MEMORY

MICHAEL ROITZSCH
• Introduction
  - Monolithic vs. microkernels
  - L4 concepts: Threads and IPC
  - Fiasco.OC/TUDOS introduction

• Today: Memory Management
  - Task creation
  - Page-fault handling
  - Flexpages
  - Hierarchical pagers
  - Region manager
  - Dataspaces
TASK CREATION
Thread needs to access code/data/stack/... to execute

- Kernel Area
- Stack
- Code
- Data

0

- ESP
- EIP

- Moe
  - Root Task

- Sigma0
  - Root Pager

- Fiasco.OC
  - Microkernel
/* Create a new task. */

l4_msgtag_t
L4::Factory::create_task (Cap< Task > const & task_cap,
l4_fpage_t const & utcb_area,
l4_utcb_t
*utcb = l4_utcb())

/
/* Create a new thread. */

l4_msgtag_t
L4::Factory::create_thread (Cap< Thread > const & target_cap,
l4_utcb_t
*utcb = l4_utcb())
/* Commit the given thread-attributes object. */

l4_msgtag_t
L4::Thread::control (Attr const & attr)

/* Exchange basic thread registers. */

l4_msgtag_t
L4::Thread::ex_regs (l4_addr_t ip, /* instruction pointer */
    l4_addr_t sp, /* stack pointer */
    l4_umword_t flags,
    l4_utcb_t *utcb = l4_utcb()}

)
4GB

Kernel Area

3GB

0

L4::Thread::ex_regs

Moe
Root Task

Sigma0
Root Pager

Fiasco.OC
Microkernel

ESP

EIP
CPU tries to fetch instruction
→ Page fault exception at EIP

Kernel Area

Moe
Root Task

Sigma0
Root Pager

Fiasco.OC
Microkernel
PAGE FAULT HANDLING
Page fault exception is caught by kernel page-fault handler

No management of user memory in kernel

Invoke user-level memory management → Pager
• Thread which is invoked on page fault
• Fiasco.OC: each thread has a (potentially different) pager assigned
• Communication with pager thread using IPC
• Kernel page fault handler sets up IPC to pager
• Pager sees faulting thread as sender of IPC
PAGE-FAULT IPC

<table>
<thead>
<tr>
<th>UT CB[0]</th>
<th>fault address / 4 (30)</th>
<th>w</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>UT CB[1]</td>
<td>faulting EIP (32)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( w = 0 \) read page fault
- \( w = 1 \) write page fault
- \( p = 0 \) no page present
- \( p = 1 \) page present
- Pager maps pages of its own address space to the application's address space
- Flexpage IPC enables these mappings
• map() creates an entry in the receiver's address space pointing to the same page frame
  – In hardware: page table entry

• Only valid pager address space entries can be mapped
• Special case: grant pages (flag: L4_FPAGE_GRANT)

→ Removes mapping from sender's address space
• Special case: grant pages (flag: L4_FPAGE_GRANT)
  ➔ Removes mapping from sender's address space
  ➔ **ATTENTION:** aliases remain
• Removes entries to a page frame (fpage is specified in invoker's address space)

• Dedicated system call: do not need partner's consent

→ Kernel tracks mappings in a database
FLEXPAGES
• Flexpages represent resources attached to an address space

• Flexpages in Fiasco.OC are used to describe:
  • Memory pages
  • I/O ports
  • Capabilities

• Today: only flexpages for memory
- Size-aligned
- Sizes are powers of two → $2^{size}$, smallest is hardware page
- Source and target area of a map IPC are described by flexpages
• Send flexpage is smaller than the receive window
  • Target position is derived from send flexpage alignment and send base
• Send flexpage is larger than receive window
  • Target position is derived from receive flexpage alignment and send base
→ Send base depends on information about the receiver

\begin{itemize}
\item \texttt{l4_ipc_send(...)}
\item \texttt{l4_ipc_receive(...)}
\end{itemize}
• Kernel page fault handler sets receive window to whole address space
  → Pager can map more than just one page, where the page fault happened to the client
- Pages are mapped as they are needed
  \[ \rightarrow \text{demand paging} \]
• Initial pager can only implement basic memory management
• No knowledge about application requirements
  • Different requirements at the same time
• Missing services for advanced memory management
  • e.g. no disk driver for swapping
• Build more advanced pagers on top of the initial one
→ Pager hierarchy
HIERARCHICAL PAGERS
• L⁴Linux implements Linux paging policy
• RT pager implements real-time paging policy (e.g. no swapping)
REGION MANAGER
- Pager has to specify send base
- Pager needs to know client's address space layout
  - No problems with only one pager (e.g. L4Linux)
- Possible conflicts if more than one pager manages an address space:

→ Virtual memory must be managed independent of pagers
• Per address space map that keeps track which part of the address space is managed by which pager

Address space

Pager 1

Pager 2

Pager 2

Region Map

<start, end>
<start, end>
<start, end>

Pager 1
Pager 2
Pager 2
• Intermediate pager that identifies which pager should handle a page fault
• Resides in the application's address space
→ Region manager is the pager of all threads of a task
• Region manager calls the pager that is responsible
• Receive window gets restricted to the area managed by that pager
→ No interference between different pagers
• Memory management in terms of pages so far
• Application’s view to memory:
  • code / data sections
  • memory mapped files
  • anonymous memory (heaps, stacks, …)
  • network / file system buffers
  • …

→ Abstraction to map this view to low-level memory management
DATASCAPES
• Dataspase: *unstructured data container*
• Abstraction for anything that contains data:
  • Files
  • Anonymous memory
  • I/O adapter memory
  • ...
• Dataspaces are implemented by *Dataspase Managers*
• Dataspaces can be attached to regions of an address space
• DS Manager determines the semantic of a dataspace
• Each DSM is the pager for its dataspaces
  → Implements the paging policy (page replacement etc.)
• Region map keeps track which dataspaces are attached to which virtual memory regions
• Region manager translates page faults to dataspace offsets

<table>
<thead>
<tr>
<th>VM Region</th>
<th>Dataspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;start, end&gt;</td>
<td>Dataspaces Manager 1, 1</td>
</tr>
<tr>
<td></td>
<td>Dataspaces Manager 2, 1</td>
</tr>
<tr>
<td></td>
<td>Dataspaces Manager 2, 2</td>
</tr>
</tbody>
</table>
- Region manager propagates fault to dataspace manager's fault handler
  → Dataspase fault (ds_manager_id, ds_id, offset)
• allocate / free dataspaces
  • create / destroy dataspace
  • semantic depends on dataspace type:
    • anonymous memory: open (size)
    • file: open (filename, mode, ...)
    • ...

• attach / detach dataspace
  • create / remove entry in region map
    ➔ Makes dataspace contents accessible to application

• propagate capability
  • grant access rights to other applications
    ➔ very easy shared memory implementation
• Application address spaces are constructed from several dataspaces:

- Application’s Address Space
- Mmap’ed File
- Code
- RO Data
- Data
- BSS
- Stack
- File System
- Files-System Buffers
- Memory Manager
- Memory
• Page Allocation Algorithms
  • List-based algorithms, bitmaps, trees, ...
• Page Replacement Algorithms
  • Least-Recently-Used (LRU)
  • Working Sets
  • Clock
  • ...
→ Page allocation and replacement are implemented by dataspace managers
→ Can have different strategies for the dataspaces of an application
• Memory sharing important for
  • Shared libraries
  • Data transfer between system components
  • ...

• Different types of sharing
  • Full sharing: all clients see modifications
    → easy to implement, pager / dataspace manager
      grants access rights to pages / dataspaces
  • Lazy copying of dataspaces
    → copy-on-write
• Closer look on tasks/threads:
  • Creation
  • Page-fault handling

• Flexpages
  • Memory pages, I/O ports, Capabilities
  • Structure
  • Offset computation

• Pager hierarchy

• Region manager & dataspaces
• Flexpages
  H.Härtig, J.Wolter, J.Liedtke: "Flexible sized page objects",
  http://os.inf.tu-dresden.de/papers_ps/flexpages.pdf

• Dataspaces
  Mohit Aron, Yoonho Park, Trent Jaeger, Jochen Liedtke,
  papers/Aron_PJLED_01.ps.gz