Fixed and Variable Length Allocation Schemes

Dr Andrew Scott

a.scott@lancaster.ac.uk

1

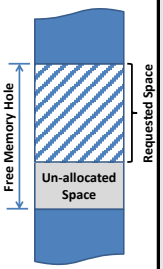
Problem with Variable Size Allocation

- Obvious approach, allocate just memory requested
- Placing things in memory (or on disk, etc.) with such approach rapidly becomes big problem…
- If we free a large area of memory, disk, …
  - Tend to place something smaller in space, leaving a gap, which prevents us from then storing something larger
- Problem known as ***Fragmentation***
  - Lots of things scattered through memory with small unusable gaps between them

2

Variable Sized: Fragmentation

- Can become severe problem
  - Sum of un-allocated space can be large
  - Requires complex de-fragmentation
    - Shuffling storage contents
      - Must be invisible to running processes
      - Not difficult with disks, etc.
      - **With memory, references in code…**
        likely impossible



3

What about Defragmentation?

- Process of shuffling code, data, etc. to coalesce/ collect together unused space
- Can work for disks
  - Have filesystems that structure/ control what's stored
    - We understand internals and can ensure consistency
    - Can 'pause' their operation while we defragment space
- Memory more difficult
  - Programs can store memory references* anywhere
  - If we miss changing just one, …

* Memory addresses (references) can be held in registers, C variables, structures, … Programs could also calculate addresses using offsets that may no longer be valid

4

Making best use of Variable Size Regions

- Dynamic storage allocation schemes…
  - First fit and Next fit
    - Always allocate first region of sufficient size found
  - Best fit
    - Always allocate smallest memory hole of sufficient size
  - Worst fit
    - Always allocate largest memory hole (of sufficient size)
    - Aims to increase chance of later allocation

We'll look at some examples of these schemes in a coming lecture

5

Variable vs. Fixed Sized Schemes

- Variable/ Dynamic allocation schemes
  - Allocate requested amount of memory
  - ***External fragmentation***
    - Space left outside/ between allocated memory
- Fixed size schemes
  - Always allocate memory in fixed-sized blocks
  - ***Internal fragmentation***
    - Processes typically get more memory than requested, rounded up to *n* * block size, which generally remains unused
    - Much less of a problem than External Fragmentation, as bounded by **block allocation size** and **process lifetime** *

* We always get an exact multiple of block size back when memory freed

6