Memory

Johan Montelius

KTH

2018
The process

Memory layout for a 32-bit Linux process
64-bit Linux on a x86_64 architecture

```
0x00... 0x00007ff.. 0xfffff800.. 0xff...
```

- code
- data
- heap
- stack
- kernel
64-bit Linux on a x86_64 architecture

- code
- data
- heap
- stack
- not used
- kernel

Address ranges:
- Code: 0x00...
- Data: 0x00007ff..
- Heap: 0xffff800..
- Stack: 0xff...
- Kernel: 0xff...
Memory virtualization

Every process has an address space from zero to some maximal address. A program contains instructions that of course rely on that code and data can be found at expected addresses. We only have one physical memory. Let's start from the beginning.
Memory virtualization

Every process has an address space from zero to some maximal address.
Every process has an address space from zero to some maximal address.

A program contains instructions that of course rely on that code and data can be found at expected addresses.
Memory virtualization

Every process has an address space from zero to some maximal address.

A program contains instructions that of course rely on that code and data can be found at expected addresses.

We only have one physical memory.
Memory virtualization

Every process has an address space from zero to some maximal address.

A program contains instructions that of course rely on that code and data can be found at expected addresses.

We only have one physical memory.
Memory virtualization

Every process has an address space from zero to some maximal address.

A program contains instructions that of course rely on that code and data can be found at expected addresses.

We only have one physical memory.
IBM System 360

- 1964, 8-64 Kbyte memory
- 12+12 bit address space
- batch operating system

Chief architect: Gene Amdahl
when things were simple

Batch processing:

0x0000

0xffff
when things were simple

Batch processing:

0x0000

0xffff
when things were simple

Batch processing:

0x0000

0xffffffff
when things were simple

Batch processing:

0x0000

0xffff
when things were simple

Batch processing:

0x0000

0xffff
when things were simple

Batch processing:

0x0000

0xffff
when things were simple

**Batch processing:**

0x0000

0xffff
when things were simple

Batch processing:

0x0000

0xffff
when things were simple

**Batch processing:**

0x0000

0xffff
Arnold Spielberg was in the team that designed the GE-235

GE-235

- 1964
- 20-bit word
- 8 Kword address space
Time-sharing:

0x0000

0xffff
Time-sharing:

0x0000

0xffff
Time-sharing:

0x0000

0xffff
Time-sharing:

0x0000

0xffffffff
Time-sharing:
Time-sharing:

0x0000

0xffff
Time-sharing:

0x0000

0xffff
Time-sharing:

0x0000

0xffffffff
Time-sharing:

0x0000

0xffff
Time-sharing:

0x0000

0xffff
why not switch between two programs

If both programs will fit in memory:

0x0000

0xffff
why not switch between two programs

If both programs will fit in memory:

0x0000

0xffff
why not switch between two programs

If both programs will fit in memory:

0x0000

0xffff

What is the problem?
why not switch between two programs

If both programs will fit in memory:

0x0000

0xffffffff
why not switch between two programs

If both programs will fit in memory:

0x0000

0xffffff
why not switch between two programs

If both programs will fit in memory:

0x0000  

0xffff
why not switch between two programs

If both programs will fit in memory:

What is the problem?
Virtual memory

Physical memory

0x0000

0xffffffff
Virtual memory

Physical memory

0x0000

0xffff
Virtual memory

Physical memory

0x0000

0xffffffff

Transparent: processes should be unaware of virtualization.
Protection: processes should not be able to interfere with each other.
Efficiency: execution should be as close to real execution as possible.
Virtual memory

Physical memory

Process view

0x0000

0xffff
Virtual memory

Physical memory

0x0000

0xffff

Process view

0x0000

0x7fff

Transparent: processes should be unaware of virtualization.

Protection: processes should not be able to interfere with each other.

Efficiency: execution should be as close to real execution as possible.
Virtual memory

Physical memory

Process view

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.
Virtual memory

- **Physical memory**
  - 0x0000
  - 0xffffffff

- **Process view**
  - 0x0000
  - 0xffffffff

- **Transparent**: processes should be unaware of virtualization.

- **Protection**: processes should not be able to interfere with each other.

- **Efficiency**: execution should be as close to real execution as possible.
Virtual memory

- **Transparent**: processes should be unaware of virtualization.
- **Protection**: processes should not be able to interfere with each other.

Efficiency: execution should be as close to real execution as possible.
Virtual memory

- **Transparent:** processes should be unaware of virtualization.
- **Protection:** processes should not be able to interfere with each other.
- **Efficiency:** execution should be as close to real execution as possible.
Let the operating system run an emulator that interprets the operations of the process and changes the memory addresses as needed.
Emulator - simple but slow

Let the operating system run an emulator that interprets the operations of the process and changes the memory addresses as needed.

This is similar to how the JVM works
When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.
When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.
When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.
When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.
When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.
When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.
When a program is loaded, all references to memory locations are changed so that they correspond to the actual location in RAM where the program is loaded.

How do we know we have changed all addresses?
Dynamic relocation

Change every memory reference, on the fly, to a region in memory allocated for the process.
Dynamic relocation

Change every memory reference, on the fly, to a region in memory allocated for the process.
Dynamic relocation

Change every memory reference, on the fly, to a region in memory allocated for the process.

CPU \rightarrow \text{virtual addr} \rightarrow \text{MMU} \rightarrow \text{physical addr} \rightarrow \text{RAM}
Dynamic relocation

Change every memory reference, on the fly, to a region in memory allocated for the process.
Base register

MMU

virtual addr.  

base
Base register

MMU

virtual addr. \[ \rightarrow \] \[ + \] \[ \rightarrow \] physical address

base
Base problem

Who is allowed to change the base register?

How do we prevent one process from overwriting another process?
Who is allowed to change the base register?
Base problem

- Who is allowed to change the base register?
- How do we prevent one process from overwriting another process?
Who is allowed to change the base register?

How do we prevent one process from overwriting another process?
Base problem

- Who is allowed to change the base register?
- How do we prevent one process from overwriting another process?
Base problem

- Who is allowed to change the base register?
- How do we prevent one process from overwriting another process?

Can we prevent this at compile or load time?
Base and bound

MMU

virtual addr.
Base and bound

MMU

virtual addr.          

base
Base and bound

MMU

virtual addr. → base + physical address
Base and bound

MMU

virtual addr.

base

bound

physical address
Base and bound

MMU

virtual addr. → < bound → base + physical address
Base and bound

MMU

virtual addr. → < → yes → within bounds

+ → bound → physical address

base
Base and bound

MMU

virtual addr. \rightarrow bound \rightarrow physical address

\begin{align*}
\text{base} &+ \quad \text{bound} \\
\end{align*}

\begin{align*}
\text{exception} \quad \uparrow \\
\text{no} &\quad \text{yes} \\
\end{align*}

\rightarrow \text{within bounds}
Pros:

- Transparent to a process.
- Simple to implement.
- Easy to change process.

Cons:

- How do we share data?
- Wasted memory.
Pros:
- Transparent to a process.
- Simple to implement.
- Easy to change process.

Cons:
- How do we share data?
- Wasted memory.
shared read-only segments

Physical memory
shared read-only segments

Physical memory

Process A
shared read-only segments

Physical memory

Process A
shared read-only segments

Physical memory

Process A

How do we write code that can be shared?
shared read-only segments

Physical memory

Process A
shared read-only segments

Physical memory

Process A
shared read-only segments

Physical memory

Process A

Process B

How do we write code that can be shared?
How do we write code that can be shared?

Physical memory

Process A

Process B

shared read-only segments
How do we write code that can be shared?
shared read-only segments

How do we write code that can be shared?
Internal fragmentation

Physical memory
Internal fragmentation

Physical memory

Process view
Internal fragmentation

Physical memory

Process view
Internal fragmentation
Internal fragmentation

Physical memory

Process view

unused
Internal fragmentation

Physical memory

Process view

unused
Internal fragmentation

Physical memory

Process view

wasted

unused
Burroughs B5000

1961

Designed for high-level languages: ALGOL-60
Memory access through a set of segment descriptors, i.e., the view of a process is not a consecutive memory rather a set of individual memory segments.

Donald Knuth was part of the design team.
Burroughs B5000

- 1961
- Designed for high-level languages: ALGOL-60
1961

- Designed for high-level languages: ALGOL-60
- Memory access through a set of segment *descriptors* i.e. the view of a process is not a consecutive memory rather a set of individual memory segments.
1961

- Designed for high-level languages: ALGOL-60
- Memory access through a set of segment descriptors i.e. the view of a process is not a consecutive memory rather a set of individual memory segments.

*Donald Knuth was part of the design team.*
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
   value n, m; array a; integer n, m, i, k; real y;

comment The absolute greatest element of the matrix a ...

begin
   integer p, q;
   y := 0; i := k := 1;
   for p := 1 step 1 until n do
      for q := 1 step 1 until m do
         if abs(a[p, q]) > y then
            begin y := abs(a[p, q]);
               i := p; k := q
            end
   end Absmax
The view of the assembler programmer.

The view of the ALGOL programmer.

procedures

data
Segmented architecture

Physical memory
Segmented architecture

Physical memory

Process A
Segmented architecture

Physical memory

Process A
Segmented architecture

Physical memory

Process A
Segmented architecture

Physical memory

Process A
Segmented architecture

Physical memory

Process A
Segmented architecture

Physical memory

Process A

Process B
Segmented architecture

Physical memory

Process A

Process B
Segmented architecture

Physical memory

Process A

Process B
Segmented architecture

Physical memory

Process A

Process B

shared code
Segmented MMU

MMU

virtual addr. → base + physical address

bound < yes → within bounds

no → exception
Segmented MMU

虚地址 → 指定偏移量 → 比较器 (≤) → 是否在界限内 → 异常处理

- 虚地址
- 指定偏移量
- 比较器 (≤)
- 是否在界限内
- 异常处理
- 界限
- 物理地址
- 段表
- 基址
Segmented MMU

MMU

virtual addr. // offset

index

segment table

exception

no

yes

within bounds

physical address
The PDP10 had two segments per process, one read only code segment and one read/write for data.
ARPANET LOGICAL MAP, MARCH 1977

(Please note that while this map shows the host population of the network according to the best information obtainable, no claim can be made for its accuracy.)

Names shown are IMP names, not necessarily host names.
Segmentation: the solution

- Segments have variable size.
Segments have variable size.

- Segments have variable size.

Reclaiming segments will cause holes (external fragmentation). Compaction needed.

Is it possible to do compaction?
- Segments have variable size.

Reclaiming segments will cause holes (external fragmentation). Compaction needed. Is it possible to do compaction?
- Segments have variable size.
Segmentation: the solution

- Segments have variable size.

Reclaiming segments will cause holes (external fragmentation). Compaction needed. Is it possible to do compaction?
Segmentation: the solution

- Segments have variable size.

Reclaiming segments will cause holes (external fragmentation). Compaction needed. Is it possible to do compaction?
Segments have variable size.
Segments have variable size.
Segments have variable size.
Segments have variable size.

Reclaiming segments will cause holes (external fragmentation).
- Segments have variable size.
- Reclaiming segments will cause holes (external fragmentation).
- Segments have variable size.
- Reclaiming segments will cause holes (external fragmentation).
- Compaction needed.
- Segments have variable size.
- Reclaiming segments will cause holes (external fragmentation).
- Compaction needed.

*Is it possible to do compaction?*
large grain vs fine grain segments

Using few large segments is easier to implement. Using many small segments would allow the compiler and operating system to do a better job.
Using few large segments is easier to implement.
Using few large segments is easier to implement.

Using many small segments would allow the compiler and operating system to do a better job.
The Altair 8800

Altair 8800 would have 4 or 8 Kbytes of memory.

Intel 8080

- 1972
- 2 MHz
- 16 bit address space (64 Kbyte)

Altair 8800 would have 4 or 8 Kbytes of memory.
The workhorse: 8086

Intel 8086

- 1978, 5 MHz
- 16 bit address space (64 Kbyte)
- 20 bit memory bus (1 Mbyte)
- no protection of segments
- segments for: code, data, stack, extra
Segment addressing in 8086 - real mode

Segment register chosen based on instruction:
- code segment
- stack segment
- data segment (and the extra segment)

The segment architecture available still today in real mode i.e. the 16-bit mode that the CPU is initially in.
Segment addressing in 8086 - real mode

Segment register chosen based on instruction: code segment, stack segment, data segment (and the extra segment.)
Segment addressing in 8086 - real mode

- Segment register chosen based on instruction: *code segment, stack segment, data segment* (and the *extra segment*).
- The segment architecture available still today in *real mode* i.e. the 16-bit mode that the CPU is initially in.
Segment addressing in 80386 - protected mode

MMU

virtual addr.  offset

<
Segment addressing in 80386 - protected mode

MMU

virtual addr.    offset

bound

base +

linear address

ok

down

no

exception
Segment addressing in 80386 - protected mode

**MMU**

virtual addr.  offset  <  bound  base  linear address

ok  exception  no  gdtr
Segment addressing in 80386 - protected mode

MMU

virtual addr.  offset

bound

base

linear address

gdtr

Global Descriptor Table (GDT)

ok

exception

no
Segment addressing in 80386 - protected mode

MMU

virtual addr. → offset → < → exception

bound → < → no → ok

descriptor

base → + → linear address

gdtr

Global Descriptor Table (GDT)
Segment addressing in 80386 - protected mode

MMU

virtual addr. → offset

segment selectors

exception

no

bound

linear address

+ → base

descriptor

gdtr

Global Descriptor Table (GDT)
Segment addressing in 80386 - protected mode

- MMU
- virtual addr.
- offset
- exception
- no
- ok
- segment selectors
- code
- descriptor
- bound
- linear address
- base
- gdtr
- Global Descriptor Table (GDT)
Segment addressing in 80386 - protected mode

**MMU**

```
virtual addr.  offset
```

- segment selectors
  - code
  - stack

- descriptor
  - base
  - bound
  - linear address

- gdtr

- Global Descriptor Table (GDT)

- ok

- exception
  - no
Segment addressing in 80386 - protected mode

MMU

virtual addr.   offset

segment selectors

- code
- stack
- data

Global Descriptor Table (GDT)

bound

linear address

base

descriptor

gdtr

ok

no

exception
Segment addressing in 80386 - protected mode

MMU

virtual addr.  offset

segment selectors
- code
- stack
- data

descriptor
- bound
- base

linear address
- gdtr

Global Descriptor Table (GDT)

no  exception
ok
The segments descriptors of code, data and stack all have base address set to 0x0 and limit to 0xffffffff i.e. they all referre to the same 4 Gibyte linear address space.
The segments descriptors of code, data and stack all have base address set to 0x0 and limit to 0xffffffff i.e. they all refer to the same 4 Gbyte linear address space.

In x86_64 long mode (64 bit mode) Intel removed some support for segments and enforce that these segments are set to 0x0 and 0xfff..ff.
The segments descriptors of code, data and stack all have base address set to 0x0 and limit to 0xffffffff i.e. they all refer to the same 4 Gibyte linear address space.

In x86_64 long mode (64 bit mode) Intel removed some support for segments and enforce that these segments are set to 0x0 and 0xff..ff.

Segmentation is still used to refer to memory that belongs to a specific core or to thread specific memory.
Virtual address space: provide a process with a view of a private address space.
Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
Summary

Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.
Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

- Emulator - two slow.
Summary

Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

- Emulator - two slow.
- Static relocation - not flexible.
Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

- Emulator - two slow.
- Static relocation - not flexible.
- Dynamic relocation:
Virtual address space: provide a process with a view of a private address space.

- **Transparent**: processes should be unaware of virtualization.
- **Protection**: processes should not be able to interfere with each other.
- **Efficiency**: execution should be as close to real execution as possible.

- **Emulator - two slow.**
- **Static relocation - not flexible.**
- **Dynamic relocation:**
  - base and bound - simple to implement
Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

- Emulator - two slow.
- Static relocation - not flexible.
- Dynamic relocation:
  - base and bound - simple to implement
  - segmentation - more flexible
Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

- Emulator - two slow.
- Static relocation - not flexible.
- Dynamic relocation:
  - base and bound - simple to implement
  - segmentation - more flexible
  - problems: fragmentation, sharing of code
Virtual address space: provide a process with a view of a private address space.

- Transparent: processes should be unaware of virtualization.
- Protection: processes should not be able to interfere with each other.
- Efficiency: execution should be as close to real execution as possible.

- Emulator - two slow.
- Static relocation - not flexible.
- Dynamic relocation:
  - base and bound - simple to implement
  - segmentation - more flexible
  - problems: fragmentation, sharing of code

_Cliffhanger - paging, the solution._