

7. Physical Memory

7.1 Preparing a Program for Execution

- Program Transformations
- Logical-to-Physical Address Binding

7.2 Memory Partitioning Schemes

- Fixed Partitions
- Variable Partitions

7.3 Allocation Strategies for Variable Partitions

7.4 Dealing with Insufficient Memory

ICS 143

1

Preparing program for execution

- program transformations
 - translation (compilation)
 - linking
 - loading

Figure 7-1

ICS 143

2

Address binding

- assign physical addresses = relocation
- static binding
 - programming time
 - compilation time
 - linking time
 - loading time
- dynamic binding
 - execution time

Compare Figures 7-2 and 7-4

ICS 143

3

Address binding

- how to implement dynamic binding
 - perform for each address at run time:
 $pa = \text{address_map}(la)$
 - simplest form of address_map:
relocation register
 $pa = la + RR$
 - more general form: page/segment table
(Chapter 8)

ICS 143

4

Memory partitioning schemes

- fixed partitions
 - single-program systems: 2 partitions (OS/user)
 - multi-programmed: partitions of different sizes
- how to assign processes to partitions Fig 7-5
 - FIFO for each partition
 - some partitions may remain unused
 - single FIFO
 - more complex but more flexible
- limitations of fixed partitions
 - program size limited to largest partition
 - internal fragmentation

ICS 143

5

Variable partitions

- memory not partitioned a priori
- each request is allocated portion of free space
- memory: sequence of variable-size blocks
 - some are occupied, some are free (holes)
 - external fragmentation occurs
- adjacent holes must be coalesced to prevent increasing fragmentation

Figure 7-6

ICS 143

6

Linked list implementation

- type/size tags at the beginning of each block
 - links (pointers) kept inside holes
 - how to check neighbors of released block b:
 - right neighbor: use size of b
 - left neighbor:
 - uses sizes to find first hole to the right
 - follow predecessor ptr to first hole on left
 - check if neighbor
- Figure 7-7(a)
- holes must be sorted by physical address

ICS 143

7

Linked list implementation

- better solution: replicate tags at end of block
 - checking neighbors of released block b:
 - right neighbor: use size of b as before
 - left neighbor: check adjacent tags
 - blocks need not be sorted

Figure 7.7(b)

ICS 143

8

Bit map implementation

- memory divided into fix-size blocks
- each block represented by a 0/1 bit in a binary string: the bit map
- can be implemented as char or int array
- operations use bit masks
 - release: $B[i] = B[i] \& '11011111'$
 - allocation: $B[i] = B[i] | '11000000'$
 - search: repeatedly check left-most bit and shift mask right: $TEST = B[i] \& '10000000'$

ICS 143

9

The Buddy system

- compromise between fixed and variable
- fixed number of possible hole sizes, e.g., 2^i
- request n bytes:
 - find smallest hole size such that $n \leq 2^i$
 - if not empty, allocate hole, else consider larger holes:
 - recursively split each hole into two buddies until smallest possible hole is created; allocate it and place other holes on appropriate lists
- release block: recursively coalesce buddies

Example: Figure 7-9

ICS 143

10

Allocation strategies

- given a request for n bytes, find hole
- constraints:
 - maximize memory utilization (minimize external fragmentation)
 - minimize search time
- first fit: simplest
- next fit: improve distribution of holes
- best fit: avoid breaking up large holes
- worst fit: avoid leaving tiny hole fragments
- first fit is generally the best choice

ICS 143

11

Measures of memory utilization

- 50% rule:
 - $\#holes = 0.5 * p * \#full_blocks$
 - p : probability of inexact match (remaining hole)
 - in practice: $p=1$, i.e., 1/3 of memory are holes
- but *how much* memory is wasted (hole size \neq block size)

ICS 143

12

Measures of memory utilization

- what is the unused fraction of space?
 - utilization depends on $k = \text{hole_size} / \text{block_size}$:
 $\text{unused_memory} = k / (k+2)$
 - intuition:
 - $k \rightarrow \infty$: unused memory $\rightarrow 1$ (100% empty)
 - $k=1$: unused memory $\rightarrow 1/3$ (50% rule)
 - $k \rightarrow 0$: unused memory $\rightarrow 0$ (100% full)
 - what determines k ?
 - block size b relative to total memory size M
 - determined experimentally
 - $b \leq M/10$: $k=0.22$; $f=0.1$
 - $b = M/3$: $k=2$; $f=0.5$
- conclusion: M must be large relative to b

ICS 143

13

Dealing with insufficient memory

- memory compaction
 - how much and what to move?
 - Figure 7-9
- swapping
 - temporarily move process to disk
 - requires dynamic relocation
- overlays
 - allows programs large than physical memory
 - programs loaded as needed according to calling structure (Figure 7-10)

ICS 143

14
