CS 134:
Operating Systems
Process Execution

CS 134:
Operating Systems
Process Execution

Overview

Patch Peer Review

Programs, Memory, & Address Space

- Running a Program
- Filling Memory
- Selecting Space
- Memory Sharing

Overview

Patch Peer Review

Programs, Memory, & Address Space
Running a Program
Filling Memory
Selecting Space
Memory Sharing

Numeric Evaluations

Group	Clarity	Concise	Fit	Correct	Docs	Total
fax	4.22	4.78	4.56	4.11	4.67	22.34
ewes	3.67	4.67	4.67	3.67	4.33	21.01
biker	4.33	4.67	4.00	3.33	4.67	21.00
nigh	4.33	4.67	5.00	3.33	3.67	21.00
loan	3.67	4.33	4.33	3.67	4.33	20.33
eat	5.00	4.33	3.67	2.00	4.67	19.67
fakes	3.67	3.67	4.00	3.33	5.00	19.67
gates	4.33	3.33	4.00	2.33	5.00	18.99
loop	4.67	3.67	4.67	2.67	2.67	18.35
halos	3.67	3.67	3.00	4.00	4.00	18.34

2013-05-19

CS34
└─Patch Peer Review
└─Numeric Evaluations

Numeric Evaluations

Group	Clarity	Concise	Fit	Correct	Docs	Total
fax	4.22	4.78	4.56	4.11	4.67	22.34
ewes	3.67	4.67	4.67	3.67	4.33	21.01
biker	4.33	4.67	4.00	3.33	4.67	21.00
nigh	4.33	4.67	5.00	3.33	3.67	21.00
loan	3.67	4.33	4.33	3.67	4.33	20.33
eat	5.00	4.33	3.67	2.00	4.67	19.67
fakes	3.67	3.67	4.00	3.33	5.00	19.67
gates	4.33	3.33	4.00	2.33	5.00	18.99
loop	4.67	3.67	4.67	2.67	2.67	18.35
halos	3.67	3.67	3.00	4.00	4.00	18.34

Ranking

Rank	Group
1.50	loan
1.67	fax
2.25	fakes
2.33	ewes
2.33	nigh
3.00	biker
3.00	halos
3.33	eat
3.50	gates
3.50	loop

Ranking

Rank	Group
1.50	loan
1.67	fax
2.25	fakes
2.33	ewes
2.33	nigh
3.00	biker
3.00	halos
3.33	eat
3.50	gates
3.50	loop

Background—How Processes Get into Memory

Class Exercise:

What transformations does the C source below need go through to become a running process?

```
int main()
{
    write(1, "Hello, world\n",  13);
    return 0;
}
```


Background—How Processes Get into Memory

Class Exercise:

What transformations does the C source below need go through to become a running process?

int main()
{
    write(1, "Hello, world\n", 13);
    return 0;
}

Assembly code—`helloworld.s`

```
.rdata
LC0:
  .ascii "Hello World\n\000"

.text
main:
  addiu sp,sp,-24 # Set up stack frame for main
  la    a1,LC0    # Params for write: a0 = 1, a1 = address
  li    a0,1      # of "Hello world" string, and a2 = 12
  sw    ra,16(sp) # Save our return address (jal overwrites)
  jal   write     # Call write
  li    a2,13     # Delay slot! Executed BEFORE instr above!
  lw    ra,16(sp) # Restore our return address
  move  v0,0      # Our return value is zero
  jr    ra        # Adjust stack and return to caller
  addiu sp,sp,24  # Delay slot! Executed BEFORE instr above!
  nop
```

Object code—`helloworld.o`

Contents of section `.text`:

```
0000 27BDFFE8 3C050000 24A50000 24040001
0010 AFBF0010 0C000000 2406000C 8FBF0010
0020 00001021 03E00008 27BD0018 00000000
```

Contents of section `.data`:

Contents of section `.rodata`:

```
% Hello World.....
  0000 48656C6C 6F2C2077 6F726C64 0A000000
```


Object code—helloworld.o
Contents of section .text:
0000 27BDFFE8 3C050000 24A50000 24040001
0010 AFBF0010 0C000000 2406000C 8FBF0010
0020 00001021 03E00008 27BD0018 00000000
Contents of section .data:
Contents of section .rodata:
% Hello World.....
0000 48656C6C 6F2C2077 6F726C64 0A000000

The `.rodata` contains "Hello, world\n"

Object code—`helloworld.o`

Contents of section `.text`:

```
0000 27BDFFE8 3C050000 24A50000 24040001
0010 AFBF0010 0C000000 2406000C 8FBF0010
0020 00001021 03E00008 27BD0018 00000000

    27BDFFE8  addiu  sp,sp,-24
    3C050000  lui    a1,0
    24A50000  addiu  a1,a1,0
    24040001  li     a0,1
    AFBF0010  sw     ra,16(sp)
    0C000000  jal    0
    2406000C  li     a2,12
    8FBF0010  lw     ra,16(sp)
    00001021  move   v0,0
    03E00008  jr     ra
    27BD0018  addiu  sp,sp,24
    00000000  nop
```


2013-05-19

CS34
└Programs, Memory, & Address Space
└Running a Program
└Object code—helloworld.o

Object code—`helloworld.o`

Contents of section `.text`:

```
0000 27BDFFE8 3C050000 24A50000 24040001
0010 AFBF0010 0C000000 2406000C 8FBF0010
0020 00001021 03E00008 27BD0018 00000000
```

Relocation records for section `.text`:

	Type	*Value*
0004	R_MIPS_HI16	.rodata
0008	R_MIPS_LO16	.rodata
0014	R_MIPS_26	write

Object code—helloworld.o
Contents of section .text:
0000 27BDFFE8 3C050000 24A50000 24040001
0010 AFBF0010 0C000000 2406000C 8FBF0010
0020 00001021 03E00008 27BD0018 00000000
Relocation records for section .text:
Type Value
0004 R_MIPS_HI16 .rodata
0008 R_MIPS_LO16 .rodata
0014 R_MIPS_26 write

Executable code—`helloworld`

Link with `libc.a` and `crt0.o`

- `crt0.o` contains startup code
- `libc.a` contains code for `write`
  - Note no dynamic/shared library support yet!
- Linker can resolve the relocation entries
- End result is an executable, or *load image*.

The OS still needs to:

- Decide if it has resources to run the program right now (long-term scheduler)
- Decide where to put the program in memory
- Perform any additional setup
- Start executing the program

Executable code—helloworld
Link with libc.a and crt0.o
- crt0.o contains startup code
- libc.a contains code for write
  - Note no dynamic/shared library support yet!
- Linker can resolve the relocation entries
- End result is an executable, or load image.
The OS still needs to:
- Decide if it has resources to run the program right now (long-term scheduler)
- Decide where to put the program in memory
- Perform any additional setup
- Start executing the program

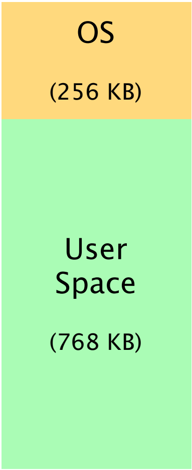
Uniprogramming OS

Only one process—can always locate running process in same place

- Static linking
- Loading is easy

Class Exercise

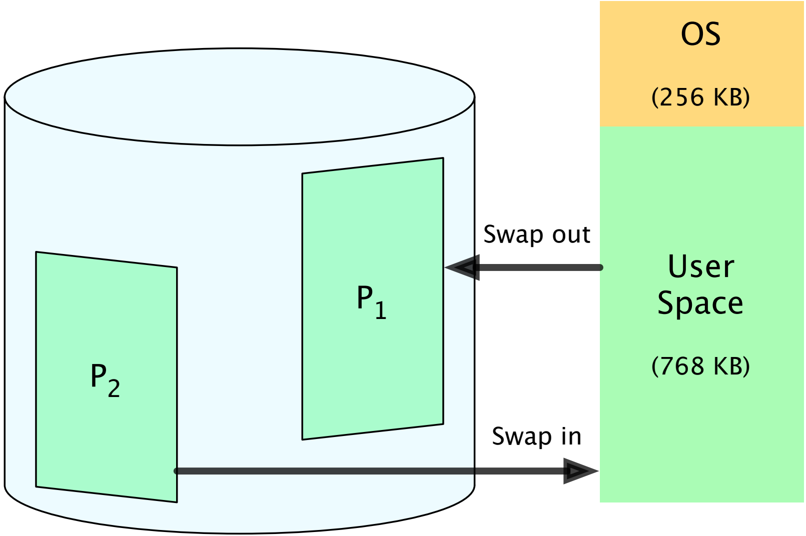
What is the *easiest* way to retrofit this model to run a second program when the first one has to wait for a while?



Uniprogramming OS
Only one process—can always locate running process in same place
Static linking
Loading is easy
Class Exercise
What is the easiest way to retrofit this model to run a second program when the first one has to wait for a while?
OS (256 KB) User Space (768 KB)

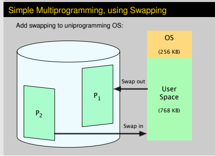
Simple Multiprogramming, using Swapping

Add swapping to uniprogramming OS:



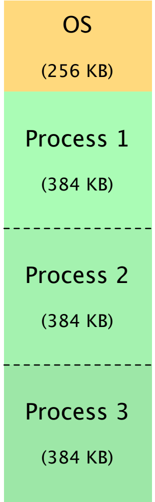
2013-05-19

CS34
└Programs, Memory, & Address Space
  └Filling Memory
    └Simple Multiprogramming, using Swapping



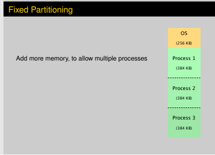
Fixed Partitioning

Add more memory, to allow multiple processes



2013-05-19

CS34
└─Programs, Memory, & Address Space
  └─Filling Memory
    └─Fixed Partitioning

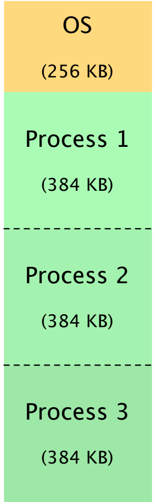


Fixed Partitioning

Add more memory, to allow multiple processes

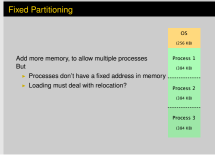
But

- Processes don't have a fixed address in memory
- Loading must deal with relocation?



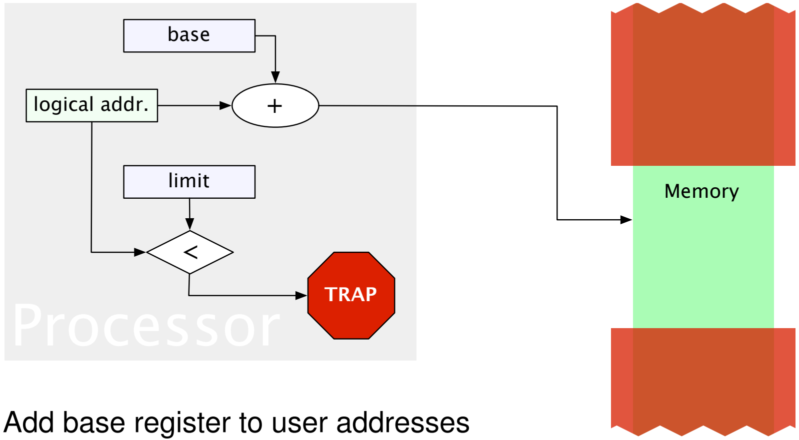
2013-05-19

CS34
└ Programs, Memory, & Address Space
  └ Filling Memory
    └ Fixed Partitioning



Runtime Relocation—Hardware to the rescue

Remember when we talked about protection?

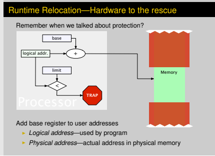


Add base register to user addresses

- *Logical address*—used by program
- *Physical address*—actual address in physical memory

2013-05-19

CS34
└Programs, Memory, & Address Space
 └Filling Memory
  └Runtime Relocation—Hardware to the rescue



Runtime Relocation—Software alternative

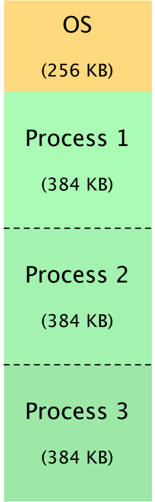
Position-independent code: either

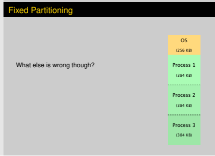
- Grab a register to use as our “base” register and add or subtract from that, or
- Calculate address based on current program counter

Runtime Relocation—Software alternative
Position-independent code: either
- Grab a register to use as our “base” register and add or subtract from that, or
- Calculate address based on current program counter

Fixed Partitioning

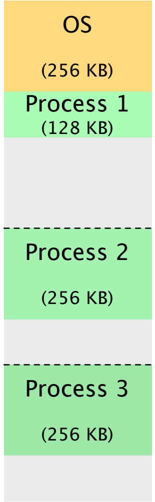
What else is wrong though?





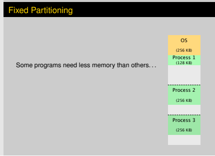
Fixed Partitioning

Some programs need less memory than others. . .



2013-05-19

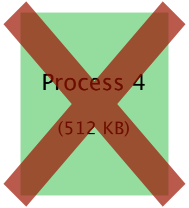
CS34
└ Programs, Memory, & Address Space
  └ Filling Memory
    └ Fixed Partitioning

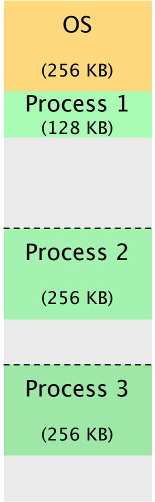


Fixed Partitioning

Some programs need less memory than others…

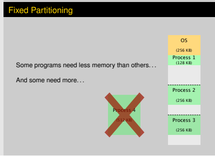
And some need more…





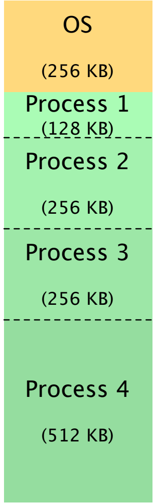
2013-05-19

CS34
└Programs, Memory, & Address Space
  └Filling Memory
    └Fixed Partitioning



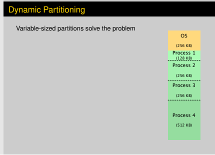
Dynamic Partitioning

Variable-sized partitions solve the problem



2013-05-19

CS34
└─Programs, Memory, & Address Space
  └─Filling Memory
    └─Dynamic Partitioning



Dynamic Partitioning

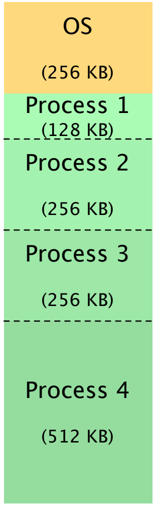
Variable-sized partitions solve the problem

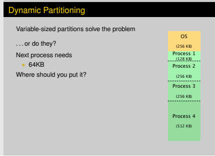
. . . or do they?

Next process needs

- 64KB

Where should you put it?





Dynamic Partitioning

Variable-sized partitions solve the problem

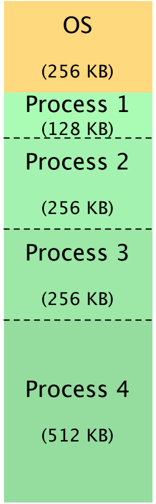
. . . or do they?

Next three processes need

- 64KB
- 64KB
- 256 KB

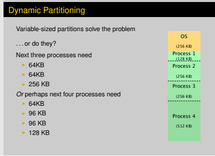
Or perhaps next four processes need

- 64KB
- 96 KB
- 96 KB
- 128 KB



2013-05-19

CS34
└Programs, Memory, & Address Space
  └Filling Memory
    └Dynamic Partitioning

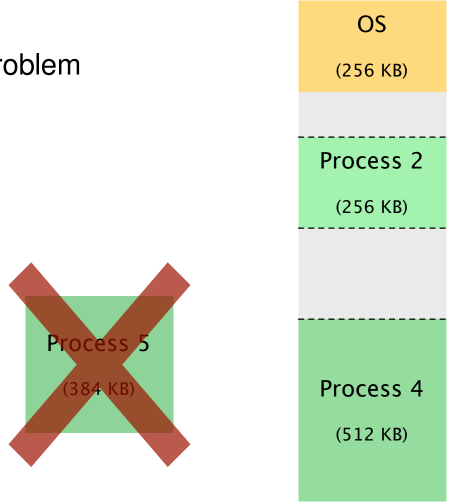


Dynamic Partitioning

Dynamic partitions solve the problem

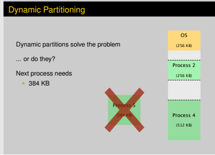
... or do they?

Next process needs
- 384 KB



2013-05-19

CS34
└ Programs, Memory, & Address Space
  └ Filling Memory
    └ Dynamic Partitioning



Which Hole?

Best fit?

- Choose smallest hole that is large enough

Worst fit?

- Choose largest hole that is large enough

First fit?

- Choose first hole that is large enough

Next fit?

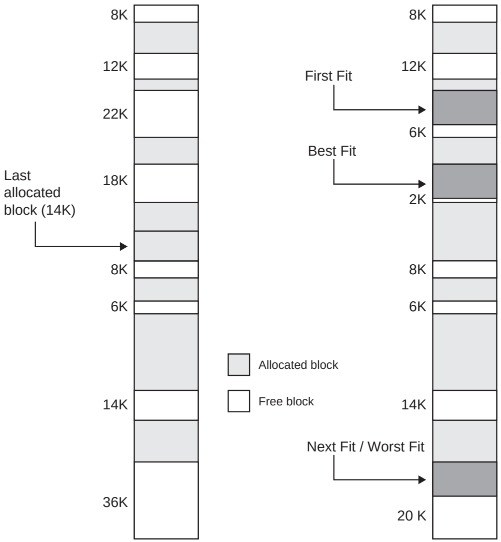
- Choose first hole that is large enough, starting search after last hole we allocated from

Which Hole?
Best fit?
Choose smallest hole that is large enough
Worst fit?
Choose largest hole that is large enough
First fit?
Choose first hole that is large enough
Next fit?
Choose first hole that is large enough, starting search after last hole we allocated from

Which hole?

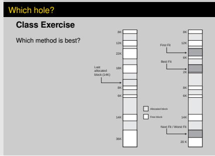
Class Exercise

Which method is best?



2013-05-19

CS34
Programs, Memory, & Address Space
Selecting Space
Which hole?



External Fragmentation

All methods are prone to fragmentation

- Best fit and first fit have least fragmentation on average

Class Exercise

How can we avoid external fragmentation?

External Fragmentation
All methods are prone to fragmentation
Best fit and first fit have least fragmentation on average
Class Exercise
How can we avoid external fragmentation?

Can eliminate fragmentation by *compaction*

Wasted Memory...?

What if two people are running the same editor?

Wasted Memory...?

What if two people are running the same editor?

Segments

We could introduce *segments*—code and data:

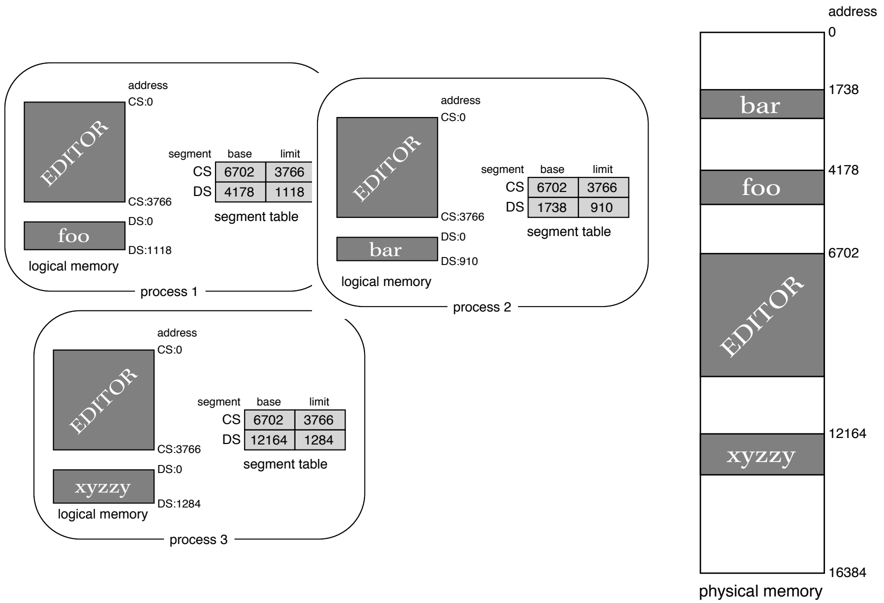
- Program code is put in a *program segment* (read only), shared between processes
- Program data is put in a *data segment*, unique to each process

Segments

We could introduce *segments*—code and data:

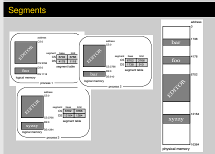
- Program code is put in a *program segment* (read only), shared between processes
- Program data is put in a *data segment*, unique to each process

Segments



2013-05-19

CS34
└Programs, Memory, & Address Space
  └Memory Sharing
    └Segments



More Segments

If two segments are a good idea, would more be even better?

More Segments

If two segments are a good idea, would more be even better?

More Segments

If two segments are a good idea, would more be even better?

How about...

- A stack segment?

Class Exercise

Any other segments that might be nice to have?

More Segments
If two segments are a good idea, would more be even better?
How about...
A stack segment?
Class Exercise
Any other segments that might be nice to have?

More Segments

If two segments are a good idea, would more be even better?
(The x86 has `CS`, `DS`, `SS` and `ES`)

More Segments
If two segments are a good idea, would more be even better?
(The x86 has CS, DS, SS and ES)

More Segments

If two segments are a good idea, would more be even better?
How about...

- A stack segment?
- A shared-data segment?
- A heap segment?
- A segment for the C library
- A thread-local storage segment
- A bonus segment?

2013-05-19

CS34
└─Programs, Memory, & Address Space
  └─Memory Sharing
    └─More Segments

More Segments

If two segments are a good idea, would more be even better?
How about...

- A stack segment?
- A shared-data segment?
- A heap segment?
- A segment for the C library
- A thread-local storage segment
- A bonus segment?

More Segments

If two segments are a good idea, would more be even better?
How about...

- A stack segment?
- A shared-data segment?
- A heap segment?
- A segment for the C library
- A thread-local storage segment
- A bonus segment?

The x86 has `CS`, `DS`, `SS`, `ES`, plus `FS` and `GS`.

Problems?

More Segments
If two segments are a good idea, would more be even better?
How about...
A stack segment?
A shared-data segment?
A heap segment?
A segment for the C library
A thread-local storage segment
A bonus segment?
The x86 has CS, DS, SS, ES, plus FS and GS.
Problems?

Confused programmers!

- Given a 32-bit address, it's hard to know which segment it points into

Are six segments enough?

Segmentation Architecture

Logical address consists of the pair

<segment-number, offset>

Example

Use 32-bit logical address

- High-order 8 bits are segment number
- Low-order 24 bits are offset within segment

256 segments, of max size 16,777,216 bytes (16MB)

Segmentation Architecture
Logical address consists of the pair
Example
Use 32-bit logical address
High-order 8 bits are segment number
Low-order 24 bits are offset within segment
256 segments, of max size 16,777,216 bytes (16MB)

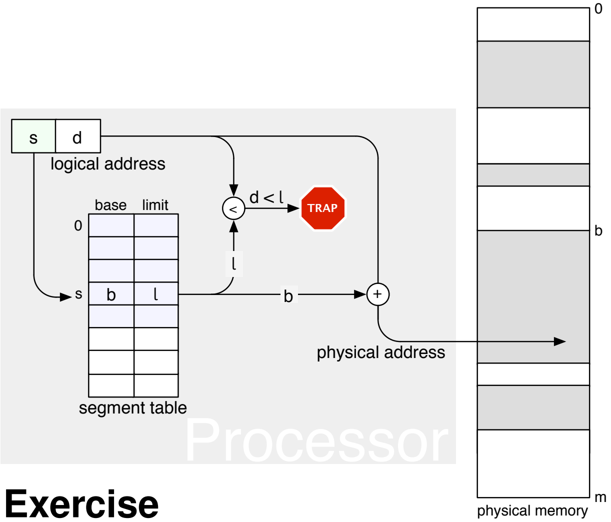
Segmentation Architecture—Segment Table

Processor needs to map 2D user-defined addresses into 1D physical addresses.
In *segment table*, each entry has:

- *Base*—Starting address of the segment in physical memory
- *Limit*—Length of the segment

Segmentation Architecture—Segment Table
Processor needs to map 2D user-defined addresses into 1D physical addresses.
In *segment table*, each entry has:
- *Base*—Starting address of the segment in physical memory
- *Limit*—Length of the segment

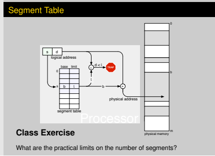
Segment Table



Class Exercise

What are the practical limits on the number of segments?

2013-05-19



Segmentation Architecture

Design Issues:

- Relocation
  - Dynamic
  - By segment table
- Sharing
  - Shared segments
  - Same segment number
- Allocation
  - First fit/best fit
  - External fragmentation

Class Exercise

Do shared segments *need* to have the same segment number?

- If so, why?
- If not, why? (And why might we give them the same segment number anyway?)

Segmentation Architecture
Design Issues:
Relocation
Dynamic
By segment table
Sharing
Shared segments
Same segment number
Allocation
First fit/best fit
External fragmentation
Class Exercise
Do shared segments *need* to have the same segment number?
If so, why?
If not, why? (And why might we give them the same segment number anyway?)

Segmentation Architecture

Class Exercise

Does our segmentation scheme capture the *difference* between code and data segments?

- If not, what would we need to fix it?

Class Exercise

What if a program wants more contiguous data space than a segment can hold? Is this a problem?

Segmentation Architecture
Class Exercise
Does our segmentation scheme capture the *difference* between code and data segments?
If not, what would we need to fix it?
Class Exercise
What if a program wants more contiguous data space than a segment can hold? Is this a problem?

With each entry in segment table, associate:

- Validation bit—0 => illegal segment
- Read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level

Segmentation Architecture—Fragmentation

Class Exercise

What kinds of fragmentation do we have?

- Internal
- External

What's the cause of the fragmentation?

Segmentation Architecture—Fragmentation
Class Exercise
What kinds of fragmentation do we have?
- Internal
- External
What's the cause of the fragmentation?

Segmentation Architecture—Fragmentation

Class Exercise

What kinds of fragmentation do we have?

- Internal—Not a problem
- External—We have a problem! (And compaction would take too long)

What's the cause of the fragmentation?

- Differing segment sizes

Segmentation Architecture—Fragmentation
Class Exercise
What kinds of fragmentation do we have?
- Internal—Not a problem
- External—We have a problem! (And compaction would take too long)
What's the cause of the fragmentation?
- Differing segment sizes

Segmentation Architecture—Fragmentation

Class Exercise

What kinds of fragmentation do we have?

- Internal—Not a problem
- External—We have a problem! (And compaction would take too long)

What's the cause of the fragmentation?

- Differing segment sizes

Crazy Solution *!?!*

Make all segments the same size!

- But now we have internal fragmentation!
- Better make the segments small, to minimize wastage—remember, we can cope with small segments

Segmentation Architecture—Fragmentation
Class Exercise
What kinds of fragmentation do we have?
- Internal—Not a problem
- External—We have a problem! (And compaction would take too long)
What's the cause of the fragmentation?
- Differing segment sizes
Crazy Solution !?!
Make all segments the same size!
- But now we have internal fragmentation!
- Better make the segments small, to minimize wastage—remember, we can cope with small segments

Tiny Segments

Properties

- All segments are the same size (e.g., 4K)
- No need for limit registers
- No longer reflect program structure

Tiny Segments
Properties
- All segments are the same size (e.g., 4K)
- No need for limit registers
- No longer reflect program structure

Paging

Properties

- All pages are the same size (e.g., 4K)
- No need for limit registers
- No longer reflect program structure
- Physical locations for pages are called ***page frames***

Paging
Properties
- All pages are the same size (e.g., 4K)
- No need for limit registers
- No longer reflect program structure
- Physical locations for pages are called ***page frames***