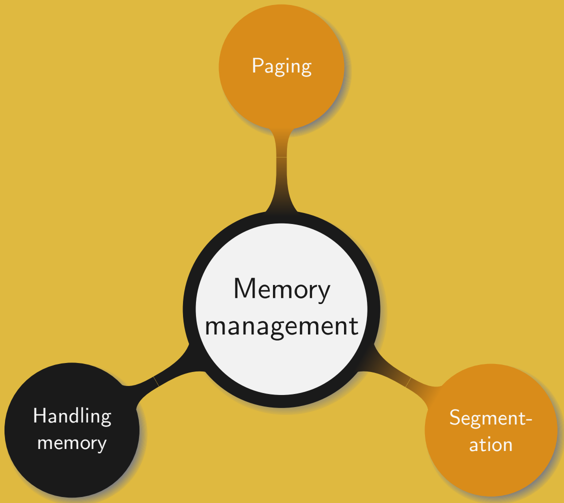
Introduction to Operating Systems

6. Memory management

Manuel – Fall 2021

Chapter organisation



Memory hierarchy

Access time		Capacity
1 ns	Registers	< 1 KB
2 ns	Cache	4 MB
10 ns	Main memory	1–8 GB
10 ms	Magnetic disk	200–1000 GB
10 s	Magnetic tape	400–800 GB

Problems related to memory:

- From expensive to cheap, fast to slow
- Job of the OS to handle the memory
- How to model the hierarchy?
- How to manage this abstraction?

Efficiently manage memory:

- Keep track of which part of the memory is used
- Allocate memory to processes when required
- Deallocate memory at the end of a process

Remark. It is the job of the hardware to manage the lowest levels of cache memory

Simplest model

No memory abstraction:

- Program sees the actual physical memory
- Programmers can access the whole memory
- Limitations when running more than one program:
  - Have to copy the whole content of the memory into a file when switching program
  - No more than one program in the memory at a time
  - More than one program is possible if using special hardware

No abstraction leads to two main problems:

- Protection: prevent program from accessing other's memory
- Relocation: rewrite address to allocate personal memory

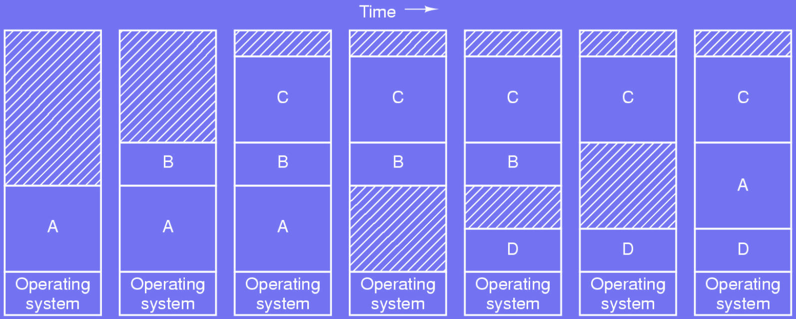
No abstraction leads to two main problems:

- Protection: prevent program from accessing other's memory
- Relocation: rewrite address to allocate personal memory

A solution is to set an address space:

- Set of addresses that a process can use
- Independent from other processes' memory

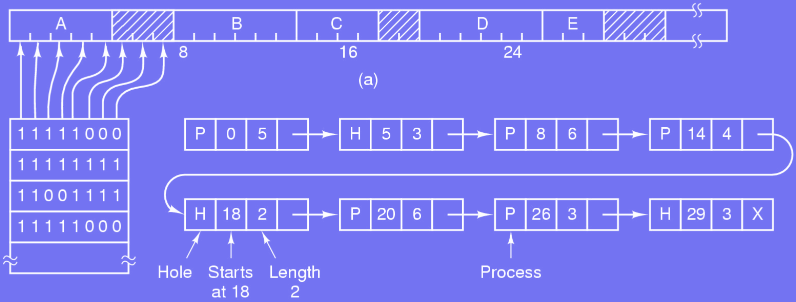
Memory size limitation



When booting many processes are started:

- As more programs are run more and more memory is needed
- More memory than available might be needed
- Processes are swapped in (out) from (to) the disk
- OS has to manage dynamically assigned memory

Bitmap and linked lists



Simple idea:

- Define some base size for an area s
- Split up the whole memory into n chunks of size s
- Keep track of the memory used in a bitmap or linked list

Ways to assign memory to processes:

- First fit: search for a hole big enough and use the first found
- Best fit: search whole list and use smallest, big enough hole
- Quick fit: maintain lists for common memory sizes, use the best

Ways to assign memory to processes:

- First fit: search for a hole big enough and use the first found
- Best fit: search whole list and use smallest, big enough hole
- Quick fit: maintain lists for common memory sizes, use the best

Characteristics:

- Speed: quick fit $>$ first fit $>$ best fit
- Locally optimal: quick fit $=$ best fit $>$ first fit
- Globally optimal: first fit $>$ quick fit $=$ best fit

Basics idea

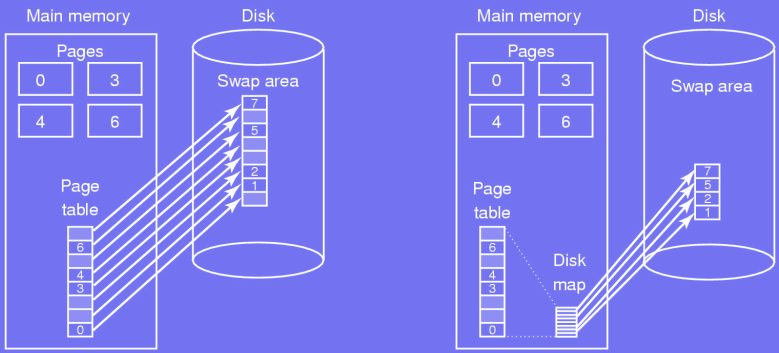
Virtual memory:

- Generalisation of the base and limit registers
- Each process has its own address space
- The address space is split into chunks called *pages*
- Each page corresponds to a range of addresses
- Pages are mapped onto physical memory
- Pages can be on different medium, e.g. RAM and swap

Swap partition principles:

- Simple way to allocate page space on the disk
- OS boots, swap is empty and defined by two numbers: its origin and its size
- When a process is started, a chunk of the partition equal to the process' size is reserved
- The new "origin" is computed
- When a process terminates its swap area is freed
- The swap is handled as a list of free chunks
- When a process starts, its swap area is initialised

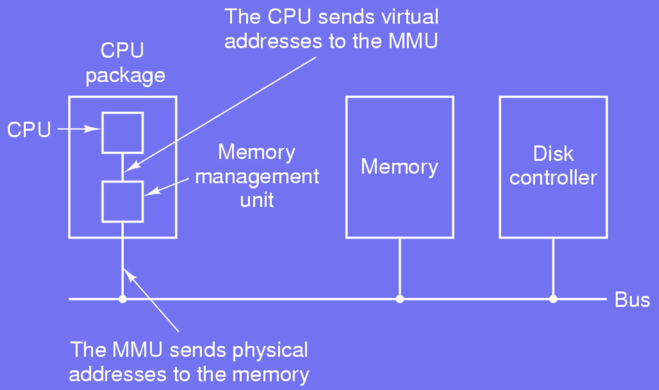
Initialising the swap area



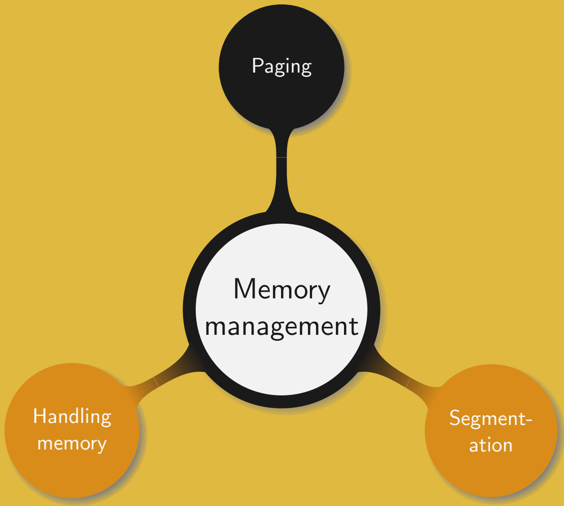
Two main strategies:

- Copy the whole process image to the swap area
- Allocate swap disk space on the fly

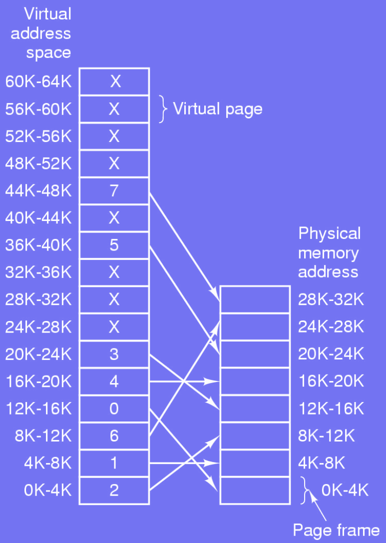
Memory Management Unit



Chapter organisation



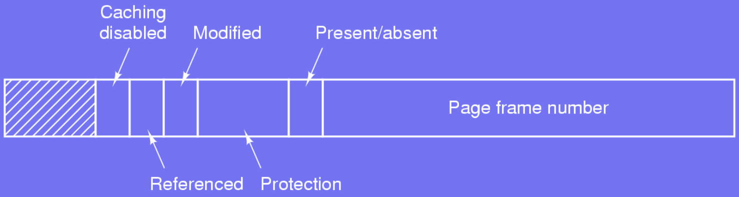
Virtual page and page frame



Organising the memory:

- Virtual address space divided into fixed-size units called *pages*
- Pages and *page frames* are usually of same size
- MMU maps virtual addresses to physical addresses
- MMU causes the CPU to trap on a page fault
- OS copies content of a little used page onto the disk
- Page frame loaded onto newly freed page

Page table



Structure of a page entry:

- Present|absent: 1|0; missing causes a page fault
- Protection: 1 to 3 bits: reading/writing/executing
- Modified: 1|0 = dirty|clean; page was modified and needs to be updated on the disk
- Referenced: bit used to keep track of most used pages; useful in case of a page fault
- Caching: important for pages that map to registers; do not want to use old copy so set caching to 0

Two main issue must be solved in a paging system:

- Mapping must be done efficiently
- A large virtual address space implies a large page table

Two main issue must be solved in a paging system:

- Mapping must be done efficiently
- A large virtual address space implies a large page table

Translation Lookaside Buffer (TLB):

- Hardware solution implemented inside the MMU
- Keeps track of few most used pages
- Features the same fields as for page table entries including the virtual page number and page frame

On a page fault the following operations are performed:

- Choose a page to remove from the memory
- If the page was modified while in the memory it needs to be rewritten on the disk; otherwise nothing needs to be done
- Overwrite the page with the new memory content

How to optimize the selection of the page to be evicted?

Page replacement – Optimal solution‘

Determining which page to remove when a page fault occurs:

- Label and order all the pages in memory
- The page with lower label is used first
- The page with larger label is swapped out of the memory

Page replacement – Optimal solution‘

Determining which page to remove when a page fault occurs:

- Label and order all the pages in memory
- The page with lower label is used first
- The page with larger label is swapped out of the memory

Can the information be known ahead of time?

Page replacement – LRU

Recently heavily used pages are very likely to be used again soon

Hardware solution, for $n \times n$ page frames:

- Initialise a binary $n \times n$ matrix to 0
- When frame k is used set row k to 1 and column k to 0
- Replace the page with the smallest value

Page replacement – LRU

Recently heavily used pages are very likely to be used again soon

Hardware solution, for $n \times n$ page frames:

- Initialise a binary $n \times n$ matrix to 0
- When frame k is used set row k to 1 and column k to 0
- Replace the page with the smallest value

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

The *M* and *R* bits

Use of the *M* and *R* bits for each page table entry:

- Software solutions require some hardware information
- OS needs to collect information on page usage
- Process starts: none of its page table entries are in memory
- Page is referenced: set the *R* bit
- Page is written: set the *M* bit
- *M* and *R* must be updated on every memory reference

Simulating LRU in software:

- For each page initialise an n-bit software counter to 0
- At each clock interrupt the OS scans all the pages in memory
- Shift all the counters by 1 bit to the right
- Add $2^{n-1} \cdot R$ to the counter

Example. $n = 8$ with 4 pages over 4 clock interrupts

t	t_0	t_1	t_2	t_3
R	$\begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$
$p1$	10000000	11000000	11100000	11110000
$p2$	00000000	10000000	11000000	01100000
$p3$	10000000	01000000	00100000	00010000
$p4$	00000000	00000000	10000000	01000000

Page replacement – Aging

Simulating LRU in software:

- For each page initialise an n-bit software counter to 0
- At each clock interrupt the OS scans all the pages in memory
- Shift all the counters by 1 bit to the right
- Add $2^{n-1} \cdot R$ to the counter

Example. $n = 8$ with 4 pages over 4 clock interrupts

t	t_0	t_1	t_2	t_3
R	$\begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$
$p1$	10000000	11000000	11100000	11110000
$p2$	00000000	10000000	11000000	01100000
$p3$	10000000	01000000	00100000	00010000
$p4$	00000000	00000000	10000000	01000000

Counter has a finite number of bits, a state is lost after $n \cdot t$

Definitions

Basic notions related to paging:

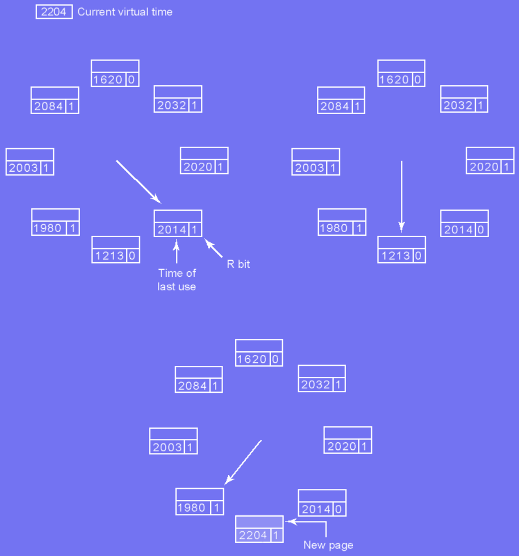
- Demand paging: pages are loaded on demand
- Locality reference: during an execution phase a process only access a small fraction of all its pages
- Working set: set of pages currently used by a process
- Thrashing: process causes many page fault due to a lack of memory
- Pre-paging: pages loaded in memory before letting process run
- Current virtual time: amount of time during which a process has used the CPU
- τ: age if the working set

Page replacement – WSClock

Using a circular list of page frames for pages which have been inserted:

- Each entry is composed of time of last use, R and M bits
- On a page fault examine the pages the hand points to
- If $R = 1$, bad candidate: set R to 0 and advance hand
- If $R = 0$, age $> \tau$
  - If page is clean, then use page frame
  - Otherwise schedule write, move the hand repeat algorithm
- If hand has completed one cycle
  - If at least one write was scheduled, keep the hand moving until a write is completed and a page frame becomes available
  - Otherwise (i.e. all the pages are in the working set) take any page ensure it is clean (or write it to the disk) and use its corresponding page frame

Page replacement – WSClock



Local vs. global allocation

Onto which set should the page replacement algorithm be applied:

- Local, i.e. within the process:
    - Allocate a portion of the whole memory to a process
    - Only use the allocated portion
    - Number of page frames for a process remains constant
- Global, i.e. within the whole memory:
    - Dynamically allocate page frames to a processes
    - Number of page frames for a process varies over time

Local vs. global allocation

Onto which set should the page replacement algorithm be applied:

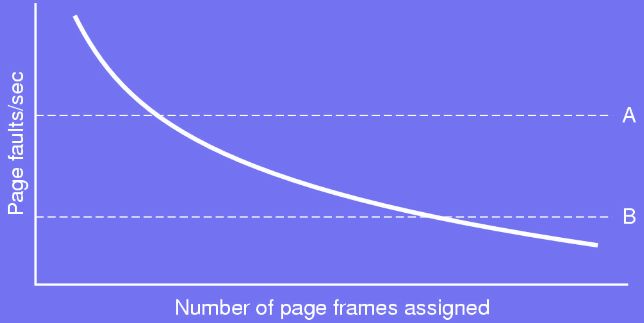
- Local, i.e. within the process:
  - Allocate a portion of the whole memory to a process
  - Only use the allocated portion
  - Number of page frames for a process remains constant
- Global, i.e. within the whole memory:
  - Dynamically allocate page frames to a processes
  - Number of page frames for a process varies over time

Which approach is best?

Page fault frequency

Adjusting the number of pages:

- Start process with a number of pages proportional to its size
- Adjust page allocation based on the page fault frequency
  - Count number of page fault per second
  - If larger than A then allocate more page frames
  - If below B then free some page frames



Finding optimal page size given a page frame size:

- In average half of the last page is used (internal fragmentation)
- The smaller the page size, the larger the page table

Page size

Finding optimal page size given a page frame size:

- In average half of the last page is used (internal fragmentation)
- The smaller the page size, the larger the page table

Page size p, process size s bytes, average size for page entry e and overhead o:

$$o = \frac{se}{p} + \frac{p}{2}$$

Differentiate with respect to p and equate to 0:

$$\frac{1}{2} = \frac{se}{p^2}$$

Optimal page size: $p = \sqrt{2se}$

Common page frame sizes: 4KB or 8KB

Page sharing

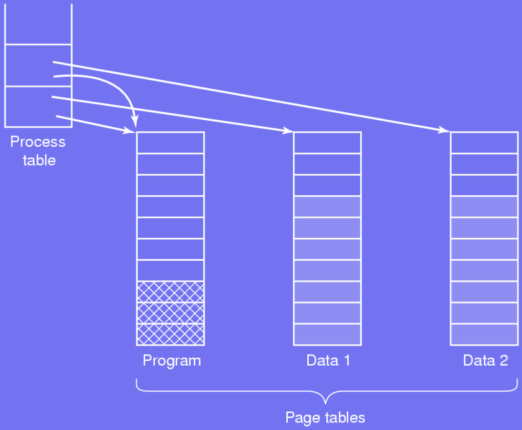
Decrease memory usage by sharing pages:

- Pages containing the program can be shared
- Personal data should not be shared

Page sharing

Decrease memory usage by sharing pages:

- Pages containing the program can be shared
- Personal data should not be shared



Several basic problems arise:

- On a process switch do not remove all pages if required by another process: would generate many page fault
- When a process terminates do not free all the memory if it is required by another process: would generate a crash
- How to share data in read-write mode?

Importance of paging in the OS

OS involved in paging related work on four occasions:

- Process creation:
  1. Determine process size
  2. Create process' page table (allocate and initialise memory)
  3. Initialise swap area
  4. Store information related to the swap area and page table in the process table
- Process execution:
  1. MMU resets for the new process
  2. Flush the TLB
  3. Make the new process' page table the current one

Importance of paging in the OS

OS involved in paging related work on four occasions:

- Page fault:
  1. Read hardware register to determine origin of page fault
  2. Compute which page is needed
  3. Locate the page on the disk
  4. Find an available page frame and replace its content
  5. Read the new page frame
  6. Rewind to the faulting instruction and re-execute it
- Process termination:
  1. Release page table entries, pages, and disk space
  2. Beware of any page that could be shared among several processes

Page fault handling

Process on a page fault:

1. Trap to the kernel is issued; program counter is saved in the stack; state of current instruction saved on some specific registers
2. Assembly code routine started: save general registers and other volatile information
3. OS search which page is requested
4. Once the page is found: check if the address is valid and if process is allowed to access the page. If not kill the process; otherwise find a free page frame
5. If selected frame is dirty: have a context switch (faulting process is suspended) until disk transfer has completed. The page frame is marked as reserved such as not to be used by another process

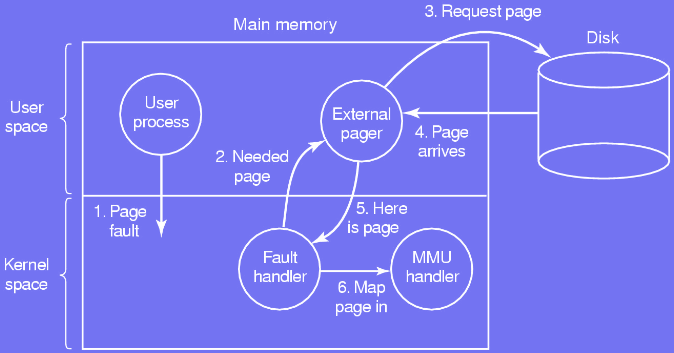
Page fault handling

6. When page frame is clean: schedule disk write to swap in the page. In the meantime the faulting process is suspended and other processes can be scheduled
7. When receiving a disk interrupt to indicate copy is done: page table is updated and frame is marked as being in a normal state
8. Rewind program to the faulting instruction, program counter reset to this value
9. Faulting process scheduled
10. Assembly code routine starts: reload registers and other volatile information
11. Process execution can continue

Policy and mechanism

Example showing how to dissociate policies from mechanisms:

- Low level MMU handler: architecture dependent
- Page fault handler: kernel space
- External handler: user space



Where should the page replacement algorithm go?

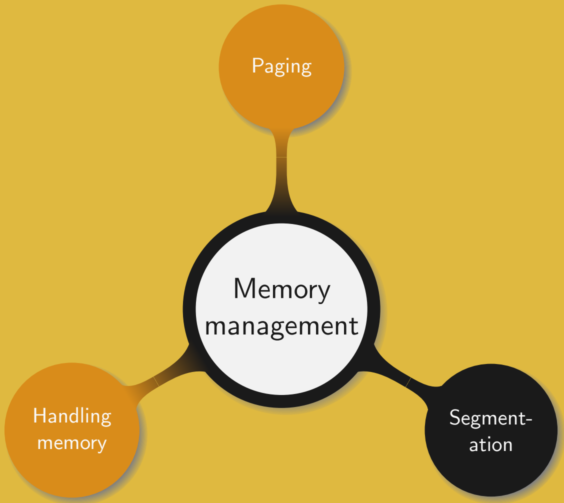
User space

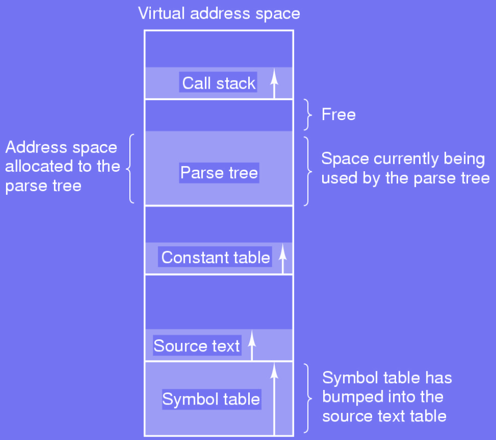
- Use some mechanism to access the R and M bits
- Clean solution
- Overhead resulting from crossing user-kernel boundary several times
- Modular code, more flexibility

Kernel space

- Fault handler sends all information to external pager (which page was selected for removal)
- External pager writes the page to the disk
- No overhead, faster

Chapter organisation

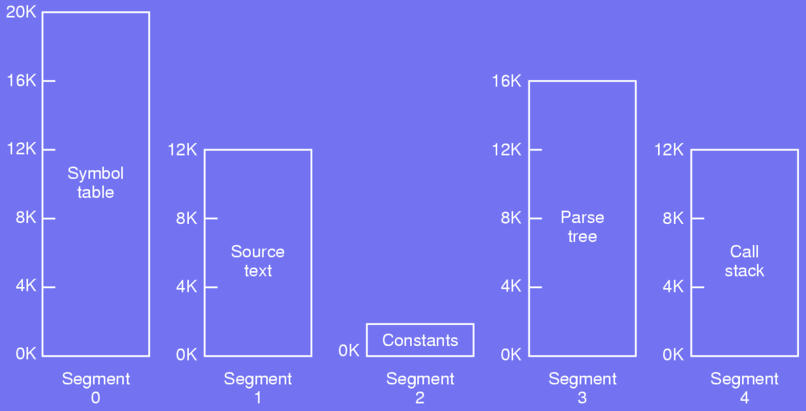




Basic paging summary:

- One dimension
- Starting address and a limit
- Example: compiler
- If many variables: symbol table expands on source text

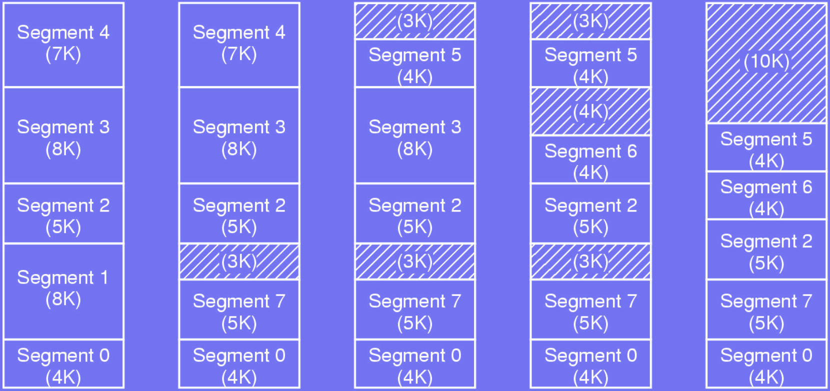
Segmentation



Handling segmentation in the OS:

- Each segment has a number and an offset
- Segment table: contains the starting physical address of each segment, the *base*, together with its size, the *limit*
- Segment table base register: points to the segment table
- Segment table length register: number of segments used in a program

External fragmentation and compaction



Paging vs. segmentation

Considerations	Paging	Segmentation
Number of linear address space	1	many
Limited by the size of the RAM	no	no
Possible to separate and protect data and procedures	no	yes
Sharing procedures between users or programs	complex	easy

Key points

- What are the two main ways to model memory?
- What is the swap area?
- Cite two main page replacement algorithms
- Discuss the differences between paging and segmentation
- Explain external and internal fragmentation

Thank you!