**CS 354 - Machine Organization & Programming**

# Tuesday, February 20, 2018

**Project p2 (6%):** DUE at 10 pm on SUNDAY, February 25th

**Homework hw2 (1.5%): DUE TOMORROW** at 10 pm on Wednesday, February 21st

**Homework hw3 (1.5%):** Assigned Friday

## Last Time

Virtual Address Space C Abstract Memory Model Where Do I Live?

Linux: Processes and Address Spaces

Meet the Heap

## Today

C’s Heap Allocator (stdlib.h)

Posix brk (unistd.h)

Allocator Design

Simple View of Heap

Free Block Organization

Implicit Free List

## Next Time

Memory Allocator Design

**Read:** B&O 9.9.7 - 9.9.11

# C’s Heap Allocator (stdlib.h)

**What’s in stdlib.h?**

## Allocator Functions

void \*malloc(size\_t size)

Allocates and returns generic ptr to block of heap memory of size bytes, or returns NULL if allocation fails.

void \*calloc(size\_t nItems, size\_t size)

Allocates, clears to 0, and returns a block of heap memory of nItems \* size bytes, or returns NULL if allocation fails.

void \*realloc(void \*ptr, size\_t size)

Reallocates to size bytes a previously allocated block of heap memory pointed to by ptr, or returns NULL if reallocation fails.

If (ptr ==NULL) return malloc(size);

Else if (size==0){free(ptr); return NULL;}

Else // try to reallocate

void free(void \*ptr)

Frees the heap memory pointed to by ptr. If ptr is NULL then does nothing.

Undfined: freeing freed heap memory. Freeing stack memory.

Void return type means free cant tell you if there is problem.

*For CS 354, If malloc/calloc/realloc returns NULL*

*just display and appropriate error message and then exit the program*

**Posix brk (unistd.h)**

**What is unistd.h?**

* Header file used with posix API functions
* IEEE CS standards that maintain compatibility among UNIX OS

## DIY Heap via Posix Calls

## *brk*

## *program break, ponter to top of heap*

## int brk(void \*addr)

Sets the top of heap to the specified address addr. Returns 0 if successful, else -1 and sets errno.

#Include<errno.h>

Printf(“error: %s”, strerror(errno));

When OS gives a program a null page of memory, it clears it out for security reasons

void \*sbrk(intptr\_t incr)

Attempts to change the program’s top of heap by incr bytes.

Returns the old brk if successful, else -1 and sets errno.

Its best to jusr stick with malloc … since the standard heap allocate is well implemented, and more portable.

# Allocator Design

**Goals**

*throughput*

*operations/SEC higher is better*

*free’s work is independent of number of heap blocks.*

*Malloc’s work is linearly depdent on number of heap blocks*

*memory utilization*

memory requested / total heap size

higher is better

trade off: typically increasing one, decreases the other

## Requirements

List the requirements of a heap allocator.

1. Must handle arbitrary sequences of requests

2. provides an immediate response

3. \* doesn’t move or change allocated blocks

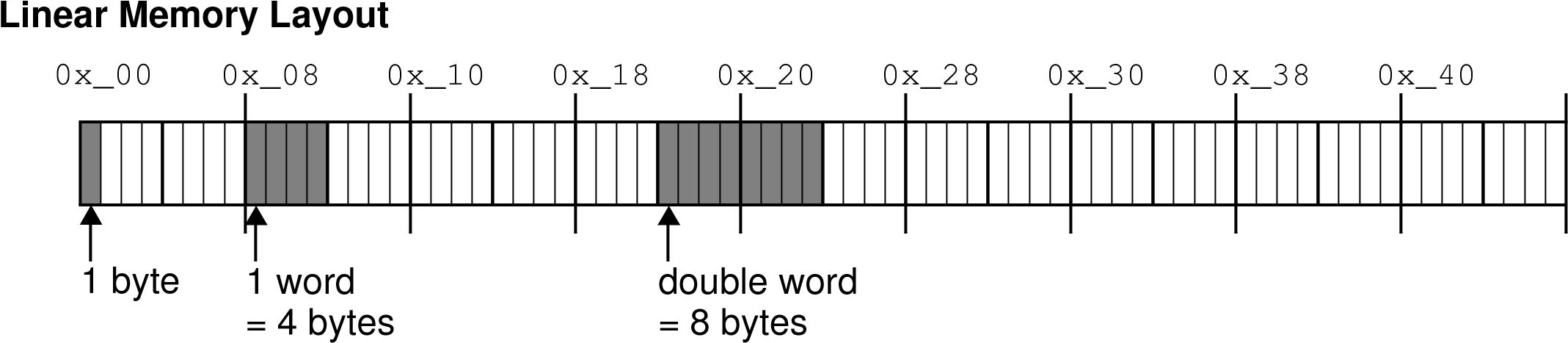
4. uses heap for allocation

5. follows alignment requirements

**Design Considerations**

* Free block organization
* Placement policy
* Splitting blocks to create a better fit
* Coalescing blocks to create a larger free block

# Simple View of Heap



*double word alignment*:

## Heap Allocation Run 1

0x\_00 0x\_08 0x\_10 0x\_18 0x\_20 0x\_28 0x\_30 0x\_38 0x\_40

Update the diagram to show the following heap allocations:

1. p1 = malloc(2 \* sizeof(int));
2. p2 = malloc(3 \* sizeof(char));
3. p3 = malloc(4 \* sizeof(int));
4. p4 = malloc(5 \* sizeof(int));

allocation must be the multiple of 8!!

What happens with the following heap operations:

1. free(p1); p1 = NULL;
2. free(p3); p3 = NULL;
3. p5 = malloc(6 \* sizeof(int)); Return Null indicating allocation Failed

*External Fragmentation*:

When there is enough free MEM in heaps but allocation fails because its been divided into smaller , non-continuous blocks

*Internal Fragmentation*:

Memory inside a block that is overhead, E.G, padding

**Explicit Free List**

# Free Block Organization

How does allocator know which parts of the heap are free and which are allocated?

Simple view doesn’t track a block’s status or size

Allocates need to track this.

How does allocator know how big each block is?

## Explicit Free List

A structure is used to store references to free blocks as well as their size

0x\_00 0x\_08 0x\_10 0x\_18 0x\_20 0x\_28 0x\_30 0x\_38 0x\_40

**Space is needed for the free list**

**Malloc’s work is linearly dependent on once the number of free blocks**

**Implicit Free List**

No separate structure, instead status and size are stored with each block.

+Allocator is simpler to go. - malloc’s work is linearly dependent on the total number of heap blocks.

# Implicit Free List

Each block starts with a header contains block size and status

## Basic Heap Block Layout

31 3|210 bits 31 24 16 8 0

A

D

C

B

D

C

B

A

Header = block size |

a

00

Payload

Possibly More Payload

Possibly Padding

3

1

2

4

1

2

3

4

## Size and status is stroed in 4 bytes of header

Status is a bit where 1

Size is the remaining value ignoring the last three bits

Int value for Free block of size 8 bytes -> 8

Allocated block 16 -> 17

## Heap Allocation Run 2 with Block Headers

0x\_00 0x\_08 0x\_10 0x\_18 0x\_20 0x\_28 0x\_30 0x\_38 0x\_40

Update the diagram to show the following heap allocations:

1. p1 = malloc(2 \* sizeof(int));
2. p2 = malloc(3 \* sizeof(char));
3. p3 = malloc(4 \* sizeof(int));
4. p4 = malloc(5 \* sizeof(int));

Why does it make sense that Java doesn’t allow primitives on the heap?