Exercise — Block Allocator

version **#bfdaecd01cbf44dfbd7800b940343997d9ad7d73**

Assistants C/Unix 2025 <assistants@tickets.assistants.epita.fr>

Copyright

Contents

*https://intra.forge.epita.fr

File Tree

```
block_allocator/
├── allocator.c  (to submit)
├── allocator.h
├── main.c
├── utils.c
└── utils.h
```

Authorized functions : You are only allowed to use the following functions

- calloc(3)
- free(3)
- malloc(3)
- putchar(3)
- sysconf(3)

Authorized headers : You are only allowed to use the functions defined in the following headers

- err.h
- errno.h
- assert.h
- stddef.h
- sys/mman.h

Compilation : Your code must compile with the following flags

- -std=c99 -pedantic -Werror -Wall -Wextra -Wvla

1 Definition

A block allocator is a structure used to allocate and manage memory. When you ask memory from the block allocator, it will allocate a bunch of pages ready to be used. After the usage, the allocator must be able to de-allocate memory without leaving traces.

To manage the different sizes of memory that can be requested, the allocator uses metadata to store information about blocks. In our case, metadata will be placed at the beginning of each block of allocated pages.

Tips

The word “metadata” refers to “data on some other data”. In this exercice, we will store several informations on our allocated blocks, so we can manipulate them more conveniently.

2 Goal

In this exercise you have to implement this allocator. To help you, the block structures are already given.

`blk_allocator` is a structure that only holds a list. This list will chain every allocated metadata. That way, holding a unique reference to the The `blk_allocator` structure is the linked list of every allocated block. The `blk_allocator` structure holds a pointer to the head of the linked list of the blocks' associated metadata, as the blocks are chained together using their metadata. The structure has to be allocated with the traditional `malloc(3)`.

```
struct blk_allocator
{
    struct blk_meta *meta;
};
```

`blk_meta` is the metadata for a given block. As defined, it will be stored at the beginning of every block of allocated pages. It contains:

- a `next` used to chain metadata blocks together. The `next` metadata will be in a different allocated page.
- a `size` corresponding to the remaining size of the page, (i.e. the size of the `data` field).
- a `data` field representing the memory requested on allocation. It is a flexible array member, which does not have a specified size and thus must be at the end of the structure.

```
struct blk_meta
{
    struct blk_meta *next;
    size_t size;
    char data[];
};
```

Tips

For every function that you have to implement, you can consider that arguments are always valid. Do not waste time checking for `NULL` arguments.

3 Block allocator new

- **Authorized functions:** `malloc(3)`, `calloc(3)`

The function must allocate and return a new `blk_allocator` structure. The meta inside the structure must also be set to *NULL*.

```
struct blk_allocator *blka_new(void);
```

4 Block allocator allocate

- **Authorized functions:** `mmap(2)`, `sysconf(3)`

The aim of this function is to allocate a `blk_meta` structure that will hold at least `size data` given as parameter. The field `size` in the metadata must be set according to the size of the page(s) without metadata. The new `blk_meta` must be inserted at the head of the metadata chain of the given `blk_allocator`, and returned.

```
struct blk_meta *blka_alloc(struct blk_allocator *blka, size_t size);
```

Tips

For a better understanding of `mmap(2)` you can check out the malloc presentation slides. In order to be able to use the right `mmap(2)` flags, you must compile with `-D_DEFAULT_SOURCE`. Do not hardcode the size of a page, when you can easily get it with `sysconf(3)`.

5 Block allocator free

- **Authorized functions:** `munmap(2)`

The purpose of this function is just to de-allocate memory of the given `blk_meta` structure. You must not do anything else.

```
void blka_free(struct blk_meta *blk);
```

6 Block allocator pop

The purpose of this function is to de-allocate memory of the first `blk_meta` structure of the `blk_allocator`. Do not forget to update the chain of metadata pointers.

```
void blka_pop(struct blk_allocator *blka);
```

7 Block allocator delete

- **Authorized functions:** `free(3)`

The `blk_allocator` allocated previously has to be freed. This is the purpose of this function. You may also pay attention to the elements of the list and release memory if needed.

```
void blka_delete(struct blk_allocator *blka);
```

8 Example

For this exercise, we provide a `main` function and a `utils.c` file for your tests. Read them, use them and add your own tests. The main is only here to help you debug, it does not mean you will pass our tests. Here is an example using our main:

```
42sh$ ls
allocator.c  allocator.h  main.c  Makefile utils.c  utils.h
42sh$ make
gcc -pedantic -Werror -Wall -Wextra -Wvla -std=c99 -D_DEFAULT_SOURCE  -c -o allocator.o␣
↪allocator.c
gcc -pedantic -Werror -Wall -Wextra -Wvla -std=c99 -D_DEFAULT_SOURCE  -c -o main.o main.c
gcc -pedantic -Werror -Wall -Wextra -Wvla -std=c99 -D_DEFAULT_SOURCE  -c -o utils.o utils.c
gcc   allocator.o main.o utils.o   -o allocator
42sh$ ./allocator
Hello world!
You are doing a great job if you see this message correctly.
But do not forget to release memory.
42sh$
```

Seek strength. The rest will follow.