Operating Systems and Concurrency

Memory Management 3

COMP2007

Geert De Maere
(Dan Marsden)
{Geert.DeMaere,Dan.Marsden}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2023

Recall

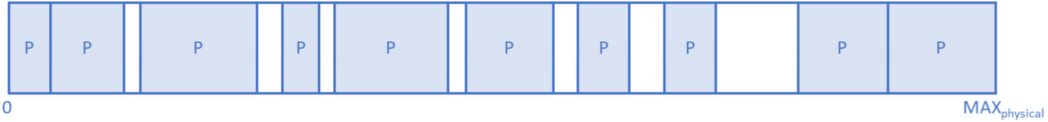
Last Lecture

- **Dynamic relocation** & **protection**, **base** & **bound** registers (**logical** $\Rightarrow$ **physical** address)
- **Dynamic partitioning & segmentation** (**internal** $\Rightarrow$ **external** fragmentation)
- **Free space** management (bitmaps and linked lists)

Sketch of the Scene

Where Did we Get To?

- **Contiguous memory allocation** methods:
  - Fixed (non-)equal size partitions
  - Dynamic partitioning
  - Segmentation (individual segments are contiguous)
  - `malloc(), free()`
- The **amount of memory** requested by processes can be **differ**
- Memory will become **fragmented** over time



Sketch of the Scene

What do we Need?

- A method to **keep track** of free memory
- A strategy to **allocate free** memory when requested by a process
- Approaches to **prevent fragmentation** whenever we can

Overview

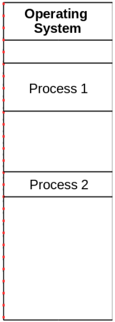
Goals for Today

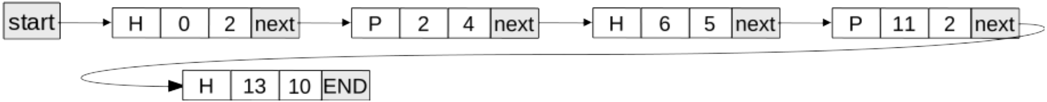
- **Memory allocation:**
  - First fit, best fit, next fit, worst fit, quick fit
  - Buddy algorithm
- **Prevent fragmentation** - **non-contiguous** memory management
  - **(Segmentation)**
  - **Paging** - page tables, address translation

Memory Allocation

First Fit

- **First fit** scans **from the start** of the list until an **sufficiently large gap** is found
  - `if` the space is **the exact size** then all the space is **allocated**
  - `else` the space is **split**:
    - The first entry is set to the **size requested** and marked “**used**”
    - The second entry is set to **remaining size** and marked “**free**”

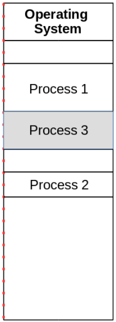


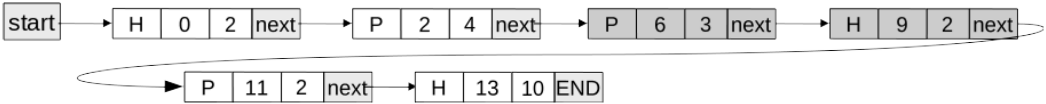


Memory Allocation

First Fit

- **First fit** scans **from the start** of the list until an **sufficiently large gap** is found
  - `if` the space is **the exact size** then all the space is **allocated**
  - `else` the space is **split**:
    - The first entry is set to the **size requested** and marked "**used**"
    - The second entry is set to **remaining size** and marked "**free**"

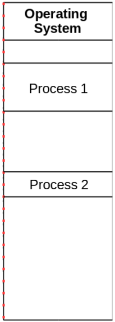


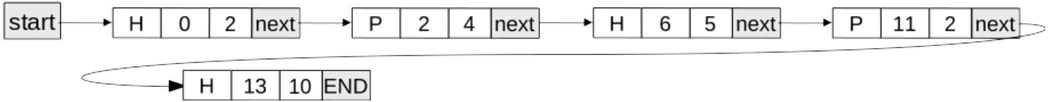


Memory Allocation

Next Fit

- The **next fit algorithm** maintains a record of where it got to:
  - It **restarts its search** from **where it stopped last time**
  - It gives an **even chance to all memory** to get allocated (first fit concentrates on the start of the list)
- Simulations have shown that next fit **performs worse** than first fit

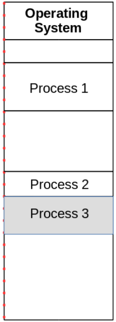


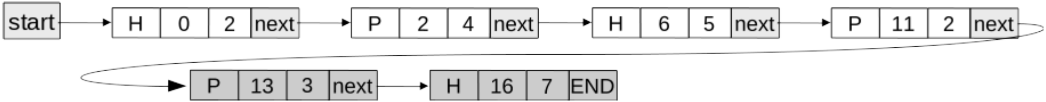


Memory Allocation

Next Fit

- The **next fit algorithm** maintains a record of where it got to:
    - It **restarts its search** from **where it stopped last time**
    - It gives an **even chance to all memory** to get allocated (first fit concentrates on the start of the list)
- Simulations have shown that next fit **performs worse** than first fit





Memory Allocation

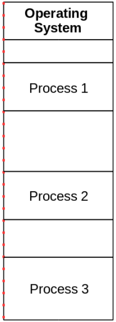
First vs. Next Fit

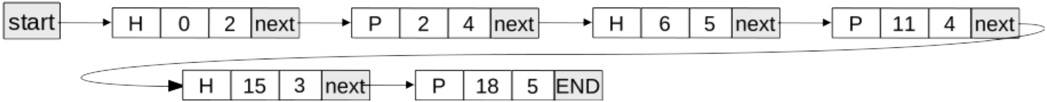
- **First fit** is a fast and looks for the **first available hole**
  - It does **not account** for a **better fitting** hole later
  - It may **break up** a **large holes** early in the list
- **Next fit** does not change this

Memory Allocation

Best Fit

- **Best fit** searches the **entire linked** list for the **smallest hole** big enough to satisfy the request
  - It is **slower** than first fit
  - Result in **small leftover holes** (wastes memory)

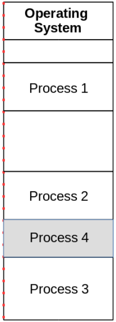


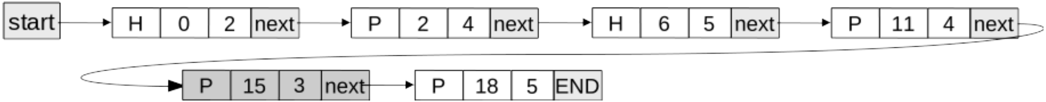


Memory Allocation

Best Fit

- **Best fit** searches the **entire linked** list for the **smallest hole** big enough to satisfy the request
  - It is **slower** than first fit
  - Result in **small leftover holes** (wastes memory)

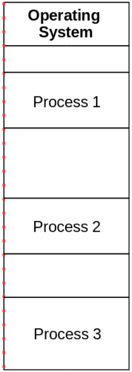


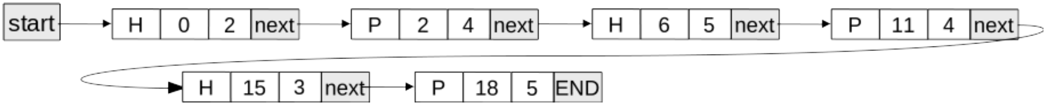


Memory Allocation

Worst Fit

- **Best fit** split an empty partition in **small holes**
- **Worst fit** finds the **largest available partition** and splits it
  - The left over part **will still be large** (potentially more useful)
  - Simulations have shown that **worst fit** is **not good** in practice

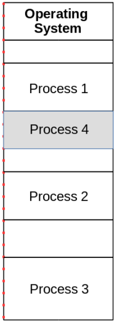


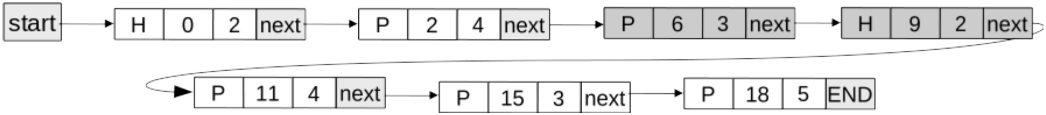


Memory Allocation

Worst Fit

- **Tiny holes** are created when **best fit** split an empty partition
- The **worst fit algorithm** finds the **largest available empty partition** and splits it
  - The **left over part will still be large** (and **potentially more useful**)
  - Simulations have also shown that worst fit is **not very good either**!





Memory Allocation

Summary

- **First fit**: allocate **first block** that is **large enough**
- **Next fit**: allocate **next block** that is large enough, i.e. **starting from the current location**
- **Best fit**: choose block that **matches** required size **closest** - O(N) complexity
- **Worst fit**: choose the **largest possible block** - O(N) complexity

Dynamic Partitioning

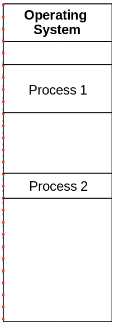
Allocating Available Memory: Quick Fit and Others

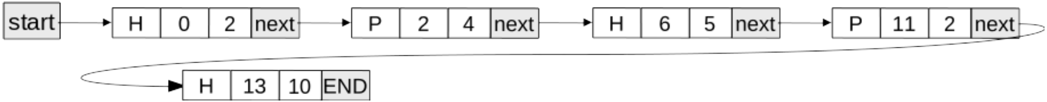
- **Quick fit** maintains **lists of commonly used sizes**
  - For example a separate list for each of 4K, 8K, 12K, 16K, etc., holes
  - **Odd sizes** can either go into the **nearest size** or into a **special separate list**
- It is **much faster** to find the required size hole using **quick fit**
- Similar to **best fit**, it has the problem of creating **many tiny holes**
- Finding neighbours for **coalescing** (combining empty partitions) becomes more difficult/time consuming

Memory Allocation

Coalescing

- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
- Both **neighbours** are examined when a **block is freed**
  - `if` either (or both) are **also free**
  - `then` the two (or three) **entries are combined** into one larger block by adding up the sizes
    - The earlier block in the linked list gives the **start point**
    - The **separate links are deleted** and a single link inserted

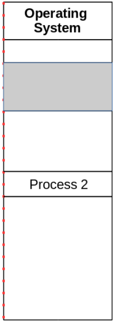


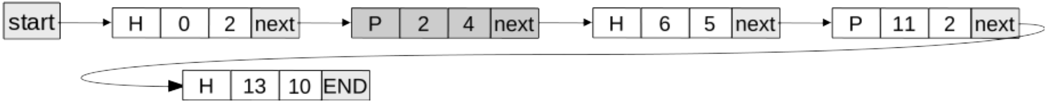


Memory Allocation

Coalescing

- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
- Both **neighbours** are examined when a **block is freed**
  - `if` either (or both) are **also free**
  - `then` the two (or three) **entries are combined** into one larger block by adding up the sizes
    - The earlier block in the linked list gives the **start point**
    - The **separate links are deleted** and a single link inserted

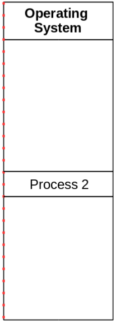


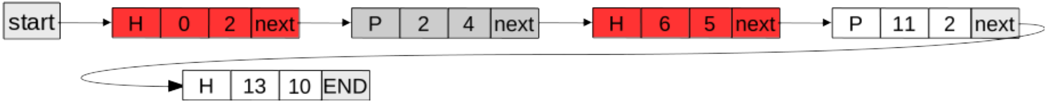


Memory Allocation

Coalescing

- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
- Both **neighbours** are examined when a **block is freed**
  - `if` either (or both) are **also free**
  - `then` the two (or three) **entries are combined** into one larger block by adding up the sizes
    - The earlier block in the linked list gives the **start point**
    - The **separate links are deleted** and a single link inserted

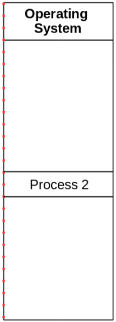




Memory Allocation

Coalescing

- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
- Both **neighbours** are examined when a **block is freed**
    - `if` either (or both) are **also free**
    - `then` the two (or three) **entries are combined** into one larger block by adding up the sizes
        - The earlier block in the linked list gives the **start point**
        - The **separate links are deleted** and a single link inserted



start → | H | 0 | 11 | next | → | P | 11 | 2 | next | → | H | 13 | 10 | END |

Memory Allocation

Compacting

- Even with coalescing, **free blocks** may still **distributed** across memory
- **Compacting** can be used but is harder to implement and **time consuming**
- Processes may have to be **moved/swapped out**, **free space coalesced**, and processes **swapped back in** at lowest available location

Contiguous Allocation Schemes

Overview and Shortcomings

- Different contiguous memory allocation schemes have different advantages/disadvantages
  - **Mono-programming** is easy but does result in **low resource utilisation**
  - **Fixed partitioning** facilitates **multi-programming** but results in **internal fragmentation**
  - **Dynamic partitioning & segmentation** facilitate **multi-programming**, reduce **internal fragmentation**, but result in **external fragmentation** (allocation methods, coalescing, and compacting help)
- Can we design a memory management scheme that **resolves the shortcomings** of **contiguous memory schemes**?

Paging

Principles

- **Paging** uses **fixed partitioning** and **code re-location** to devise a new ***non-contiguous*** management scheme:
  - Memory is split into **much smaller blocks**
  - A process is allocated **one or more blocks** (e.g., a 11KB process would take up three 4KB blocks)
  - Blocks do **not** have to be stored **in contiguous in physical memory**
  - The process **perceives** them to be **contiguous**
- Benefits of non-contiguous schemes include:
  - **Internal fragmentation** is reduced to the **last "block"** only
  - **No external fragmentation** (blocks in physical address space are **stacked** directly on top of each other)

Paging

Principles (Cont'ed)

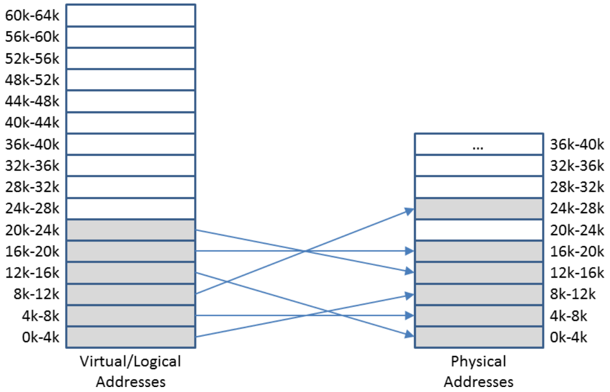


Figure: Paging in physical memory with multiple processes

Paging

Principles (Cont'ed)

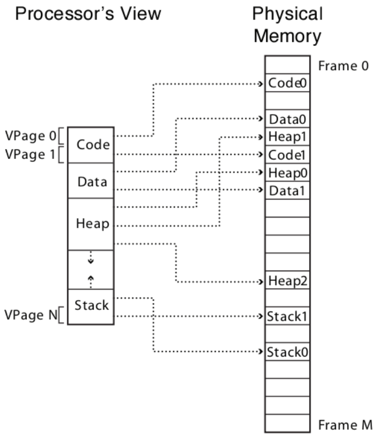


Figure: Paging in main memory with multiple processes (Anderson)

Paging

Principles (Cont'ed)

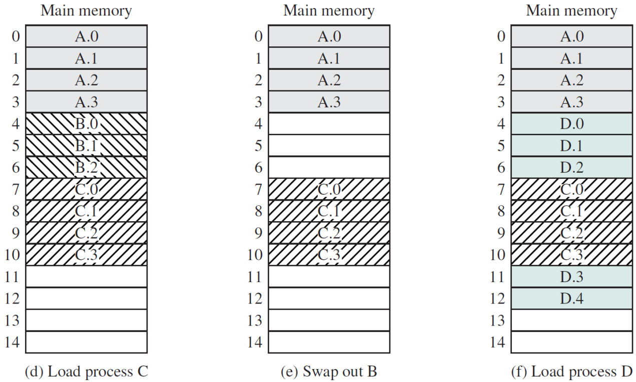


Figure: Concept of Paging (Stallings)

Paging

Principles: Definitions

- A **page** is a small block of **contiguous memory** in the **logical address space**, i.e. as seen by the process
- A **frame** is a **small contiguous block** in **physical memory**
- Pages and frames (commonly) have the **same size**:
  - The size is usually a power of 2
  - Sizes range between 512 bytes and 1Gb

Paging

Relocation

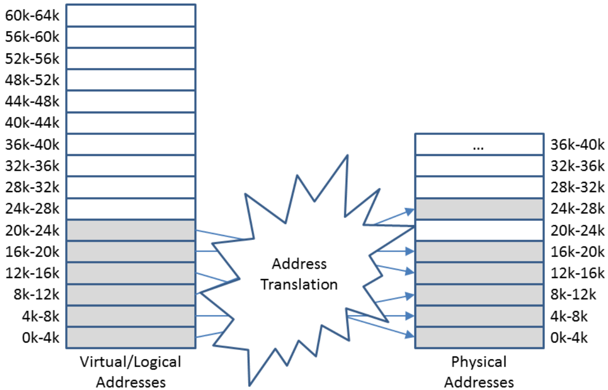


Figure: Address Translation

Segmentation

Segmentation Table

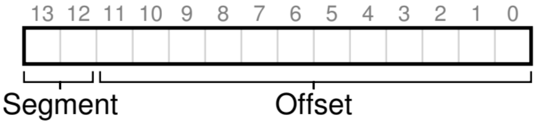


Figure: Logical address for segmentation (offset = position relative to start of segment)

Segment	**index**	**Base**	**Bound**	**RWX**
Code	00	...	...	...
Data	01	...	...	...
Heap	10	...	...	...
Stack	11	...	...	...

Paging
Page Table

Figure: Logical address for page (offset = position relative to start of segment)

Page Number	**Frame Number**	**...**	**...**	**RWX**
0000	1010	...	...	...
0001	0100	...	...	...
0010	1111	...	...	...
...	...	...	...	...

Paging

Page Table

Figure: Logical address for page (offset = position relative to start of segment)

Page Number	**Frame Number**	**...**	**...**	**RWX**
0000	1010	...	...	...
0001	0100	...	...	...
0010	1111	...	...	...
...	...	...	...	...

Paging

Page Table

Figure: Logical address for page (offset = position relative to start of segment)

Page Number	Frame Number	...	...	RWX
0000	1010	...	...	...
0001	0100	...	...	...
0010	1111	...	...	...
...	...	...	...	...

Paging

Page Table

Figure: Logical address for page (offset = position relative to start of segment)

Page Number	Frame Number	...	...	RWX
0000	1010	...	...	...
0001	0100	...	...	...
0010	1111	...	...	...
...	...	...	...	...

Paging

Page Table

Figure: Logical address for page (offset = position relative to start of segment)

Page Number	**Frame Number**	**. . .**	**. . .**	**RWX**
0000	1010	. . .	. . .	. . .
0001	0100	. . .	. . .	. . .
0010	1111	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .

Paging

Relocation

- **Logical address** (page number, offset within page) needs to be **translated** into a **physical address** (frame number, offset within frame)
- **Multiple "base registers"** will be required:
  - Each **page** has a **"base register"** that identifies the start of the associated **frame**
  - A **set of base registers** must be maintained for every process
- The **set of base registers** is stored in the **page table**

Paging

Relocation: Address Translation

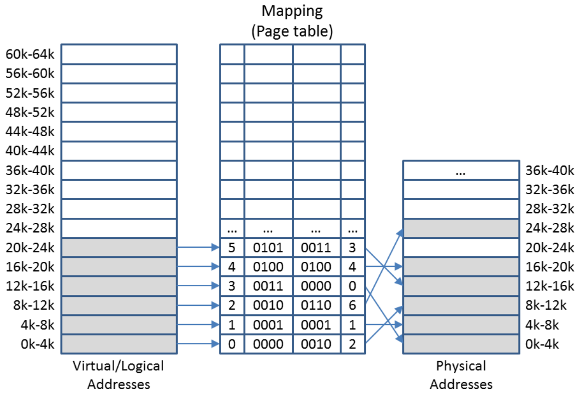


Figure: Address Translation

Paging

Relocation: Page Tables

- The page table maps the **page number** onto the **frame number** (logical $\Rightarrow$ physical address)
  - frameNumber=f(pageNumber)
- The **page number** is used as **index to the page table** that lists the **number of the associated frame**
- From the **frame number** one can calculate the base (offset remains unchanged)is used as **index to the page table** that lists the **number of the associated frame**
- Every process has its **own page table** containing its own **set of "base registers"**
- The **operating system** maintains a **list of free frames**

Paging

Address Translation: Implementation

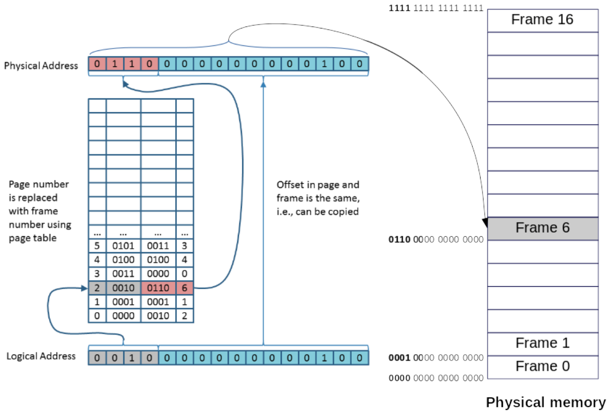


Figure: Address Translation

Test Your Understanding

Page tables and memory addressing

Page Tables

Given a **64-bit machine** that uses paging, and a page/frame size of **4KB**.

- What would be the maximum **number of frames**? 2^64 / (4*1024)
- How many **entries** will we have in the **page table**? 2^64 / (4*1024)
- How many **pages** do we have in a **17KB process**?
- How much **memory is wasted** in the last frame?

Recap

Take-Home Message

- Memory **allocation**, **coalescing** and **compacting** in dynamic partitioning
- **Paging**, **page tables**, and **address translation**