Operating Systems

Memory management

Stephan Sigg

Distributed and Ubiquitous Systems
Technische Universität Braunschweig

February 4, 2011
Overview and Structure

- Introduction to operating systems
  - History
  - Architectures
- Processes
  - Processes, Threads, IPC, Scheduling
  - Synchronisation
  - Deadlocks
- Memory management
  - Paging
  - Segmentation
- Filesystems
- Security and Protection
- Distributed systems
- Cryptography
Outline
Memory management

1 Memory management
- Background
- Swapping
- Contiguous memory allocation
- Paging
- Structure of the page tree
- Segmentation
- Summary
Memory management

Introduction

- We have seen that the CPU can be shared by processes
- To enable this, several processes have to be kept in memory
- We have to share memory
- Memory management strategies are required
Outline

Memory management

1 Memory management
- Background
- Swapping
- Contiguous memory allocation
- Paging
- Structure of the page tree
- Segmentation
- Summary
Memory management

Background

- Memory is central to the operation of a modern computer system
- Memory consists of a large array of words or bytes
- Each with its own address
Memory management

Background

- CPU fetches instructions from memory according to value of program counter
- Memory unit sees only a stream of memory addresses
- It does not know how they are generated or what they are for
Memory management

Background

- Datapath of a simple 'DLX'-datapath

\[\text{Hennessey and Patterson, Computer Architecture, a quantitative approach, Morgan Kaufmann, 2002}\]
Memory management

Background

- Pipelining of the DLX datapath²

²Hennessey and Patterson, Computer Architecture, a quantitative approach, Morgan Kaufmann, 2002
Memory management

Background

Basic hardware

- Main memory and CPU registers are accessed directly by the CPU.
- Machine instructions, however, do not contain disk addresses.
- Data must be moved into registers or memory before the CPU can operate on it.
Memory management

Background

Basic hardware

CPU registers are generally accessible within one CPU clock cycle

Main memory is accessed via a transaction on the memory bus

Memory access may take many CPU cycles
Memory management

Background

Basic hardware

- At memory access, the CPU would then usually be forced to stall
- This situation is unacceptable
- One solution can be to introduce fast memory between CPU and main memory (e.g. a cache)
Memory management
Background

Basic hardware

- User processes and OS-processes have to be protected against each other
- Else, user process might overwrite memory, cache or register contents of another process
- This protection must be provided by hardware
- Each process is allocated a separate memory space
  - Base register holds the smallest legal physical memory address
  - Limit register specifies the range
Memory management

Background

Basic hardware

- Base register and limit register for each process is stored in registers
- Memory access of processes is controlled by CPU hardware
- An attempt to access other processes memory results in a fatal error
- Access to the values of base and limit registers is possible only in kernel mode
- Therefore, only the operating system can modify the registers
Memory management

Background

Address binding

- Programs usually reside on disk as binary executable files
- For execution, programs are brought into memory and placed within a process
- The process might be moved between memory and disk during execution
- Processes on disk that wait for execution are placed into the input queue
- When a process is finished, its allocated memory space is declared available
Memory management

Background

Address binding

- Processes can reside at arbitrary places in the physical memory
- The addresses used by such a process might be symbolic or absolute
- Symbolic: E.g.: 14 bytes from the beginning of this module
- Linkage editor will bind these addresses to absolute addresses (e.g. 74006)
Memory management

Background

Logical versus physical address space

- Addresses generated by the CPU are typically logical addresses
- A hardware device (the memory-management unit) maps virtual to physical addresses
- One possible approach is a relocation register that adds a value to the logical address
- User programs operate on logical addresses
Memory management

Dynamic loading

- With increasing program size, it may become infeasible to load the whole program into memory.
- With dynamic loading, a routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format.
- The operating system may help the programmer to support dynamic loading by providing library routines.
Outline
Memory management

1 Memory management
- Background
- Swapping
- Contiguous memory allocation
- Paging
- Structure of the page tree
- Segmentation
- Summary
Memory management

Swapping

- Processes must resign in memory to be executed
- A CPU-scheduling algorithm may temporarily remove processes from memory and store them to the ready queue on disk
- Normally, processes are swapped back into the same memory space they occupied previously
- This restriction is dictated by address binding
The context switch time is considerably high:

**Example**

- Assume a user process of 100 MB
- Assume a transfer rate of 50 MB from disk to memory
- The actual transfer time is then
  \[
  \frac{100\text{MB}}{50\frac{\text{MB}}{\text{second}}} = 2\text{seconds}
  \]
- With an average latency of 8 ms the swap time is 2008 ms
- The total swap time (in/out) is then 4016 ms
Other factors that impact swapping:

- Pending I/O
  - A process waiting for I/O operation cannot be swapped
  - Otherwise, the I/O operation might then attempt to use the memory that now belongs to a different process

- Swapping time of standard swapping is considerably high
  - Therefore, many actual operating systems utilise modified versions
  - In many UNIX versions, swapping is only utilised when a considerable number of processes is active and else disabled
Outline

Memory management

1. Memory management
   - Background
   - Swapping
   - Contiguous memory allocation
   - Paging
   - Structure of the page tree
   - Segmentation
   - Summary
Memory management
Contiguous memory allocation

- The main memory is typically divided into regions for the operating system and for user processes
- User processes are dynamically allocated
Memory management
Contiguous memory allocation

Memory allocation

- Memory allocation strategies:
  - **Multiple partition method**: Divide memory into fixed-size partitions that are allocated to processes (e.g. in OS/360 operating system).
  - **Variable partition method**: Operating system keeps table of available and occupied segments.
    - New processes are put into an input queue. OS takes expected memory requirements into account (e.g. in batch systems)
    - Scheduling algorithm can order the input queue
    - Memory allocated to processes until the first process requires more memory than available.
Memory management

Contiguous memory allocation

Memory allocation

- The variable partition method is an instance of the dynamic storage allocation problem
  - A request of size $n$ is to be satisfied from a list of free holes
- Example solutions to solve this problem
  - First fit:
    - Allocate first hole big enough
    - Minimises search time
  - Best fit:
    - Allocate smallest hole that is big enough
    - Produces the smallest possible 'leftover-hole'
  - Worst fit:
    - Allocate largest hole
    - Produces the largest possible 'leftover-hole'
Memory management
Contiguous memory allocation

Memory allocation

- **Performance measures**
  - In simulations, first fit and best fit outperform worst fit in storage utilisation
  - First fit and best fit utilise storage equally good
  - First fit is generally faster
  - Worst fit is less seriously affected by fragmentation

<table>
<thead>
<tr>
<th></th>
<th>0-7</th>
<th>8-15</th>
<th>16-23</th>
<th>24-31</th>
<th>32-39</th>
<th>40-47</th>
<th>48-55</th>
<th>56-63</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>C</td>
<td></td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>
Memory management

Contiguous memory allocation

Fragmentation

- First fit and best fit strategies suffer from external fragmentation
  - External fragmentation: The data item is smaller than the allocated memory region
- Free memory space is broken into small pieces
- 50-percent rule: Statistical analysis of first fit revealed, that from $N$ allocated blocks another $0.5N$ blocks are lost to fragmentation.
Memory management
Contiguous memory allocation

Fragmentation
- When memory holes are arbitrary, the cost to keep track of many arbitrary sized holes might be undesirable
- Therefore, standard block sizes are used
- Processes are allocated memory segments that are slightly larger than the required memory
- This difference is called internal fragmentation
Memory management
Contiguous memory allocation

Fragmentation
- A solution to external fragmentation is compaction
- Place all memory contents next to each other to avoid memory holes
- Compaction is only possible, if relocation of processes is dynamic (and not static)
- Another solution is to permit the allocated memory space to be noncontiguous
Outline
Memory management

1 Memory management
- Background
- Swapping
- Contiguous memory allocation
- Paging
- Structure of the page tree
- Segmentation
- Summary
Memory management

Paging

- Paging is a memory-management scheme that allows for noncontiguous physical address spaces
- Paging avoids external fragmentation
- Paging is used in most operating systems
- Traditionally, paging was supported by hardware
- In recent designs, paging is implemented by integrating the hardware and operating system
Memory management

Paging

Basic method

- Physical memory is separated into fixed-sized blocks (frames)
- Logical memory is separated into blocks of the same size (pages)
- When a process is executed, its pages are loaded into any available memory frames
- Every address generated by the CPU is divided into
  - page number (p)
  - page offset (d)
Memory management

Paging

Basic method

- Logical address separated into page number and page offset
  - Logical address space: $2^m$
  - Page size: $2^n$
  - Page number: $2^{m-n}$
Memory management

Paging

Basic method

- With paging, no external fragmentation occurs
- Any free frame can be allocated to a process
- Internal fragmentation may occur
- With smaller page sizes, the wasted space is potentially minimised
- However, the overhead to maintain each page-table entry also increases with smaller page sizes
Memory management

Paging

Basic method

- Over time, page sizes have grown with memory sizes
- Today, 4KB or 8KB page size are typical
Memory management

Paging

Basic method

- When a process arrives at the system, its size in pages is estimated
- If sufficient frames are available in memory, they are allocated
- Processes are separated:
  - A process has no way of accessing memory outside of its own page table
  - A page table includes only those pages the process owns

\[\text{Page number} \quad \text{Page offset}\]
\[\begin{array}{c|c}
p & d \\
\hline
m-n & n \\
\end{array}\]
Memory management
Paging

Basic method
- The operating system is always aware of the allocation of physical memory.
- A frame table indicates for each physical page frame whether it is allocated and to which page of which process.
Memory management

Paging

Hardware support

- Most operating systems allocate a page table for each process
- A pointer to the table is stored in the PCB
- Hardware implementation in several ways possible
  - Dedicated registers
  - Stored in main memory
  - Transaction look-aside buffer (TLB)
Hardware support – Dedicated registers

- A page table can be implemented as a set of dedicated registers
- High-speed logic to make the paging-address translation efficient
  - Every memory access must go through the paging map
- Example: DEC PDP-11
  - 16 bit addresses
  - page size 8KB
Memory management

Paging

Hardware support – Main memory

- use of registers for the page table feasible only for small page tables (e.g. 256 entries)
- Modern computers allow larger page tables (e.g. 1 million entries)
- Page table is then kept in main memory
- page-table base register (PTBR) points to page table
- Short context switch time
  - Changing page table by changing this register only
- Long delay to access memory
  - Two memory accesses for each information
  - Speed reduction by factor 2
Memory management

Paging

Hardware support – TLB

- Standard solution to speedup access to page table:
  - Translation look-aside buffer (TLB)
  - High-speed memory
  - Each entry consists of key and value
  - Comparison of keys is possible simultaneously
  - Fast access

- Expensive hardware
Memory management

Paging

Hardware support – TLB

- Translation look-aside buffer stores some entries of the page table
- Entry is first searched in the TLB
- If successful, the frame number is available immediately
- **Hit ratio**: percentage of times that a particular page number is found
Memory management

Paging

Example

- 20 ns to search TLB
- 100 ns for memory access
- hit ratio: 80%

\[
effective\ access\ time = 0.8 \cdot 120\text{ns} + 0.2 \cdot 220\text{ns} = 140\text{ns}
\]

- hit ratio: 98%

\[
effective\ access\ time = 0.98 \cdot 120\text{ns} + 0.02 \cdot 220\text{ns} = 122\text{ns}
\]
Memory management

Paging

Paging with TLB

Stephan Sigg Operating Systems 45/69
Memory management

Paging

Protection

- Memory protection might be necessary when not the whole logical address space is used by a process
- Valid-invalid bits are utilised to indicate invalid frame numbers in the page table
Memory management

Paging

Shared pages

- Due to paging, common code can be shared
  - Reentrant code: non-self-modifying code
  - Two or more processes can access the same code simultaneously

- Sharing of processes reduces the memory required in a system with multiple processes and users
Memory management

Paging

Shared pages

```
<table>
<thead>
<tr>
<th>Logical memory</th>
<th>Process P1</th>
<th>Process P2</th>
<th>Process P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>editor 1.1</td>
<td>editor 1.1</td>
<td>editor 1.1</td>
<td>editor 1.1</td>
</tr>
<tr>
<td>editor 1.2</td>
<td>editor 1.2</td>
<td>editor 1.2</td>
<td>editor 1.2</td>
</tr>
<tr>
<td>editor 1.3</td>
<td>editor 1.3</td>
<td>editor 1.3</td>
<td>editor 1.3</td>
</tr>
<tr>
<td>data P1</td>
<td>data P2</td>
<td>data P3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>page table</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>editor 1.1</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>editor 1.2</td>
<td>4</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>editor 1.3</td>
<td></td>
<td>6</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>data P1</td>
<td></td>
<td></td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>editor 1.3</td>
</tr>
<tr>
<td>data P3</td>
</tr>
<tr>
<td>editor 1.1</td>
</tr>
<tr>
<td>editor 1.2</td>
</tr>
</tbody>
</table>
```

Physical memory:

```
<table>
<thead>
<tr>
<th>Frame number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>data P1</td>
</tr>
<tr>
<td>data P3</td>
</tr>
<tr>
<td>editor 1.1</td>
</tr>
<tr>
<td>editor 1.2</td>
</tr>
<tr>
<td>editor 1.3</td>
</tr>
<tr>
<td>data P2</td>
</tr>
</tbody>
</table>
```
Outline
Memory management

1 Memory management
- Background
- Swapping
- Contiguous memory allocation
- Paging
- Structure of the page tree
- Segmentation
- Summary
Memory management

Structure of the page tree

- The page table can be structured in various ways:
  - Hierarchical Paging
  - Hashed page tables
  - Inverted page tables
Memory management

Structure of the page tree

Hierarchical paging

- Modern computer systems have large logical address spaces (e.g. $2^{32}$, $2^{64}$)
- The page table then becomes excessively large
  - With a 4KB ($2^{12}$) page size, the page table may contain
    \[ \frac{2^{32}}{2^{12}} = 2^{20} \] entries
  - When each entry consists of 4 bytes, a single entry may require 4MB of physical space
- A solution is to split the page table into several sub-page tables
- These have smaller addresses and are loaded only when required
Memory management
Structure of the page tree

Hierarchical paging
- Logical address divided into
  - Page number outer page table
  - Page number page table
  - Page offset
Memory management
Structure of the page tree

Hierarchical paging

- With greater address space, more hierarchy stages possible
- Example: the 64-bit UltraSPARC requires 7 levels of paging
- With additional stages the number of memory accesses increases
- Therefore, hierarchical page tables are then generally considered inappropriate
Memory management

Structure of the page tree

Hashed page tables

- A common approach to handle large address spaces:
  - Use hashed page table
  - Hash values are the virtual page numbers
  - Each entry contains linked list of elements that hash to the same location
    - to handle collisions
Memory management

Structure of the page tree

Hashed page tables

- Three entries in the linked lists of elements of the hashed page table:
  1. Virtual page number
  2. Value of the mapped page frame
  3. Pointer to the next element in the list
Memory management
Structure of the page tree

Hashed page tables

[Diagram showing logical address mapping to physical address through hash table]
Memory management
Structure of the page tree

Hashed page tables

- Algorithm:
  1. Virtual page number in the virtual address is hashed into the hash table
  2. Virtual page number is compared with field 1 in the first element of the linked list
  3. If match: corresponding page frame is used to form desired physical address
  4. Else: Continue with subsequent entries in the list

- Variation: Cluster fixed number of pages (e.g. 16) together in the same manner

- Method useful for sparse address spaces
Memory management
Structure of the page tree

Inverted page tables

- Normally, each process owns its own page table
- Since these page tables may become very large, large amounts of physical memory may be allocated only to page tables
- A possible solution is an inverted page table
  - One entry for each real page (frame) of memory
  - Each entry consists of virtual address of the page and information about the process that owns the page
  - Only one page table then exists in the system
Memory management
Structure of the page tree

Inverted page tables
Memory management

Structure of the page tree

Inverted page tables

- Issues with inverted page tables
  - Shared memory is more complicated to implement
  - Since there is only one virtual page entry, one physical page cannot have several virtual addresses
  - To allow shared memory, more advanced paging strategies are required

![Diagram of page table and memory mapping]
Outline

Memory management

1. Memory management
   - Background
   - Swapping
   - Contiguous memory allocation
   - Paging
   - Structure of the page tree
   - Segmentation
   - Summary
Memory management

Segmentation

- Due to paging, the user’s view of memory might differ from the actual physical memory
- Logical memory and physical memory differ
- Segmentation is a memory-management scheme that supports this user view of memory
Memory management

Segmentation

Basic method

- A logical address space is a collection of segments
- Each segment has a name and a length
- The addresses specify the segment name and offset within the segment
- Segments are numbered and referred to by a segment number

Logical address:

\(< \text{segment} - \text{number}, \text{offset} >\)
Memory management

Segmentation

Basic method

- Normally, a compiler automatically constructs segments reflecting the input program, e.g.
  - Code
  - Global variables
  - Heap for memory allocation
  - Stacks used by each thread
  - Standard C library

- The loader takes these segments and assigns them segment numbers
Memory management

Segmentation

Hardware

- Two dimensional logical addresses have to be mapped to one-dimensional physical addresses
- This is done by a segment table
  - base : Start of the entry
  - limit : segment length
Memory management
Segmentation

Hardware

segment 0
segment 1
segment 2
segment 3
segment 4

sqrt
main program
stack
symbol table
subroutine

segment table

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>6300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>4300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>3200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>4700</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical memory

segment 0
segment 1
segment 2
segment 3
segment 4
Memory management

Summary

- Swapping
- Contiguous memory allocation
- Paging
- Structure of the page tree
- Segmentation
Questions?
Literature
Recommended literature