Memory Management Basics

Don Porter

Portions courtesy Emmett Witchel and Kevin Jeffay
Review: Address Spaces

- **Physical address space** — The address space supported by the hardware
  - Starting at address 0, going to address $\text{MAX}_{\text{sys}}$

- **Logical/virtual address space** — A process’s view of its own memory
  - Starting at address 0, going to address $\text{MAX}_{\text{prog}}$

But where do addresses come from?

```
MOV r0, @0xfffffa620e
```
Which is bigger, physical or virtual address space?

- A. Physical address space
- B. Virtual address space
- C. It depends on the system.
Address Space Generation

- The compilation pipeline

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

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

Library Routines

Library Routines

```

Compilation  Assembly  Linking  Loading
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...
2 register translation

Logical Addresses $\leq$ Limit Register $\rightarrow$ Base Register $\rightarrow$ Physical Addresses

CPU

Instructions

Limit Register 500

Base Register 1000

MEMORY EXCEPTION no yes

MAX$_{sys}$ 1500

Program $P$'s physical address space

MAX$_{prog}$ 1000

Program $P$'s logical address space
• With base and bounds registers, the OS needs a hole in physical memory at least as big as the process.
  – A. True
  – B. False
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
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
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.
First Fit: Rationale and Implementation

- Simplicity!

- 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**
- Simple
- Tends to produce larger free blocks toward the end of the address space

**Disadvantages**
- Slow allocation
- External fragmentation
Best Fit Allocation

To allocate $n$ bytes, use the *smallest* available free block such that the block size is larger than *(or equal to)* $n$.

To allocate 400 bytes, we use the 3rd free block available *(smallest)*.
Best Fit: Rationale and Implementation

- 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)
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)
Worst Fit: Rationale and Implementation

- 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
Allocation strategies

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

• Compaction
  – Relocate programs to coalesce holes

flammable

◆ Swapping
  ➢ Preempt processes & reclaim their memory

![Diagram](image-url)

- Ready
- Running
- Waiting
- Suspended
- Semaphore/condition queues
- Suspended queue
- Ready queue
- MAX
- Programs P₁, P₂, P₃, P₄
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?
Multiple (sub) Name Spaces

- **Program Text**: $2^{n_1} - 1$ spaces
  - (shared)
  - $0$ to $2^{n_1} - 1$

- **Program Data**: $2^{n_2} - 1$ spaces
  - (not shared)
  - $0$ to $2^{n_2} - 1$

- **Run-Time Stack**: $2^{n_3} - 1$ spaces
  - (not shared)
  - $0$ to $2^{n_3} - 1$

- **Heap**: $2^{n_4} - 1$ spaces
  - (not shared)
  - $0$ to $2^{n_4} - 1$

- **Libraries**: $2^{n_5} - 1$ spaces

- **User Code**: $2^{n_6} - 1$ spaces
Segmentation

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

- A process now addresses objects — a pair \((s, \text{addr})\)
  - \(s\) — segment number
  - \(\text{addr}\) — an offset within an object

- Don’t know size of object, so 32 bits for offset?

Two ways to encode a virtual address
Implementing Segmentation

- Add a segment table containing base & limit register values

```
STBR
+-----------------------------+
| base | limit | 500 | 1000 |
+-----------------------------+
```

```
Program P
```

```
Logical Addresses
```

```
CPU
```

```
Segment Table
```

```
Limit Register
```

```
physical memory
```

```
Program P's Segment
```

```
STBR
```

```
Physical Memory
```

```
MEMORY EXCEPTION
```

```
= 500
```

```
≤
```

```
+ 1000
```

```
o
```

```
no
```

```
yes
```

```
s
```

```
0 32 0
```

```
n
```

```
1000
```

```
1500
```

```
P
```

```
STBR
```

```
0
```

```
s
```
Are we done?

• Segmentation allows sharing
  – And dead simple hardware
    • Can easily cache all translation metadata on-chip
  – Low latency to translate virtual addresses to physical addresses
    • Two arithmetic operations (add and limit check)

• … 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?
  – stay tuned…