Dynamic Memory Allocation

CSE 220: Systems Programming

Ethan Blanton & Carl Alphonce

Department of Computer Science and Engineering

University at Buffalo

Dynamic Memory Allocation

We have discussed two kinds of memory allocation:

- Static allocation
  - Global variables
  - Static local or global variables
- Dynamic allocation
  - Automatic variables
  - Manually allocated memory

We covered automatic variables in depth, now it’s time for manual allocations!

The Dynamic Allocator

The interface to the dynamic allocator is `malloc()` *et al.*

However, the underlying mechanism is more complex.

The operating system kernel provides only large allocations.

Its minimum allocation on x86-64 is typically 4 KB.

The dynamic allocator must efficiently parcel out these allocations.

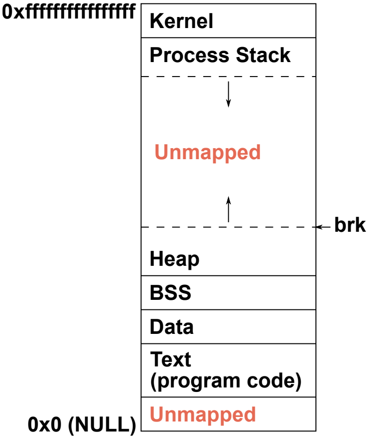
The System Break

The OS heap occupies the memory above the BSS.

To the OS, it is one large block of memory.

The dynamic allocator must manage it.

The OS provides one tool for this: the system break.



Managing the System Break

The system break marks the boundary of the heap.

It is the address of the first byte that *isn't* on the heap.

It can be moved with two system calls:

- `brk()`: set the break to an address
- `sbrk()`: move the break a relative number of bytes

A dynamic allocator can use this to request memory from the OS.

sbrk()

The sbrk() system call moves the system break:

```
void *sbrk(intptr_t increment);
```

It returns the old location of the system break.

A positive break value expands the heap.

This means sbrk() works a little bit like malloc:

```
void *mem = sbrk(size);
```

Allocation API

The original Unix allocator required explicit sizes: both allocating and freeing memory took a size.

The `malloc()` allocator does not.

This means that it must store that size somewhere!

There are many allocation strategies with different solutions to this problem.

Metadata

Metadata[1] is stored for heap allocations.

This metadata allows for:

- Identifying available memory on the heap
- Determining the size of allocated memory for `free()`
- Locating regions of memory that make up the heap
- …

How this metadata is stored and managed can vary.

[1] Metadata is data about data.

Allocation Blocks

Assume that the dynamic allocator allocates blocks of memory.

Each block contains:

- Any metadata that is required for the allocator
- Memory available to the user to serve an allocation

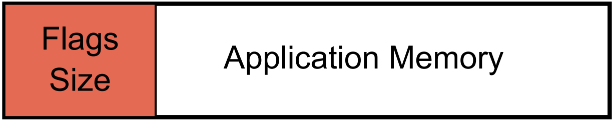
The set of all of these blocks makes up the heap.

Explicit Metadata

When explicit metadata is in use, the block might contain:

- An integer containing the size of the block
- A flag indicating whether it is free or in use

This data is stored adjacent to, or nearby, the user memory:¶



Heap Block

Free Lists

A common allocation management technique is:

- Blocks containing explicit sizes
- Free blocks placed on a linked list

Sometimes allocated blocks may also be on a list.

When the user asks for memory, available memory is located on this list.

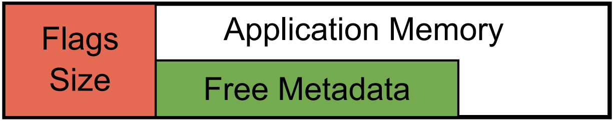
Sharing Space

Sometimes metadata is required only when a block is free.

This metadata can be stored inside the application memory portion of the block.

Since the block is not in use, this memory is available.

This reduces the overhead of free memory blocks.



Overhead

Allocators have overhead.

This is extra memory used only by the allocator.

It is important to minimize this overhead.

There are two primary sources of overhead:

- Metadata
- Fragmentation

Fragmentation

Fragmentation is space used by the allocator that is not useful to the application.

Sometimes metadata is included in fragmentation.

There are two kinds of fragmentation:

- Internal: unused memory inside a heap block
- External: unused memory between heap blocks

Internal Fragmentation

Internal fragmentation is like packing in a structure.

It is memory that is required by the allocator, but not useful.

It often arises because allocator blocks either:

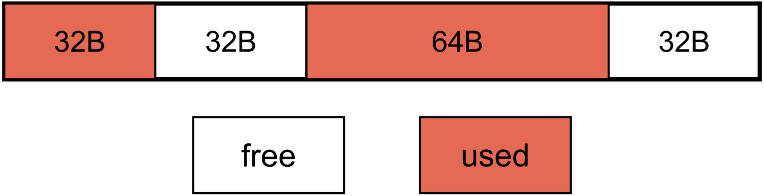
- Must be aligned
- Have limited possible sizes

For example: an allocator only creates blocks of size 2^k, but the user asks for $2^k - 1$ bytes of memory.

External Fragmentation

External fragmentation is due to user allocation patterns.

The allocator has free blocks, but they are not suitable.



What if:

- this is the entire heap
- the user wants a 64B block?

Exploration

Let's explore some of these concepts in diagrams.

Summary

- The OS notion of the heap is very simplistic.
- The dynamic allocator has to manage the heap.
- Metadata is required for management.
- The heap can become fragmented:
    - Internal fragmentation is inside heap blocks.
    - External fragmentation is between heap blocks.

References I

Required Readings

[1] Randal E. Bryant and David R. O'Hallaron. *Computer Science: A Programmer's Perspective*. Third Edition. Chapter 9: 9.9, 9.11. Pearson, 2016.

License