Assessed Exercise 1: Uxntal and Uxn

Part A: Uxntal programming exercise: dynamic memory allocation

Aim

In this coursework you will create a mechanism for dynamic memory allocation in Uxntal, similar to the calls `malloc()` and `free()` in C. What this means is that, if you need a certain amount of memory `n_bytes`, calling `ptr = malloc(n_bytes)` should return a pointer `ptr` to the start of an area of at least that size. When you call `free(ptr)` the memory pointed to by `ptr` should be freed up so it can be reused. In Uxntal this could for example look as follows:

```
#10 malloc ;ptr1 STA2 ( allocate 16 bytes )
( use ptr1 )
#20 malloc ;ptr2 STA2 ( allocate 32 bytes )
( use ptr2 )
;ptr1 LDA2 #10 free ( free 16 bytes )
#40 malloc ;ptr3 STA2 ( allocate 64 bytes )
( use ptr3 )
;ptr2 LDA2 #20 free ( free 32 bytes )
;ptr3 LDA2 #40 free ( free 64 bytes )
```

In C, the `free()` call only needs the pointer, because the size of the allocated memory is handled internally. For this exercise, we use a version of `free` where you provide the number of pages to deallocate because it is simpler to implement.

In practice, a part of the Uxn VM's memory must be divided into chunks of a fixed size, typically called pages. Depending on how many bytes an allocation needs, one or more pages may be needed, and it is the assumption for this exercise that they are contiguous.

You need datastructures to keep track of the free pages and the pages in use. A very common datastructure is a bitmap where every bit indicates if a page is free or allocated, and this is what you should use for the exercise.

The problem with allocation and deallocation of contiguous blocks of pages is that the memory gets fragmented: even thought there may be enough free memory in total, there might not be a large enough block left for a given allocation. In this exercise, you do not need to address this problem.

Suggested approach

Creating and manipulating a bitmap A bitmap is simply an array of bytes which represents the array of pages that makes up the dynamically allocatable memory. A `0` bit means the page is free, a `1` means it's claimed. Given a page index, you need to find the byte in which it occurs, and the find the bit in that byte. You then need to access (get, set or clear) this individual bit. For that,

you need to use bit masking and shifting operations. These are the operations `&`, `|`, `<<` and `>>` in Python, in Uxntal the instructions are `AND`,`ORA` and `SFT`.

Allocating and de-allocating a number of bytes

- Allocating a number of bytes means first working out how many pages are needed, and then look for this number of contiguous pages in memory, using the bitmap. The result of the allocation is that the bitmap contains a contiguous sequence of `1` bits.
- Dellocating a number of bytes means setting all the corresponding bits in the bitmap to `0`.

Python reference implementation

I provide a Python reference implementation of `malloc()` and `free()` as defined in this exercise. This serves as the functional specification. I also provide an Uxntal code skeleton with a suitable structure and helper functions for printing.

Your task

Your task is:

- to create an equivalent Uxntal implementation which assembles and runs correctly. You don't have to follow the same code structure as the Python reference, but the functional behaviour must be the same, and the Uxntal subroutines must have the following signature:

  ```
  <size in bytes> malloc
  <16-bit address> <size in bytes> free
  ```

  - You must define `PAGE_SZ` , `N_PAGES` and `VMEM_START` as constants in the program, this has been done in the skeleton code.
  - `malloc` returns a pointer to the allocated area, i.e. a 16-bit unsigned number which is a valid address in Uxntal main memory, or `0` if the allocation failed.
  - `free` always succeeds and returns nothing
  - You *must* use a bitmap with the following low-level API

    ```
    <byte-sized index> get_bit ( returns 1 or 0 )
    <byte-sized index> set_bit ( sets the bit to 1 )
    <byte-sized index> clear_bit ( clears the bit to 0)
    ```

- to provide a unit test program with unit tests for every subroutine used in your program. I provide a Python reference implementation and an Uxntal implementation which does not contain the actual implementations.

- to provide an integration test program demonstrating that your `malloc()` and `free()` work as expected. I provide a Python reference implementation which is not complete at all. "As expected" means:
  - if `malloc` requests `n` bytes and there is enough contiguous memory to claim, returns the pointer to the start of this memory area
  - if `malloc` requests `n` bytes and there is not enough contiguous memory to claim, returns `0`
  - if `free` takes a valid pointer (i.e. returned by a succesful malloc) and the corresponding size, the corresponding contiguous memory area will be deallocated
  - if `free` takes an invalid pointer (i.e. from a succesful malloc), the behaviour is undefined and does not need to be verified
  - if `free` takes a valid pointer but incorrect size, the behaviour is undefined and does not need to be verified
  - allocation and deallocation should work correctly for any size between `1` and `PAGE_SZ*N_PAGES-1` bytes

Marking scheme

Total marks out of 50:

- [2 marks] Identifying information (in the form of comments at the beginning of the program). The first comments identify the program, giving your name and student ID, and saying what the program does. These may be the easiest marks you'll ever get!
- [3 marks] Your status report. State clearly whether the program works. If parts are not working, say so.
- [5 marks] The bitmap and its low-level API: get-bit, set-bit, clear-bit (Uxntal code)
- [5 marks] Unit tests for the bitmap and its low-level API: get-bit, set-bit, clear-bit (Uxntal code)
- [5 marks] The `malloc()` routine (including any auxiliary routines) (Uxntal code)
- [5 marks] Unit tests for the `malloc()` routine (including any auxiliary routines) (Uxntal code)
- [5 marks] The `free()` routine (including any auxiliary routines) (Uxntal code)
- [5 marks] Unit tests for the `free()` routine (including any auxiliary routines) (Uxntal code)
- [5 marks] The code assembles and executes
- [10 marks] An integration test program demonstrating correct functionality of `malloc` and `free`.

Due date

12 April 2024 via the GA Workbook submission link