Content Quiz 7-1: Memory Management Parts 1-3 (Partitioning)

Question 1

2 / 2 pts

All of the memory resources needed by a process are referred to as [process] [data] . For a process to be able to execute, its data must be in [main] [memory] .

Answer 1:
process

Answer 2:
data

Answer 3:
main

Answer 4:
memory

Question 2

1 / 1 pts

A mechanism that can be used to allow multiple processes to reside in memory simultaneously is [memory] [partitioning] .

Answer 1:
memory

Answer 2:
partitioning

Question 3

1 / 1 pts

When using static, equal sized partitions, a given process is always mapped to the same physical partition .

Answer 1:
always mapped to the same physical partition

PartialQuestion 4

0.5 / 1 pts

An advantage of static equal sized partition scheme is its protection . A disadvantage is that it suffers from internal fragmentation.

Answer 1:
protection

Answer 2:
internal

Question 5

1 / 1 pts

In a scheme with static unequal sized partitions, the system needs to store the base address and the limit in order to enforce protection.

Answer 1:
base

Answer 2:
limit

Question 6

1 / 1 pts

A scheme that allows dynamic partitions of varying sizes starts with unpartitioned memory, called free space , and then creates partitions dynamically based on process needs.

Answer 1:
free

Answer 2:
space

Question 7

1 / 1 pts

A possible mechanism to keep track of free spaces is to use a linked list.

Answer 1:
mechanism

Question 8

1 / 1 pts

In the first fit approach, the system chooses the first free space that is large enough to fit the process being placed.

Answer 1:
first

Answer 2:
large

Question 9

1 / 1 pts

Both the best fit and the worst fit approaches require traversal of the entire free space list. True or False.

True

False

Question 10

1 / 1 pts

In general, programs issue logical or relative addresses and not physical or absolute addresses.

Answer 1:
logical or relative

Answer 2:
physical or absolute

Content Quiz 8-2: File Management

Due No due date **Points** 18 **Questions** 18 **Time Limit** None
Allowed Attempts Unlimited

Instructions

Answer this quiz based on your understanding of the **File Management** videos and assigned readings.

You may attempt this quiz multiple times. Only the highest score will be kept.

Note that your answers must be exact. Correct spelling and spacing are needed.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	125 minutes	18 out of 18

Correct answers are hidden.

Score for this attempt: **18** out of 18
Submitted Mar 1 at 7:06pm
This attempt took 125 minutes.

Answer the following questions after watching the **File Management** videos.

Question 1

1 / 1 pts

Data on a disk is organized as a sequence of equal-sized ___________.

blocks

Question 2

1 / 1 pts

The following are characteristics of non-volatile , long-term storage:

- Must store very large amounts of information
- Must persist beyond process lifetime
- Must be accessible by multiple processes at once

Answer 1:

non-volatile

Answer 2:

long-term

Question 3

1 / 1 pts

What is one of the **benefits** of using contiguous allocation?

- Reading a file is very slow
- It results in external fragmentation
- (selected) Figuring out the location of a specific file block requires a very simple calculation
- The maximum size of a file must be known

Question 4

1 / 1 pts

When using a File Allocation Table that is loaded in memory, when is a disk access is needed?

- To locate a random block within a file
- (selected) To actually fetch a block after it has been located
- Both to locate random blocks within a file AND to fetch their contents after they are located

Question 5

1 / 1 pts

Operating systems that use **i-node** file allocation typically impose a limit on:

- The number of files that a process can open at any given time
- The total number of files that can be open at any given time on a system
- (selected) The number of files that a process can open at any given time AND the total number of files that can be open at any given time on a system

Answer the following questions after reading the following **Unix - File Management** tutorial: https://www.tutorialspoint.com/unix/unix-file-management.htm (https://www.tutorialspoint.com/unix/unix-file-management.htm)

Question 6

1 / 1 pts

What is the Unix command to list the files and directories in the current directory?

Provide the command only, without any options.

ls

Question 7

1 / 1 pts

The owner of a file is listed in the ________ column of a detailed listing, i.e., one produced using the **-l** option.

third

Question 8

1 / 1 pts

What does the **d** prefix in a file listing indicate?

- Regular file, such as an ASCII text file, binary executable, or hard link.
- Block special file. Block input/output device file such as a physical hard drive.
- Character special file. Raw input/output device file such as a physical hard drive.
- Symbolic link file. Links on any regular file.

- (selected) Directory file that contains a listing of other files and directories.
- Named pipe. A mechanism for interprocess communications.
- Socket used for interprocess communication.

Question 9

1 / 1 pts

In Unix, we can use the metacharacter [*] to match zero or more characters and the metacharacter [?] to match a single character.

Answer 1:

*

Answer 2:

?

Question 10

1 / 1 pts

The __ editor can be used to create ordinary (text) files in Unix.

Provide the name of the editor or the command used to launch it.

[vi]

Question 11

1 / 1 pts

The ___ command can be used to rename a file in Unix.

mv

Answer the following questions after reading the following **Unix - Directory Management** tutorial: https://www.tutorialspoint.com/unix/unix-directories.htm (https://www.tutorialspoint.com/unix/unix-directories.htm)

Question 12

1 / 1 pts

What Unix command can be used to determine your location in the filesystem, i.e., the current working directory?

pwd

Question 13

1 / 1 pts

What Unix command can be used to create new directories in Unix?

mkdir

Question 14

1 / 1 pts

What Unix command can be used to change to another directory in Unix?

cd

Answer the following questions after reading the following **Unix - File Permission / Access Modes** tutorial: [https://www.tutorialspoint.com/unix/unix-file-permission.htm (https://www.tutorialspoint.com/unix/unix-file-permission.htm)](https://www.tutorialspoint.com/unix/unix-file-permission.htm)

Question 15

1 / 1 pts

Match the given description file with its corresponding access mode.

Grants the capability to view the contents of the file.	Read
Grants the capability to modify, or remove the content of the file.	Write
User with execute permissions can run a file as a program.	Execute

Question 16

1 / 1 pts

When using absolute permissions, what **number** can be used to grant permissions to view and modify the contents of a file and also to run it if it is a program?

Do not use the symbolic mode to answer this question, you must provide a number.

7

Question 17

1 / 1 pts

What is the Unix command to change file or directory permissions?

chmod

Question 18

1 / 1 pts

What is the Unix command to change the owner or a file or directory?

chown

Quiz Score: **18** out of 18

Embedded Systems File Management

Recall that embedded systems are typically found in devices that are not commonly seen as computers. Often referred to as Internet of Things (IoT) devices, these include consumer products such as MP3 players, TV sets, DVD players, pacemakers and cars. Similarly to desktop computers and servers, embedded systems use file systems to permanently store information. However, the file management requirements of embedded systems are different due to the specialized nature and more limited resources of these systems. Among these we include the following:

- Increased bandwidth, which may be needed to allow faster boot times.
- Reliability, which includes more resiliency in case of a power loss and the ability to self-recover from data corruption.
- Efficiency, which helps ensure better utilization of power, an important consideration in battery-operated devices.
- CPU utilization, which is more critical in systems with slower and less capable processors.
- Support for storage hardware based on flash (solid-state) memory.

Even though the operational needs of embedded systems are specialized and typically more modest than those of personal computers, security is a critical aspect that must be considered as part of their design and implementation. Embedded systems and in particular IoT devices can be vulnerable to several types of attacks that target sensitive data and the device's ability to operate correctly. Hence, the key security concerns for file systems in embedded devices include:

- Secure storage, which deals with ensuring the confidentiality and integrity of data stored in the embedded device.
- Identity management and user identification, which deals with ensuring that only authorized users and components have access to the data and functionality of the device.
- Resistance to tampering, which deals with maintaining the integrity of the device in case of physical or software-based attacks.

The ability to use standard methods to address security challenges can be significantly affected by the limited resources and constraints that are typical of embedded systems. Methods such as cryptography to secure data and complex security algorithms may not be applicable due the need to conserve battery power or the lack of sufficient computing power to implement them. Modern approaches to address these challenges focus on the use of lightweight and highly efficient cryptographic algorithms and secured file systems that are part of the core operating system. Secure file systems focus on protecting access to encryption keys and privileged data, as well as guaranteeing the integrity of data in the event of a power loss. Other approaches include the use of security extensions at the hardware level that make it possible to implement access controls.

Exercise 8-2: File Management

Due Mar 11 at 11:59pm **Points** 24 **Questions** 20
Available until Mar 11 at 11:59pm **Time Limit** None **Allowed Attempts** 2

Instructions

For this exercise, you will work through a series of short **File Management** command-line exercises in the Cloud9 environment and will complete the quiz questions shown below.

You can submit this exercise two (2) times. Only the highest score will be kept.

Note: Your answers must be exact, i.e., correct spelling and spacing are necessary.

This exercise assumes that you have already configured a Cloud9 environment for C++ (see **Cloud9SetUp slide show** (https://studentuncc-my.sharepoint.com/personal/swatso50_uncc_edu/_layouts/15/guestaccess.aspx?docid=0736c1193558844da9e8073ff4bcad902&authkey=AeUaKt1RFUSFJZf_dMVFI5w) for instructions).

This quiz was locked Mar 11 at 11:59pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	220 minutes	23.25 out of 24

Correct answers are hidden.

Score for this attempt: **23.25** out of 24
Submitted Mar 1 at 10:46pm
This attempt took 220 minutes.

General Instructions

- You will use the BASH shell in your Cloud9 environment to execute various Linux file management commands and view their output.
- Every time that you run a new command, you should make sure that you are not getting an error. Also, you should get a directory listing to verify

the results of the command you just ran.

- **Answers where you are asked to provide a command must be EXACT. Do NOT include unnecessary whitespace.**
  - Copy-paste works best.

Objectives

- Develop an understanding of Linux **file management**, **directory** and **file permission** commands
- Practice **manipulating files and directories** using Linux commands
- Practice **setting permissions** using Linux commands

Readings

- **https://www.tutorialspoint.com/unix/unix-file-management.htm (https://www.tutorialspoint.com/unix/unix-file-management.htm)**
- **https://www.tutorialspoint.com/unix/unix-directories.htm (https://www.tutorialspoint.com/unix/unix-directories.htm)**
- **https://www.tutorialspoint.com/unix/unix-file-permission.htm (https://www.tutorialspoint.com/unix/unix-file-permission.htm)**

Assignment Setup

For the following questions you will need to download a folder with test files. Using your Cloud9 C++ environment:

1. Type the following command into the terminal window to pull the project repository from GitLab:

```
git clone https://cci-git.uncc.edu/jbahamon/ITSC_3146_A_9_1
```

2. Change directory into the newly created directory (folder) named **ITSC_3146_A_9_1**

Before working on the following questions, you should work through the **Unix File Management** tutorial: **https://www.tutorialspoint.com/unix/unix-file-**

management.htm (https://www.tutorialspoint.com/unix/unix-file-management.htm)

Question 1

2 / 2 pts

Use a linux command that will output a **DETAILED list** of files to your screen. Answer the following questions about it based on the output:

1. Which file has the largest size in bytes? Part 2.pdf
2. Which file that has the smallest size in bytes? README.md
3. How many files are listed? 6
4. What types of files are listed? Both pdf AND md

Answer 1:

Part 2.pdf

Answer 2:

README.md

Answer 3:

6

Answer 4:

Both pdf AND md

Question 2

1 / 1 pts

Type the following command:

```
ls *.md
```

What is the output?

README.md

Question 3

1 / 1 pts

Use a linux command that will output a list of all files, including hidden ones. Answer the following questions about it based on the output:

What hidden **file** is revealed? .Some-hidden-file.p

What is the size of the hidden git folder in bytes? 4096

Answer 1:

.Some-hidden-file.pdf

Answer 2:

4096

Question 4

1 / 1 pts

Use **vi** to create a new file called “file1” with the following text:

```
This is a new file.
I created it in vi.
```

Save the file.

Use a Linux command to output the contents of the file with line numbers. The output should look like the following:

```
     1  This is a new file.
     2  I created it in vi.
```

Copy-paste the command that you used: ___________________

cat -b file1

Question 5

1.5 / 1.5 pts

Use a command to output the number of lines, words and file size of the **README.md** file.

1. How many lines are in the file according to this output?
   2
2. How many words? 26
3. How large is this file in bytes? 155

Please enter numbers only, be exact.

Answer 1:

2

Answer 2:

26

Answer 3:

155

Partial

Question 6

0.75 / 1.5 pts

Use a Linux command to print the word count of every **PDF** file in this directory **without specifically listing each file name in the command**.
Hint: Read the *metacharacters* section

Copy-paste the command you used: wc*.pdf

What is the **total number of words** in all of these files? 75865

Answer 1:

wc*.pdf

Answer 2:

75865

Question 7

2 / 2 pts

Use Linux commands to do each of the operations listed below. Provide the **exact command** that you used each time.

(a) Make 3 copies of file1. The copied files should be called copyfile1, copyfile2, copyfile3. Copy-paste the command you used (o*nly provide the command to make the first copy):* cp file1 copyfile1

(b) Rename file1 to renamedfile1. Copy-paste the command you used: mv file1 renamedfil

(c) Delete copyfile1. Copy-paste the command you used: rm copyfile1

(d) Delete copyfile2 and copyfile3 with a **single** command. **Warning: Do NOT use * or you will delete more than these two files!** Copy-paste the command you used: rm copyfile?

Answer 1:

cp file1 copyfile1

Answer 2:

mv file1 renamedfile1

Answer 3:

rm copyfile1

Answer 4:

rm copyfile?

Before working on the following questions, you should work through the **Unix Directory Management** tutorial: **https://www.tutorialspoint.com/unix/unix-directories.htm (https://www.tutorialspoint.com/unix/unix-directories.htm)**

Question 8

1 / 1 pts

Use a Linux command to create a directory called *temp*. Copy-paste the command you used: _________

mkdir temp

Question 9

1 / 1 pts

Use a Linux command to rename the *temp* directory to *newtemp*. Copy-paste the command you used: _________

mv temp newtemp

Question 10

1 / 1 pts

Use a Linux command to delete the *newtemp* directory. Copy-paste the command you used: __________

rmdir newtemp

Question 11

1 / 1 pts

Use a **single Linux command** to create a directory structure where a child folder is inside a parent folder, which is inside a grandparent folder. The structure would look like this in tree format:

```
grandparent
└── [drwxr-xr-x]  parent
    └── [drwxr-xr-x]  child

2 directories, 0 files
```

You can run the command **tree -p grandparent** to verify that it worked.

The **tree** command may not run in your AWS virtual machine. To install it, type the following command at the terminal

sudo yum install tree

Copy-paste the command you used to create the directory structure:

____

mkdir -p grandparent/parent/child

Before working on the following questions, you should work through the **Unix File Permissions** tutorial: https://www.tutorialspoint.com/unix/unix-file-permission.htm (https://www.tutorialspoint.com/unix/unix-file-permission.htm)

Question 12

1 / 1 pts

Make three copies of **renamedfile1** called permfile1, permfile2, permfile3.

Execute the **ls -l** command and examine the permissions for these files.

Who has **write permissions** for all three of these files?

- Other
- Group
- Owner (selected)

Question 13

1 / 1 pts

Use chmod in **symbolic mode** to **ADD** write permissions of permfile1 for all group members.

Copy-paste the command you used to do this: __________

chmod g+w permfile1

Question 14

1 / 1 pts

Use chmod in **symbolic mode** to **REMOVE** read permissions of permfile2 for all other users (not group members or the owner).

Copy-paste the command you used to do this: __________

chmod o-r permfile2

Question 15

1 / 1 pts

Use chmod in **symbolic mode** to **SET** read, write and execute permissions of permfile3 for the owner.

Copy-paste the command you used to do this: __________

chmod u=rwx permfile3

Question 16

1 / 1 pts

Use chmod **absolute mode** to **SET** read and execute permissions of permfile1 for the owner, read only for all group members and no permissions for all other users.

Copy-paste the command you used to do this: __________

chmod 540 permfile1

Question 17

1 / 1 pts

Use chmod **absolute mode** to **SET** read, write and execute permissions of permfile2 for the owner, read and write for all group members and read only for all other users.

Copy-paste the command you used to do this: __________

chmod 764 permfile2

Question 18

1 / 1 pts

Use chmod **absolute mode** to **SET** full (or all) permissions for permfile3 for the owner, all group members and all other users.

Copy-paste the command you used to do this: _________

chmod 777 permfile3

Question 19

2 / 2 pts

Read this short article on the **Principle of Least Privilege**: https://kb.iu.edu/d/amsv (https://kb.iu.edu/d/amsv) .

According to this principle, users and processes should have the

least authority needed to perform their duties.

Doing this reduces the "attack surface" and improves the security of a system.

Answer 1:

least

Answer 2:

authority

Question 20

1 / 1 pts

Consider the earlier question where we **SET** full permissions for permfile3 for the owner, all group members and all other users. Assuming that not all of the users require read, write, and execute permissions to do their job, are these permissions following the **Principle of Least Privilege**?

- Yes
- No

Quiz Score: **23.25** out of 24

Memory management

Context

- Process needs resources for execution
  - CPU cycles, I/O devices, etc.
  - *Memory* resources to store
    - Program code ("text")
    - Data
    - OS structures for process

We will hereafter refer to the set of all process memory resources as process data

Context

- For process to execute, its data must be in *main memory*
  - Recall that main memory is *volatile*
  - Process data stored in *non-volatile* memory, e.g., disk
  - Process data must be *loaded* into main memory from disk

Simple scenario

- Allow only *one process* to be active at a time
  - *Load* process data from disk to main memory
  - Once process is done, *store* its contents to disk

- In reality, a process
  - May wait for I/O → poor *CPU utilization*
  - May not use all memory → poor *memory utilization*

More realistic scenario…

- Just like *CPU* multiplexed among multiple processes…
  - …*memory* must be multiplexed among multiple processes
- What we need
  - *Multiple* processes should reside in memory *at a time*
  - Processes should not *collide* in memory
  - Process should not access other process' memory (*protection*)
  - Processes should be able to *share* memory if desired
- I.e., we need *memory protection* & *controlled overlap*

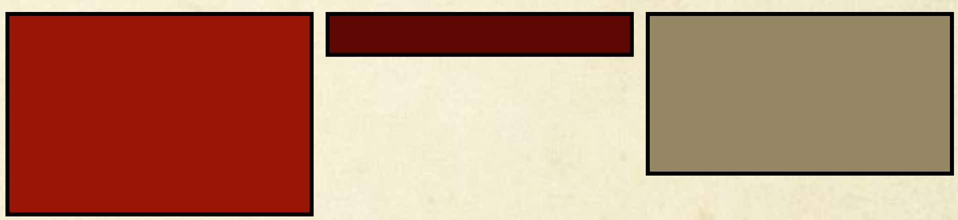
Memory partitioning

- Basic idea
  - Divide memory into multiple *partitions*
  - *Allocate* processes to partitions
  - Ensure *protection* across partitions
- Several questions arise
  - How should memory be divided into partitions?
  - Which process should be allocated to which partition?
  - Should partitions be allowed to change dynamically?
- Let's consider multiple options…

Static, equal-sized partitions

- *Statically* divide main memory into *equal-sized* partitions
- Map *each* process to *one* partition
    - For now, assume mapping does not change

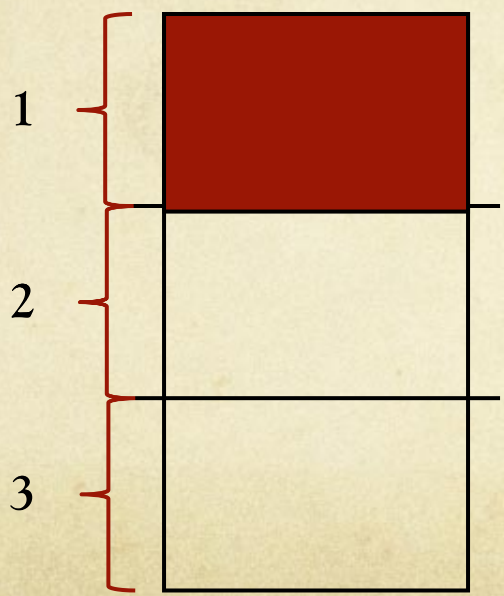
Partitioned memory



Process memory footprints

Static, equal-sized partitions

- *Statically* divide main memory into *equal-sized* partitions
- Map *each* process to *one* partition
  - For now, assume mapping does not change

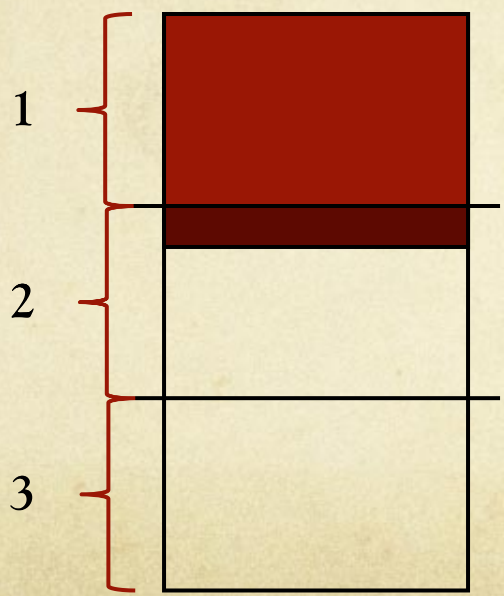


Partitioned memory

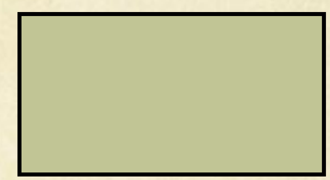
Process memory footprints

Static, equal-sized partitions

- *Statically* divide main memory into *equal-sized* partitions
- Map *each* process to *one* partition
  - For now, assume mapping does not change



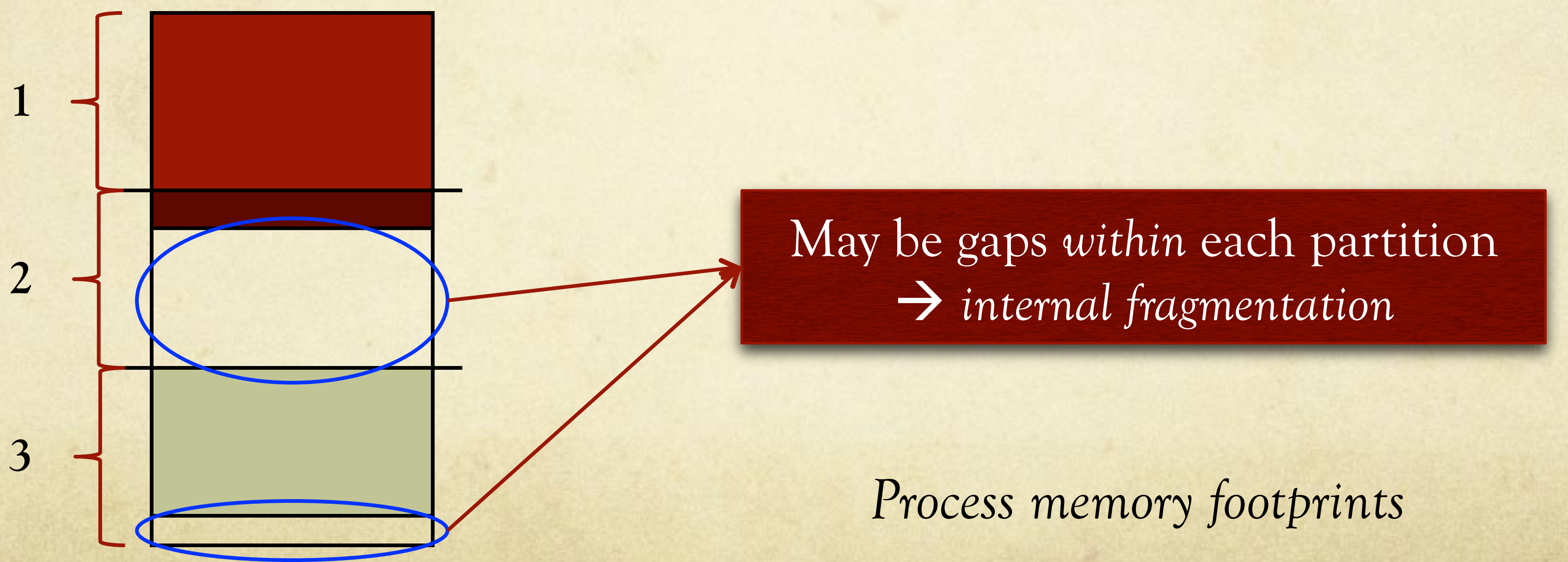
Partitioned memory



Process memory footprints

Static, equal-sized partitions

- *Statically* divide main memory into *equal-sized* partitions
- Map *each* process to *one* partition
  - For now, assume mapping does not change

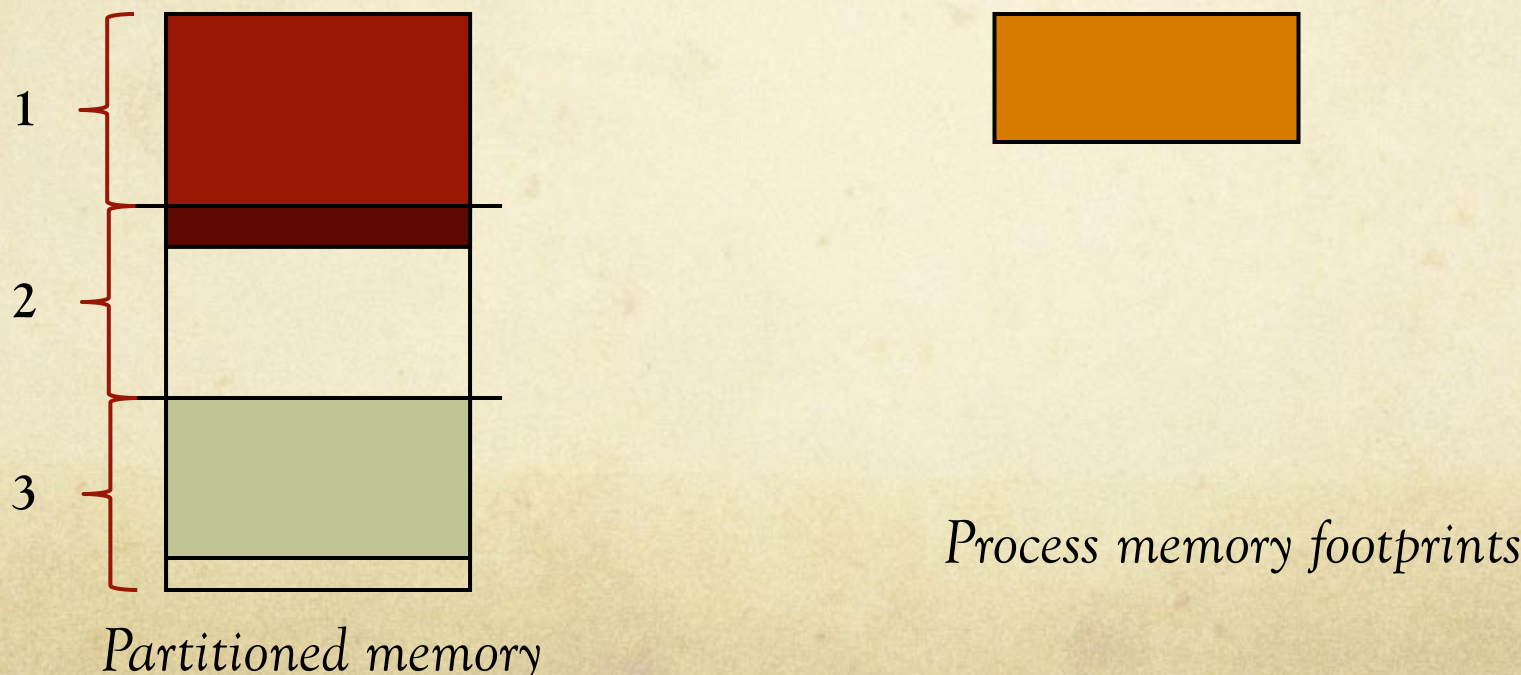


Partitioned memory

Process memory footprints

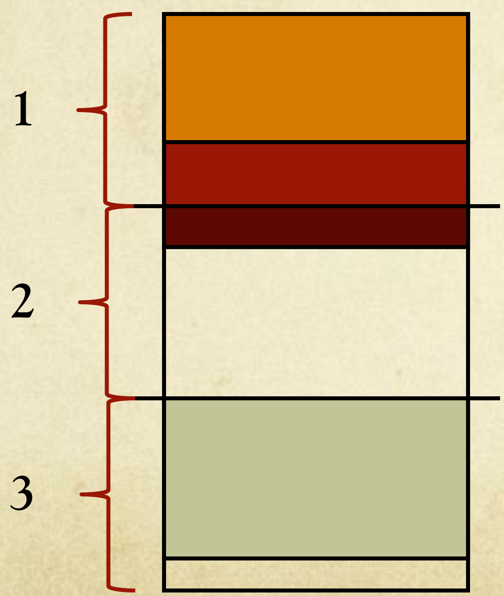
Static, equal-sized partitions

- What if there are more processes than partitions?
  - Map multiple processes to *same* partition
  - *Store* existing process data to disk & *de-allocate* its partition
  - *Allocate* new process to partition & *load* its data from disk

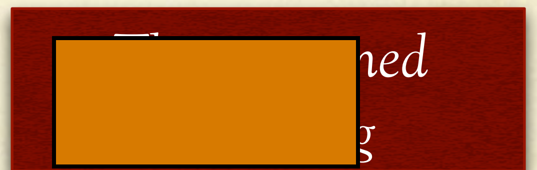


Static, equal-sized partitions

- What if there are more processes than partitions?
  - Map multiple processes to *same* partition
  - *Store* existing process data to disk & *de-allocate* its partition
  - *Allocate* new process to partition & *load* its data from disk



Partitioned memory



Process memory footprints

Protection

- Ensure process only accesses locations in its partition
- Store start address or *base address* (*BA*) for each process
- Check every address issued by process
  - Must lie between *BA* & (*BA + partition size*)

Example

- Memory address range: 0 – 4999; Partitions of size 1000
- Partition to process mapping
  - Partition 1: P1, P6
  - Partition 2: P2, P7
  - Partition 3: P3
  - Partition 4: P4
  - Partition 5: P5
- Is P1 allowed to access address 1004?
  - *No*!
- Is P4 allowed to access address 3000?
  - *Yes*

Discussion

- *Pros*
  - Very simple
- *Cons*
  - Results in *internal* fragmentation
    - Results in *under-utilization* of memory
  - Cannot fit process *larger* than partition size
  - Takes a *one-size-fits-all* kind of approach

Static, unequal-sized partitions

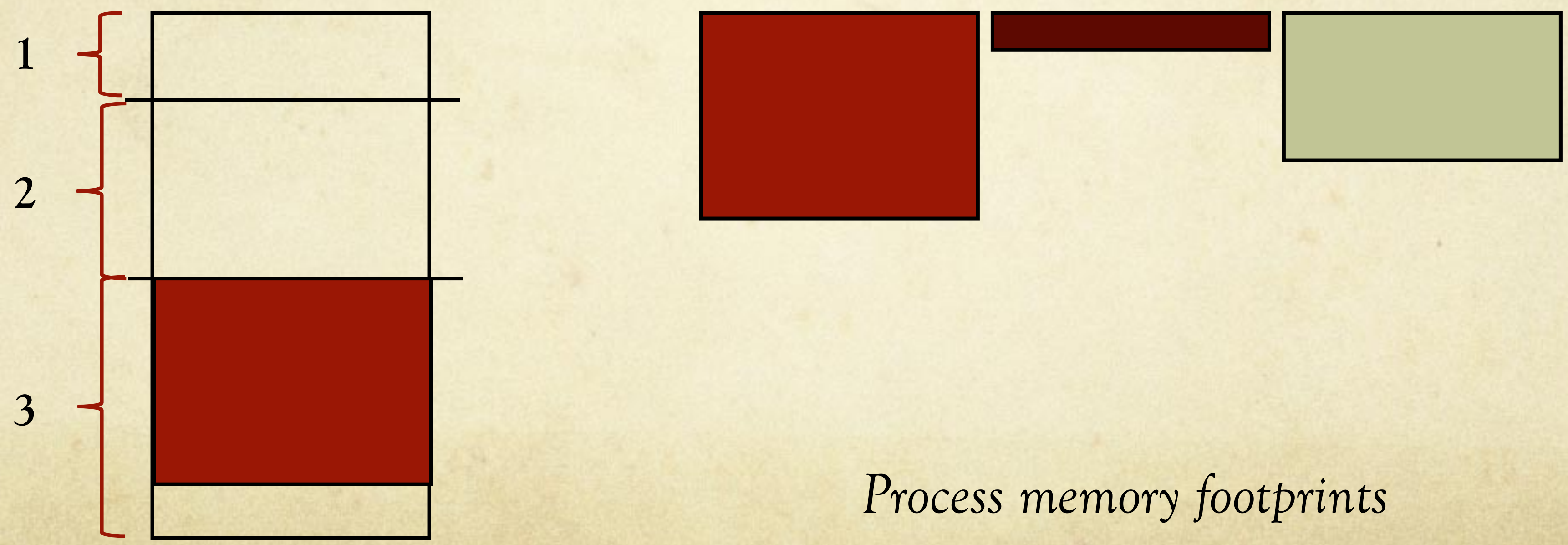
- Statically create partitions of *different* sizes
- Map each process to a partition that's *large enough* for it
  - For now, assume mapping does not change

Partitioned memory

Process memory footprints

Static, unequal-sized partitions

- Statically create partitions of *different* sizes
- Map each process to a partition that's *large enough* for it
    - For now, assume mapping does not change

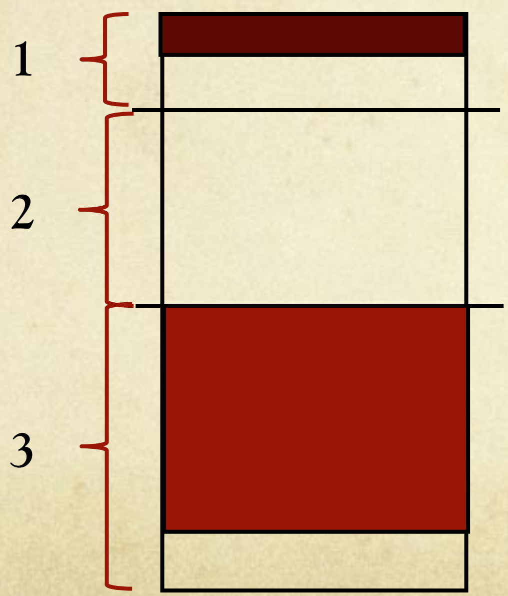


Process memory footprints

Partitioned memory

Static, unequal-sized partitions

- Statically create partitions of *different* sizes
- Map each process to a partition that's *large enough* for it
  - For now, assume mapping does not change

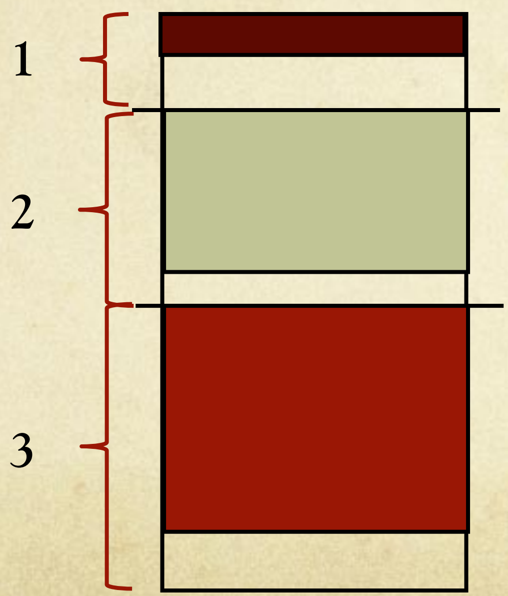


Partitioned memory

Process memory footprints

Static, unequal-sized partitions

- Statically create partitions of *different* sizes
- Map each process to a partition that's *large enough* for it
  - For now, assume mapping does not change

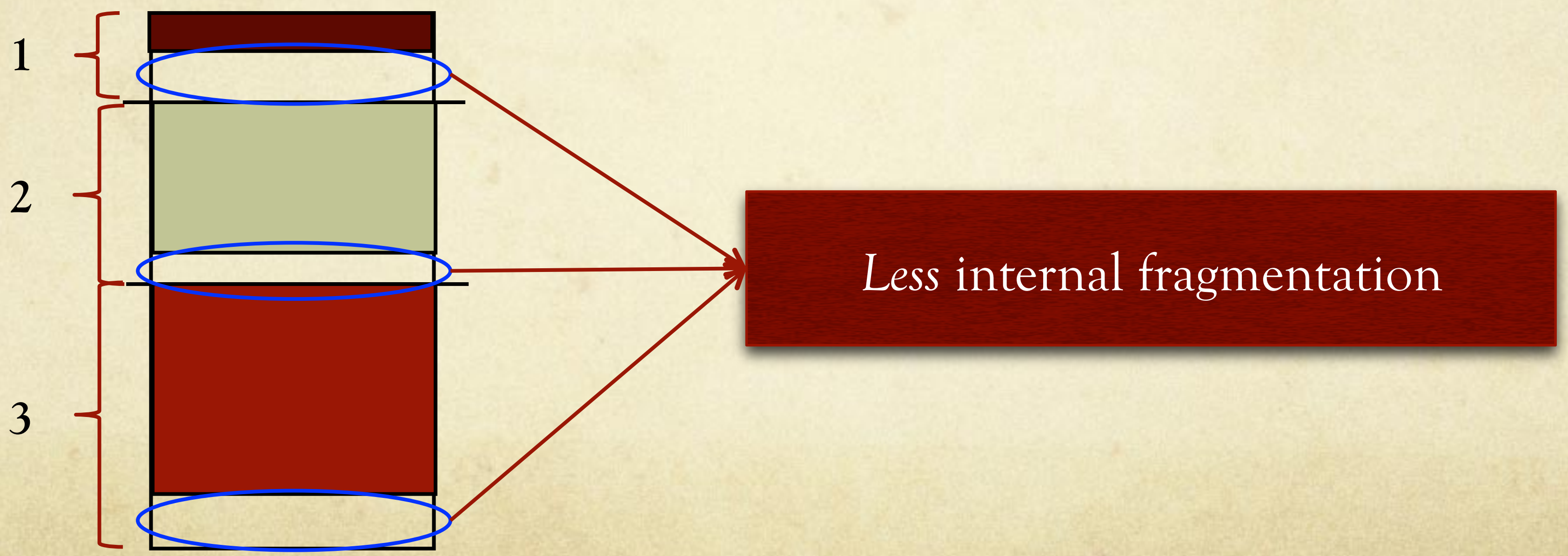


Partitioned memory

Process memory footprints

Static, unequal-sized partitions

- Statically create partitions of *different* sizes
- Map each process to a partition that's *large enough* for it
  - For now, assume mapping does not change



Partitioned memory

Static, unequal-sized partitions

- As before, this can be combined with *swapping*

Partitioned memory

Static, unequal-sized partitions

- As before, this can be combined with *swapping*

Partitioned memory

Static, unequal-sized partitions

- As before, this can be combined with *swapping*

Partitioned memory

Protection

- Store *base address* (*BA*) + *limit* for each process' partition
- Check every address issued by process
    - Must lie between *BA* & (*BA + limit*)

Discussion

- *Pros*
  - Supports processes of *different* sizes better
  - Less *internal* fragmentation
- *Cons*
  - Slightly more *complex* than equal-sized partitions
    - Needs policy for mapping processes to partitions (*partition placement*)
    - Needs slightly enhanced protection mechanism
  - Needs partition to be created *statically*
    - Chosen sizes *may not suit* processes that may need to be supported
  - Still can't support processes *larger* than *largest* partition

Dynamic, variable-sized partitions

- Basic idea
  - Start with *un-partitioned* memory (a.k.a. "*free space*")
  - Create partitions *dynamically* based on process *data size*
    - *Number* of partitions changes dynamically
    - *Lengths* of partitions may be different

Dynamic, variable-sized partitions

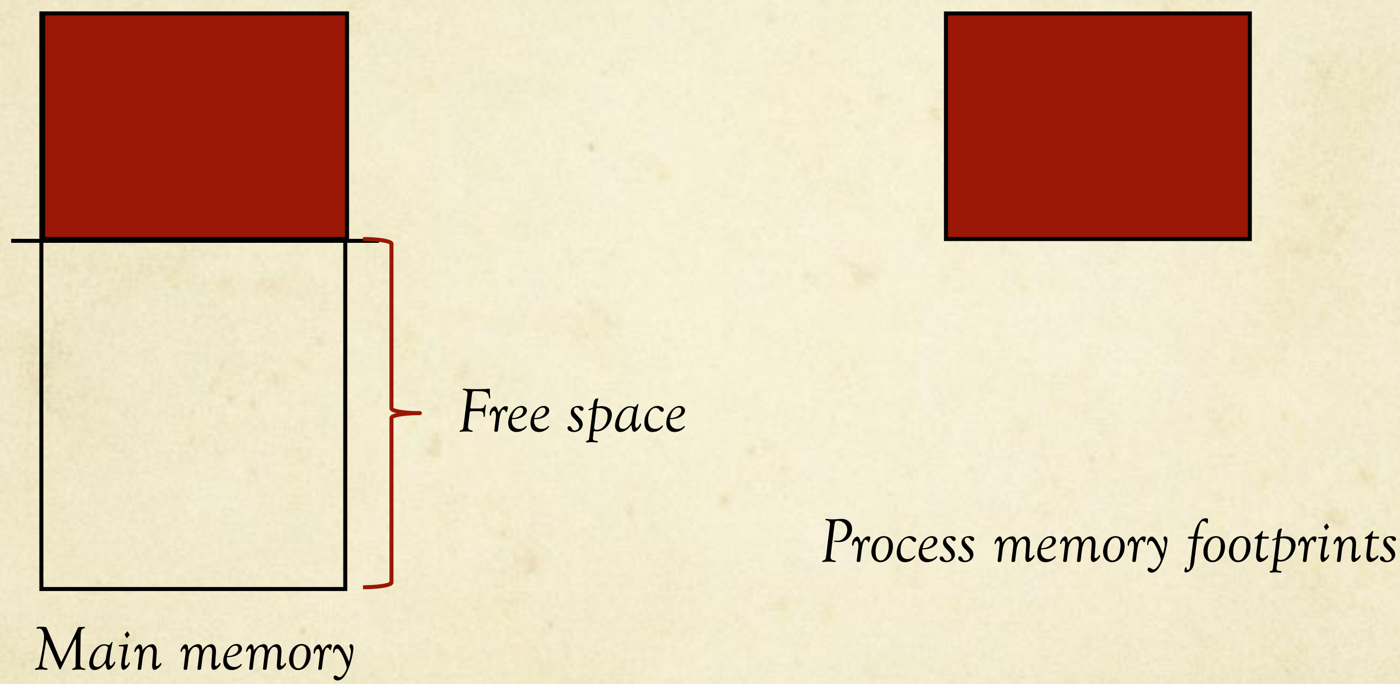
- Place initial partitions *contiguously* in main memory

Process memory footprints

Main memory

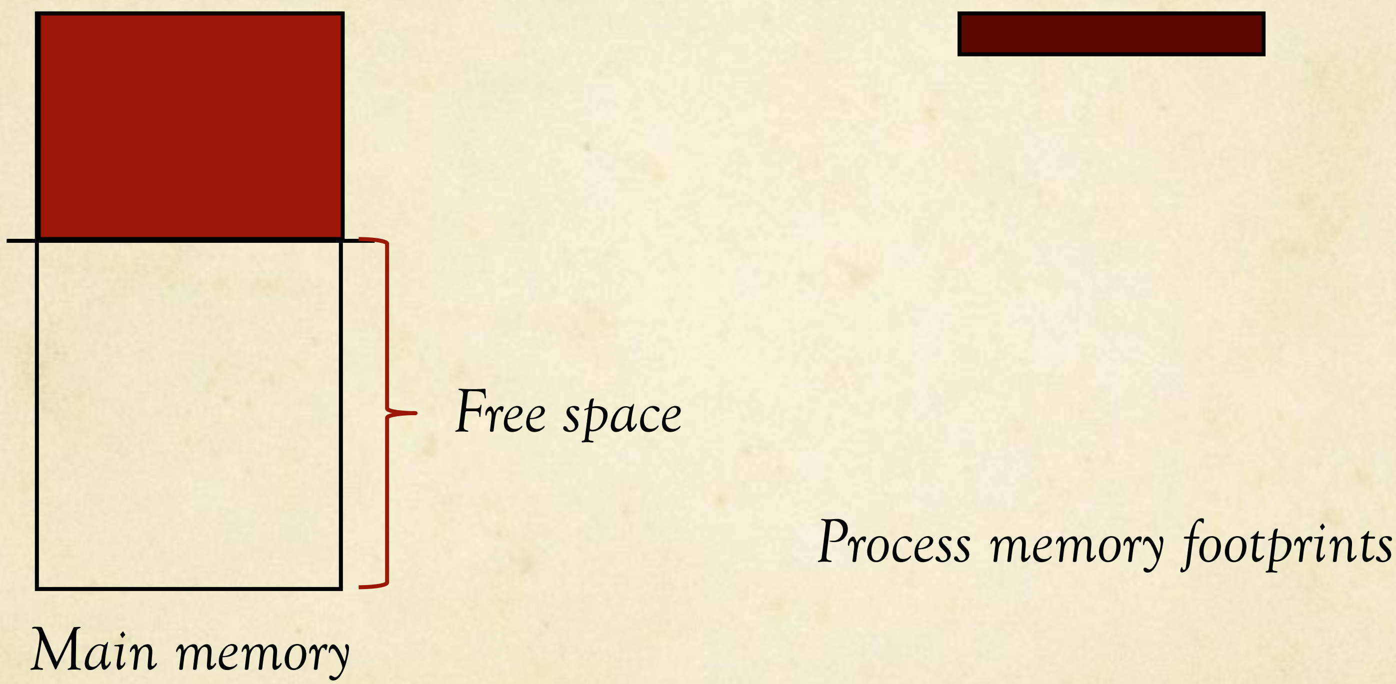
Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory



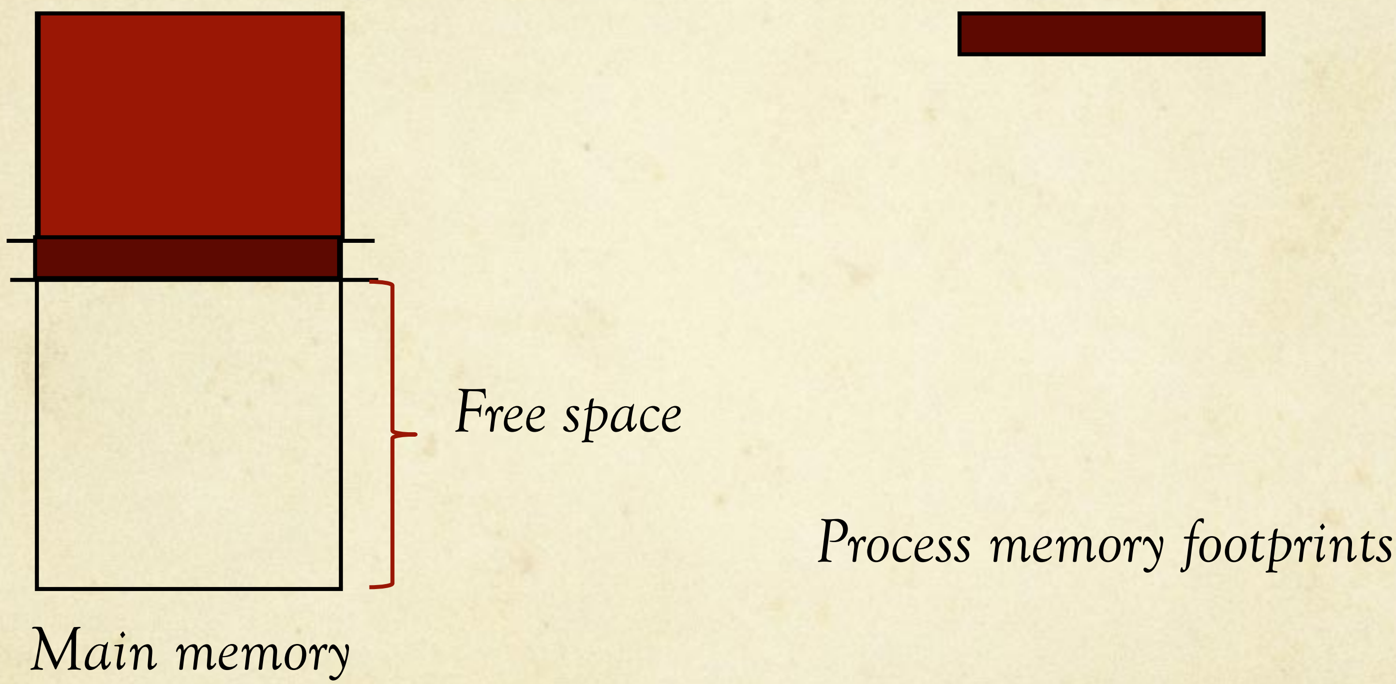
Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory



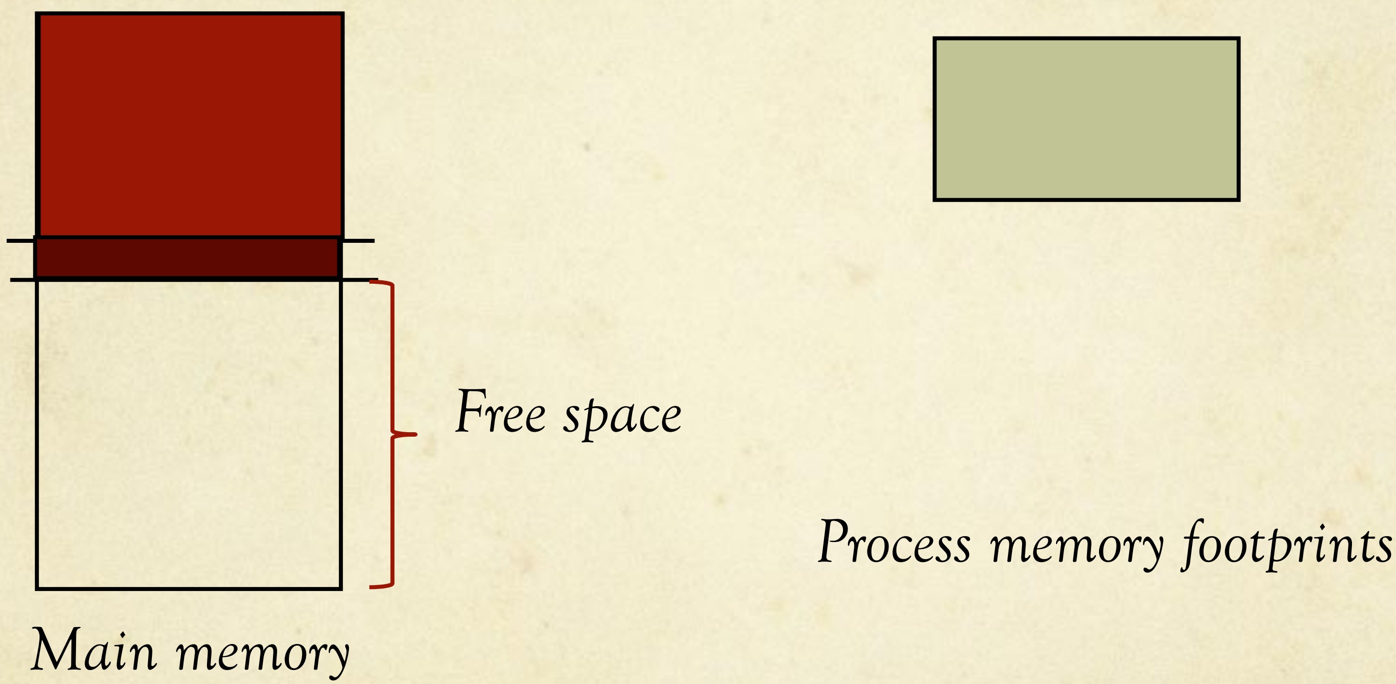
Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory



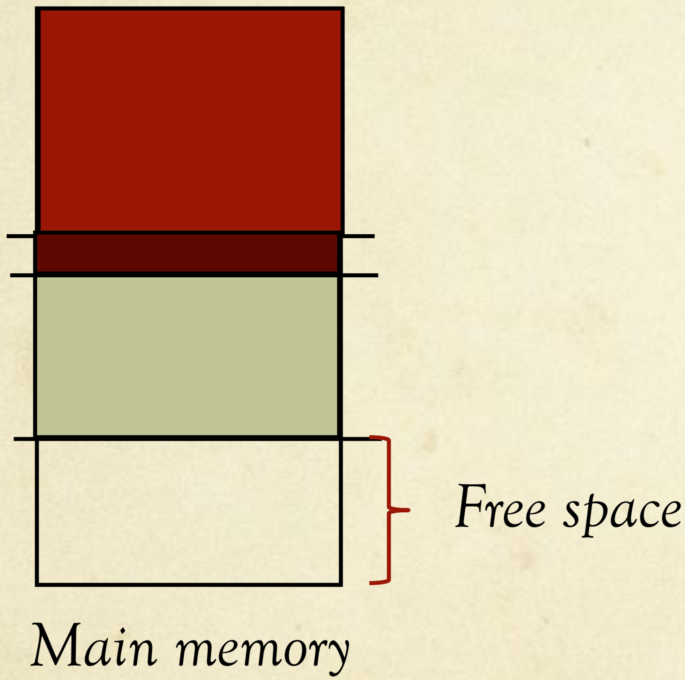
Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory



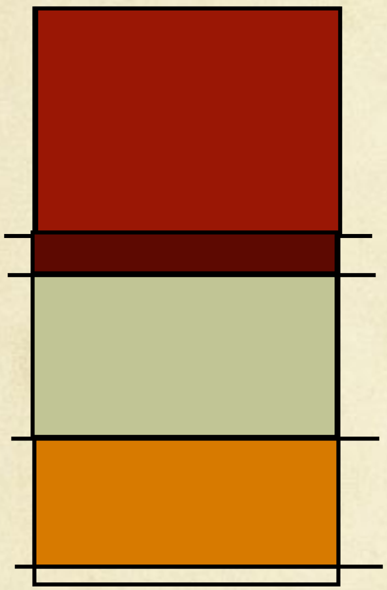
Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory



Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory

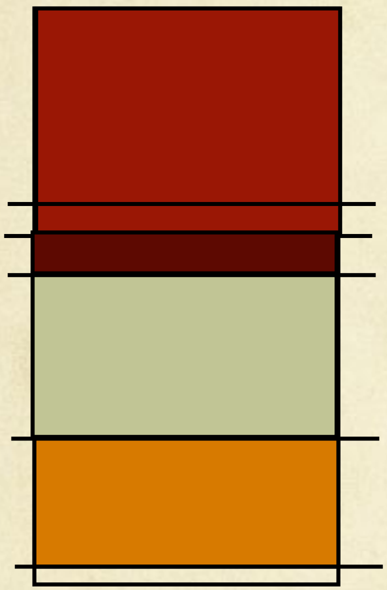


Main memory

Process memory footprints

Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory

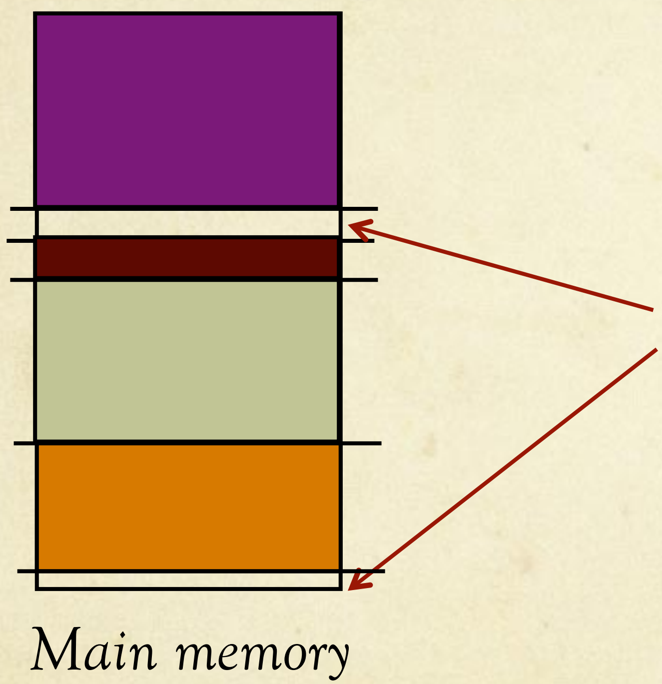


Main memory

Process memory footprints

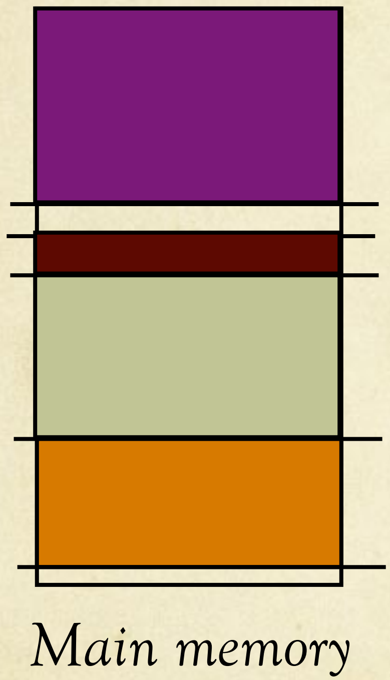
Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory



Dynamic, variable-sized partitions

- Place initial partitions *contiguously* in main memory



External fragmentation

- As processes get swapped in & out of main memory
  - Memory ends up with "*free spaces*"/"*holes*" b/w partitions

Dynamic, variable-sized partitions

- Consequences
  - Need *mechanism* to *keep track* of free spaces!
  - Need *policy* for *partition placement*

- Possible *mechanism* to keep track of free spaces
  - Maintain *linked list* of free spaces
  - Each list element stores *base address* & *size* of free space

BA: 100, Size: 600 → BA: 850, Size: 1300 → BA: 3000, Size: 800 → BA: 4200, Size: 600

Polices for partition placement

- *First fit* (*FF*)
  - Scan list, choose *first* free space *large enough* for partition
  - *Pros*: simple; fast
  - *Cons*: partitions may *crowd* initial regions of main memory
- *Next fit* (*NF*)
  - Similar to first fit; scanning starts where previous scan ended
  - *Pros*: more distributed allocation

Polices for partition placement

- *Best fit* (*BF*)
  - Scan entire list; choose *smallest* free space *large enough*
  - *Cons*: slow; leaves many small unusable free spaces

- *Worst fit* (*WF*)
  - Scan entire list; choose *largest* free space that fits process
  - *Pros*: leaves larger free spaces
  - *Cons*: slow

First fit

Memory address range: 0 – 4999

Process memory requirements

P1: 500 P3: 750

P2: 800 P4: 1200

P5: 900

BA: 100, Size: 600 → BA: 850, Size: 1300 → BA: 3000, Size: 800 → BA: 4200, Size: 600

P2 Address 850

BA: 100, Size: 600 → BA: 1650, Size: 500 → BA: 3000, Size: 800 → BA: 4200, Size: 600

P1 Address 100

BA: 600, Size: 100 → BA: 1650, Size: 500 → BA: 3000, Size: 800 → BA: 4200, Size: 600

Best fit

Memory address range: 0 – 4999

Process memory requirements

P1: 500 P3: 750

P2: 800 P4: 1200

P5: 900

BA: 100, Size: 600 → BA: 850, Size: 1300 → BA: 3000, Size: 800 → BA: 4200, Size: 600

P2 Address 3000

BA: 100, Size: 600 → BA: 850, Size: 1300 → BA: 4200, Size: 600

P1 Address 100

BA: 600, Size: 100 → BA: 850, Size: 1300 → BA: 4200, Size: 600

Discussion

- *Pros*
  - More *flexible* than static partitioning
- *Cons*
  - Management more *complex* than with static partitioning
  - Has *external fragmentation* (extent varies with policy)

Now consider this…

- How should a program specify memory addresses?
  - Absolute addresses?
  - Definitely not for *dynamic* partitioning!
  - Probably not even for *static* partitioning → loses portability

In practice...

- Program uses offsets *relative* to start address of 0
  - I.e., program uses *logical* address
  - Range of logical addresses for process: **logical address space**
- System *translates* relative offset into *absolute* address
  - I.e., system generates *physical* address
  - Range of absolute addresses of process: **physical address space**
- Translation mechanism
  - Maintain *base address* & *limit* for process partition
  - Add base address to logical address issued by program
  - Check whether result is within limit

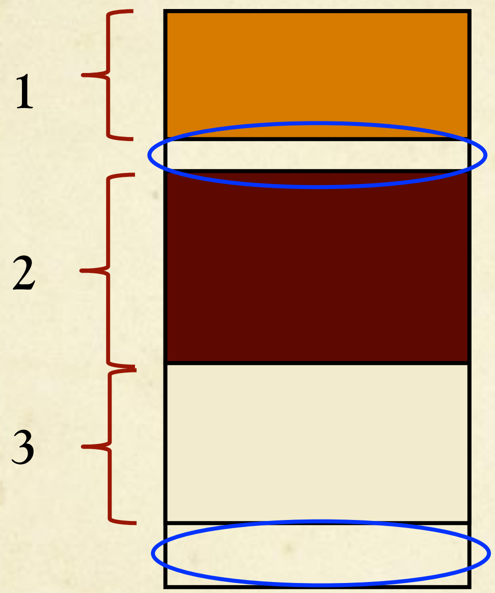
Consequence of relative addressing

- Process *need not* be mapped to same physical partition all the time…
- …partition *relocation* is possible
- Very useful, especially when using *dynamic* partitioning

External fragmentation

- Reduces memory *utilization*
  - Even if enough free memory *exists* to allocate new process…
  - …not useful since memory is not *contiguous*
- Possible approaches
  - Reduce external fragmentation using *compaction*
  - Find way to allow *non-contiguous* partitions
    - I.e., split process footprint into multiple parts

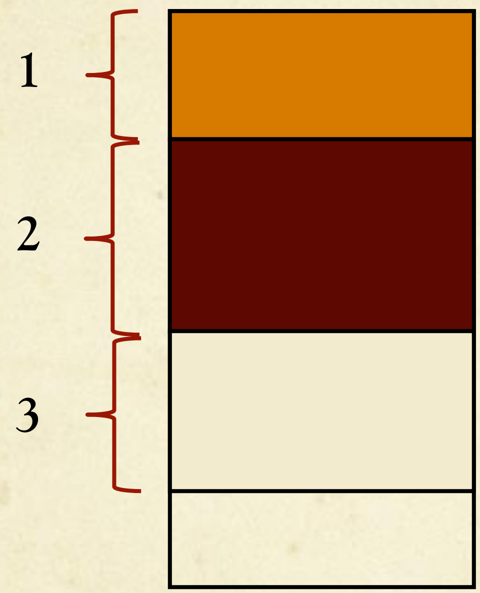
Compaction



Partitioned memory

- Move partitions to abut each other

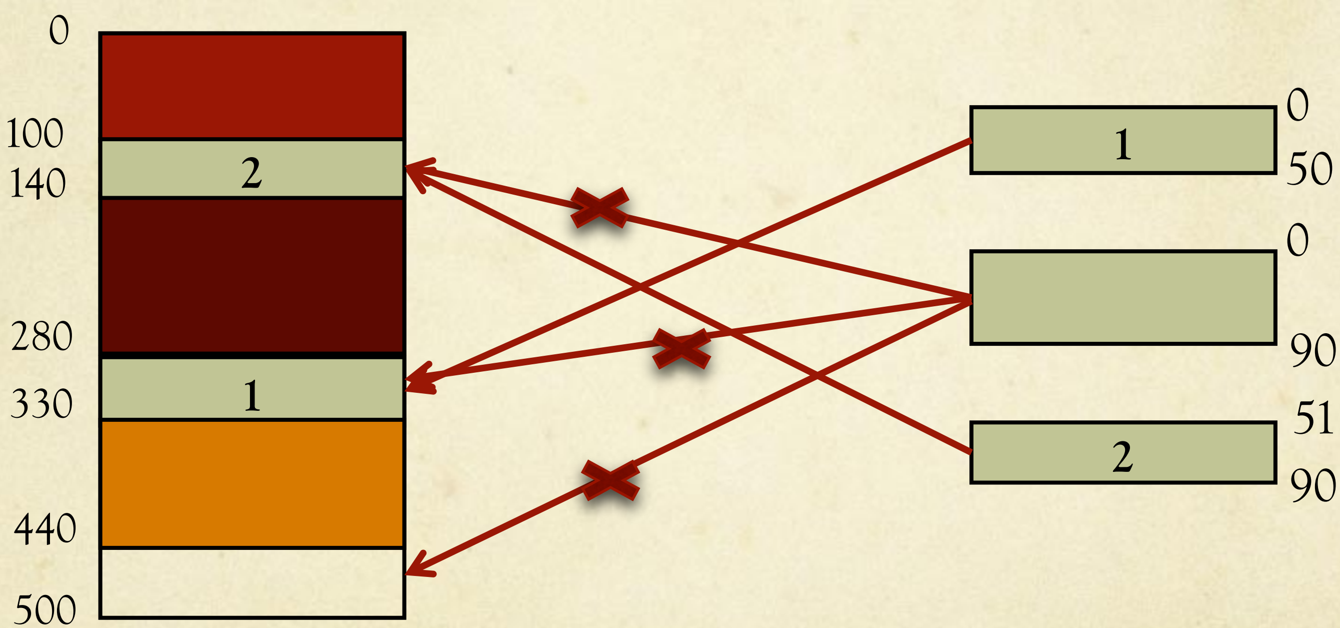
Compaction



Partitioned memory

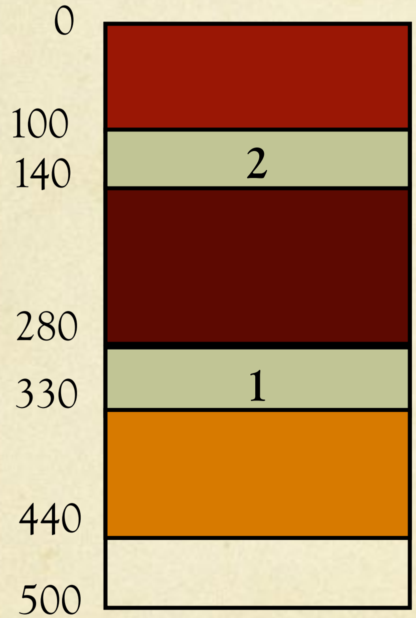
- Move partitions to abut each other
- Needs partition *relocation*

Allowing non-contiguous allocation...



Partitioned memory

Allowing non-contiguous allocation…



Partitioned memory

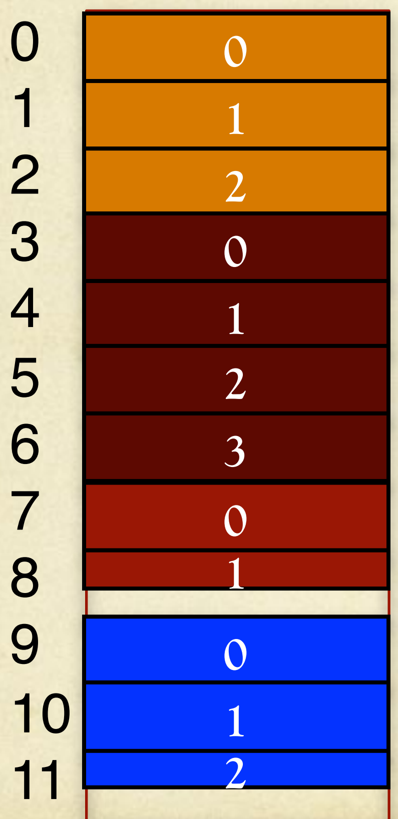
- What is the *base address*?
- What is the *limit*?

What we need…

- …is a more *sophisticated* memory management mechanism!
  - Maintain information for *multiple* parts of process
- One such mechanism is **paging**
- Divide physical memory into equal-sized blocks: *page frames*
- Divide logical memory into same sized blocks: *pages*
- Process pages *mapped* to physical page frames
- Page frame sizes are powers of *2*
  - E.g., 4096 bytes, 8192 bytes

Paging

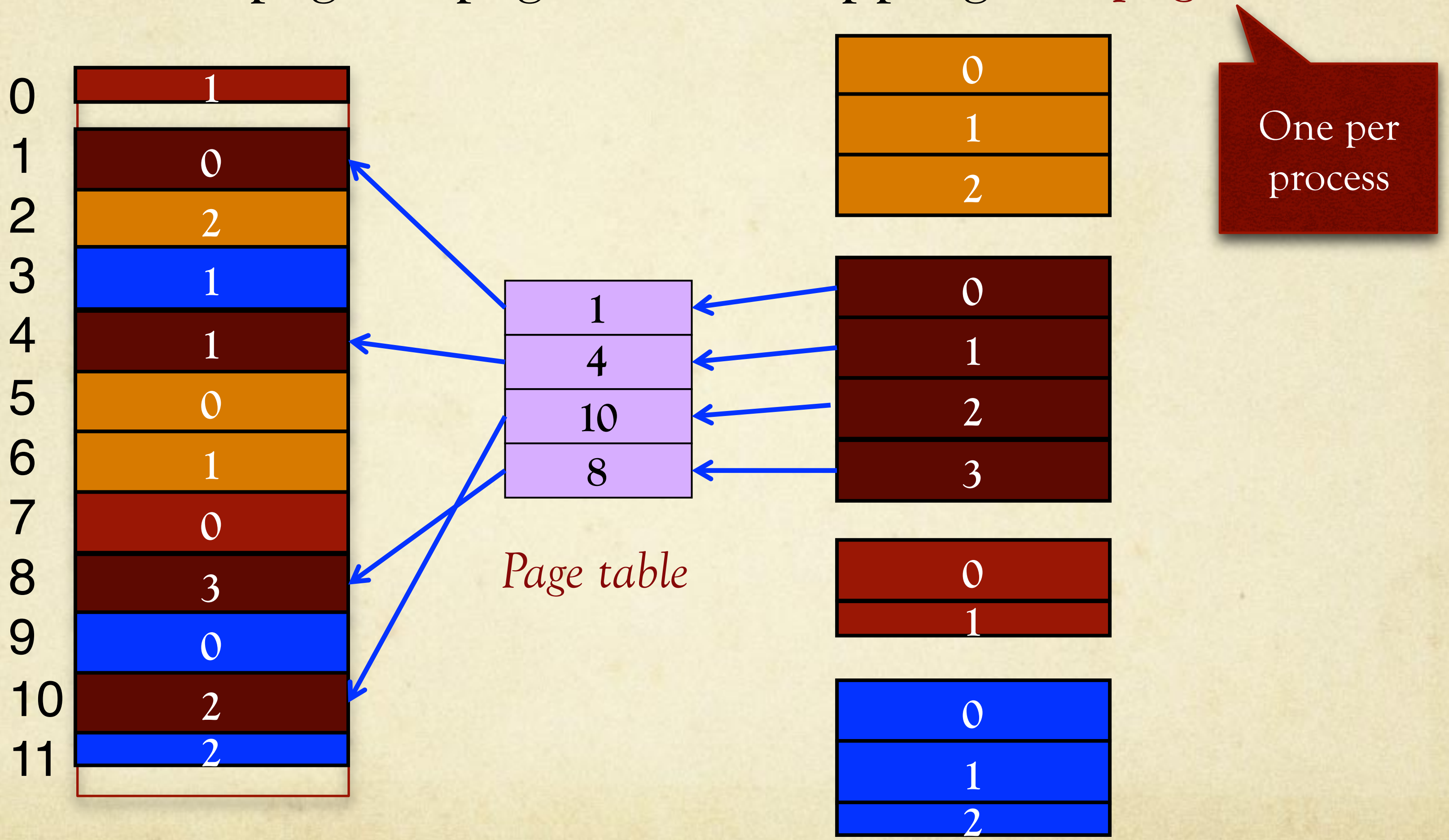
Paging



- Mapping of process pages could end up being contiguous
- But it need not be...
- ...depends on *state* of memory when process is allocated...
- ...and on *policy* used for allocation

Paging

- OS maintains page to page frame mappings → *page tables*



Paging

- OS keeps track of *free* page frames → *linked list*
- To allocate process that needs *n* pages
  - Find *n* free page frames; *allocate* process pages to frames
- *Internal* fragmentation possible due to fixed-sized frames
  - Manageable by carefully choosing *size* of page/page frame

Part of OS that takes care of mapping, address translation etc. → memory management unit

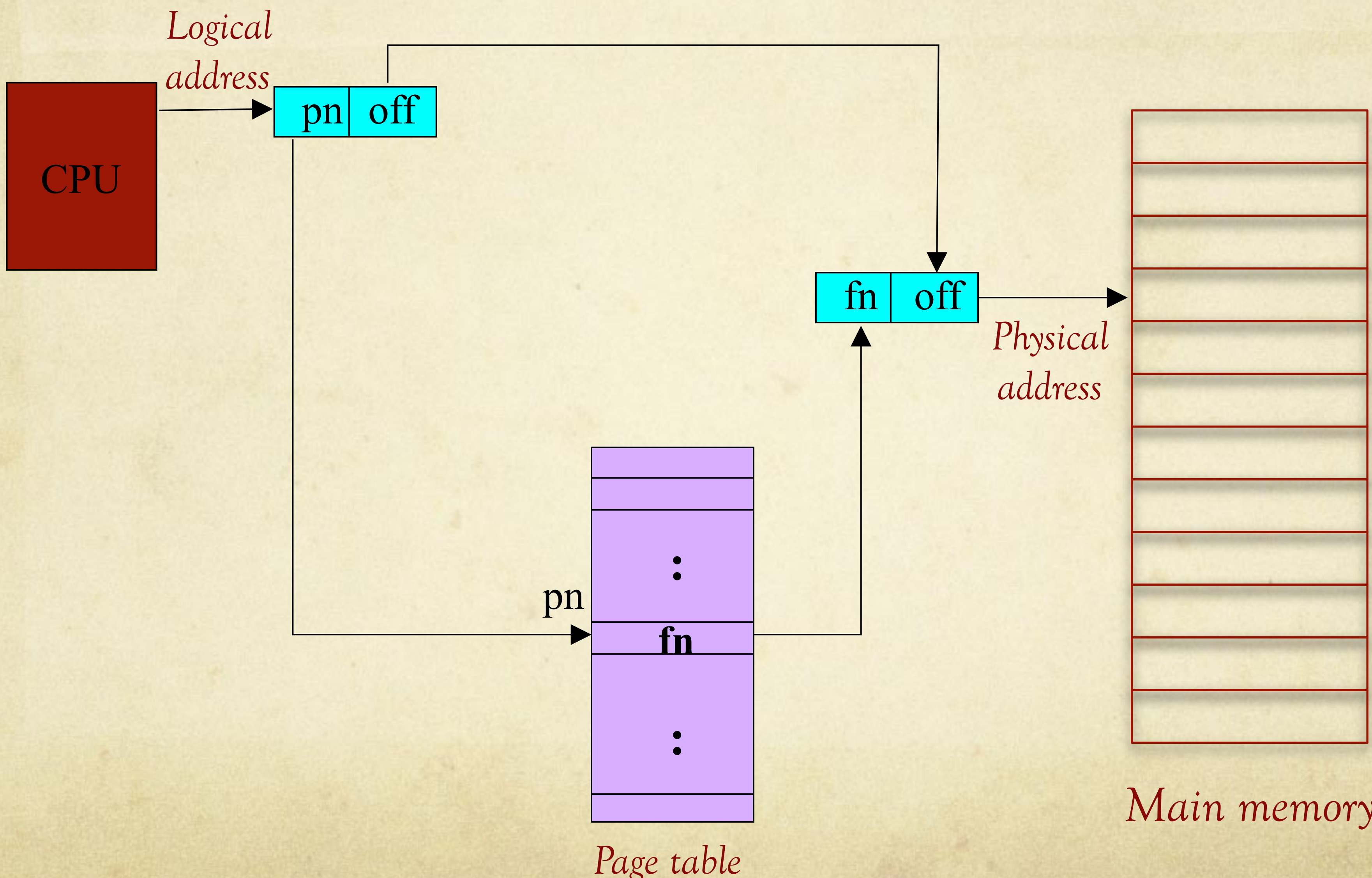
Address translation in paged systems

- *Logical* address includes two parts
  - Page number (pn)
  - Offset within page (off)

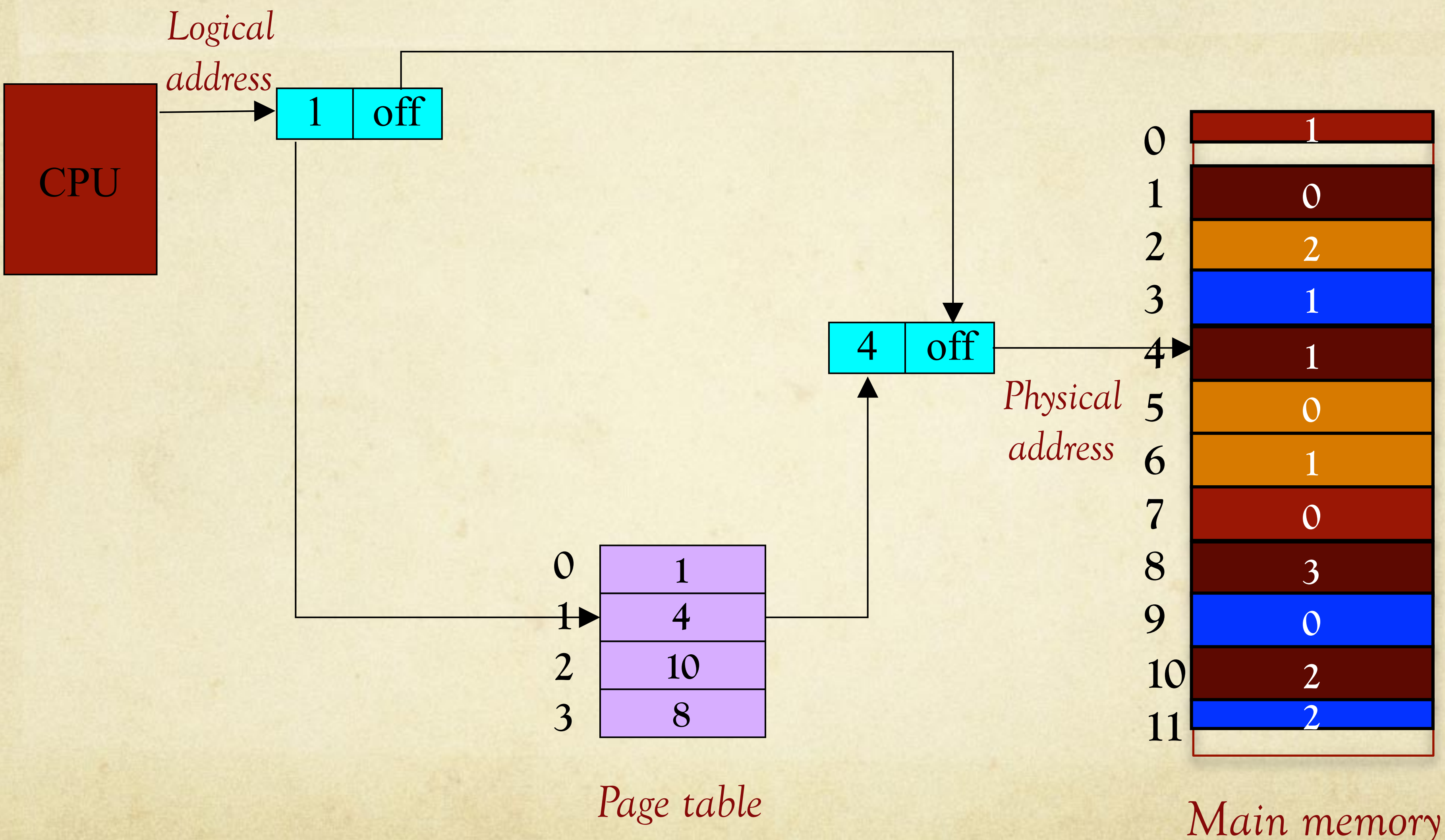
Page table

- *Physical* address includes corresponding parts
  - Page frame number (fn)
  - Offset within frame (off)

Address translation in paged systems



Address translation in paged systems



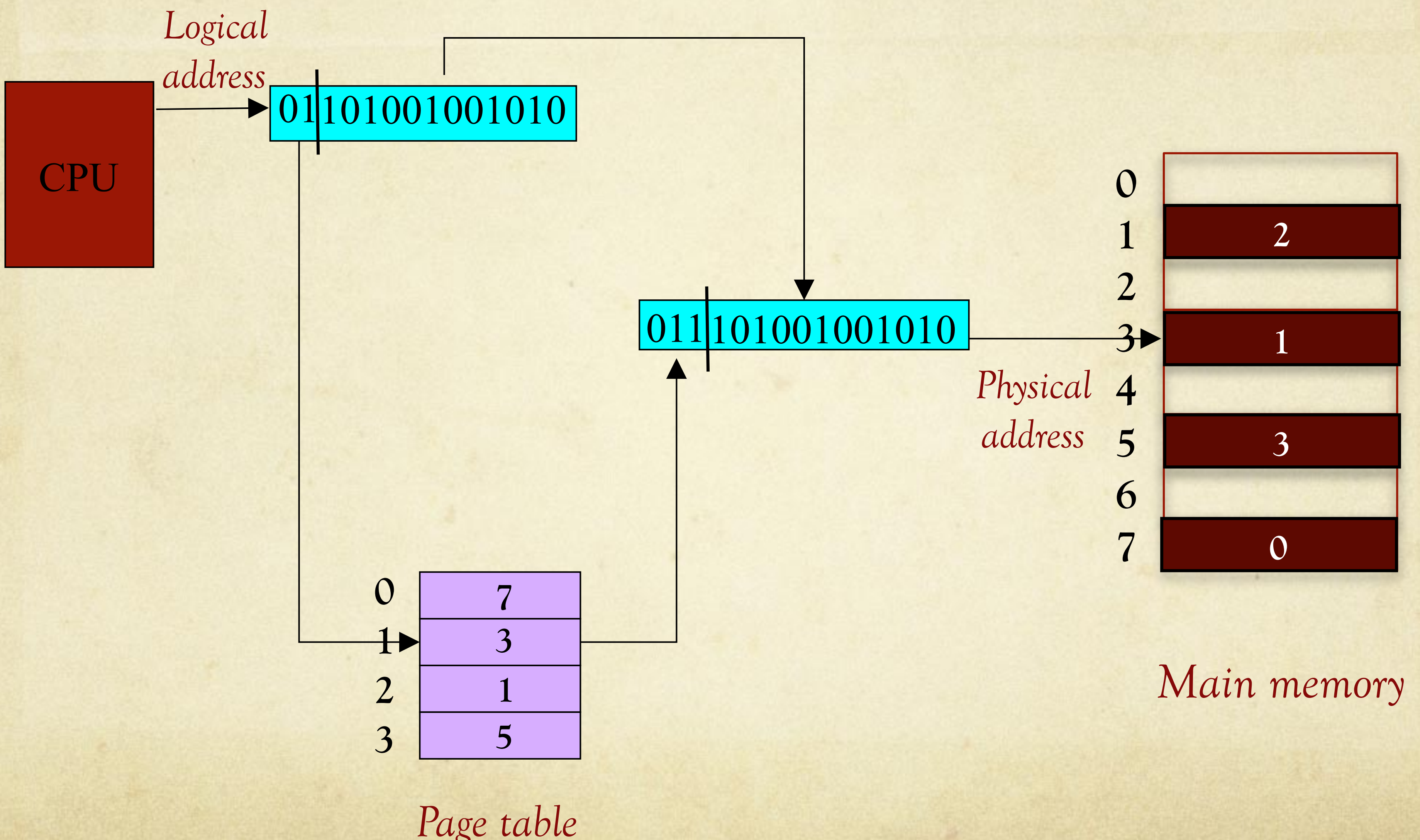
In a computer ...

- Addresses represented in *binary*
  - Main memory & page frame sizes are powers of 2
  - *Hexadecimal* → more compact
- *Knowledge of number systems & conversions needed*
- Binary → bit values 0 and 1
  - 2 bits: 4 (2^2) combinations (00, 01, 10, 11)
  - 3 bits: 8 (2^3) combinations (000, 001, 010, 011, 100, 101, 110, 111)
  - n bits: 2^n combinations

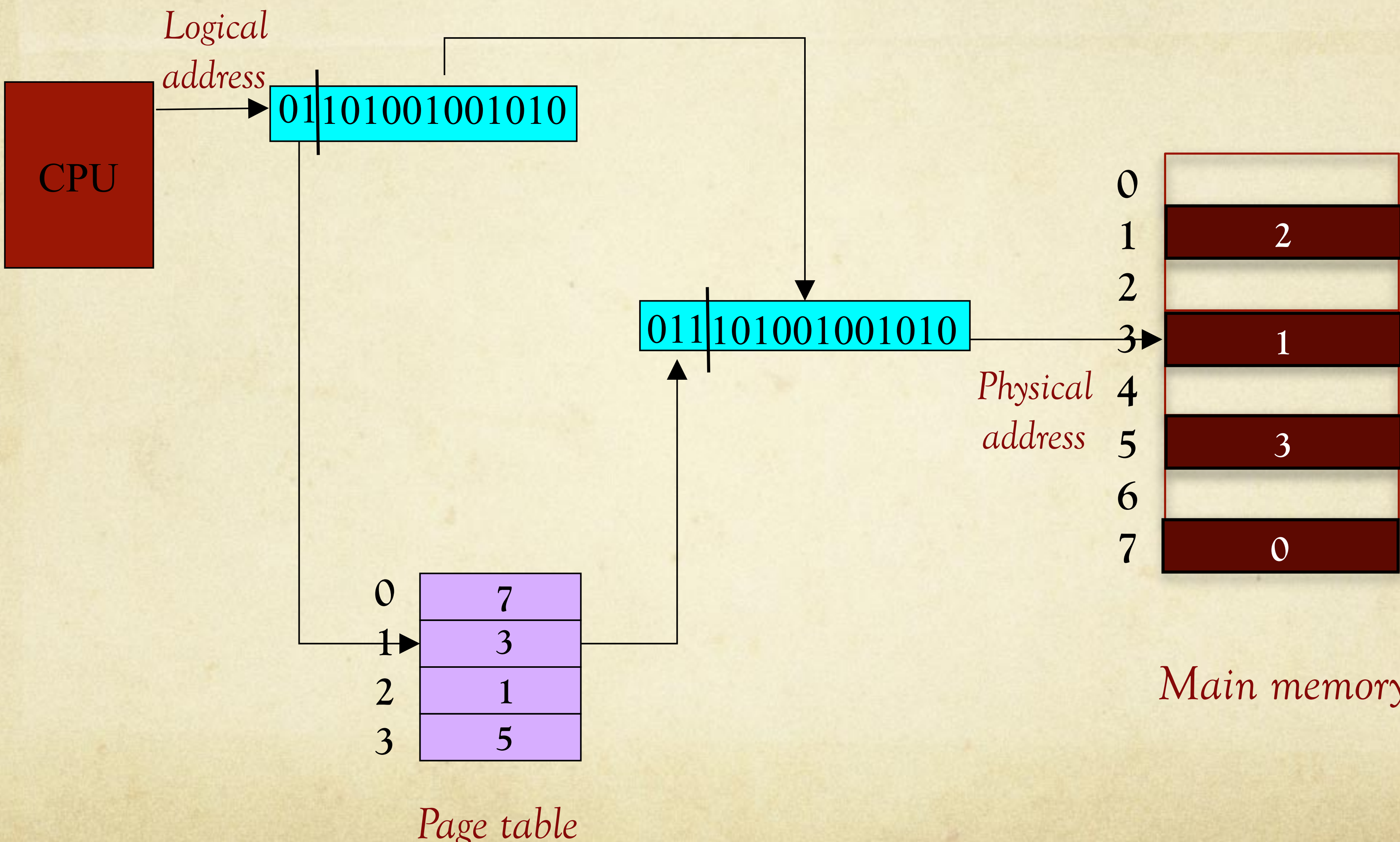
Memory System

- 32KB (2^{15}) main memory
  - 15 bit physical address
- 16KB (2^{14}) logical address space
  - 14 bit logical address
- 4KB (2^{12}) page frame /page
  - 12 bit offset [*LSB*]
  - $2^{15} / 2^{12} = 2^3$ page frames
    - 3 bit page frame number
  - $2^{14} / 2^{12} = 2^2$ pages
    - 2 bit logical page number

Address translation in paged systems

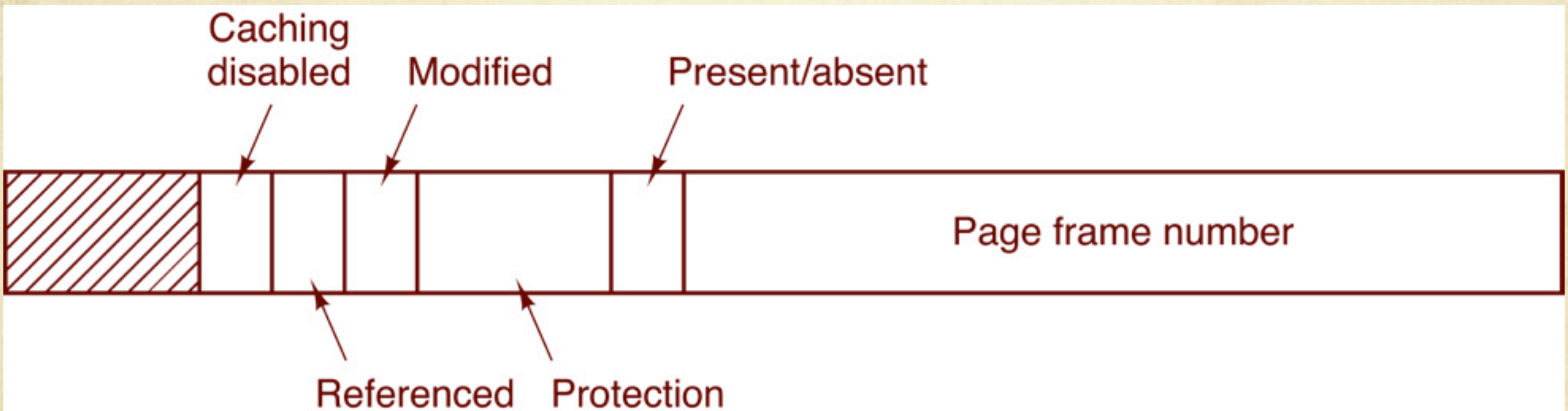


Address translation in paged systems



Page table entry structure

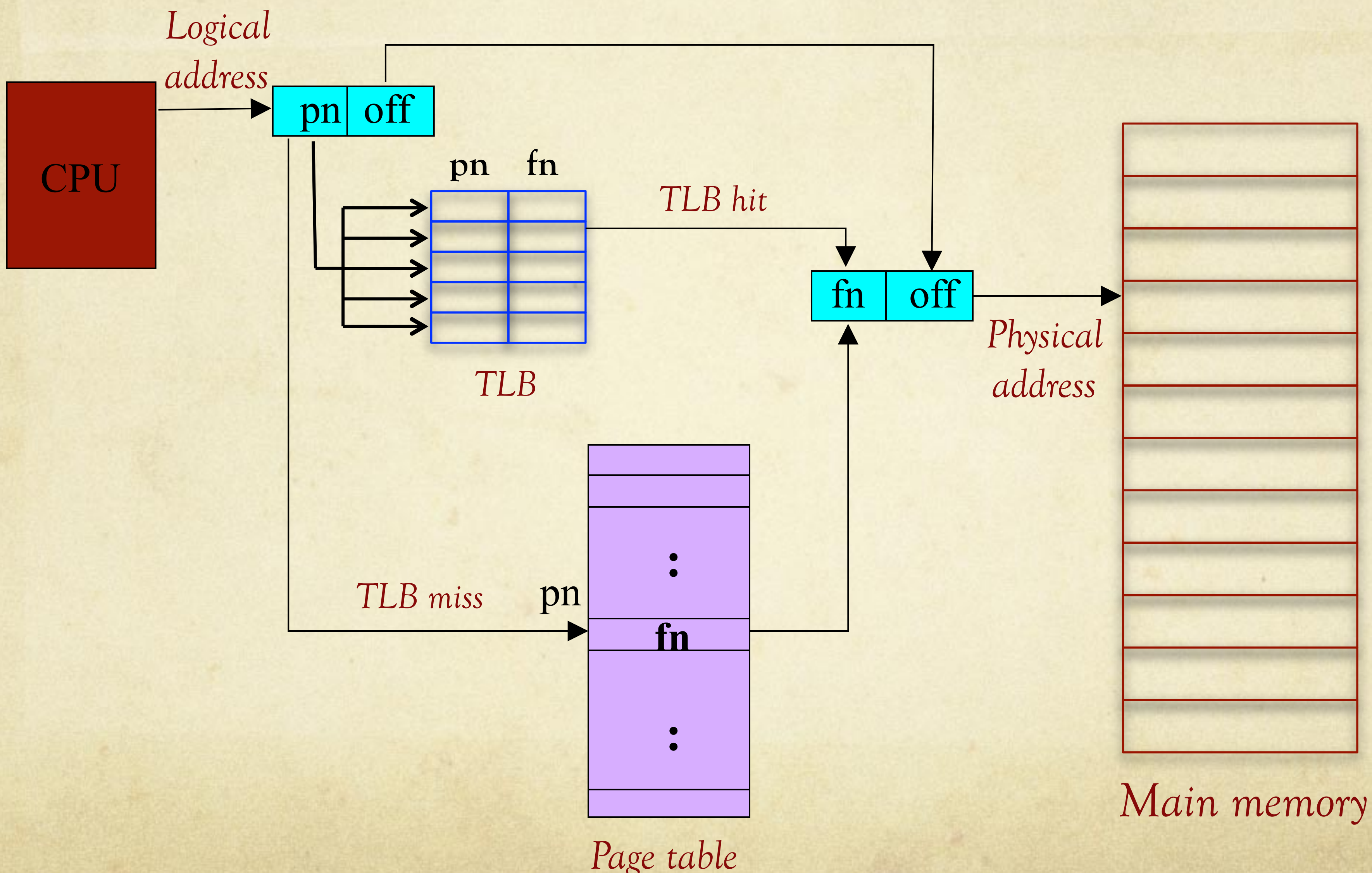
- Structure is dependent on system
- Let's look at one possible structure



More about paging

- Page table itself stored in *main memory*
  - *Expensive* to access it for each memory request
- Solution → *cache* subset of page table entries
- Translation look-aside buffer (*TLB*)
  - Small set of *associative* registers
  - Provides *fast* access

Address translation with TLB



We still have one restrictive assumption....

- *All* process pages must be in *main memory* to execute process
  - Limits *num processes* in main memory & process *data size*
- Observation
  - Process does not use *all* its data *all* the time
- Better option
  - Keep only *subset* of process pages in main memory
  - Bring others as needed
  - This is called **demand paging**

Demand paging

- When process requests for data
  - Check if page with data *already* in main memory
  - If yes, proceed as usual
  - If not, **page fault** is set to occur
- When page fault occurs
  - *Load* missing page into main memory
    - May add *TLB* entry for new page
  - *Restart* instruction that caused page fault

A consequence of demand paging…

- …process data *not limited* to physical main memory size!
  - I.e., size of process virtually unlimited
- This is referred to as **virtual memory**
  - Did not support this in our earlier *examples*!
- What can we do?
  - Allow more logical *pages* in process than physical *page frames*
  - Logical page # can be longer physical page frame #
  - Logical (*virtual*) address can be *longer* than physical address

Memory System

- 32KB (2^{15}) main memory
  - 15 bit physical address
- 64KB (2^{16}) logical address space
  - *16 bit* logical address
- 4KB (2^{12}) page frame /page
  - 12 bit offset [*LSB*]
  - $2^{15} / 2^{12} = 2^3$ page frames
    - 3 bit page frame number
  - $2^{16} / 2^{12} = 2^4$ pages
    - *4 bit* logical page number

Content Quiz 7-2: Memory Management- Parts 4-5b (Paging)

Embedded real-time systems

Early real-time operating systems (RTOSs) did not have any mechanisms to enforce memory protection among multiple processes. They employed a simple ***shared memory*** model that allowed all processes to access all shared memory and they relied on process' trustworthiness to ensure secure operation. This model worked quite well for closed embedded real-time systems where all programs were known ahead of time and went through rigorous certification processes to guarantee their trustworthiness.

As embedded real-time systems become more sophisticated and more interconnected, the above approach does not work well. So, modern embedded real-time systems typically have a ***memory management unit*** that enforces memory protection among multiple processes. However, even modern embedded systems tend to use ***simpler*** memory management mechanisms such as partitioning (typically static partitioning). More sophisticated memory management mechanisms such as demand paging are typically not used because they have higher overheads, which is undesirable in a system with real-time requirements.

Question 1

1 / 1 pts

Compaction is an option to reduce [external] fragmentation of main memory. The disadvantage of this approach is that it is [expensive] to keep doing it every now and then.

Answer 1:
external

Answer 2:
expensive

Question 2

2 / 2 pts

Paging is a [mechanism] that allows a process to be split up into multiple parts and allocated to multiple partitions. Here, the main memory is divided into equal sized blocks called [page] [frames] and the logical address space of a process is divided into blocks of the same size called [pages] .

Answer 1:
mechanism

Answer 2:
page

Answer 3:
frames

Answer 4:
pages

Question 3

1 / 1 pts

Logical pages of a given process must always be allocated to contiguous page frames in the physical main memory. True or False?

True

False

Question 4

1 / 1 pts

To keep track of what logical page of which process maps to what physical page frame in the main memory, the Operating Systems uses structures called [page] [tables] .

Answer 1:
page

Answer 2:
tables

Question 5

2.5 / 2.5 pts

In a paged system, the logical address issued by a process includes two parts, namely, the logical [page] number and the [offset] within it. Similarly, the physical address includes the physical [page] [frame] number and the [offset] within it.

Answer 1:
page

Answer 2:
offset

Answer 3:
page

Answer 4:
frame

Answer 5:
offset

Question 6

0.5 / 0.5 pts

The operating system is responsible for translating the logical address issued by a process into a physical address. While doing so, it can retain the [offset] as it is.

Answer 1:
offset

Question 7

1 / 1 pts

To speed up access to page table entries, systems typically store frequently used page table entries in a special cache called the [translation] [look-aside] buffer.

Answer 1:
translation

Answer 2:
look-aside

Question 8

3 / 3 pts

A process may not use all of its data all the time. An approach in which pages may be brought into the main memory as needed is called [demand] [paging] . When using this approach, if a process requests for data on some page and that page is not in the main memory yet, a [page] [fault] is said to occur.

In a system using this approach, logical - also know as [virtual] addresses may be [longer] than physical addresses.

Answer 1:
demand

Answer 2:
paging

Answer 3:

page

Answer 4:
fault

Answer 5:
virtual

Answer 6:
longer

Content Quiz 8-1: Memory Management - Parts 6 + 7 (Page Replacement)

Question 1

3 / 3 pts

Choose answers from each of the dropdown boxes below that will correctly complete the "algorithm" shown below for the *Optimal* page replacement policy.

if (newly requested page is already in main memory) {

access is a hit

} else {

access causes a page fault

identify page whose next access will be farthest in the future

replace identified page with newly requested page

}

Answer 1:
already in main memory

Answer 2:
is a hit

Answer 3:
causes a page fault

Answer 4:
page whose next access will be farthest in the future

Answer 5:
identified page

Answer 6:
newly requested page

Question 2

3 / 3 pts

Choose answers from each of the dropdown boxes below that will correctly complete the "algorithm" shown below for the *First In First Out* page replacement policy.

if (newly requested page is already in main memory) {

access is a hit

} else {

access causes a page fault

identify page that was brought into main memory the longest time ago

replace identified page with newly requested page

}

Answer 1:
already in main memory

Answer 2:
is a hit

Answer 3:
causes a page fault

Answer 4:
page that was brought into main memory the longest time ago

Answer 5:
identified page

Answer 6:
newly requested page

Question 3

4 / 4 pts

Choose answers from each of the dropdown boxes below that will correctly complete the "algorithm" shown below for the *Modified First In First Out* page replacement policy, also known as the *Second Chance* page replacement policy.

if (newly requested page is already in the main memory) {

access is a hit

set reference bit of page to ["", ""]

} else {

access ["", ""]

do {

identify page currently marked as the ["", "", ""]

if (identified page has its reference bit set to true) {

mark identified page as the ["", "", ""] page

set reference bit of identified page to ["", ""]

mark the second oldest page as the oldest page

} else {

replace identified page with newly requested page

break out of loop

}

}

Answer 1:
is a hit

Answer 2:
true

Answer 3:

causes a page fault

Answer 4:
oldest page

Answer 5:
newest page

Answer 6:
false

Answer 7:
identified page

Answer 8:
newly requested page

Question 4

3 / 3 pts

Choose answers from each of the dropdown boxes below that will correctly complete the "algorithm" shown below for the *Least Recently Used* page replacement policy.

if (newly requested page is already in main memory) {

access is a hit

} else {

access causes a page fault

identify page whose previous access was farthest in the past

replace identified page with newly requested page

}

Answer 1:
already in main memory

Answer 2:
is a hit

Answer 3:
causes a page fault

Answer 4:
page whose previous access was farthest in the past

Answer 5:
identified page

Answer 6:
newly requested page

Unix / Linux - Directory Management

Previous Page Next Page

In this chapter, we will discuss in detail about directory management in Unix.

A directory is a file the solo job of which is to store the file names and the related information. All the files, whether ordinary, special, or directory, are contained in directories.

Unix uses a hierarchical structure for organizing files and directories. This structure is often referred to as a directory tree. The tree has a single root node, the slash character (**/**), and all other directories are contained below it.

Home Directory

The directory in which you find yourself when you first login is called your home directory.

You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

You can go in your home directory anytime using the following command −

```
$cd ~
$
```

Here **~** indicates the home directory. Suppose you have to go in any other user's home directory, use the following command −

```
$cd ~username
$
```

To go in your last directory, you can use the following command −

```
$cd -
$
```

Absolute/Relative Pathnames

Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.

Following are some examples of absolute filenames.

```
/etc/passwd
/users/sjones/chem/notes
/dev/rdsk/Os3
```

A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood's home directory, some pathnames might look like this −

```
chem/notes
personal/res
```

To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory −

```
$pwd
/user0/home/amrood

$
```

Listing Directories

To list the files in a directory, you can use the following syntax −

```
$ls dirname
```

Following is the example to list all the files contained in **/usr/local** directory −

```
$ls /usr/local

X11      bin        gimp     jikes     sbin
ace      doc        include  lib       share
atalk    etc        info     man       ami
```

Creating Directories

We will now understand how to create directories. Directories are created by the following command −

```
$mkdir dirname
```

Here, directory is the absolute or relative pathname of the directory you want to create. For example, the command −

```
$mkdir mydir
$
```

Creates the directory **mydir** in the current directory. Here is another example −

```
$mkdir /tmp/test-dir
$
```

This command creates the directory **test-dir** in the **/tmp** directory. The **mkdir** command produces no output if it successfully creates the requested directory.

If you give more than one directory on the command line, **mkdir** creates each of the directories. For example, −

```
$mkdir docs pub
$
```

Creates the directories docs and pub under the current directory.

Creating Parent Directories

We will now understand how to create parent directories. Sometimes when you want to create a directory, its parent directory or directories might not exist. In this case, **mkdir** issues an error message as follows −

```
$mkdir /tmp/amrood/test
mkdir: Failed to make directory "/tmp/amrood/test";
No such file or directory
$
```

In such cases, you can specify the **-p** option to the **mkdir** command. It creates all the necessary directories for you. For example −

```
$mkdir -p /tmp/amrood/test
$
```

The above command creates all the required parent directories.

Removing Directories

Directories can be deleted using the **rmdir** command as follows −

```
$rmdir dirname
$
```

Note − To remove a directory, make sure it is empty which means there should not be any file or sub-directory inside this directory.

You can remove multiple directories at a time as follows −

```
$rmdir dirname1 dirname2 dirname3
$
```

The above command removes the directories dirname1, dirname2, and dirname3, if they are empty. The **rmdir** command produces no output if it is successful.

Changing Directories

You can use the **cd** command to do more than just change to a home directory. You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as given below −

```
$cd dirname
$
```

Here, **dirname** is the name of the directory that you want to change to. For example, the command −

```
$cd /usr/local/bin
$
```

Changes to the directory **/usr/local/bin**. From this directory, you can **cd** to the directory **/usr/home/amrood** using the following relative path −

```
$cd ../../home/amrood
$
```

Renaming Directories

The **mv (move)** command can also be used to rename a directory. The syntax is as follows −

```
$mv olddir newdir
$
```

You can rename a directory **mydir** to **yourdir** as follows −

```
$mv mydir yourdir
$
```

The directories . (dot) and .. (dot dot)

The **filename .** (dot) represents the current working directory; and the **filename ..** (dot dot) represents the directory one level above the current working directory, often referred to as the parent directory.

If we enter the command to show a listing of the current working directories/files and use the **-a option** to list all the files and the **-l option** to provide the long listing, we will receive the following result.

```
$ls -la
drwxrwxr-x    4    teacher   class   2048  Jul 16 17.56 .
drwxr-xr-x    60   root              1536  Jul 13 14:18 ..
----------    1    teacher   class   4210  May 1 08:27 .profile
-rwxr-xr-x    1    teacher   class   1948  May 12 13:42 memo
$
```

Previous Page Next Page

Unix / Linux - File Management

Previous Page Next Page

In this chapter, we will discuss in detail about file management in Unix. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

When you work with Unix, one way or another, you spend most of your time working with files. This tutorial will help you understand how to create and remove files, copy and rename them, create links to them, etc.

In Unix, there are three basic types of files −

- **Ordinary Files** − An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
- **Directories** − Directories store both special and ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.
- **Special Files** − Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

Listing Files

To list the files and directories stored in the current directory, use the following command −

```
$ls
```

Here is the sample output of the above command −

```
$ls

bin        hosts  lib     res.03
```

```
ch07       hw1    pub     test_results
ch07.bak   hw2    res.01  users
docs       hw3    res.02  work
```

The command **ls** supports the **-l** option which would help you to get more information about the listed files −

```
$ls -l
total 1962188

drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
drwxr-xr-x  2    200    300      4096 Nov 25  2007 webthumb-1.01
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf
drwxr-xr-x 11 amrood amrood      4096 May 29  2007 zlib-1.2.3
$
```

Here is the information about all the listed columns −

First Column − Represents the file type and the permission given on the file. Below is the description of all type of files.

Second Column − Represents the number of memory blocks taken by the file or directory.

Third Column − Represents the owner of the file. This is the Unix user who created this file.

Fourth Column − Represents the group of the owner. Every Unix user will have an associated group.

Fifth Column − Represents the file size in bytes.

Sixth Column − Represents the date and the time when this file was created or modified for the last time.

Seventh Column − Represents the file or the directory name.

In the **ls -l** listing example, every file line begins with a **d**, **-**, or **l**. These characters indicate the type of the file that's listed.

Sr.No.	Prefix & Description

1	**-** Regular file, such as an ASCII text file, binary executable, or hard link.
2	**b** Block special file. Block input/output device file such as a physical hard drive.
3	**c** Character special file. Raw input/output device file such as a physical hard drive.
4	**d** Directory file that contains a listing of other files and directories.
5	**l** Symbolic link file. Links on any regular file.
6	**p** Named pipe. A mechanism for interprocess communications.
7	**s** Socket used for interprocess communication.

Metacharacters

Metacharacters have a special meaning in Unix. For example, ***** and **?** are metacharacters. We use ***** to match 0 or more characters, a question mark (**?**) matches with a single character.

For Example −

```
$ls ch*.doc
```

Displays all the files, the names of which start with **ch** and end with **.doc** −

```
ch01-1.doc   ch010.doc  ch02.doc    ch03-2.doc
ch04-1.doc   ch040.doc  ch05.doc    ch06-2.doc
ch01-2.doc ch02-1.doc c
```

Here, ***** works as meta character which matches with any character. If you want to display all the files ending with just **.doc**, then you can use the following command −

```
$ls *.doc
```

Hidden Files

An invisible file is one, the first character of which is the dot or the period character (.). Unix programs (including the shell) use most of these files to store configuration information.

Some common examples of the hidden files include the files −

- **.profile** − The Bourne shell (sh) initialization script
- **.kshrc** − The Korn shell (ksh) initialization script
- **.cshrc** − The C shell (csh) initialization script
- **.rhosts** − The remote shell configuration file

To list the invisible files, specify the **-a** option to **ls** −

```
$ ls -a

.         .profile       docs     lib     test_results
..        .rhosts        hosts    pub     users
.emacs    bin            hw1      res.01  work
.exrc     ch07           hw2      res.02
.kshrc    ch07.bak       hw3      res.03
$
```

- **Single dot (.)** − This represents the current directory.
- **Double dot (..)** − This represents the parent directory.

Creating Files

You can use the **vi** editor to create ordinary files on any Unix system. You simply need to give the following command −

```
$ vi filename
```

The above command will open a file with the given filename. Now, press the key **i** to come into the edit mode. Once you are in the edit mode, you can start writing your content in the file as in the following program −

```
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
```

Once you are done with the program, follow these steps −

Press the key **esc** to come out of the edit mode.

Press two keys **Shift + ZZ** together to come out of the file completely.

You will now have a file created with **filename** in the current directory.

```
$ vi filename
$
```

Editing Files

You can edit an existing file using the **vi** editor. We will discuss in short how to open an existing file −

```
$ vi filename
```

Once the file is opened, you can come in the edit mode by pressing the key **i** and then you can proceed by editing the file. If you want to move here and there inside a file, then first you need to come out of the edit mode by pressing the key **Esc**. After this, you can use the following keys to move inside a file −

l key to move to the right side.

h key to move to the left side.

k key to move upside in the file.

j key to move downside in the file.

So using the above keys, you can position your cursor wherever you want to edit. Once you are positioned, then you can use the **i** key to come in the edit mode. Once you are done with the editing in your file, press **Esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

Display Content of a File

You can use the **cat** command to see the content of a file. Following is a simple example to see the content of the above created file −

```
$ cat filename
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
$
```

You can display the line numbers by using the **-b** option along with the **cat** command as follows −

```
$ cat -b filename
1   This is unix file....I created it for the first time.....
2   I'm going to save this content in this file.
$
```

Counting Words in a File

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is a simple example to see the information about the file created above −

```
$ wc filename
2  19 103 filename
$
```

Here is the detail of all the four columns −

- **First Column** − Represents the total number of lines in the file.
- **Second Column** − Represents the total number of words in the file.
- **Third Column** − Represents the total number of bytes in the file. This is the actual size of the file.
- **Fourth Column** − Represents the file name.

You can give multiple files and get information about those files at a time. Following is simple syntax −

```
$ wc filename1 filename2 filename3
```

Copying Files

To make a copy of a file use the **cp** command. The basic syntax of the command is −

```
$ cp source_file destination_file
```

Following is the example to create a copy of the existing file **filename**.

```
$ cp filename copyfile
$
```

You will now find one more file **copyfile** in your current directory. This file will exactly be the same as the original file **filename**.

Renaming Files

To change the name of a file, use the **mv** command. Following is the basic syntax −

```
$ mv old_file new_file
```

The following program will rename the existing file **filename** to **newfile**.

```
$ mv filename newfile
$
```

The **mv** command will move the existing file completely into the new file. In this case, you will find only **newfile** in your current directory.

Deleting Files

To delete an existing file, use the **rm** command. Following is the basic syntax −

```
$ rm filename
```

Caution − A file may contain useful information. It is always recommended to be careful while using this **Delete** command. It is better to use the **-i** option along with **rm** command.

Following is the example which shows how to completely remove the existing file **filename**.

```
$ rm filename
$
```

You can remove multiple files at a time with the command given below −

```
$ rm filename1 filename2 filename3
$
```

Standard Unix Streams

Under normal circumstances, every Unix program has three streams (files) opened for it when it starts up −

- **stdin** − This is referred to as the *standard input* and the associated file descriptor is 0. This is also represented as STDIN. The Unix program will read the default input from STDIN.
- **stdout** − This is referred to as the *standard output* and the associated file descriptor is 1. This is also represented as STDOUT. The Unix program will write the default output at STDOUT
- **stderr** − This is referred to as the *standard error* and the associated file descriptor is 2. This is also represented as STDERR. The Unix program will write all the error messages at STDERR.

Previous Page Next Page

Unix / Linux - File Permission / Access Modes

Previous Page Next Page

In this chapter, we will discuss in detail about file permission and access modes in Unix. File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

- **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows −

```
$ls -l /home/amrood
-rwxr-xr--   1 amrood   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 amrood   users 1024  Nov 2 00:10  mydir
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

- The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.

The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.

The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

Read

Grants the capability to read, i.e., view the contents of the file.

Write

Grants the capability to modify, or remove the content of the file.

Execute

User with execute permissions can run a file as a program.

Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

Read

Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

Write

Access means that the user can add or delete files from the directory.

Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

Sr.No.	Chmod operator & Description
1	**+** Adds the designated permission(s) to a file or directory.
2	**-** Removes the designated permission(s) from a file or directory.
3	**=** Sets the designated permission(s).

Here's an example using **testfile**. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod g = rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Here's how you can combine these commands on a single line −

```
$chmod o+wx,u-x,g = rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Here's an example using the testfile. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
$ls -l testfile
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Changing Owners and Groups

While creating an account on Unix, it assigns a **owner ID** and a **group ID** to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files −

- **chown** − The **chown** command stands for **"change owner"** and is used to change the owner of a file.
- **chgrp** − The **chgrp** command stands for **"change group"** and is used to change the group of a file.

Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows −

```
$ chown user filelist
```

The value of the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

The following example will help you understand the concept −

```
$ chown amrood testfile
$
```

Changes the owner of the given file to the user **amrood**.

NOTE − The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own.

Changing Group Ownership

The **chgrp** command changes the group ownership of a file. The basic syntax is as follows −

```
$ chgrp group filelist
```

The value of group can be the **name of a group** on the system or **the group ID (GID)** of a group on the system.

Following example helps you understand the concept −

```
$ chgrp special testfile
$
```

Changes the group of the given file to **special** group.

SUID and SGID File Permission

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file **/etc/shadow**.

As a regular user, you do not have **read** or **write** access to this file for security reasons, but when you change your password, you need to have the write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file **/etc/shadow**.

Additional permissions are given to programs via a mechanism known as the **Set User ID (SUID)** and **Set Group ID (SGID)** bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is the case with SGID as well. Normally, programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter **"s"** if the permission is available. The SUID **"s"** bit will be located in the permission bits where the owners' **execute** permission normally resides.

For example, the command −

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x  1   root   bin  19031 Feb 7 13:47  /usr/bin/passwd*
$
```

Shows that the SUID bit is set and that the command is owned by the root. A capital letter **S** in the execute position instead of a lowercase **s** indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users −

- The owner of the sticky directory
- The owner of the file being removed
- The super user, root

To set the SUID and SGID bits for any directory try the following command −

```
$ chmod ug+s dirname
$ ls -l
drwsr-sr-x 2 root root  4096 Jun 19 06:45 dirname
$
```

Previous Page

Next Page