Microkernel Construction

Threads and Address Spaces

Nils Asmussen

04/16/2020

Outline

- **Threads**
  - Definition
  - Concepts in NOVA
  - Thread Switch in NOVA
- FPU Handling
- Address Spaces

What is a Thread?

- An independent flow of control inside an address space
- Communicates with other threads using IPC
- Characterized by a set of registers and the thread state
- Dispatched by the kernel according to a defined schedule

What is a Thread?

- An independent flow of control inside an address space
- Communicates with other threads using IPC
- Characterized by a set of registers and the thread state
- Dispatched by the kernel according to a defined schedule
- Each thread is bound to one core at a time
- Only one thread per core is running at one point in time
- With *n* cores, *n* threads can run at once
- All other threads are inactive, waiting inside the kernel

Implementation in NOVA

Execution Context:

- Register state
- Continuation
- Address Space (PD)
- UTCB (message buffer)
- IPC partner
- FPU state
- prev/next pointer

Scheduling Context:

- Execution Context
- Priority
- Budget
- Remaining budget
- prev/next pointer

Thread Variants

Global Thread

- Needs an scheduling context, i.e., CPU time, to execute
- Causes exception on startup to let creator set register state

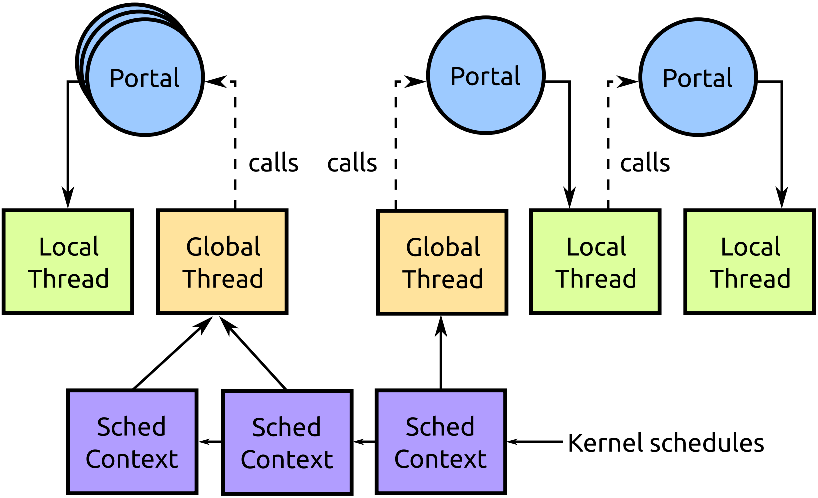
Local Thread

- Has no scheduling context
- Are only used to handle portal calls
- Waits in the kernel until someone called an associated portal

Portals

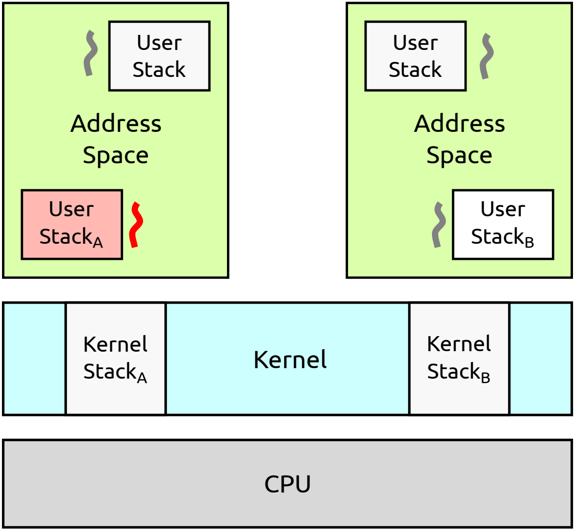
- A portal is an IPC endpoint
- Executed by local threads
- CPU time is donated from caller
- Called via system call
- Message is transferred from sender UTCB to receiver UTCB

Overview

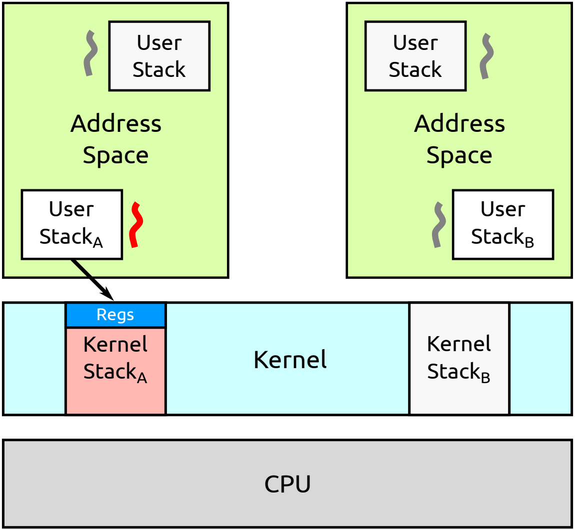


Portal
Portal
Portal
calls
calls
calls
Local Thread
Global Thread
Global Thread
Local Thread
Local Thread
Sched Context
Sched Context
Sched Context
Kernel schedules

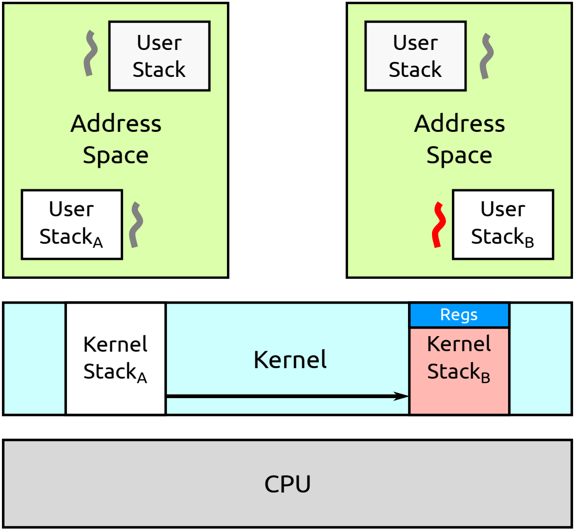
Thread Switch: Conventional



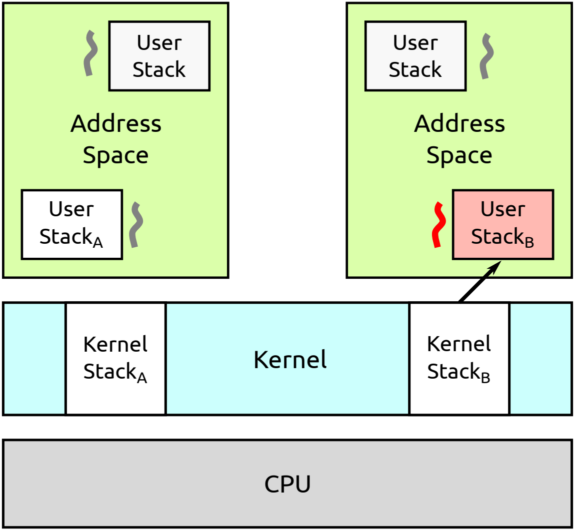
Thread Switch: Conventional



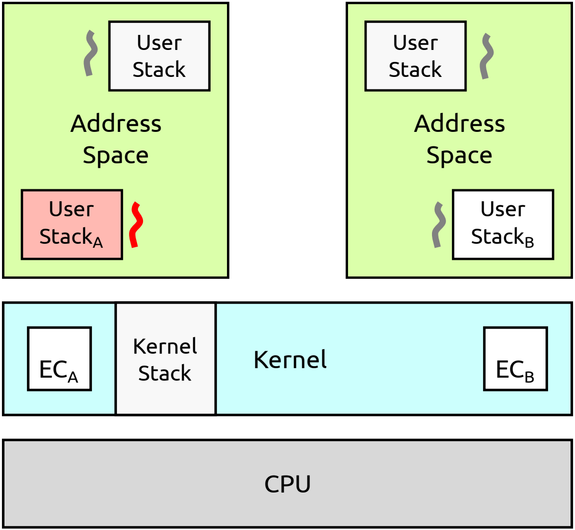
Thread Switch: Conventional



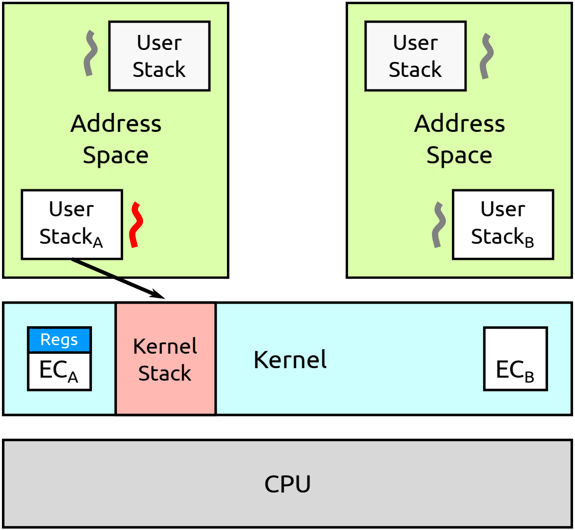
Thread Switch: Conventional



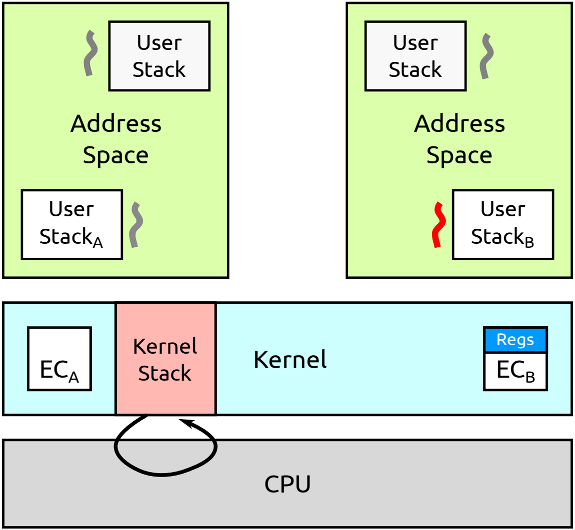
Thread Switch: Continuation Style



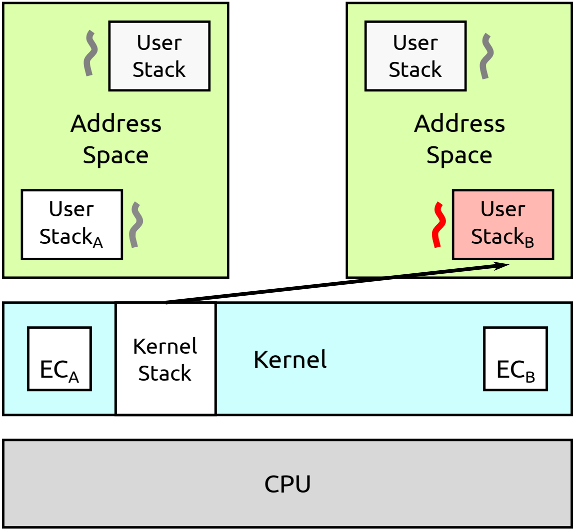
Thread Switch: Continuation Style



Thread Switch: Continuation Style



Thread Switch: Continuation Style



Thread Switch: In-Kernel Switch

- Traditional kernels save/restore the current CPU state
- Each thread has own stack → stack frames are kept
- In NOVA, stack frames and CPU state are lost

Part of `sys_call`

```
current->cont = ret_user_sysexit;
current->set_partner (ec);
ec->cont = recv_user;
ec->regs.set_ip (pt->ip);
ec->regs.set_pt (pt->id);
ec->make_current();
```

Thread Switch: In-Kernel Switch

Switching to an Ec

```
void Ec::make_current()
{
    current = this;
    Tss::run.sp0 = reinterpret_cast<mword>(exc_regs());
    pd->make_current();
    asm volatile (
      "mov %0, %%rsp;"
      "jmp *%1;"
      :
      : "g" (CPU_LOCAL_STCK + PAGE_SIZE),
        "rm" (cont)
      : "memory"
    );
    UNREACHED;
}
```

Outline

- Threads
- **FPU Handling**
  - General Idea
  - x86 Details
  - Implementation in NOVA
- Address Spaces

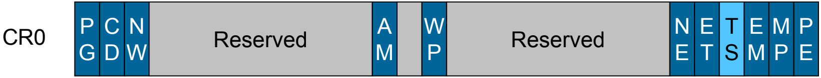
Floating Point Unit

- CPU has dedicated functional units for FP computations
- Are accessed with specific instructions
- Have their own state, which is large (512 bytes)
- Each thread has its own FPU state
- $\rightarrow$ Save/restore FPU on each context switch is expensive
- $\rightarrow$ However, many OSes on x86 today save it on every switch (vector instructions, LazyFPU vulnerability)

FPU Switch: General Idea

- We want to know if/when a thread uses the FPU
- We only want to save the FPU state if it has been modified
- We don't want to save the FPU state when switching from a thread that used the FPU to a thread that is not going to use the FPU and then later restore the old (unmodified) FPU state

Lazy FPU Switch on x86



If CR0.TS (Task Switched) flag is set, FPU instructions are not executed, but cause #NM exception.

Implementation in NOVA

Handling the #NM exception

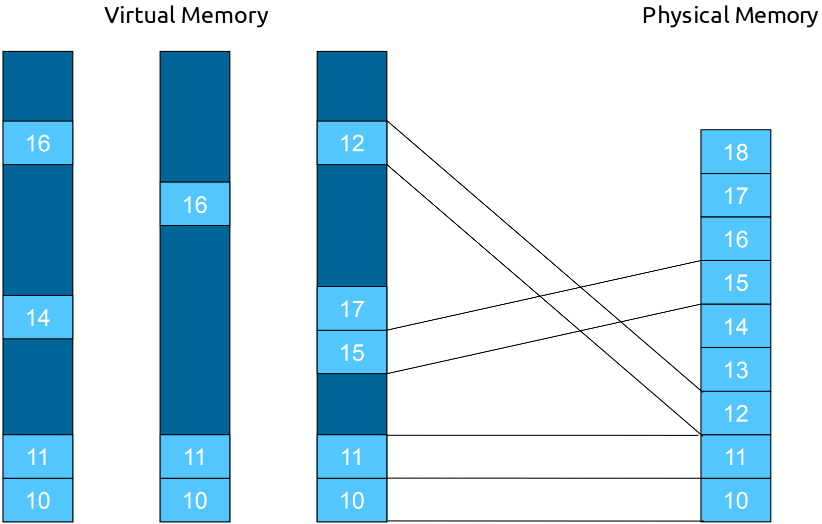
```
void handle_exc_nm() {
  CR0.TS = 0;
  hzd |= HZD_FPU;
  if (current == fpowner)
    return;
  if (fpowner)
    fpowner->fpu->save();
  if (current->fpu)
    current->fpu->load();
  else {
    current->fpu = new Fpu;
    Fpu::init();
  }
  fpowner = current;
}
```

Before leaving to user

```
void handle_hazards() {
  if ((hzd & HZD_FPU) &&
        current != fpowner) {
    CR0.TS = 1;
    hzd &= ~HZD_FPU;
  }
}
```

Outline

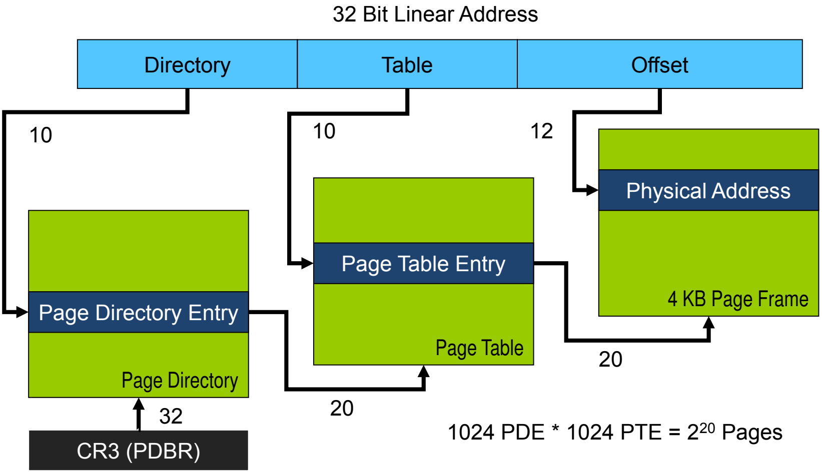
- Threads
- FPU Handling
- **Address Spaces**
  - Virtual Memory Recap
  - x86 Data Structures
  - x86 TLB
  - Implementation in NOVA


Virtual Memory
Physical Memory
16
14
11
10
16
11
10
12
17
15
11
10
18
17
16
15
14
13
12
11
10

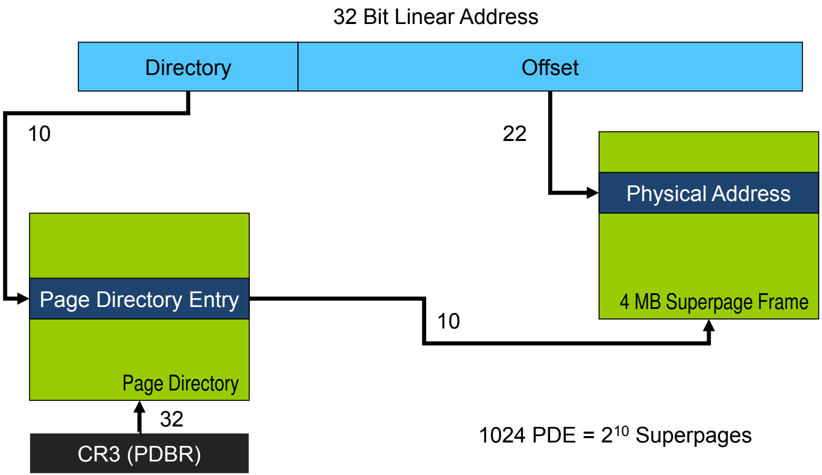
Paging (x86)

- Translation of linear to physical addresses
- Done by memory management unit (MMU)
- Hardware defines data structures:
  - Page Directory Base Register (CR3)
  - Page Directory (PDIR)
    - 4KiB page containing 1024 page directory entries (PDEs)
  - Page Table (PTAB)
    - 4KiB page containing 1024 page table entries (PTEs)
- Paging data structures use physical addresses

Address Translation: 4 KiB pages (x86)



Address Translation: 4 MiB superpages (x86)



PDEs and PTEs (x86)

PDE (4 KB Page Table)

Page Table Base Address	Avail	(G)	PS	0	A	PCD	PWT	U/K	R/W	P
			0							

PDE (4 MB Superpage)

Page Base Address	Reserved	PAT	Avail	G	PS	D	A	PCD	PWT	U/K	R/W	P
					1							

PTE (4 KB Page)

Page Base Address	Avail	G	PAT	D	A	PCD	PWT	U/K	R/W	P

Translation Lookaside Buffer (x86)

- Caches recent linear-to-physical translations
- Avoids expensive page-table walk
- Must be kept consistent with the page tables by the OS
- No TLB coherency protocol
- On modifications, OS must flush relevant TLB entries
- TLB flush triggered by CR3 reload or INVLPG instruction
- No TLB flush required when upgrading page attributes
- CR3 reload does not flush global pages
- TLB shootdowns for page tables active on other cores
    - Expensive signaling and synchronization
    - Inter-Processor-Interrupt (IPI)

Memory space of a protection domain

```
class Space_mem : public Space {
public:
    Hpt hpt;            // master page table
    Hpt loc[NUM_CPU];   // per-core PTs; synced from master
    Dpt dpt;            // DMA PT for IOMMU
    union {
        Ept ept;        // nested PT for Intel (VMX)
        Hpt npt;        // nested PT for AMD (SVM)
    };
};
```

Implementation in NOVA – PTEs

Generic page table entry handling

```
template <typename P, typename E, unsigned L, unsigned B>
class Pte {
    E val;

    P *walk (E virt, unsigned long level, bool add);
    size_t lookup (E virt, Paddr &phys, mword &attr);
    void update (E virt, mword size, E phys,
                 mword attr, bool add);
};

class Hpt : public Pte<Hpt, uint32, 2, 10>;
class Dpt : public Pte<Dpt, uint64, 4, 9>;
class Ept : public Pte<Ept, uint64, 4, 9>;
```

Implementation in NOVA – TLB shootdowns

- `cpus` mask stores cores that use the address space
- core-bit in `cpus` is set as soon as Ec is started on a core
- `htlb` is set to `cpus` on permission downgrades
- TLB shootdown sends IPI to all cores in `htlb`
- IPI causes a scheduling to set CR3
- . . . and clear core-bit in `htlb`

Start	End	core-local	Usage
0000_0000	BFFF_FFFF	No	User space
C000_0000	CFBF_FFFF	No	Code, static data, heap
CFFF_D000	CFFF_DFFF	Yes	Kernel stack
CFFF_E000	CFFF_EFFF	Yes	LAPIC
CFFF_F000	CFFF_FFFF	Yes	Kernel data
D000_0000	D000_1FFF	No	I/O Bitmap
E000_0000	FFFF_FFFF	No	Capabilities