Introduction

Page Frames

This Lab continues preparation for building a full simulation of how an OS manages virtual memory. Virtual memory systems divide memory into fixed-size blocks, called page frames. We'll use 65536 = 0x10000 bytes as the page frame size. Each process will keep track of page frames that are allocated to it (for this Lab, you'll only simulate one process, but think ahead to future assignments where multiple processes will share the memory).

For this assignment, we'll use a vector<uint8\_t> variable to simulate system memory, as in Lab 2. Memory addresses are indices into this vector. In a future assignment, we'll replace this vector with an interface to a simulated memory management unit.

Free Page Frames

The free list should be maintained as a linked list of page frames. Links are page frame addresses (links must be to the start of page frames, multiples of the page frame size). Links are store in the page frames; you shouldn't maintain a separate linked list structure. Each page frame on the free list has, in its first 4 bytes, the address of the next page frame on the list. The last page frame on the free list has link 0xFFFFFFFF. The head "pointer" is just the address of the first page frame in the list (0xFFFFFFFF if the list is empty). This format allows a page frame to be allocated from the free list in constant time, and also allows the return of a page frame to the free list in constant time.

Allocated Page Frames

The OS also needs to keep track of page frames allocated to each process. Since the page contents are controlled by the running process, the OS needs to keep track of the pages elsewhere. For this assignment, use a std::vector<uint32\_t> to store all the page addresses allocated to the single process.

Reserved Memory Area for Memory Allocation

The first page frame (starting at address 0) should always be reserved for use by the memory allocator.

Programming Task

MemoryAllocator class

Create a class named MemoryAllocator. This class will manage the allocation and deallocation of the page frames. No input or output should be done within the class.

Class Data Members

Your class should not use any global variables for memory allocation. The only variable data member in your MemoryAllocator class should be the vector<uint8\_t> containing the memory. Other data used by the memory allocator must be stored in page frame 0 of memory, not in class member variables (you may define class-member static const values specifying the address of each variable value in page frame 0):

* page\_frames\_total (32 bits):
  + A count of the total number of page frames in memory (memory size divided by page frame size)
* page\_frames\_free (32 bits):
  + The current number of free page frames
* free\_list\_head (32 bits):
  + The page frame address of the first page frame in the free list (0xFFFFFFFF if list empty)

You may copy these values temporarily to local variables within class member functions while you are working with them.

Constructor

The constructor should accept a single argument, the number of page frames in the memory. It should resize the memory vector to the number of page frames multiplied by the page frame size. The constructor should then build the free list consisting of all page frames in memory except page 0, which is reserved for use by the memory allocator. The free list should start with page frame 1 (address 0x10000), linked to page frame 2 (0x20000), etc., up to the last page frame, which will have its link set to 0xFFFFFFFF. The constructor should initialize the other data in page 0 as appropriate.

There should be no default constructor, and other constructors and assignments should be defined such that move and copy are not allowed. An empty destructor should be supplied.

Allocator

Define the public class method specified as

bool AllocatePageFrames(uint32\_t count, std::vector<uint32\_t> &page\_frames);

This method takes as its first argument a count of the number of page frames to allocate from the free list. The method should push the addresses of all the allocated page frames onto the back of the vector page\_frames specified as the second argument. If the number of free page frames is less than the count argument, then no page frames should be allocated, and the method should return false. If the page frames are successfully allocated, the method should return true. The contents of all bytes of allocated page frames should be set to 0.

Deallocator

Define the public class method specified as

bool FreePageFrames(uint32\_t count, std::vector<uint32\_t> &page\_frames);

The last *count* page frame addresses from the vector *page\_frames* should be returned to the free list. These page frame addresses should be popped from the back of the *page\_frames* vector as they are returned to the free list. Returns true if *count* <= *page\_frames*.size(), otherwise returns false without freeing any page frames.

Accessing Data Members

The memory vector class data member should be private. You should supply public member functions to return values needed by users of this class (these are sometimes called "getters"). For example, to access the current number of free page frames, in the .h file you could define the public class member function

uint32\_t get\_page\_frames\_free() const;

Other Class Member Functions

You may define other private class function members as needed.

Main Program

Your main program should read input from the file specified by the first command line argument. Output should be written to standard output.

Input File Format

The first line of the input file will contain a single hexadecimal integer specifying the total number of page frames (including page frame 0, which is reserved for use by the memory allocator). Your program should read this number from the input file and use the value to construct a MemoryAllocator object.

Subsequent lines in the input file will contain one or two hex numeric values:

1. The first value on each line will be a 1 to allocate page frames, 0 to deallocate page frames, 2 to print the contents of the free list
2. The second hex value on each line will be a count of page frames to allocate or deallocate (not present when first value = 2)

Your main program should read the input file a line at a time, and process the allocation or deallocation specified by each line in sequence. You don't need a separate "Process" class for this assignment (although you may define one if desired).

Output Format

As you read each input file line (including the first line), write the line to output, preceded by the '#" character.

After processing each allocate or deallocate request, write a line in the following fomat:

*S FC*

The first character of the output line should be a blank. The *S* value should be T if Allocate returned true, and F if Allocate returned false. The *FC* value should be the hex value of the number of free page frames after the Allocate call is made.

For print free list, write a blank at the beginning of the line, followed by the hex numbers of all the pages in the free list, separated by spaces.

Example Input and Output

Two sample input files (.txt files) and corresponding output files (.out files) can be found in [Lab3Data.zip](https://canvas.du.edu/courses/65842/files/3739198/download?verifier=aQ2vTKtHtzb4XblmYwYHv7XE8ocjW5Es4OXCwjFh&wrap=1).

Here's sample input allocator1.txt:

5

2

1 1

2

0 1

2

1 5

2

1 1

1 2

0 6

0 7

0 1

0 1

0 1

2

0 1

0 1

0 1

1 6

2

The corresponding output should look like this:

#5

#2

10000 20000 30000 40000

#1 1

T 3

#2

20000 30000 40000

#0 1

T 4

#2

10000 20000 30000 40000

#1 5

F 4

#2

10000 20000 30000 40000

#1 1

T 3

#1 2

T 1

#0 6

F 1

#0 7

F 1

#0 1

T 2

#0 1

T 3

#0 1

T 4

#2

10000 20000 30000 40000

#0 1

F 4

#0 1

F 4

#0 1

F 4

#1 6

F 4

#2

10000 20000 30000 40000