Slide 1

Memory Management Basics
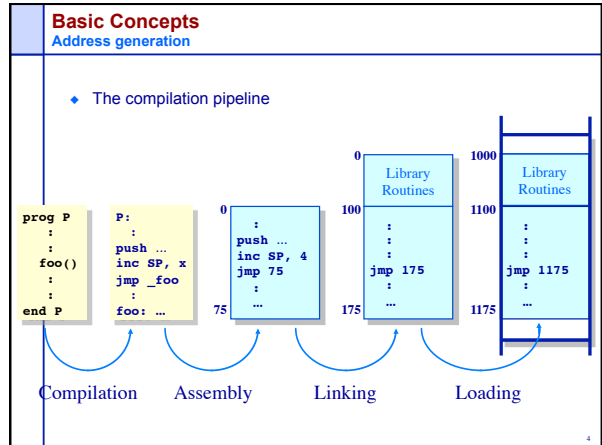

Slide 2
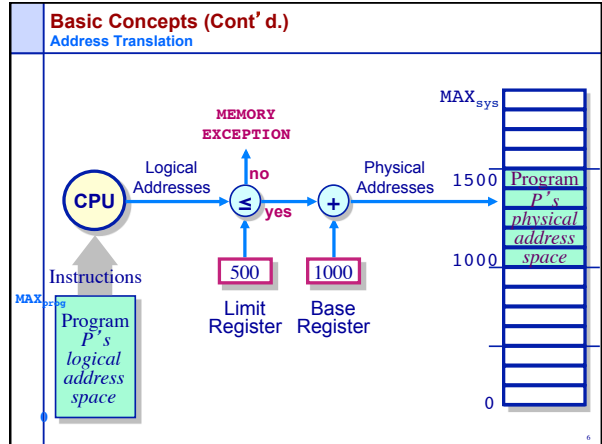
Basic Memory Management Concepts
Address spaces

- *Physical address space* — The address space supported by the hardware
  - Starting at address 0, going to address MAX_{sys}
- *Logical/virtual address space* — A process's view of its own memory
  - Starting at address 0, going to address MAX_{prog}

But where do addresses come from?
```
MOV r0, @0xfffa620e
```

MAX_{sys}

MAX_{prog}

Program
P

0

0

Slide 3

- Which is bigger, physical or virtual address space?
  - A. Physical address space
  - B. Virtual address space
  - C. It depends on the system.

Slide 4

Basic Concepts
Address generation

- The compilation pipeline

```
prog P
  :
  :
foo()
  :
  :
end P
```

```
P:
  :
  push ...
  inc SP, x
  jmp _foo
  :
foo: ...
```

```
0
  :
  push ...
  inc SP, 4
  jmp 75
  :
75  ...
```

```
100
  :
  :
  :
jmp 175
  :
175  ...
```
Library
Routines

```
0
```
```
1000
```
```
1100
  :
  :
  :
jmp 1175
  :
1175  ...
```
Library
Routines

Compilation   Assembly   Linking   Loading

Slide 5

Program Relocation

- Program issues virtual addresses
- Machine has physical addresses.
- If virtual == physical, then how can we have multiple programs resident concurrently?
- Instead, relocate virtual addresses to physical at run time.
  - While we are relocating, also bounds check addresses for safety.
- I can relocate that program (safely) in two registers…

Slide 6

Basic Concepts (Cont'd.)
Address Translation

MEMORY
EXCEPTION

CPU

Logical
Addresses

≤ no
yes

+

Physical
Addresses

Instructions

MAX_{prog}

Program
P's
logical
address
space

500
Limit
Register

1000
Base
Register

MAX_{sys}

1500 Program
P's
physical
address
1000 space

0

Slide 7

- With base and bounds registers, the OS needs a hole in physical memory at least as big as the process.
  - A. True
  - B. False

Slide 8

Evaluating Dynamic Allocation Techniques
The fragmentation problem

- External fragmentation
  - Unused memory between units of allocation
  - E.g, two fixed tables for 2, but a party of 4
- Internal fragmentation
  - Unused memory within a unit of allocation
  - E.g., a party of 3 at a table for 4

| Program Code ("text") |
| Data |
| Execution Stack |

MAX

Program $Q's$ PAS

Program $R's$ PAS

0

Slide 9

Simple Memory Management Schemes
Dynamic allocation of partitions

- Simple approach:
  - Allocate a partition when a process is admitted into the system
  - Allocate a contiguous memory partition to the process

OS keeps track of...
  - Full-blocks
  - Empty-blocks ("holes")

Allocation strategies
  - First-fit
  - Best-fit
  - Worst-fit

MAX

| Program P_1 |
| Program P_2 |
| Program P_3 |
| Program P_4 |

P_5

0

Slide 10

First Fit Allocation

To allocate *n* bytes, use the *first* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 1st free block available

1K bytes	
2K bytes	2K bytes
500 bytes	500 bytes

Slide 11

Rationale & Implementation

- Simplicity of implementation
- Requires:
  - Free block list sorted by address
  - Allocation requires a search for a suitable partition
  - De-allocation requires a check to see if the freed partition could be merged with adjacent free partitions (if any)

Advantages	Disadvantages
- Simple	- Slow allocation
- Tends to produce larger free blocks toward the end of the address space	- External fragmentation

Slide 12

Best Fit Allocation

To allocate *n* bytes, use the *smallest* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 3rd free block available (smallest)

1K bytes	1K bytes
2K bytes	2K bytes
500 bytes	

Rationale & Implementation

- To avoid fragmenting big free blocks
- To minimize the size of external fragments produced
- Requires:
  - Free block list sorted by size
  - Allocation requires search for a suitable partition
  - De-allocation requires search + merge with adjacent free partitions, if any

Advantages
- Works well when most allocations are of small size
- Relatively simple

Disadvantages
- External fragmentation
- Slow de-allocation
- Tends to produce many useless tiny fragments (not really great)

- Doug Lea's malloc "In most ways this malloc is a best-fit allocator"

13

Worst Fit Allocation

To allocate n bytes, use the *largest* available free block such that the block size is larger than n.

To allocate 400 bytes, we use the 2nd free block available (largest)

1K bytes

2K bytes

500 bytes

1K bytes

14

Rationale & Implementation

- To avoid having too many tiny fragments
- Requires:
  - Free block list sorted by size
  - Allocation is fast (get the largest partition)
  - De-allocation requires merge with adjacent free partitions, if any, and then adjusting the free block list

Advantages
- Works best if allocations are of medium sizes

Disadvantages
- Slow de-allocation
- External fragmentation
- Tends to break large free blocks such that large partitions cannot be allocated

15

Allocation strategies

- First fit, best fit and worst fit all suffer from external fragmentation.
  - A. True
  - B. False

16

Dynamic Allocation of Partitions
Eliminating Fragmentation

- Compaction
  - Relocate programs to coalesce holes

- Swapping
  - Preempt processes & reclaim their memory

MAX

Program P_1

Program P_2

Program P_3

Program P_4

0

Ready — Running

ready queue

Waiting

Suspended

suspended queue
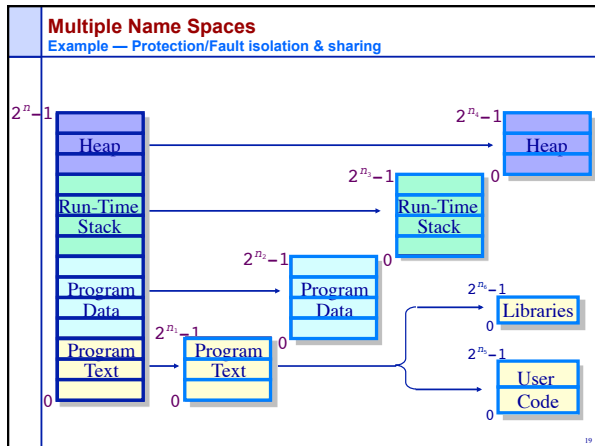
semaphore/condition queues

17

Memory Management
Sharing Between Processes

- Schemes so far have considered only a single address space per process
  - A single *name space* per process
  - No sharing

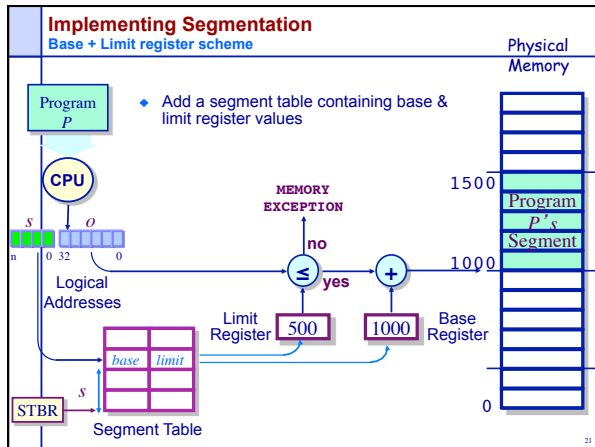How can one share code and data between programs without paging?

2^n-1

Heap

Run-Time Stack

Program Data

Program Text

0

Program P's VAS

18

Multiple Name Spaces
Example — Protection/Fault isolation & sharing

2^n-1

Heap

Run-Time Stack

Program Data

$2^{n_1}-1$

Program Text

0

0

Program Text

0

$2^{n_2}-1$

0

Program Data

0

$2^{n_3}-1$

Run-Time Stack

0

$2^{n_4}-1$

Heap

0

$2^{n_5}-1$

Libraries

0

$2^{n_6}-1$

User Code

0

19

Supporting Multiple Name Spaces
Segmentation

- New concept: A *segment* — a memory "object"
  - A virtual address space

- A process now addresses objects —a pair (*s*, *addr*)
  - *s* — segment number
  - *addr* — an offset within an object
    - Don't know size of object, so 32 bits for offset?

n_2 0 n_1 0

s   *addr*

Segment + Address register scheme

n 0

s   *addr*

Single address scheme

20

Implementing Segmentation
Base + Limit register scheme

Program *P*

CPU

s   *o*

n   0 32   0

Logical Addresses

- Add a segment table containing base & limit register values

MEMORY EXCEPTION

no

≤   yes

+

Limit Register   500

Base Register   1000

base	limit

s

STBR

Segment Table

Physical Memory

1500

Program *P's* Segment

1000

0

21

Memory Management Basics
Are We Done?

- Segmentation allows sharing

- … but leads to poor memory utilization
  - We might not use much of a large segment, but we must keep the whole thing in memory (bad memory utilization).
  - Suffers from external fragmentation
  - Allocation/deallocation of arbitrary size segments is complex

- How can we improve memory management?
  - Paging

22