Principper for Samtidighed og Styresystemer

Memory Management

René Rydhof Hansen

11 MAR 2019

Learning Goals

After the last lecture you

- ... can explain the notion of limited direct execution and how it relates to scheduling
- ... will know the simplified process model
- ... will know and can explain important metrics for measuring a scheduling policy:
  - Fairness
  - Turnaround time
  - Response time
- ... can explain important scheduling policies and their pros and cons
  - FIFO
  - SJF
  - STCF
  - Round robin
  - MLFQ
  - Lottery scheduling

Learning Goals

After today's lecture you

- ... will know and can discuss the three goals of memory management
  - Transparency
  - Efficiency
  - Protection (isolation)
- ... can explain what an address space is
- ... define and explain the notion of virtual memory
- ... perform simple address translation from virtual to physical
- ... can explain the need for and use of base/bound registers
- ... define and explain the use of segmentation

Simple Memory Management

Memory management: Goals and Assumptions

Goals

- Transparency
- Efficiency
- Protection

Definition (Address Space)

Running program's (abstract) view of memory

Simplifying Assumptions

- Contiguous allocation
- Small address space (smaller than physical memory)
- Fixed size address spaces

A Note on Efficiency: Memory Hierarchy

Access times for memory

- CPU registers (< 1 cycle)
- Internal cache (1 cycle)
- Secondary (external) cache (5 cycles)
- Tertiary cache (if any) (10 cycles)
- Physical memory (25–50 cycles)
- Swap disk and virtual memory ($\sim$ 1.000.000 cycles)

Exercise

Compare to service in a restaurant (assume 10 min. normally)...

A Note on Efficiency: Memory Hierarchy

Access times for memory

- CPU registers (< 1 cycle)
- Internal cache (1 cycle)
- Secondary (external) cache (5 cycles)
- Tertiary cache (if any) (10 cycles)
- Physical memory (25–50 cycles)
- Swap disk and virtual memory ($\sim$ 1.000.000 cycles)

Exercise

Compare to service in a restaurant (assume 10 min. normally)... 19+ years

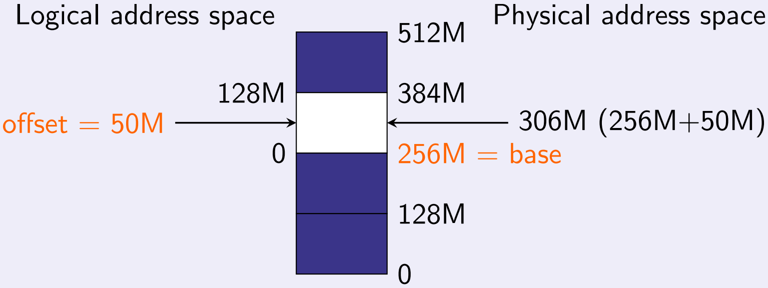
Simple virtual addresses: Relative addresses

Relative Addresses

A logical address defined relative to a known base address

- Base address is a known physical address
- Relative address is the offset to the base address
- $\mathtt{PHY} = \mathtt{BASE} + \mathtt{VIRT}$

Example (Address translation: relative addresses)



Relocation, Protection, and Sharing

Abstracting away addresses

- Remove dependence on physical location
- Introduce a layer of indirection

Physical Addresses (aka. absolute addresses)

Defines specific physical memory cell in primary memory

Virtual Addresses (aka. logical addresses)

References to memory location independently of physcial location

- Indirection ("Every software problem can be solved by adding another layer of indirexction" (Steven M. Bellovin))
- Virtual addresses are mapped to physical addresses at runtime
- Requires hardware support (MMU); often (always) integrated into CPU
- Virtual addressing is transparent to the programmer

How to Manage Memory for an Operating System?

Challenges

- How can processes (code) be relocated in memory?
- How are processes protected from each other?
- How is the operating system isolated from processes?
- How can (primary) memory be shared between processes?
- How are processes organised in primary memory?
- How can programmers use memory optimally?
- What if a process requires more memory than physically available?
- How is fragmentation avoided in primary memory?

How to Manage Memory for an Operating System?

Challenges

- How can processes (code) be relocated in memory?
- How are processes protected from each other?
- How is the operating system isolated from processes?
- How can (primary) memory be shared between processes?
- How are processes organised in primary memory?
- How can programmers use memory optimally?
- What if a process requires more memory than physically available?
- How is fragmentation avoided in primary memory?

1. Relocation
2. Protection
3. Sharing
4. Logical organisation
5. Physical organisation

How to Manage Memory for an Operating System?

Challenge: Relocation

- Where to (re-)load a process/swapping
- Fixed addresses (at compile time)
- Fixed addresses (at link/load time)
- Dynamic addresses (dynamic rewriting)

Challenge: Protection/Isolation

- Protect processes/OS from interference
  - Deliberate
  - Accidental
- All memory references generated at run-time must be checked
- Special hardware needed(?)

How to Manage Memory for an Operating System?

Challenge: Sharing

- Shared data
  - Essential for interprocess communication (IPC)
- Shared code

Challenge: Logical organisation

- Should be easy for programmers (abstraction)
- Independent on underlying platform
  - Independent modules/libraries/plug-ins/...
  - Facilitate code sharing

Challenge: Physical organisation

- Managing (limited) primary memory
- When to switch between primary and secondary memory: OS decision

How to Manage Memory for an Operating System?

Definition (Address Space)

An address space is a set of addresses. Example: The set of CPU addresses adressable by the CPU is an address space

Simple Solution?

- Put all processes in a common address space
- Make all memory references in program code absolute
- Let the OS re-write all memory references to unique addresses

Problems

How to Manage Memory for an Operating System?

Definition (Address Space)

An address space is a set of addresses. Example: The set of CPU addresses adressable by the CPU is an address space

Simple Solution?

- Put all processes in a common address space
- Make all memory references in program code absolute
- Let the OS re-write all memory references to unique addresses
- Example: Singularity experimental kernel

Problems

- Program code can not be shared between processes
- Processes are not really isolated (or are they?)

Relocation, Protection, and Sharing

Relocation, Protection, and Sharing

Abstracting away addresses

- Remove dependence on physical location
- Introduce a layer of indirection

Physical Addresses (aka. absolute addresses)

Defines specific physical memory cell in primary memory

Virtual Addresses (aka. logical addresses)

References to memory location independently of physcial location

- Indirection ("Every software problem can be solved by adding another layer of indirexction" (Steven M. Bellovin))
- Virtual addresses are mapped to physical addresses at runtime
- Requires hardware support (MMU); often (always) integrated into CPU
- Virtual addressing is transparent to the programmer

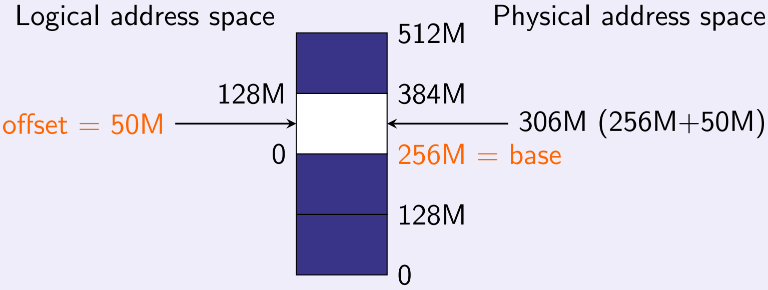
Simple virtual addresses: Relative addresses

Relative Addresses

A logical address defined relative to a known base address

- Base address is a known physical address
- Relative address is the offset to the base address
- The physical address is the sum of base address and offset

Example (Address translation: relative addresses)



Address Translation

Example

```
a = a + b

LDR r1,0        ; read cell 0 to register r1
LDR r2,4        ; read cell 4 to register r2
ADD r1,r1,r2    ; r1 = r1 + r2
STR 0,r1        ; store result in cell 0
```

- If 0 and 4 are logical addresses, they are translated to physical addresses by the MMU
- For different processes, this will result in different physical addresses
- Facilitates code relocation, protection, and sharing of program code between processes (what about data?)
- What about logical and physical organisation?

Simple allocation

Static allocation

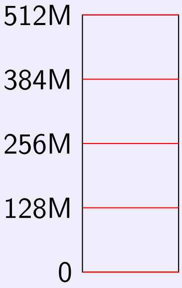
Definition (static allocation)

Allocation in blocks of pre-determined fixed size

Easiest: uniform, fixed size

- Every block (partition) has same size

Example (Uniform block size)



Pro's and Con's

- Easy to implement/manage
- Little/no OS overhead

- Internal fragmentation (a lot)
- Inflexible (fixed no of proc.)
- Big programs problematic (overlay)

Internal fragmentation

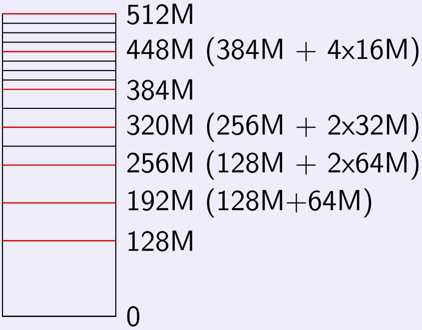
Unused space inside a block

Static allocation

Non-uniform block size

- Split memory into blocks of different size

Example (Non-uniform: 128MB+2x64MB+4x32MB+8x16MB)



Static allocation

Pro's and Cons': Non-uniform block size

- Requires strategy for allocation of processes to blocks
- Smallest-fit
  - May result in processes waiting for blocks of the “appropriate” size
- Smallest-available
  - May result in more internal fragmentation
- Problem: memory requirement must be known beforehand

Summary: Static allocation

- Main challenges: block size, process placement
- Main advantages: easy to implement, little overhead
- Main disadvantages: inflexible, inefficient

Dynamic allocation

Definition

- Allocation in blocks dynamically sized during load-time
- OS allocates appropriate block
  - OS maintains start address and length
  - Processes must specify memory requirements at start-up

Finding an appropriate block (placement strategy)

- Placement strategies
  - Best-fit: find block that is closest in size
  - First-fit: find first block big enough
  - Next-fit: find first block big enough, after last placed block
- Best strategy?

Dynamic allocation

Definition

- Allocation in blocks dynamically sized during load-time
- OS allocates appropriate block
  - OS maintains start address and length
  - Processes must specify memory requirements at start-up

Finding an appropriate block (placement strategy)

- Placement strategies
  - Best-fit: find block that is closest in size
  - First-fit: find first block big enough
  - Next-fit: find first block big enough, after last placed block
- Best strategy?
  - Depends, but usually first-fit (fastest, best performing)
  - ... next-fit (slightly worse performance)
  - ... best-fit (more fragmentation)

Dynamic allocation

Pro's

- No (visible) internal fragmentation
- Adapts to current needs (few big or many small processes)

Con's

- Memory requirements must be known beforehand
- Compaction
  - Relocation of processes to reduce external fragmentation
  - Similar to de-fragmentation of harddisks
- Compaction without virtual memory requires re-writing of addresses in program code and data (pointers)

Definition (External fragmentation)

Unused memory between blocks