# Malloc Presentation

ACU 2025 Team

# Copyright



**Rules**

- You must have downloaded your copy from the Assistants' Intranet <https://intra.forge.epita.fr>.
- This document is strictly personal and must **not** be passed on to someone else.
- Non-compliance with these rules can lead to severe sanctions.

# Table of content

# Table of content

1. Introduction
2. Malloc 101
3. System-side memory management
4. Algorithms
5. Improvements
6. Survival kit
7. Submission

# Introduction

# A user/system interface

- `malloc` is a *wrapper* around system-provided functionalities for memory management.
- It asks the system to reserve memory then uses various algorithms to allocate blocks for the user within that memory.

# A user/system interface

**Memory allocation functions**

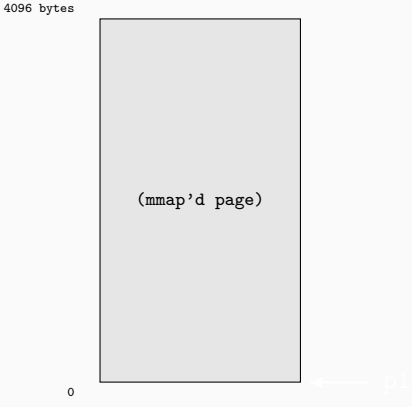
- `malloc`
- `free`
- `realloc`
- `calloc`

**Memory management**

- `sbrk`
- `mmap`

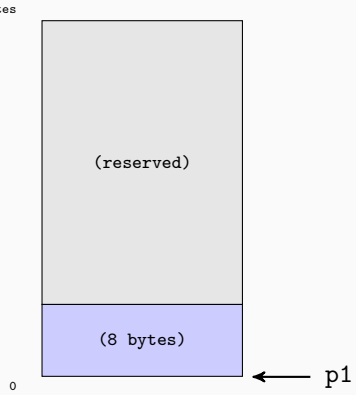
# Malloc 101

# Example



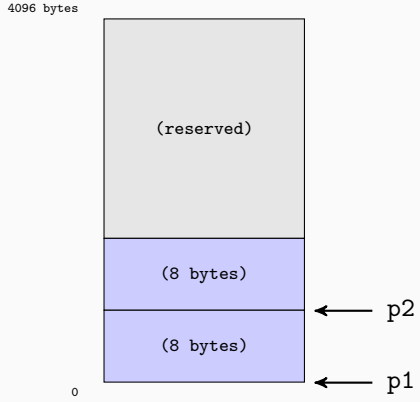
# Example

```
void *p1 = malloc(8);
```



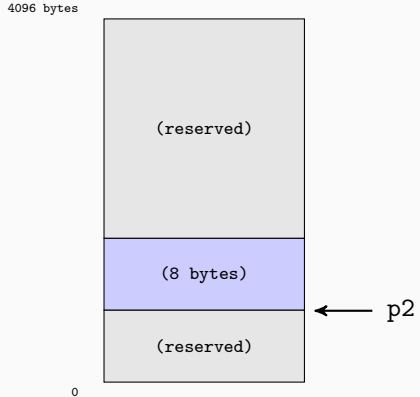
# Example

```
void *p1 = malloc(8);
void *p2 = malloc(8);
```



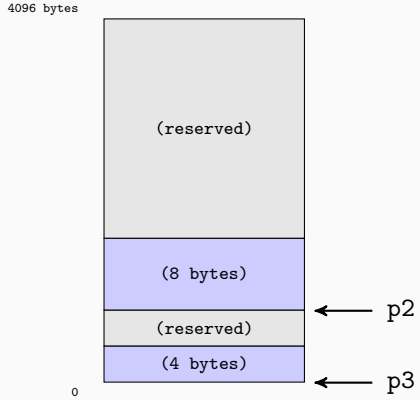
# Example

```
void *p1 = malloc(8);
void *p2 = malloc(8);
free(p1);
```



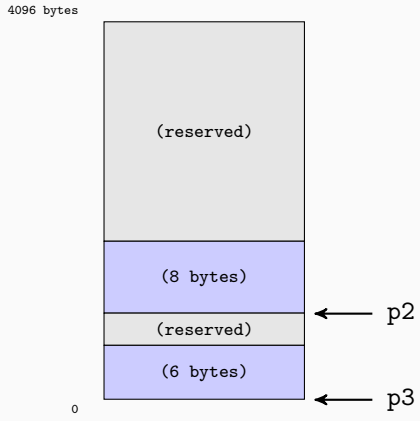
# Example

```
void *p1 = malloc(8);
void *p2 = malloc(8);
free(p1);
void *p3 = malloc(4);
```



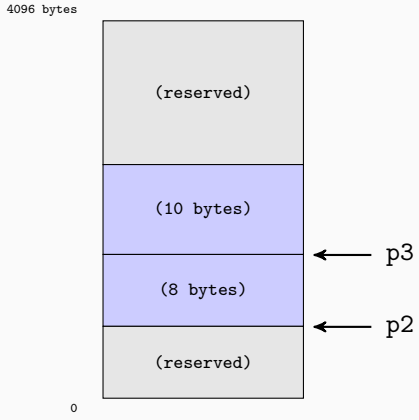
# Example

```
void *p1 = malloc(8);
void *p2 = malloc(8);
free(p1);
void *p3 = malloc(4);
p3 = realloc(p3, 6);
```



# Example

```
void *p1 = malloc(8);
void *p2 = malloc(8);
free(p1);
void *p3 = malloc(4);
p3 = realloc(p3, 6);
p3 = realloc(p3, 10);
```



# System-side memory management

# Dynamic memory allocation

Two ways of getting memory from the system:

- `sbrk(2)`
- `mmap(2)`

# sbrk (2)

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

*Change the data segment size.*

- Interface created to match *segmented memory management* systems available on most CPUs.
- Legacy interface, we will not use it.

# mmap(2)

```
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);
```

Map pages to a virtual address

- `addr`: starting address (hint)
- `length`: number of bytes to map
- `prot`: permissions (`PROT_READ`, `PROT_WRITE`, ...)
- `flags`: options (`MAP_PRIVATE`, `MAP_ANONYMOUS`, ...)
- `fd`: descriptor of file to be mapped; -1 with `MAP_ANONYMOUS`
- `offset`: starting offset in the mapped file

# mmap(2)

- Do not forget to define the right feature test macros to be able to use all the flags. For more information, see `man 2 mmap`.
- Be careful, you **MUST** check mmap return value (see `MAP_FAILED` macro defined in `sys/mman.h`).

# Memory Mapping

- You can use `mmap` (2) to map plain memory, thus reserving memory for the process. This is called an *anonymous mapping*.
- `mmap`'ed memory can be released with `munmap` (2) and resized using `mremap` (2).
- A lot of *syscalls* related to memory management: see `sys/mman.h`.

## Example

```c
#include <stddef.h>
#include <sys/mman.h>

void *my_get_page(void)
{
    void *addr = mmap(NULL, 4096, PROT_READ | PROT_WRITE,
                      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (addr == MAP_FAILED)
        return NULL;
    return addr;
}
```

# Algorithms

# Metadata

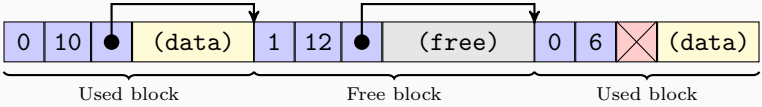
- `malloc` returns blocks matching the required size.
- `mmap` allocates an empty memory zone.

$\Rightarrow$ To manage these memory zones, **metadata is needed**.

# Metadata

Metadata usually includes:

- Block state
- Block size
- Pointer to the next block



# Strategies

- Several allocation strategies, each with its own features.
- What matters:
  - Simplicity of implementation
  - Allocation speed
  - Memory efficiency (little fragmentation)
  - Portability

# Strategies: Forbidden

Doing `malloc = mmap` is not an algorithm and will result in a 0 !

# Implementation

**Recommended**

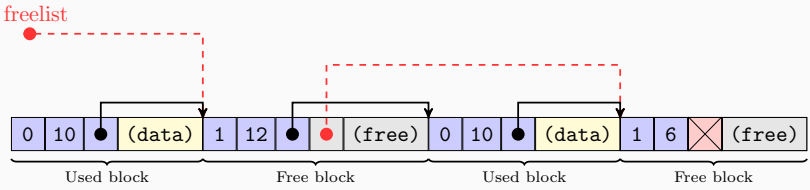
- Linked-List
- Bit Bucket

**Advanced**

- Binary Buddies
- Slab Allocator
- ...

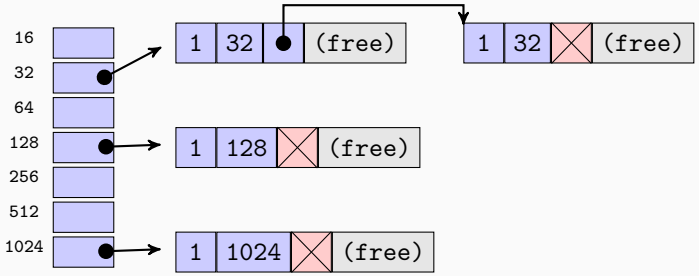
# Improvements

# Free-list

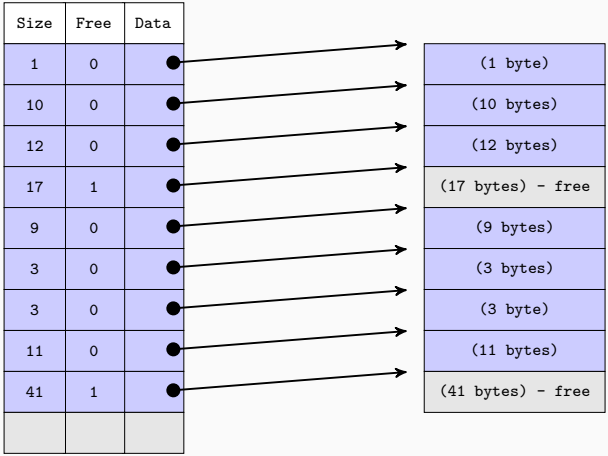



- Allocation is faster since we only search through free blocks.

# Sized free-lists



# External metadata



# Survival kit

# Using your allocator

Your memory allocation functions should be exported in a `libmalloc.so` shared object.

How to use your malloc:

- Link against `libmalloc.so` and use `LD_LIBRARY_PATH`:

```
42sh$ gcc main.c -L. -lmalloc -o main
42sh$ LD_LIBRARY_PATH=. ./main
```

- When you want to use your malloc with external programs, just use the `LD_PRELOAD` environment variable:

```
42sh$ LD_PRELOAD=./libmalloc.so ls
```

# Testing your allocator

- We will use LD_PRELOAD to run several existing programs with your allocator.
- Be sure to test it thoroughly by doing the same.

We want to see you use a testsuite. You should create your testsuite early on to avoid regression. A shell script is acceptable and is enough for this project. But we want multiple and different tests.

# Testing your allocator

- Testing is important and at this time of the year it's a skill you must have.
- If you don't have any test, we won't debug you!

## Debugging your allocator

- You want the debugged program to use your library but you do not want gdb to do so as gdb will make some allocations (they may fail if your library is buggy). That's why you cannot launch gdb as shown previously.
- You have to start gdb normally and then set the debugged program environment using gdb commands to ensure that it will use your library.

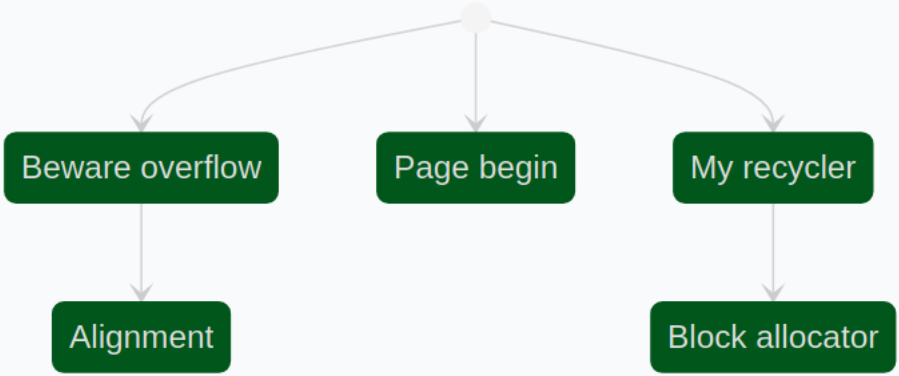
```
42sh$ gdb ./main
Reading symbols from ./main...
(gdb) set env LD_LIBRARY_PATH=.
(gdb) start
```

**Note:** Do not forget to compile your binary and your library with `-g`.

## Debugging other programs with your library preloaded

```
42sh$ gdb ls
Reading symbols from ls...
(No debugging symbols found in ls)
(gdb) set exec-wrapper env 'LD_PRELOAD=./libmalloc.so'
(gdb) start
```

# Exercises




**Note:** Malloc can only be accessed after all the exercises have been completed!

## Given binaries

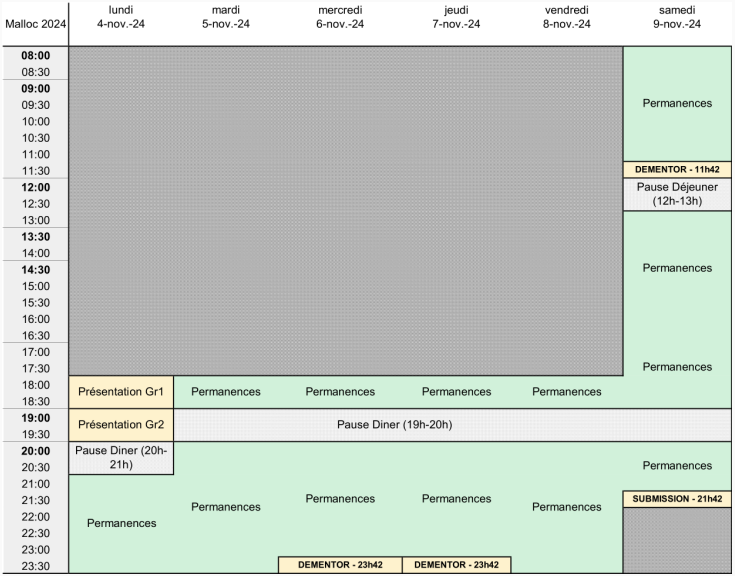
- corruptionproof: check external metadata
- memoryfootprint: check memory optimization
- speed: check speed optimization

# Miscellaneous advice

- Start **early**.
- Read the **entire** subject *AND* the man pages.
- Do not spend too much time on complicated algorithms.
- **Test**, test and re-test.

# Planning

# Planning


Malloc 2024
lundi 4-nov.-24
mardi 5-nov.-24
mercredi 6-nov.-24
jeudi 7-nov.-24
vendredi 8-nov.-24
samedi 9-nov.-24
08:00
08:30
09:00
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
17:00
17:30
18:00
18:30
19:00
19:30
20:00
20:30
21:00
21:30
22:00
22:30
23:00
23:30
Permanences
DEMENTOR - 11h42
Pause Déjeuner (12h-13h)
Permanences
Permanences
Présentation Gr1
Permanences
Permanences
Permanences
Permanences
Présentation Gr2
Pause Diner (19h-20h)
Pause Diner (20h-21h)
Permanences
Permanences
Permanences
Permanences
Permanences
SUBMISSION - 21h42
Permanences
DEMENTOR - 23h42
DEMENTOR - 23h42

# Submission and dementors

**Dementor 1:** November 06, 23:42

**Dementor 2:** November 07, 23:42

**Dementor 3:** November 09, 11:42

**Submission:** November 09, 21:42

# Recap

**Newsgroup** `Assistants ING - Projets`
**Tag** `[MLL]`
**Deadline** November 09, 21:42

As usual:

- Your project must comply with the coding style.
- Cheating will be sanctioned.

# Questions

Any Questions ?