Microkernel System Calls

KernelInterface
IPC
Unmap
ExchangeRegisters
ThreadSwitch
Schedule
SystemClock
ThreadControl
SpaceControl
ProcessorControl
MemoryControl
Address Spaces and Mapping
Basic Operations

- Map
- Unmap
- Grant
Page Fault Handling

Application

Kernel

Pager

"PF" msg

map msg

PF → IPC

res ← IPC

© 2009 University of Karlsruhe, System Architecture Group
Page Fault Protocol

"PF" msg
MR₀ = access type
MR₁ = faulting address
MR₂ = IP of faulting instruction

map msg
Contains MapItem

Application → Pager

© 2009 University of Karlsruhe, System Architecture Group
Page Fault Message

- Short message
  - No further page faults
- Applications can synthesize page fault messages
  - Not a problem - the application could do it anyway by directly accessing the memory it wishes to cause a fault on
Mapping Questions

- How is the mapping to be sent specified?
- How is the mapping to be received specified?
- How do they combine? What is the result?
Fpage Data Type

- **Fpage**
  - fpage size = $2^s$

- Specifies a region of the address space that is
  - A power of 2 in size
  - Aligned to its size

- Note: Smallest supported size is architecture specific
  - IA-32 supports 4K ($s = 12$)
Fpage Data Type

- Complete Address Space

- Nilpage

- See l4/types.h
Receiving a mapping

"PF" msg
MR₁ = 0x00002002
MR₂ = 0xXXXXXXXX

Application

Pager

BR₀ Rcv Fpage (window)

0x2000
0x3000

0x3000
0x2000
0x8

12(6) 0(4)
Buffer Register 0

- $BR_0$ Specifies
  - Willingness to receive StringItems
    - $s = 1$
    - Target string locations in other BRs
  - The receive window for mappings
    - Region of the address space to accept mappings
    - Nilpage: No mappings accepted

Rcv Window (fpage) | 000s
Normal Page Fault

"PF" msg
MR\(_1\) = 0x00002002
MR\(_2\) = 0xXXXXXXXX

Application

0x2000

BR\(_0\) Rcv Fpage (window)
Set by kernel to complete address space

Pager

0 1\(_{(6)}\) 0\(_{(4)}\)
Permissions

- r: read
- w: write
- x: execute

Note: Not all architectures support all combinations

- IA-32: rx and rwx are supported by hardware

- g: mapping (0) or granting (1)
Receive window > mapping size

Map Item

<table>
<thead>
<tr>
<th>Address</th>
<th>Permissions</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>0rwx</td>
<td>s=12</td>
</tr>
<tr>
<td>0x8</td>
<td>0(6)</td>
<td>1000</td>
</tr>
</tbody>
</table>

Sndbase determines location in receive window
Sending a mapping

Map Item

<table>
<thead>
<tr>
<th>Address</th>
<th>Permissions</th>
<th>Permissions Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>0rwx</td>
<td>s=12</td>
</tr>
<tr>
<td>0x123</td>
<td>0(6)</td>
<td>1000</td>
</tr>
</tbody>
</table>

Application

Pager

map msg

0x2000

0x10000
Mappings and Window Sizes

- See reference manual for precise definition of what happens for mismatched mappings and window sizes
- Advice:
  - Simply use 4K pages for all mappings
A Map Message

- For page faults, the kernel expects the following map message response
  - No untyped words
  - 1 MapItem
Explicit Mapping Receive

- $BR_0$ determines whether a receive/wait IPC can include strings or a mapping
- Set it prior to invoking IPC receive/wait

```c
L4_Acceptor_t L4_UntypedWordsAcceptor
L4_Acceptor_t L4_StringItemsAcceptor
L4_Acceptor_t L4_MapGrantItems (
    L4_Fpage_t RcvWindow
)
void L4_Accept (L4_Acceptor_t a)
```
Unmap

- Revoke mappings
  - That were derived from mappings in the current address space

- Revoke access rights
  - To existing mappings
  - Example: RW -> RO

- The mappings to revoke are specified by fpages in MRs
Unmap Arguments

Control

- **f**: specifies whether fpages are flushed from the current address space in addition to revoking derived mapping
- **k**: specifies the highest number MR that contains an Fpage to unmap

Fpages

- Fpages specify the regions in the local address space
- **rwx**: the access rights to revoke

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fpage</td>
<td>0rwX</td>
<td>MR₂</td>
</tr>
<tr>
<td>Fpage</td>
<td>0rwX</td>
<td>MR₁</td>
</tr>
<tr>
<td>Fpage</td>
<td>0rwX</td>
<td>MR₀</td>
</tr>
</tbody>
</table>
### Unmap Results

- **RWX**
  - Reference (r), Dirty (w), and Executed (x) bits
    - Bit returned set if corresponding access has occurred on any derived mapping
    - Reset as a result of the unmap operation
  - Supported combinations are arch-dependent

<table>
<thead>
<tr>
<th>Fpage</th>
<th>0RWX</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR₂</td>
<td></td>
</tr>
<tr>
<td>MR₁</td>
<td></td>
</tr>
<tr>
<td>MR₀</td>
<td></td>
</tr>
</tbody>
</table>
Unmap

- Derived mappings in other address spaces
  L4_Fpage_t L4_UnmapFpage (L4_Fpage_t f)

- Derived mappings in own address space as well
  L4_Fpage_t L4_Flush (L4_Fpage_t f)

void L4_UnmapFpages (L4_Word_t n,
                      L4_Fpage_t * fpages)

void L4_FlushFpages (L4_Word_t n,
                      L4_Fpage_t * fpages)

L4_Bool_t L4_WasWritten (L4_Fpage_t f)
L4_Bool_t L4_WasReferenced (L4_Fpage_t f)
L4_Bool_t L4_WasExecuted (L4_Fpage_t f)
SpaceControl

- Used to control the layout of newly created address spaces
  - Specifically
    - Location of Kernel Info Page - fpage
    - Location of UTCB region - fpage

- Redirector
  - All IPC from threads within the address space is redirected to a controlling thread
    - Used to enforce security policy
  - Note: Should not need to change what is already done in the example code
Microkernel System Calls

KernelInterface
IPC
Unmap
ExchangeRegisters
ThreadSwitch
Schedule
SystemClock
ThreadControl
SpaceControl
ProcessorControl
MemoryControl
ProcessorControl

- Privileged system call
- Sets processor frequency, voltage and other processor specific stuff
  - ... once implemented
MemoryControl

- Privileged system call
- Set cache architecture attributes on pages in memory
  - Machine specific
  - Not implemented for IA-32
  - Obsoleted by MapControl proposal
Microkernel System Calls

KernelInterface
IPC
Unmap
ExchangeRegisters
ThreadSwitch
Schedule
SystemClock
ThreadControl
SpaceControl
ProcessorControl
MemoryControl

That’s it.
Protocols

- Page Fault
- Thread Start
- Interrupt
- Preemption
- Exception
- Sigma0
Exception Protocol

- Exception IPC to exception handler thread
  - On behalf of faulting thread
  - The IPC contains
    - IP of where to resume the thread after handling the exception
    - Exception type
    - Other machine specific stuff
  - The exception handler can respond with an IPC specifying a new IP and other state to recover from the exception
    - See the IA-32 appendix in the manual
Sigma0

- Owns all physical memory in the machine
  - Except that reserved for kernel use
  - Mapped idempotently
    - One-to-one
- Various memory classes
  - Conventional
  - Shared (VGA screen memory, ...)
  - Architecture-specific (ACPI tables, ...)
  - Boot-loader specific (modules, initial servers, ...)
- Maps each page once (and once only)
- Sigma0 protocol
  - Request specific page
  - Request any page
  - Request larger regions
Sigma0 – Root Pager

- Pager of initial threads (root task)
  - Also implements page fault protocol
  - Responds with idempotent mapping
## Sigma0 Request Message

<table>
<thead>
<tr>
<th>Requested attributes</th>
<th>MR₂</th>
<th>MR₁</th>
<th>MR₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>B = Requested Fpage/1024</td>
<td>s(6)</td>
<td>0rw(4)</td>
<td></td>
</tr>
<tr>
<td>-6</td>
<td>0(4)</td>
<td>0(4)</td>
<td>0(6)</td>
</tr>
</tbody>
</table>

- **Requested attributes**
  - **Architecture specific**
    - Use default = 0
- **Requested Fpage**
  - B != -1
    - Request a specific region of physical memory
Some Sigma0 Helpers

- Found in l4/sigma0.h

- Request a specific page
  \[
  L4_Fpage_t L4_Sigma0_GetPage (L4_ThreadId_t s0, 
  \quad L4_Fpage_t f, 
  \quad L4_Fpage_t RcvWindow)
  \]

- Request some page
  \[
  L4_Fpage_t L4_Sigma0_GetAny (L4_ThreadId_t s0, 
  \quad L4_Word_t s, 
  \quad L4_Fpage_t RcvWindow)
  \]
Example Code

- Kernel
  - L4Ka::Pistachio 0.4
  - ia32-kernel
- Supporting applications
  - Kickstart
  - Sigma0
- Custom applications
  - Roottask
  - Test client
Custom Applications

- Roottask
  - Locator
  - Log server
  - Pager
- Test Client
  - Locates log server
  - Sends a message to the log server

Diagram:
- testclient
- roottask
- Pager
- Locator
- Logger
- sigma0
- ia32-kernel
The Boot Sequence (1)

- BIOS loads the boot block
  - Small loader, GRUB stage1
  - Only 512 bytes available
- Stage1 starts
  - Searches disks for stage2
  - Loads GRUB stage2
The Boot Sequence (2)

- **Stage2**
  - more complex part of GRUB
    - Understands various file systems
    - Supports network
    - Supports a menu
    - 60kb – 80kb in size
    - Supports ELF loading
  - Searches for menu.lst
The Boot Sequence (3)

- Stage 2 continues
  - ELF-loads kickstart
  - Appends additional modules after kickstart
  - Generates multiboot info
  - Passes pointer to multiboot info to kickstart
  - Hands over execution to kickstart
The Boot Sequence (4)

- Kickstart starts
  - Converts multiboot info into generic bootinfo
- ELF-loads
  - L4 kernel
  - Sigma 0
  - Roottask
- Configures L4 via Kernel Configuration Page
- Hands over control to kernel
The Boot Sequence (5)

- **L4 starts**
  - allocates some upper memory for internal data structures
  - starts sigma0 and roottask

- **Roottask starts**
  - parses generic bootinfo
  - ELF-loads test client
  - starts test client
The Directory Structure

- /include
  - Global header files
  - Subdirectories
    - /l4 - L4 systemcalls
    - /sdi - Header for our system library
- /if
  - Global interface descriptions
- /lib/sdi
  - Our system library code
- /src
  - Custom application code
Next Week

- Tuesday: OS Interfaces
- Thursday: IDL4, Debugging on L4

Homework

- Change your group’s password in the lab (R.149)
  - We will lock accounts with default passwords on Thursday
- Get your build environment going
  - See the SDI Wiki at http://i30www.ira.uka.de/~sdi/wiki/
- Create two threads in the roottask’s address space
  - Let them send a few untyped words back and forth
- Create an additional testclient-like binary
  - Run it in its own address space
- Also available as assignment01.pdf