Operating Systems

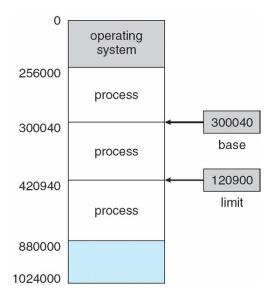
Main Memory

Background

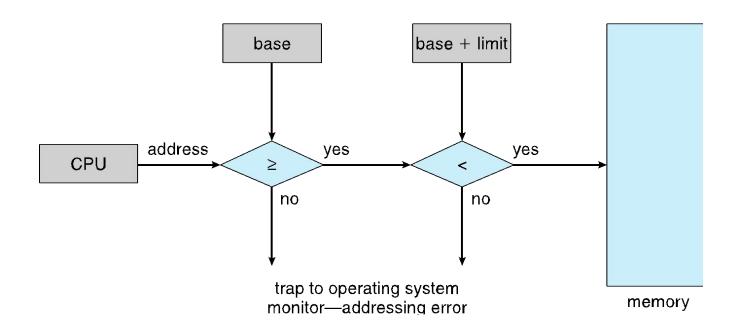
- Program must be brought (from disk) into memory and placed within a process for it to be run
- ☐ Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**, since it does not have the data required to complete the instruction that it is executing
- ☐ Cache sits between main memory and CPU registers for fast access
- ☐ Protection of memory required to ensure correct operation

Base and Limit Registers

- A pair of base and limit registers define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



Hardware Address Protection



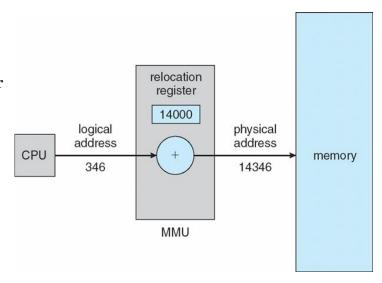
Operating Systems **Address Space**

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
 - Logical address generated by the CPU; also referred to as virtual address
 - Physical address address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- Logical address space is the set of all logical addresses generated by a program
- Physical address space is the set of all physical addresses generated by a program

Memory-Management Unit (MMU)

- MMU Hardware device that at run time maps virtual to physical address
- consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
 - Base register now called **relocation register**
 - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
 - Execution-time binding occurs when reference is made to location in memory
 - Logical address bound to physical addresses



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
 - Implemented through program design
 - OS can help by providing libraries to implement dynamic loading

Dynamic Linking

- Static linking system libraries and program code combined by the loader into the binary program image
- Dynamic linking linking postponed until execution time
- Small piece of code, stub, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
 - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as shared libraries

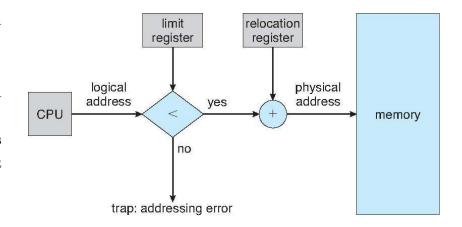
Operating Systems Contiguous Allocation

Contiguous Allocation

- Main memory must accommodate both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually divided into two partitions:
 - Resident operating system, usually held in low memory with interrupt vector
 - User processes then held in high memory
 - ☐ Each process contained in single contiguous section of memory

Contiguous Allocation

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
 - Base register contains value of smallest physical address
 - Limit register contains range of logical addresses
 each logical address must be less than the limit register
 - MMU maps logical address *dynamically*
 - Can then allow actions such as kernel code being transient and kernel changing size



Multiple-partition allocation

- Multiple-partition allocation:
 - Fixed-sized partition: Degree of multiprogramming limited by number of partitions
 - Variable-partition sizes for efficiency (sized to a given process' needs)
 - Hole block of available memory; holes of various size are scattered throughout memory
 - When a process arrives, it is allocated memory from a hole large enough to accommodate it
 - Process exiting frees its partition, adjacent free partitions combined
 - Operating system maintains information about:
 - a) allocated partitions b) free partitions (hole)

Operating Systems Dynamic Memory Allocation

Dynamic Storage-Allocation Problem

How to satisfy a request of size *n* from a list of free holes?

- First-fit: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size
 - Produces the smallest leftover hole
- Worst-fit: Allocate the *largest* hole; must also search entire list
 - Produces the largest leftover hole

Fragmentation

- External Fragmentation total memory space exists to satisfy a request, but it is not contiguous
- Internal Fragmentation allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- For First fit, statistical analysis reveals that given *N* blocks allocated, another 0.5 *N* blocks lost to fragmentation
 - 1/3 may be unusable -> **50-percent rule**
- Reduce external fragmentation by compaction
 - Shuffle memory contents to place all free memory together in one large block
 - Compaction is possible *only* if relocation is dynamic, and is done at execution time

Operating Systems

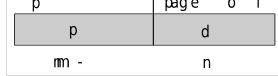
Paging

Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
 - Avoids external fragmentation
 - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called frames
 - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size N pages, need to find N free frames and load program
- Set up a page table to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

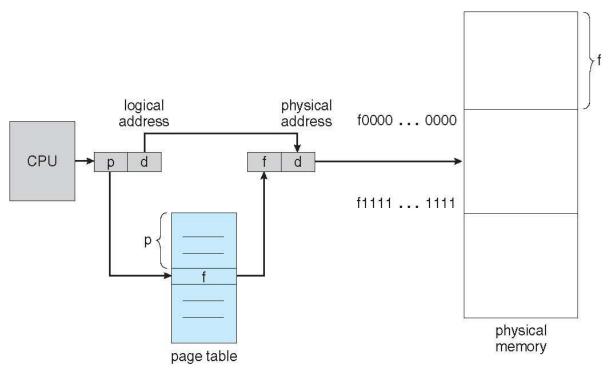
Address Translation Scheme

- Address generated by CPU is divided into:
 - Page number (p) used as an index into a page table which contains base address of each page in physical memory
 - Page offset (d) combined with base address to define the physical memory address that is sent to the memory unit page of

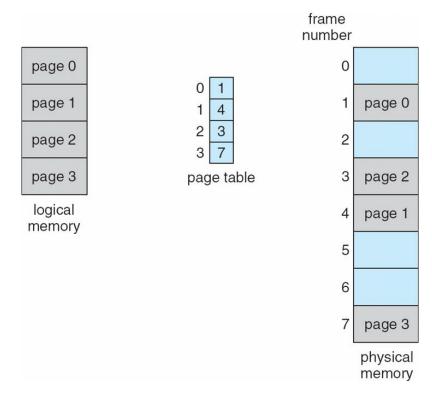


• For given logical address space 2^m and page size 2^n

Paging Hardware



Paging Model of Logical and Physical Memory



Paging Example

n=2 and *m*=232-byte memory and 4-byte pages

0	a
1	b
2	С
3	d
4	е
5	f
6	g h
7	h
8	i
9	j k
10	k
11	-
12	m
13	n
14	0
15	р

1	6	
2	1	
3	2	
oage	ta	ble

0 5

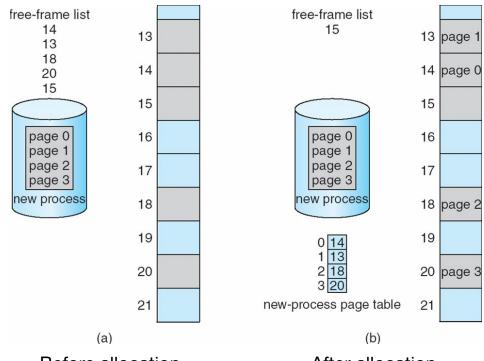
0	
4	i j k
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

physical memory

Size of Page

- Calculating internal fragmentation
 - \circ Page size = 2,048 bytes
 - \circ Process size = 72,766 bytes
 - 35 pages + 1,086 bytes
 - O Internal fragmentation of 2,048 1,086 = 962 bytes
 - \circ Worst case fragmentation = 1 frame 1 byte
 - On average fragmentation = 1 / 2 frame size
 - So small frame sizes desirable?
 - But each page table entry takes memory to track
 - Page sizes growing over time
 - Solaris supports two page sizes 8 KB and 4 MB
- Process view and physical memory now very different
- By implementation process can only access its own memory

Free Frames



Before allocation

After allocation

Operating Systems Implementation of Page Table

Implementation of Page Table

- Page table is kept in main memory
- Page-table base register (PTBR) points to the page table
- Page-table length register (PTLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
 - One for the page table and one for the data / instruction
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs)

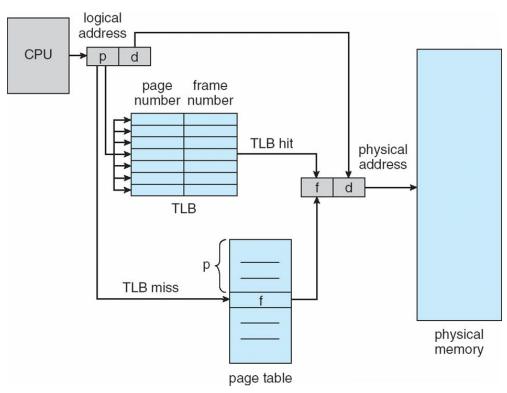
Associative Memory

• Associative memory – parallel search

Page #	Frame #	

- Address translation (p, d)
 - \circ If p is in associative register, get frame # out
 - Otherwise get frame # from page table in memory

Paging Hardware With TLB



Effective Access Time

- Associative Lookup = ε time unit
 - Can be < 10% of memory access time
- Hit ratio = α
 - Hit ratio percentage of times that a page number is found in the associative registers;
- Consider $\alpha = 80\%$, $\varepsilon = 20$ ns for TLB search, 100ns for memory access
- Effective Access Time (EAT):
- Consider $\alpha = 80\%$, $\epsilon = 20$ ns for TLB search, 100ns for memory access
 - \circ EAT = 0.80 x 120 + 0.20 x 200 = 136ns
- Consider more realistic hit ratio -> $\alpha = 99\%$, $\epsilon = 20$ ns for TLB search, 100ns for memory access
 - \circ EAT = 0.99 x 120 + 0.01 x 200 = 120.8ns

Operating Systems

Shared Pages

Shared Pages

Shared code

- One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
- Similar to multiple threads sharing the same process space
- Also useful for inter-process communication if sharing of read-write pages is allowed

Private code and data

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example

