Operating Systems Design (CS 423)

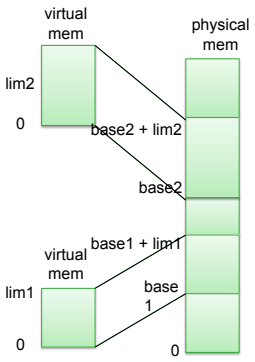
Elsa L Gunter
2112 SC, UIUC
http://www.cs.illinois.edu/class/cs423/

Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum

Base and Limit

- Load each process into contiguous regions of physical memory

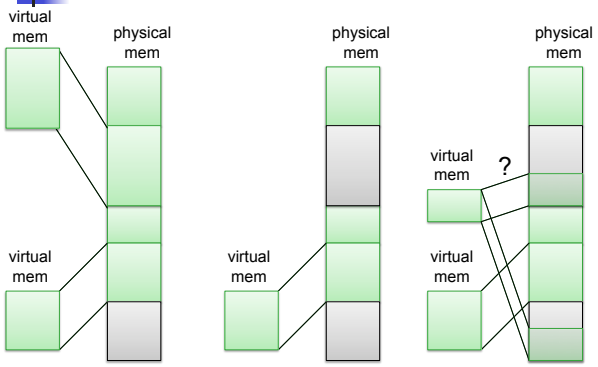
```
If(virt addr > bound){
   trap to kern
} else {
   phys addr =
    virt addr + base
}
```



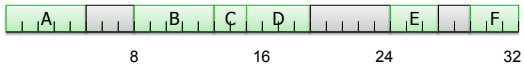
Base and limit

- What must change during a context switch?
- Can a process change its own base and limit?
- Can you share memory with another process?
- How does the kernel handle the address space growing
  - You are the OS designer, come up with an algorithm for allowing processes to grow

Memory Allocation in the small

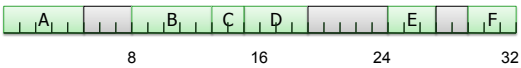


Memory Management w/ Linked Lists



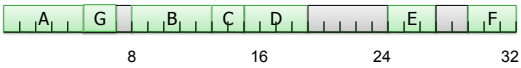
- Info for each hole
  - Both ends:
    - Size of hole
    - Bit saying free or allocated
    - Together
  - Next and Previous pointers to other holes
- Info for each allocated "chunk:
  - Size and allocation bit, together, on each end

Memory Allocation Algorithms

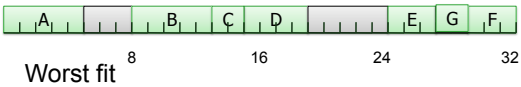


Allocated 2 units:

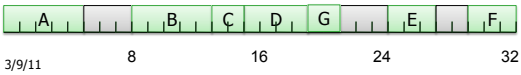
- First fit



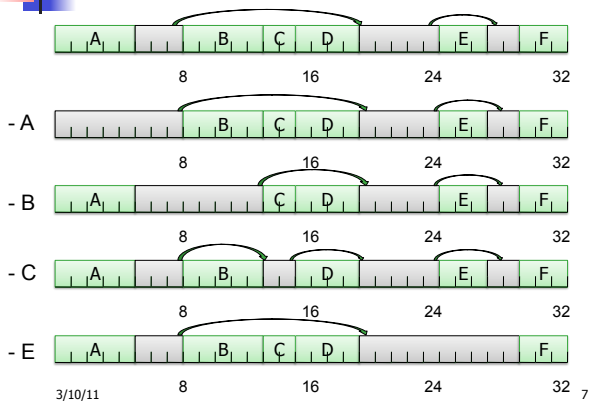
- Best fit



- Worst fit



Memory Deallocation – Linked Lists



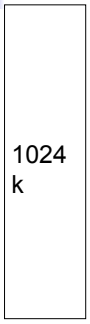
Buddy System

- Blocks of memory allocated in sizes that are 2^x
- Determine the largest size block 2^u that fits in available memory
- Fix a smallest size to be allocated 2^l
- When *y* memory needed, determine smallest k such that $2^k \geq y$
- Find free block of that size
- If none exists, find smallest one large enough
  - Divide it in half repeatedly until small enough
  - Allocate in first piece, marking all others as free

Buddy System - Deallocating

- Each block has a "buddy" to its left or right
- When block is released, its buddy is checked
- When two buddies are free, they are merged
- Minimizes external fragmentation - gaps in allocated memory
- May still have much internal fragmentation - gaps between allocation and actual use

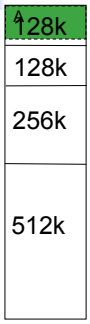
Buddy System



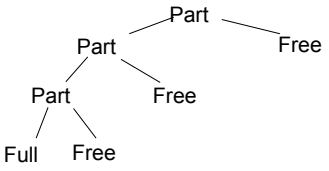
- System Initialization
- u= 10; 2^uk = 1024k = 1m
- l = 2; 2^2 = 4k
- Initialize search tree

Free

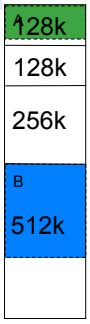
Buddy System



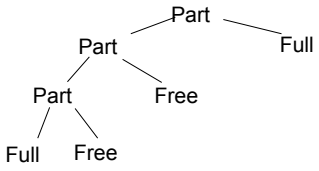
- A needs 100k
- Divide memory into two 512K blocks
- Divide first into two 256k blocks
- Divide first into two 128k blocks
- Assign A first of these



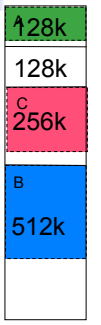
Buddy System



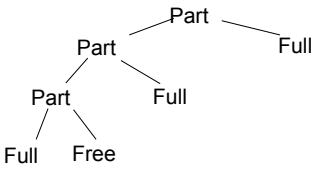
- B needs 300k
- Search left
- Stop at second branch because it is not free
- Search right
- Assign B 512K block



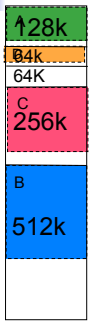
Buddy System



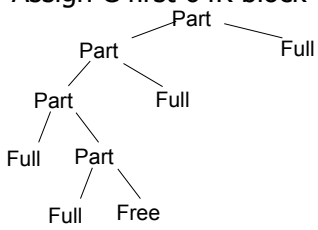
- C needs 200k
- Find leftmost block that fits
- Assign C 256K block



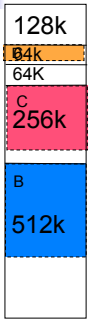
Buddy System



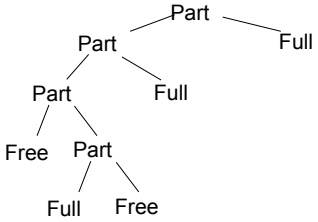
- D needs 50k
- Split 128K block
- Assign C first 64K block



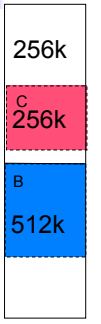
Buddy System



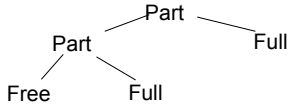
- Deallocate A
- Buddy is part full; just free A



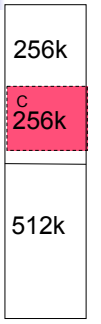
Buddy System



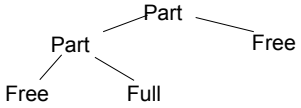
- Deallocate D
- Immediate buddy is free; merge
- One level up, buddy is free; merge
- Mark 256K free



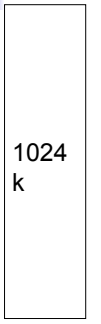
Buddy System



- Deallocate B
- Immediate buddy is not free
- Just free 512K



Buddy System



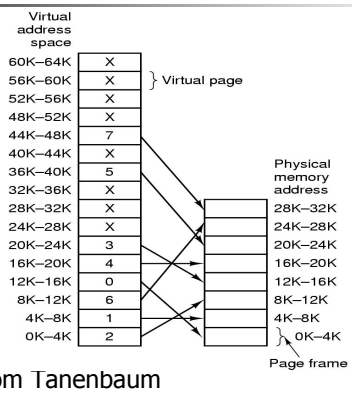
- Deallocate C
- Immediate buddy is free; merge
- Next buddy is free; merge
- Mark all 1024k free

Free

Paging

- Allocate physical memory in terms of fixed-size chunk
  - Fixed unit easier to allocate
  - Any free physical page (page frame) can store any virtual page
- Virtual address
  - Virtual page # (high bits of address)
  - Offset (low bits of address, e.g., bits 11-0 for 4k page)

Translations table



- Fig 3.9 from Tanenbaum

Translation Process

```
If(virtual page is invalid or non-
  resident or protected) {
   trap to OS fault handler
} else {
   physical page # =
   pageTable[virtpage#].physPageNum
}
```


Translation Process

- What must change on a context switch?

Translation Process

- What must change on a context switch?
  - Page table must be replaces
- Each virtual page can be in physical memory or swapped out to disk (called paged)

Resident Pages

- How does the processor know that a virtual page is not in memory?

Resident Pages

- How does the processor know that a virtual page is not in memory?
  - A bit in the page table entry
- Resident means a virtual page is in memory
- NOT an error for a program to access non-resident page

Valid Pages Accesses

- Pages can have different protections
  - Read, write, execute
- Valid means that a virtual page is legal for the program to access
  - E.g. page not part of the address space is invalid page
- IS an error to try to access an invalid page

Valid vs Resident

- Who makes a page resident / non-resident?
- Who makes a virtual page valid / invalid?
- Why would a process want one if its virtual pages to be invalid?

Valid vs Resident

- Who makes a page resident / non-resident?
  - OS
- Who makes a virtual page valid / invalid?
  - (user) program
- Why would a process want one of its virtual pages to be invalid?
  - Security, and general protection from self