensure isolation between different processes, programs must be prevented from writing, reading, or executing memory which is not theirs.
- Two ways to do this
  - Visibility: prevent programs from seeing memory which they should not access
  - Permissions: provide finer-grained control for partial access, e.g., read but not write
- Memory protection provided by paging or segmentation
- x86 architecture has a variety of modes which affect memory protection

Memory layout



- User space from 0 to UserMaxAddr-1
- kernel space from KernelMinAddr to KernelMaxAddr
- one unused page at top of address space so that we can write iterations

```
for (a=KernelMinAddr;
     a <= KernelMaxAddr;
     a+= pageSize)
  // do something for each
  // page in kernel memory
```

- What happens w/o the unused page?
- Depending on the memory model, Xen hypervisor is mapped above the kernel

Part II

Regions

Regions

- A process's address space is divided into regions
  - text region is read only which holds both constants and instructions
  - stack region contains local variables of procedures
  - heap is dynamically allocated (non-stack) storage
- regions are logical constructs, protections typically provided by paging

Region semantics

- Read-only regions may be shared
- Writable regions can be shared if writes are rare, called **Copy on Write (CoW)**
  - regions are marked copy on write
  - page table entries are marked read-only and point to a shared page
  - on a write, a page fault occurs, the kernel is entered, and the memory is replicated (coped) and made writable
  - successive writes need not be intercepted
- Regions may allocated pages on demand (e.g., heap)
- Ethos assumes that VM address space is large, and thus uses fixed sized regions.

Region use

fork Use CoW to copy parent process's regions
writes will be caught by interrupt mechanism and kernel makes copies

exec Replace the process's region with new regions from file

exit Removes all process's regions

heap in Ethos, static region given for heap.

stack in Ethos, static region given for stack.

Access to page in heap or stack region causes allocation.

Part III

Process Structure

Process structure

- A process structure is maintained in the kernel for each process in user space
- It contains information which cannot be trusted to user space
- For example:
  - the user on whose behalf the process executes
  - the executable label
  - the process-specific state of all resources in use by the process
  - the memory regions, page table, kernel stack associated with the process
  - the events of the process
  - the current state of the process
- Because the process structure is maintained in the kernel, no matter what the process does, the kernel can use it to clean up after process exits

Per-process kernel stack

- In a traditional monitor model, the process can sleep on an event.
- When the process resumes in the kernel, it needs to resume with its own state.
- Which is kept on the process's kernel stack.
- Right now, Ethos has a kernel stack per process
- Ethos system calls, however, are designed for the most part to be non-blocking
- Event processing is where (most of) the blocking is, so

Event stacks

- We are considering providing per-event stacks
- Single stack can be shared between all processes since no system call blocks
  This is almost true, but we do have a few blocking syscalls (e.g.,`block`)
- Blocking primarily occurs in event processing, so
  - Create a stack for processing
  - The stack evokes an Event Driven Thread (EDT)
  - E.g. `createDirectoryEvent` might be the blocking code which is used to create a directory.
  - The EDT is non-preemptive
  - The EDT may block on events (rpc, networking, ...)
  - When the EDT is finished, it marks the event as completed and frees the stack

Part IV

Buddy Allocator

Buddy allocator

In Ethos

- After the kernel loads, it places all unused pages under the control of the buddy allocator
- These pages are all mapped to kernel addresses
- Some of them are needed for PTEs—and as per Xen—each PTE must be read only.
- Ethos is intended for 64-bit machines which has sufficient virtual address space
- (Ethos currently is 32-bit)
- Even when allocated to user space, pages are mapped to the kernel

Buddy allocator (cont'd)

- Buddy allocator enables allocations in powers of two (called the *order*)
- The allocator partitions and coalesces allocations
- Pages allocated for kernel use are a power of two in size and are allocated in place
- Pages allocated for user space are individually remapped and thus singleton pages are used for this purpose

Part V

Slob Allocator

Slob allocator

- The page allocator is the most efficient way of allocating large objects
- For smaller objects, the slob allocator is used
- The slob allocator can be used in one of two ways
  - Fixed sized and type objects which can have initialization constructors
  - Variable sized string

Part VI

Interrupts and Exceptions

Interrupts and exceptions

- To perform I/O, processes need to request the OS to perform operations on their behalf
- Cannot branch into the OS for at least three reasons:
  - Cannot access kernel memory
  - Must enter only at well-defined entry points (allowed functionality) where inputs are checked
  - Fixing kernel addresses in a process is a problem if there are changes to the kernel
- Solution is to have the kernel define an interrupt vector, and allow processes to specify an interrupt number within a specified range.

Types of exceptions

faults: precise exceptions which allow instruction restart. Any machine changes prior to fault are undone.

traps: precise exceptions in which the instruction completes execution. Restart begins at the next instruction.

aborts: imprecise exception which do not allow reliable instruction restart.

System calls

- System calls performed using an interrupt
- Need to pass parameters
- In Ethos, three registers are used for syscall
  - syscall number
  - pointer to input parameter to syscall
  - pointer to output parameter to syscall
- The input and output parameters may be structures, enabling multiple values to be passed in each direction.
- the call is made by an int 80 (interrupt 80)
- the status is passed back on the stack
- this calling sequence is not at all optimized, but it is flexible

System calls

- System calls can return variable size values
- (This is not the case with UNIX, in which storage is preallocated).
- Ethos pre-allocates the storage
- If there is insufficient space, userspace will allocate storage and then the syscall wrapper will get the return value
- this mechanism is integrated with Ethos's `retire` semantics
- the mechanism is designed to conveniently allow many return values

Interrupts and Exceptions

0	Integer Divide-by-zero
1	Debug exception
2	Non-maskable interrupt
3	Breakpoint exception
4	Overflow exception
5	Bound range exception (BOUND instruction)
6	Invalid opcode
7	Device not available exception
8	Double-fault exception
9	Co-processor overrun exception (Reserved in X86-64)
10	Invalid TSS exception
11	Segment-not-present exception
12	Stack exception
13	General protection exception
14	Page fault exception
15	(Reserved)

Interrupts and Exceptions (cont'd)

16	x87 floating point exception
17	Alignment check exception
18	Machine check exception
19	SIMD floating point exception
0-255	software interrupt
Any	Hardware maskable interrupt

Interrupt types

terminal errors	machine check exceptions
ill formed program	invalid opcode, general protection exception, alignment check exception
ill formed kernel	invalid TSS
features	integer divide by zero, debug, breakpoint, overflow, floating point exceptions, SIMD exceptions
paging	page fault exception

Ethos interrupt handling

- panic on terminal errors, ill-formed kernel, or ill formed program when operating in kernel mode
- handle page faults
- ignore some unhandled interrupts when they occur in processes
- terminate some processes on unhandled interrupt

Xen events and Ethos usage

- Xen intercepts interrupts and produces events for the appropriate VM
- Most of these events are currently ignored by Ethos
- Except those which are so bad that they are used to panic the kernel