


TCCS 422: OPERATING SYSTEMS

Memory Virtualization with Segments, Introduction to Paging

Wes J. Lloyd
 School of Engineering and Technology
 University of Washington - Tacoma



May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

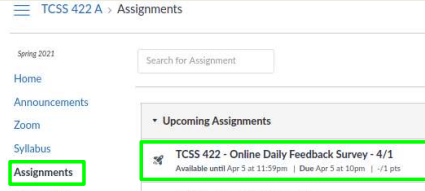
OBJECTIVES – 5/20

- **Questions from 5/18**
- Assignment 2
- Quiz 3 – Synchronized Array
- Tutorial 2 – Pthread, locks, conditions tutorial
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.2

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys **ON TIME**
- Tuesday surveys: due by ~ Wed @ 11:59p
- Thursday surveys: due ~ Mon @ 11:59p



May 20, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.3

TCCS 422 - Online Daily Feedback Survey - 4/1

Quiz Instructions

Question 1 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

1	2	3	4	5	6	7	8	9	10
Mostly Review To Me			Equal New and Review				Mostly New To Me		

Question 2 0.5 pts

Please rate the pace of today's class:

1	2	3	4	5	6	7	8	9	10
slow	just right				fast				

May 20, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (46 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.44 (↑ - previous 6.32)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.69 (↑ - previous 5.63)**

May 20, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.5

FEEDBACK

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.6

OBJECTIVES – 5/20

- Questions from 5/18
- **Assignment 2**
- Quiz 3 – Synchronized Array
- Tutorial 2 – Pthread, locks, conditions tutorial
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging

May 20, 2021 TCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.7

OBJECTIVES – 5/20

- Questions from 5/18
- Assignment 2
- **Quiz 3 – Synchronized Array**
- Tutorial 2 – Pthread, locks, conditions tutorial
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging

May 20, 2021 TCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.8

OBJECTIVES – 5/20

- Questions from 5/18
- Assignment 2
- Quiz 3 – Synchronized Array
- **Tutorial 2 – Pthread, locks, conditions tutorial**
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging


May 20, 2021 TCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.9

OBJECTIVES – 5/20

- Questions from 5/18
- Assignment 2
- Quiz 3 – Synchronized Array
- Tutorial 2 – Pthread, locks, conditions tutorial
- **Chapter 14: The Memory API**
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging

May 20, 2021 TCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.10

CHAPTER 14: THE MEMORY API



May 20, 2021 TCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.11

OBJECTIVES – 5/18

- **Chapter 13: Introduction to memory virtualization**
 - The address space
 - Goals of OS memory virtualization
- **Chapter 14: Memory API**
 - Common memory errors

May 20, 2021 TCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.12

REALLOC()

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size)
```

- Resize an existing memory allocation
- Returned pointer may be same address, or a new address
 - New if memory allocation must move
- `void *ptr`: Pointer to memory block allocated with `malloc`, `calloc`, or `realloc`
- `size_t size`: New size for the memory block(in bytes)

■ EXAMPLE: `realloc.c`
 ■ EXAMPLE: `nom.c`

May 20, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.13
--------------	---	--------

DOUBLE FREE

```
int *x = (int *)malloc(sizeof(int)); // allocated
free(x); // free memory
free(x); // free repeatedly
```

- Can't deallocate twice
- Second call core dumps

May 20, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.14
--------------	---	--------

SYSTEM CALLS

- `brk()`, `sbrk()`
 - Used to change data segment size (the end of the heap)
 - Don't use these
- `Mmap()`, `munmap()`
 - Can be used to create an extra independent "heap" of memory for a user program
- See man page

May 20, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.15
--------------	---	--------

OBJECTIVES – 5/20

- Questions from 5/18
- Assignment 2
- Quiz 3 – Synchronized Array
- Tutorial 2 – Pthread, locks, conditions tutorial
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging

May 20, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.16
--------------	---	--------

CHAPTER 15: ADDRESS TRANSLATION

May 20, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.17
--------------	---	--------

OBJECTIVES – 5/18

- Chapter 15: Address translation
 - Base and bounds
 - HW and OS Support

May 20, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.18
--------------	---	--------

ADDRESS TRANSLATION

- 64KB Address space example
- Translation: mapping virtual to physical

May 20, 2021
 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
 L15.19

BASE AND BOUNDS

- Dynamic relocation
- Two registers base & bounds: **on the CPU**
- OS places program in memory
- Sets base register

$$\text{physical address} = \text{virtual address} + \text{base}$$

- Bounds register
 - Stores size of program address space (16KB)
- OS verifies that every address:

$$0 \leq \text{virtual address} < \text{bounds}$$

May 20, 2021
 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
 L15.20

INSTRUCTION EXAMPLE

```
128 : movl 0x0(%ebx), %eax
```

- Base = 32768
- Bounds = 16384
- Fetch instruction at 128 (virt addr) ↑
 - Phy addr = virt addr + base reg
 - 32896 = 128 + 32768 (base)
- Execute instruction
 - Load from address (var x is @ 15kb=15360)
 - 48128 = 15360 + 32768 (base) -- found x...
- Bounds register: terminate process if
 - ACCESS VIOLATION:** Virtual address > bounds reg

$$\text{physical address} = \text{virtual address} + \text{base}$$

May 20, 2021
 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
 L15.21

MEMORY MANAGEMENT UNIT

- MMU
 - Portion of the CPU dedicated to address translation
 - Contains base & bounds registers
- Base & Bounds Example:
 - Consider address translation
 - 4 KB (4096 bytes) address space, loaded at 16 KB physical location

Virtual Address	Physical Address
0	16384
1024	17408
3000	19384
FAULT 4400	20784 (out of bounds)

May 20, 2021
 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
 L15.22

DYNAMIC RELOCATION OF PROGRAMS

- Hardware requirements:

Requirements	HW support
Privileged mode	CPU modes: kernel, user
Base / bounds registers	Registers to support address translation
Translate virtual addr; check if in bounds	Translation circuitry, check limits
Privileged instruction(s) to update base / bounds regs	Instructions for modifying base/bound registers
Privileged instruction(s) to register exception handlers	Set code pointers to OS code to handle faults to register exception handlers
Ability to raise exceptions	For out-of-bounds memory access, or attempts to access privileged instr.

May 20, 2021
 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
 L15.23

OS SUPPORT FOR MEMORY VIRTUALIZATION

- For base and bounds OS support required
 - When process starts running
 - Allocate address space in physical memory
 - When a process is terminated
 - Reclaiming memory for use
 - When context switch occurs
 - Saving and storing the base-bounds pair
 - Exception handlers
 - Function pointers set at OS boot time

May 20, 2021
 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
 L15.24

OS: WHEN PROCESS STARTS RUNNING

- OS searches for free space for new process
- Free list: data structure that tracks available memory slots

The OS lookup the free list

Free list

16KB

48KB

Physical Memory

0KB Operating System

16KB (not in use)

32KB Code

Heap

48KB Stack

64KB (not in use)

Physical Memory

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.25

OS: WHEN PROCESS IS TERMINATED

- OS places memory back on the free list

Free list

16KB

32KB

48KB

Physical Memory

0KB Operating System

16KB (not in use)

32KB Process A

48KB (not in use)

64KB (not in use)

Physical Memory

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.26

OS: WHEN CONTEXT SWITCH OCCURS

- OS must save base and bounds registers
- Saved to the Process Control Block PCB (task_struct in Linux)

Context Switching

Process A PCB

base: 32KB

bounds: 48KB

Physical Memory

0KB Operating System

16KB (not in use)

32KB Process A Currently Running

48KB Process B

64KB (not in use)

Physical Memory

0KB Operating System

16KB (not in use)

32KB Process A

48KB Process B Currently Running

64KB (not in use)

Physical Memory

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.27

DYNAMIC RELOCATION

- OS can move process data when not running

1. OS deschedules process from scheduler
2. OS copies address space from current to new location
3. OS updates PCB (base and bounds registers)
4. OS reschedules process

- When process runs new base register is restored to CPU
- **Process doesn't know it was even moved!**

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.28

Consider a 64KB computer the loads a program. The BASE register is set to 32768, and the BOUNDS register is set to 4096. What is the physical memory address translation for a virtual address of 6000 ?

- 34768
- 38768
- 32769
- 36864
- Out of bounds

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollux.com/app


May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.29

OBJECTIVES – 5/20

- Questions from 5/18
- Assignment 2
- Quiz 3 – Synchronized Array
- Tutorial 2 – Pthread, locks, conditions tutorial
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.30

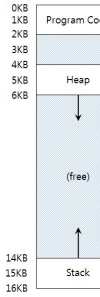
CHAPTER 16: SEGMENTATION



May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.31

BASE AND BOUNDS INEFFICIENCIES

- Address space
 - Contains significant unused memory
 - Is relatively large
 - Preallocates space to handle stack/heap growth
- Large address spaces
 - Hard to fit in memory
- How can these issues be addressed?



May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.32

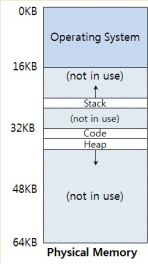
MULTIPLE SEGMENTS

- Memory segmentation
- Manage the address space as (3) separate segments
 - Each is a contiguous address space
 - Provides logically separate segments for: code, stack, heap
- Each segment can be placed separately
- Track base and bounds for each segment (registers)

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.33

SEGMENTS IN MEMORY

■ Consider 3 segments:



Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

↓
Much smaller

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.34

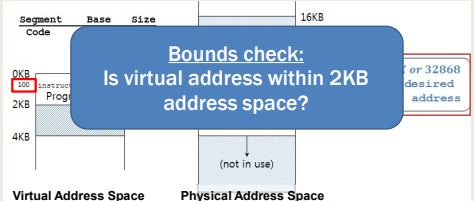
ADDRESS TRANSLATION: CODE SEGMENT

$physical\ address = offset + base$

- Code segment - physically starts at 32KB (base)
- Starts at "0" in virtual address space

Segment	Base	Size
Code	32KB	16KB

Bounds check:
Is virtual address within 2KB address space?



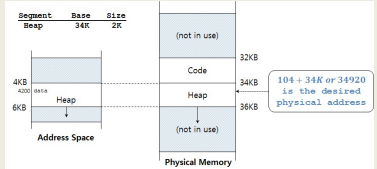
May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.35

ADDRESS TRANSLATION: HEAP

Virtual address + base is not the correct physical address.

- Heap starts at virtual address 4096
- The data is at 4200
- Offset = 4200 - 4096 = 104 (virt addr - virt heap start)
- Physical address = 104 + 34816 (offset + heap base)

Segment	Base	Size
Heap	34K	2K



May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.36

SEGMENTATION FAULT

- Access beyond the address space
- Heap starts at virtual address: 4096
- Data pointer is to 7KB (7168)
- Is data pointer valid?

Heap starts at $4096 + 2 \text{ KB seg size} = 6144$
 Offset = $7168 > 4096 + 2048 (6144)$

Address Space

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.37

SEGMENT REGISTERS

- Used to dereference memory during translation

- First two bits identify segment type
- Remaining bits identify memory offset
- Example: virtual heap address 4200 (01000001101000)

Segment	bits
Code	00
Heap	01
Stack	10
-	11

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.38

SEGMENTATION DEREFERENCE

```

1 // get top 2 bits of 14-bit VA
2 Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
3 // now get offset
4 Offset = VirtualAddress & OFFSET_MASK
5 if (Offset >= Bounds[Segment])
6     RaiseException(PROTECTION_FAULT)
7 else
8     PhysAddr = Base[Segment] + Offset
9     Register = AccessMemory(PhysAddr)
    
```

- VIRTUAL ADDRESS = 01000001101000 (on heap)
- SEG_MASK = 0x3000 (11000000000000)
- SEG_SHIFT = 01 → **heap** (mask gives us segment code)
- OFFSET_MASK = 0xFFF (00111111111111)
- OFFSET = 000001101000 = 104 (isolates segment offset)
- OFFSET < BOUNDS : 104 < 2048

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.39

STACK SEGMENT

- Stack grows backwards (FILO)
- Requires hardware support:
- Direction bit: tracks direction segment grows

Segment	Base	Size	Grows	Positive?
Code	32K	2K	1	
Heap	34K	2K	1	
Stack	28K	2K	0	

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.40

SHARED CODE SEGMENTS

- Code sharing: enabled with HW support
- Supports storing shared libraries in memory only once
- DLL: dynamic linked library
- .so (linux): shared object in Linux (under /usr/lib)
- Many programs can access them
- Protection bits: track permissions to segment

Segment Register Values(with Protection)					
Segment	Base	Size	Grows	Positive?	Protection
Code	32K	2K	1		Read-Execute
Heap	34K	2K	1		Read-Write
Stack	28K	2K	0		Read-Write

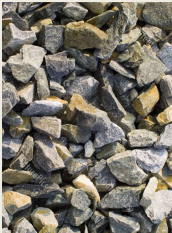
May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.41

Consider a program with 2KB of code, a 1 KB stack, and a 2 KB heap. This program runs on a 64 KB computer that manages memory with 4 kb segments. If the computer is empty and segments were allocated as: code, stack, heap, how large can the heap grow to?

Start the presentation to see live content. For screen share software, share the entire screen. Get help at poller.com/app

SEGMENTATION GRANULARITY


- Coarse-grained
- Manage memory as large purpose based segments:
 - Code segment
 - Heap segment
 - Stack segment



May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.43

SEGMENTATION GRANULARITY - 2

- Fine-grained
- Manage memory as list of segments
- Code, heap, stack segments composed of multiple smaller segments
- Segment table
 - On early systems
 - Stored in memory
 - Tracked large number of segments



May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.44

MEMORY FRAGMENTATION

- Consider how much free space?
- We'll say about 24 KB
- Request arrives to allocate a 20 KB heap segment
- Can we fulfil the request for 20 KB of contiguous memory?

Not compacted	
0KB	
8KB	Operating System
16KB	(not in use)
24KB	Allocated
32KB	(not in use)
40KB	Allocated
48KB	(not in use)
56KB	Allocated
64KB	

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.45

COMPACTION

- Supports rearranging memory
- Can we fulfil the request for 20 KB of contiguous memory?
- **Drawback:** Compaction is slow
 - Rearranging memory is time consuming
 - 64KB is fast
 - 4GB+ ... slow
- Algorithms:
 - Best fit: keep list of free spaces, allocate the most snug segment for the request
 - Others: worst fit, first fit... (in future chapters)

Compacted	
0KB	
8KB	Operating System
16KB	
24KB	Allocated
32KB	
40KB	
48KB	(not in use)
56KB	
64KB	

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.46

WE WILL RETURN AT 4:50PM

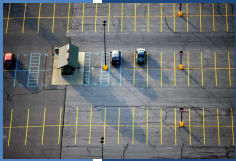


May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.47

OBJECTIVES – 5/20

- Questions from 5/18
- Assignment 2
- Quiz 3 – Synchronized Array
- Tutorial 2 – Pthread, locks, conditions tutorial
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- Chapter 18: Introduction to Paging

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.48



CHAPTER 17: FREE SPACE MANAGEMENT

May 20, 2021 TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.49

OBJECTIVES – 5/18

- **Chapter 17: Free Space Management**
 - Fragmentation, Splitting, coalescing
 - The Free List
 - Memory Allocation Strategies

May 20, 2021 TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.50

FREE SPACE MANAGEMENT

- How should free space be managed, when satisfying variable-sized requests?
- What strategies can be used to minimize fragmentation?
- What are the time and space overheads of alternate approaches?

May 20, 2021 TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.51

FREE SPACE MANAGEMENT

- Management of memory using
 - Only fixed-sized units
 - Easy: keep a list
 - Memory request → return first free entry
 - Simple search
 - With variable sized units
 - More challenging
 - Results from variable sized malloc requests
 - Leads to fragmentation

May 20, 2021 TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.52

FRAGMENTATION

- Consider a 30-byte heap

30-byte heap: free used free

0 10 20 30
- Request for 15-bytes

free list: head → addr:0
len:10 → addr:20
len:10 → NULL
- Free space: 20 bytes
- No available contiguous chunk → return NULL

May 20, 2021 TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.53

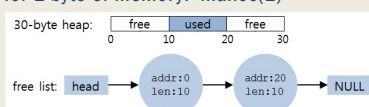
FRAGMENTATION - 2

- **External:** OS can compact
 - Example: Client asks for 100 bytes: malloc(100)
 - OS: No 100 byte contiguous chunk is available: returns NULL
 - Memory is externally fragmented - - Compaction can fix!
- **Internal:** lost space – OS can't compact
 - OS returns memory units that are too large
 - Example: Client asks for 100 bytes: malloc(100)
 - OS: Returns 125 byte chunk
 - Fragmentation is *in* the allocated chunk
 - Memory is lost, and unaccounted for – can't compact

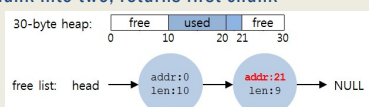
May 20, 2021 TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.54

ALLOCATION STRATEGY: SPLITTING

- Request for 1 byte of memory: malloc(1)




- OS locates a free chunk to satisfy request
- Splits chunk into two, returns first chunk




May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.55

ALLOCATION STRATEGY: COALESCING

- Consider 30-byte heap
- Free() frees all 10 bytes segments (list of 3-free 10-byte chunks)



- Request arrives: malloc(30)
- **SPLIT DOES NOT WORK** - no contiguous 30-byte chunk exists!
- Coalescing regroups chunks into contiguous chunk

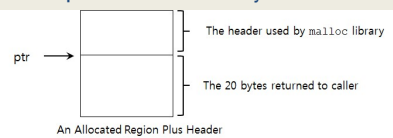


- Allocation can now proceed
- Coalescing is defragmentation of the free space list

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.56

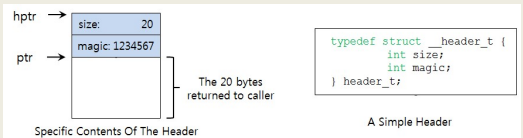
MEMORY HEADERS

- free(void *ptr): Does not require a size parameter
- How does the OS know how much memory to free?
- Header block
 - Small descriptive block of memory at start of chunk



May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.57

MEMORY HEADERS - 2



```

typedef struct __header_t {
    int size;
    int magic;
} header_t;
    
```

- Contains size
- Pointers: for faster memory access
- Magic number: integrity checking

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.58

MEMORY HEADERS - 3

- Size of memory chunk is:
 - Header size + user malloc size
 - N bytes + sizeof(header)
- Easy to determine address of header

```

void free(void *ptr) {
    header_t *hptr = (void *)ptr - sizeof(header_t);
}
    
```

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.59

THE FREE LIST

- Simple free list struct

```

typedef struct __node_t {
    int size;
    struct __node_t *next;
} node_t;
    
```

- Use mmap to create free list
- 4kb heap, 4 byte header, one contiguous free chunk

```

// mmap() returns a pointer to a chunk of free space
node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
    MAP_ANON|MAP_PRIVATE, -1, 0);
head->size = 4096 - sizeof(node_t);
head->next = NULL;
    
```

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.60

FREE LIST - 2

- Create and initialize free-list "heap"


```
// mmap() returns a pointer to a chunk of free space
node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_ANON|MAP_PRIVATE, -1, 0);
head->size = 4096 - sizeof(node_t);
head->next = NULL;
```
- Heap layout:

[virtual address: 16KB]
header: size field
header: next field(NULL is 0)
the rest of the 4KB chunk

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.61

FREE LIST: MALLOC() CALL

- Consider a request for a 100 bytes: malloc(100)
- Header block requires 8 bytes
 - 4 bytes for size, 4 bytes for magic number
- Split the heap – header goes with each block

A 4KB Heap With One Free Chunk A Heap: After One Allocation

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.62

FREE LIST: FREE() CALL

- Addresses of chunks
 - Start=16384
 - + 108 (end of 1st chunk)
 - + 108 (end of 2nd chunk)
 - = 16708
-

Free Space With Three Chunks Allocated

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.63

FREE LIST: FREE() CHUNK #2

- Free(sptr)
- Our 3 chunks start at 16 KB (@ 16,384 bytes)
- Free chunk #2 - sptr
- Sptr = 16500
 - addr - sizeof(node_t)
 - = 16492
- Actual start of chunk #2 = 16492

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.64

FREE LIST- FREE ALL CHUNKS

- Now free remaining chunks:
 - Free(16392)
 - Free(16608)
- Walk back 8 bytes for actual start of chunk
- External fragmentation
- Free chunk pointers out of order
- Coalescing of next pointers is needed

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.65

GROWING THE HEAP

- Start with small sized heap
- Request more memory when full
- sbrk(), brk()

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.66

MEMORY ALLOCATION STRATEGIES

- **Best fit**
 - Traverse free list
 - Identify all candidate free chunks
 - Note which is smallest (has best fit)
 - When splitting, “leftover” pieces are small (and potentially less useful – fragmented)
- **Worst fit**
 - Traverse free list
 - Identify largest free chunk
 - Split largest free chunk, leaving a still large free chunk

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.67

EXAMPLES

- Allocation request for 15 bytes

- Result of Best Fit

- Result of Worst Fit

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.68

MEMORY ALLOCATION STRATEGIES - 2

- **First fit**
 - Start search at beginning of free list
 - Find first chunk large enough for request
 - Split chunk, returning a “fit” chunk, saving the remainder
 - Avoids full free list traversal of best and worst fit
- **Next fit**
 - Similar to first fit, but start search at last search location
 - Maintain a pointer that “cycles” through the list
 - Helps balance chunk distribution vs. first fit
 - Find first chunk, that is large enough for the request, and split
 - Avoids full free list traversal

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.69

Which memory allocation strategy is more likely to distribute free chunks closer together which could help when coalescing the free space list?

Best Fit

Worst Fit

First Fit

None of the above

All of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at poller.com/app

SEGREGATED LISTS

- For popular sized requests
e.g. for kernel objects such as locks, inodes, etc.
- Manage as segregated free lists
- Provide object caches: stores pre-initialized objects
- How much memory should be dedicated for specialized requests (object caches)?
- If a given cache is low in memory, can request “slabs” of memory from the general allocator for caches.
- General allocator will reclaim slabs when not used

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.71

BUDDY ALLOCATION

- Binary buddy allocation
 - Divides free space by two to find a block that is big enough to accommodate the request; the next split is too small...
- Consider a 7KB request

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.72

BUDDY ALLOCATION - 2

- Buddy allocation: suffers from internal fragmentation
- Allocated fragments, typically too large
- Coalescing is simple
 - Two adjacent blocks are promoted up

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.73

A computer system manages program memory using three separate segments for code, stack, and the heap. The codesize of a program is 1KB but the minimal segment available is 16KB. This is an example of:

- External fragmentation
- Binary buddy allocation
- Internal fragmentation
- Coalescing
- Splitting

Start the presentation to see live content. For screen share software, share the entire screen. Get help at polllev.com/app

A request is made to store 1 byte. For this scenario, which memory allocation strategy will always locate memory the fastest?

- Best fit
- Worst fit
- Next fit
- None of the above
- All of the above


Start the presentation to see live content. For screen share software, share the entire screen. Get help at polllev.com/app

OBJECTIVES – 5/20

- Questions from 5/18
- Assignment 2
- Quiz 3 – Synchronized Array
- Tutorial 2 – Pthread, locks, conditions tutorial
- Chapter 14: The Memory API
- Chapter 15: Address Translation
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- **Chapter 18: Introduction to Paging**

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.76

CHAPTER 18: INTRODUCTION TO PAGING



May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.77

OBJECTIVES – 5/21

- Questions from 5/19
- Tuesday Class Activity: (Submit by May 22 11:59pm AOE)
- Tutorial 2 posted (pthreads, locks, conditions)
- Quiz 3 posted – Active Reading Chapter 19
- Assignment 2 (based on Ch. 30)
- Chapter 17: Free Space Management
- **Chapter 18: Introduction to Paging**
- Chapter 19: Translation Lookaside Buffer (TLB)
 - TLB Algorithm, Tradeoffs, Context Switch
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Hybrid Tables, Multi-level Page Tables

May 20, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L15.78

PAGING

- Split up address space of process into fixed sized pieces called **pages**
- Alternative to variable sized pieces (Segmentation) which suffers from significant fragmentation
- Physical memory is split up into an array of fixed-size slots called **page frames**.
- Each process has a **page table** which translates virtual addresses to physical addresses

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.79

ADVANTAGES OF PAGING

- Flexibility
 - Abstracts the process address space into pages
 - No need to track direction of HEAP / STACK growth
 - Just add more pages...
 - No need to store unused space
 - As with segments...
- Simplicity
 - Pages and page frames are the same size
 - Easy to allocate and keep a free list of pages

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.80

PAGING: EXAMPLE

Page Table:
 VP0 → PF3
 VP1 → PF7
 VP2 → PF5
 VP3 → PF2

- Consider a 128 byte (2^7) address space with 16-byte (2^4) pages
- Consider a 64-byte (2^6) program address space

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.81

PAGING: ADDRESS TRANSLATION

- PAGE: Has two address components
 - VPN: Virtual Page Number (serves as the page ID)
 - Offset: Offset within a Page (indexes any byte in the page)

VPN				offset			
Va5	Va4	Va3	Va2	Va1	Va0		

- Example:
 Page Size: 16-bytes (2^4),
 Program Address Space: 64-bytes (2^6)

VPN			offset		
0	1	0	1	0	1

Here program can have just four pages...

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.82

EXAMPLE: PAGING ADDRESS TRANSLATION

- Consider a 64-byte (2^6) program address space (4 pages → 2^2)
- Stored in 128-byte (2^7) physical memory (8 frames → 2^3)
- Offset is preserved
 - 4 bits indexes any byte
 - Page size is 16 bytes (2^4)
- Page table translates a Virtual Page Number (VPN) to a Physical Page Frame Number (PFN)

Page Table:
 VP0 → PF3
 VP1 → PF7
 VP2 → PF5
 VP3 → PF2

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.83

PAGING DESIGN QUESTIONS

- (1) Where are page tables stored?
- (2) What are the typical contents of the page table?
- (3) How big are page tables?
- (4) Does paging make the system too slow?

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.84

(1) WHERE ARE PAGE TABLES STORED?

- Example:
 - Consider a 32-bit process address space ($4GB=2^{32}$ bytes)
 - With 4 KB pages ($4KB=2^{12}$ bytes)
 - 20 bits for VPN (2^{20} pages)
 - 12 bits for the page offset (2^{12} unique bytes in a page)
- Page tables for each process are stored in RAM
 - Support potential storage of 2^{20} translations = 1,048,576 pages per process
 - Each page has a page table entry size of 4 bytes

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.85

PAGE TABLE EXAMPLE

- With 2^{20} slots in our page table for a single process
- Each slot (i.e. entry) dereferences a VPN
- Each entry provides a physical frame number
- Each entry requires 4 bytes (32 bits)
 - 20 for the PFN on a 4GB system with 4KB pages
 - 12 for the offset which is preserved
 - (note we have no status bits, so this is unrealistically small)
- How much memory is required to store the page table for 1 process?
 - Hint: # of entries x space per entry
 - 4,194,304 bytes (or 4MB) to index one process

VPN ₀
VPN ₁
VPN ₂
...
...
VPN _{1,048,576}

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.86

NOW FOR AN ENTIRE OS

- If 4 MB is required to store one process
- Consider how much memory is required for an entire OS?
 - With for example 100 processes...
- Page table memory requirement is now $4MB \times 100 = 400MB$
- If computer has 4GB memory (maximum for 32-bits), the page table consumes 10% of memory

400 MB / 4000 GB

- Is this efficient?

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.87

(2) WHAT'S ACTUALLY IN THE PAGE TABLE

- Page table is data structure used to map virtual page numbers (VPN) to the physical address (Physical Frame Number PFN)
 - Linear page table → simple array
- Page-table entry
 - 32 bits for capturing state

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
PFN																						U	S	R	W	D	A	P	M	U	S	R	W	P	A	L

An x86 Page Table Entry(PTE)

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.88

PAGE TABLE ENTRY

- P: present
- R/W: read/write bit
- U/S: supervisor
- A: accessed bit
- D: dirty bit
- PFN: the page frame number

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
PFN																						P	R	W	U	S	A	D	P	M	U	S	R	W	P	A	L

An x86 Page Table Entry(PTE)

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.89

PAGE TABLE ENTRY - 2

- Common flags:
 - **Valid Bit:** Indicating whether the particular translation is valid.
 - **Protection Bit:** Indicating whether the page could be read from, written to, or executed from
 - **Present Bit:** Indicating whether this page is in physical memory or on disk(swapped out)
 - **Dirty Bit:** Indicating whether the page has been modified since it was brought into memory
 - **Reference Bit(Accessed Bit):** Indicating that a page has been accessed

May 20, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L15.90

(3) HOW BIG ARE PAGE TABLES?

- Page tables are too big to store on the CPU
- Page tables are stored using physical memory
- Paging supports efficiently storing a sparsely populated address space
 - Reduced memory requirement
Compared to base and bounds, and segments

May 20, 2021 TCCS422: Operating Systems [Spring 2021] L15.91
 School of Engineering and Technology, University of Washington - Tacoma

(4) DOES PAGING MAKE THE SYSTEM TOO SLOW?

- Translation
 - Issue #1:** Starting location of the page table is needed
 - HW Support: Page-table base register
 - stores active process
 - Facilitates translation
 - Issue #2:** Each memory address translation for paging requires an extra memory reference
 - HW Support: TLBs (Chapter 19)

Page Table:
 VP0 → PF3
 VP1 → PF7
 VP2 → PF5
 VP3 → PF2

Stored in RAM →

May 20, 2021 TCCS422: Operating Systems [Spring 2021] L15.92
 School of Engineering and Technology, University of Washington - Tacoma

PAGING MEMORY ACCESS

```

1. // Extract the VPN from the virtual address
2. VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3.
4. // Form the address of the page-table entry (PTE)
5. PTEAddr = PTBR + (VPN * sizeof(PTE))
6.
7. // Fetch the PTE
8. PTE = AccessMemory(PTEAddr)
9.
10. // Check if process can access the page
11. if (PTE.Valid == False)
12.     RaiseException(SEGMENTATION_FAULT)
13. else if (CanAccess(PTE.ProtectBits) == False)
14.     RaiseException(PROTECTION_FAULT)
15. else
16.     // Access is OK: form physical address and fetch it
17.     offset = VirtualAddress & OFFSET_MASK
18.     PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19.     Register = AccessMemory(PhysAddr)
    
```

May 20, 2021 TCCS422: Operating Systems [Spring 2021] L15.93
 School of Engineering and Technology, University of Washington - Tacoma

COUNTING MEMORY ACCESSES

- Example: Use this Array initialization Code


```

int array[1000];
...
for (i = 0; i < 1000; i++)
    array[i] = 0;
            
```
- Assembly equivalent:


```

0x1024 movl $0x0, (%edi,%eax,4)
0x1028 incl %eax
0x102c cmpl $0x03e8,%eax
0x1030 jne 0x1024
            
```

May 20, 2021 TCCS422: Operating Systems [Spring 2021] L15.94
 School of Engineering and Technology, University of Washington - Tacoma

VISUALIZING MEMORY ACCESSES: FOR THE FIRST 5 LOOP ITERATIONS

- Locations:
 - Page table
 - Array
 - Code
- 50 accesses for 5 loop iterations

May 20, 2021 TCCS422: Operating Systems [Spring 2021] L15.95
 School of Engineering and Technology, University of Washington - Tacoma

Consider a 4GB Computer with 4KB (4096 byte) pages. How many pages would fit into physical memory?

$$2^{32} / 2^{20} = 2^{12} \text{ pages}$$

$$2^{32} / 2^{12} = 2^{20} \text{ pages}$$

$$2^{32} / 2^{16} = 2^{16} \text{ pages}$$

$$2^{32} / 2^8 = 2^{24} \text{ pages}$$

None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at polllev.com/app

For the 4GB computer example, how many bits are required for the VPN?

24 VPN bits (indexes
 2^{24} locations)

16 VPN bits (indexes
 2^{16} locations)

20 VPN bits (indexes
 2^{20} locations)

12 VPN bits (indexes
 2^{12} locations)

None of the above

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.7

For the 4GB computer example, how many bits are available for page status bits?

32 - 12 VPN bits
 = 20 status bits

32 - 24 VPN bits
 = 8 status bits

32 - 16 VPN bits
 = 16 status bits

32 - 20 VPN bits
 = 12 status bits

None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at poller.com/app

For the 4GB computer, how much space does this page table require? (number of page table entries x size of page table entry)

2^{20} entries x 4b = 4 MB

2^{12} entries x 4b = 16 KB

2^{16} entries x 4b = 256 KB

2^{24} entries x 4b = 64 MB

None of the above

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.9

For the 4GB computer, how many page tables (for user processes) would fill the entire 4GB of memory?

4 GB / 16 KB = 65,536

4 GB / 64 MB = 256

4GB / 256 KB = 16,384

4GB / 4MB = 1,024

None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at poller.com/app

PAGING SYSTEM EXAMPLE

- Consider a 4GB Computer:
 - With a 4096-byte page size (4KB)
 - How many pages would fit in physical memory?
- Now consider a page table:
 - For the page table entry, how many bits are required for the VPN?
 - If we assume the use of 4-byte (32 bit) page table entries, how many bits are available for status bits?
 - How much space does this page table require?
 # of page table entries x size of page table entry
 - How many page tables (for user processes) would fill the entire 4GB of memory?

May 20, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L15.101

QUESTIONS

