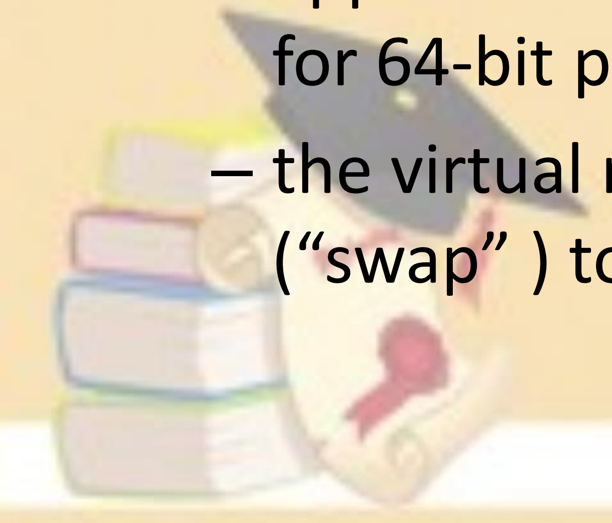
İSTANBUL KEMERBURGAZ ÜNİVERSİTESİ

Mac OS X Memory management

Name : BAKR KAMAL JASIM
Student No : 163103005
Email : bakir_alani92@yahoo.com

MEMORY MANAGEMENT

- Mac OS X includes a fully-integrated virtual memory system that you cannot turn off
- It is always on, providing up to 4 gigabytes of addressable space per 32-bit process and approximately 18 Exabyte of addressable space for 64-bit processes
- the virtual memory system uses hard disk storage (“swap”) to hold data not currently in use



MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - In Mac OS X, each process has its own sparse 32-bit or 64-bit virtual address space
    - » **For 32-bit processes**: each process has an address space that can grow dynamically up to a limit of 4 gigabytes
    - » **For 64- bit processes**: the address space can grow dynamically up to a limit of approximately 18 Exabyte

MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
    - The virtual address space of a process consists of mapped regions of memory
    - Each region of memory in the process represents a specific set of virtual memory pages
    - regions contain a given number of pages, they are **page-aligned**

MEMORY MANAGEMENT

– **Virtual Memory in Mac OS X**

– The VM object maps regions in the backing store through the **default pager** and maps file-mapped files through the **vnode pager**

» **default pager-** a system manager that maps the nonresident virtual memory pages to backing store and fetches those pages when requested

» **vnode pager-** implements file mapping and uses the paging mechanism to provide a window directly into a file

MEMORY MANAGEMENT

– **Virtual Memory in Mac OS X**

  – **copy-on-write -** a form of page-level sharing that allows multiple blocks of code (including different processes) to share a page as long as none write to that page

  - allows the system to copy large quantities of data efficiently

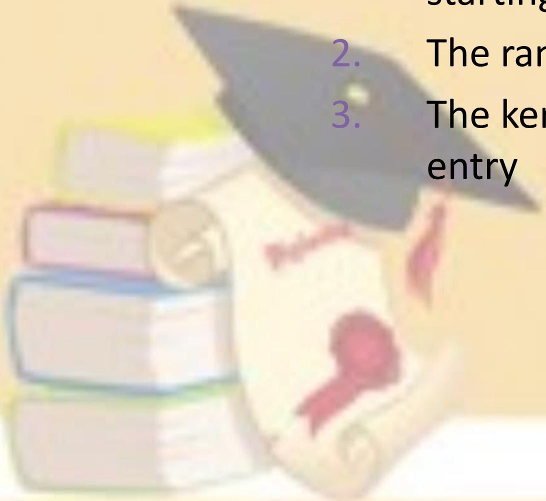
MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Allocating and Accessing Virtual Memory**
    - » Uses the malloc routine
    - » This routine finds free space on an existing page or allocates new pages using vm_allocate to create space for the new memory block

MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Allocating and Accessing Virtual Memory**
    - Through the vm_allocate routine, the kernel performs a series of initialization steps

1. It maps a range of memory in the virtual address space of this process by creating a **map entry**; the map entry is a simple structure that defines the starting and ending addresses of the region.
2. The range of memory is backed by the default pager.
3. The kernel creates and initializes a VM object, associating it with the map entry



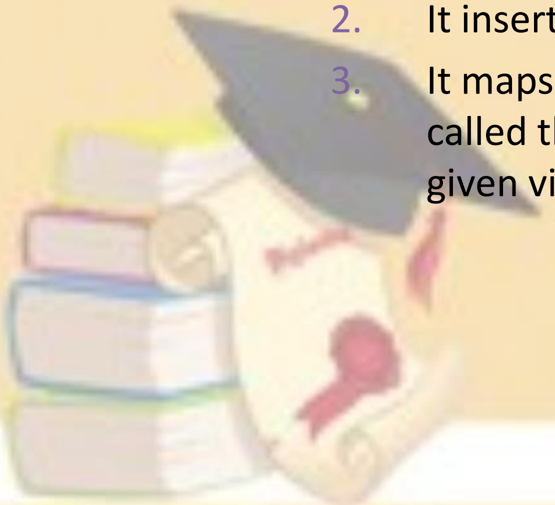
MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Allocating and Accessing Virtual Memory**
    - » At this point there are no pages resident in physical memory and no pages in the backing store. Everything is mapped virtually within the system
    - » When a program accesses the region, by reading or writing to a specific address in it, a fault occurs because that address has not been mapped to physical memory.

MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Allocating and Accessing Virtual Memory**
    - The kernel also recognizes that the VM object has no backing store for the page on which this address occurs. The kernel then performs the following steps for each page fault

1. It acquires a page from the free list and fills it with zeroes
2. It inserts a reference to this page in the VM object’s list of resident pages
3. It maps the virtual page to the physical page by filling in a data structure called the **pmap(**contains the page table used by the processor to map a given virtual address to the actual hardware address**)**



MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Shared Memory-** memory that can be written to or read from by two or more processes

    - can be inherited from a parent process, created by a shared memory server, or explicitly created by an application for export to other applications

    - fragile;If one program corrupts a section of shared memory, any programs that also use that memory share the corrupted data.

MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Shared memory**
    - » Uses for shared memory include the following:

      -sharing large resources such as icons or sounds

      -fast communication between one or more processes

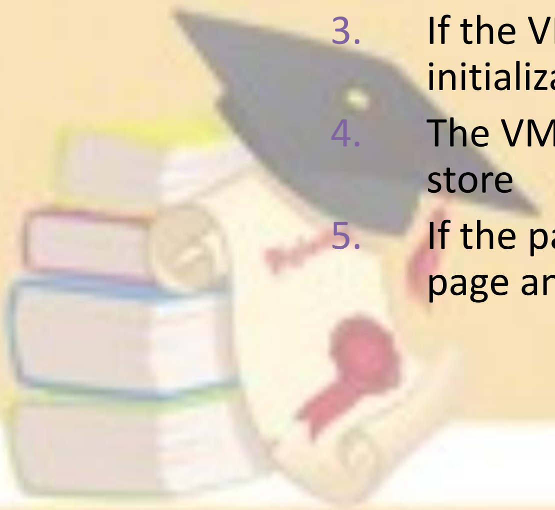
MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Paging Virtual Memory Out**
    - The kernel continuously compares the number of physical pages in the free list against a threshold value
    - When the number of pages in the free list dips below this threshold, the kernel reclaims physical pages for the free list by swapping inactive pages out of memory

MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Paging Virtual Memory Out**
    - the kernel iterates all resident pages in the active and inactive lists, performing the following steps:

1. If a page in the active list is not recently touched, it is moved to the inactive list
2. If a page in the inactive list is not recently touched, the kernel finds the page's VM object
3. If the VM object has never been paged before, the kernel calls an initialization routine that creates and assigns a default pager object
4. The VM object's default pager attempts to write the page out to the backing store
5. If the pager succeeds, the kernel frees the physical memory occupied by the page and moves the page from the inactive to the free list



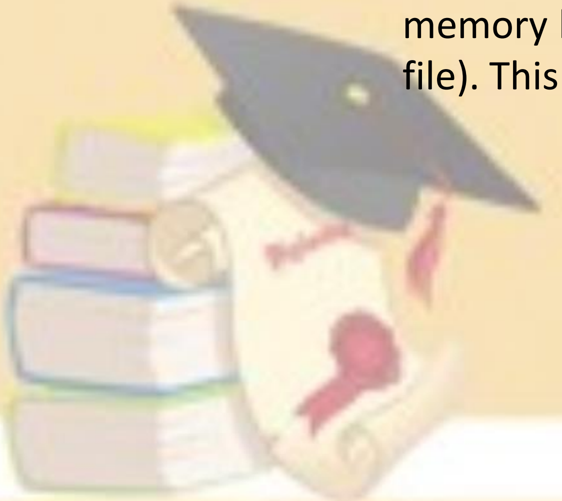
MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Paging Virtual Memory In**
    - The final phase of virtual memory management moves pages in the backing store back into physical memory.
    - A memory access fault initiates the page-in process. Memory access faults occur when code tries to access data at a virtual address that is not mapped to physical memory

MEMORY MANAGEMENT

- **Virtual Memory in Mac OS X**
  - **Paging Virtual Memory In**
    - There are two kinds of faults:

1. **soft fault-** occurs when the page of the referenced address is resident in physical memory but is currently not mapped into the address space of this process
2. **hard fault-** occurs when the page of the referenced address is not in physical memory but is swapped out to backing store (or is available from a mapped file). This is what is typically known as a page fault.



Thank you
for
listening!

[illegible]