COMP3430 **Assignment 4** Due: April 6th, 2018 @11:59PM

Assignment Objectives:

- *To learn something about the implementation of basic free space management (which is applicable to both RAM and persistent storage (such as hard disks) but which will be limited to RAM in this assignment.*
- *To improve your understanding of how paged virtual memory systems work.*
- *To improve your understanding of i-node based disk space management.*

Assignment Instructions:

- *You must hand in your assignment electronically via UMLearn using the "Assignments" functionality under the "Assessments" drop-down menu. A folder for Assignment 4 has been created there. You can hand in multiple files if so desired but we would ask that you be kind to the markers and hand in everything in one batch at the end if at all possible. Recall that you must agree to the online honesty document before the dropbox will become visible. You can read back in the announcements on UMLearn for details on how to do this though you should have already done this to hand in you first lab.*
- *In general, for written questions PDF (.pdf), Word (.docx), PowerPoint (.pptx), Rich Text Format (.rtf) and Text (.txt) format files are acceptable. Handwritten then scanned PDF files are **not** acceptable. For any programming questions, please hand in all **source** files and any requested results as well as a Makefile to generate the executable file(s) for the target platform. You **must** also include a text file named 'README' for each programming question that briefly describes how to use your program.*
- *Your programs must run on the Linux lab machines, but you can try to develop your code on other platforms if you like though this is **not** recommended. In some cases, this may simply not be possible since your running programs will be making Linux system calls. Should you choose to use different platforms for development, you **alone** will be responsible for dealing with any differences in the compilers, tools, etc. that you may encounter. In such a case, be sure to leave plenty of time to port your code to Linux. Normally this takes less than 30 minutes but sometimes it may take several hours if your code is still a little buggy. (For example, the C compiler on some systems sets all uninitialized memory to zeroes. Hence, bad pointers are treated as NULLs and thus, sometimes incorrect code works there but not with other compilers and operating systems such as gcc on Linux.)*
- *Note that assignments are to be done **entirely independently** unless otherwise explicitly stated and that inclusion of materials from online sites is entirely forbidden.*

Assignment Questions:

[10] **1**. You are to extend the simple free list-based memory management system written in C provided with this assignment on the course homepage. Your system should extend the basic first-fit algorithm code to support free space **amalgamation** (i.e. when an area is freed that is adjacent to other free areas, they should be combined into a single larger free area). The provided memory management system provides three functions:

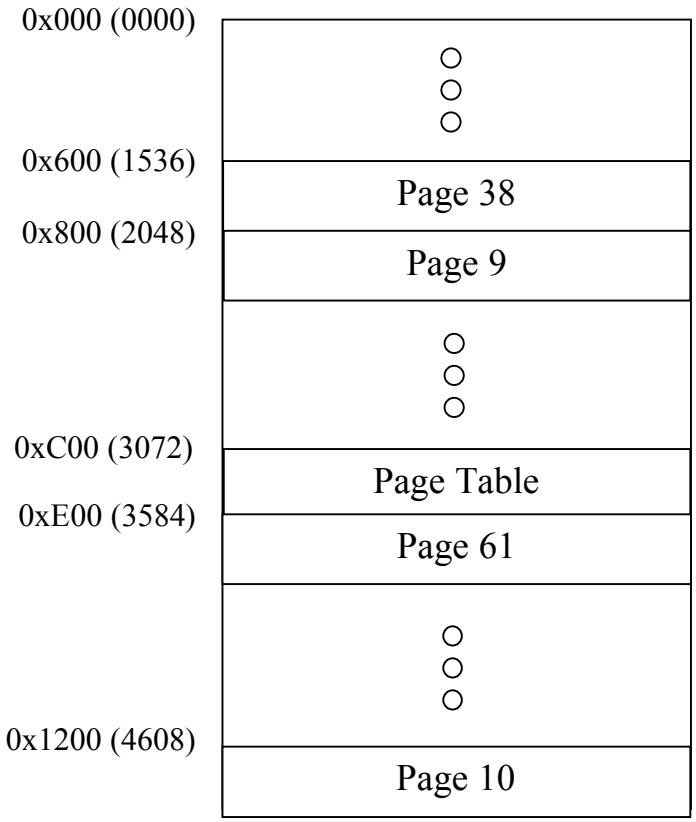
```
void myinit(unsigned long areasz);
void *mymalloc(unsigned long sz);
void myfree(void * addr, unsigned long sz);
```

A call to the function `myinit(areasz)` initializes the memory management system. It begins by using Linux's `malloc()` call to allocate `areasz` bytes of memory from which subsequent calls to `mymalloc()` are processed. It then builds an initial free list (consisting of a single node with size `areasz`) **within** the allocated space. (Each node on the free list is physically stored **in** the first part of the corresponding free area and contains an `unsigned long` specifying the size of the area and a `void *` pointer to the next free area on the list.) Once the system is initialized, it services allocation and freeing requests (via calls to `mymalloc()` and `myfree()`). Each time a call to `mymalloc(sz)` is made, the **first-fit** algorithm is applied to the free list to find the first block large enough in size (i.e. `sz` or more bytes) from which the allocation is made. The code then updates the free list to reflect the allocation and returns the address of the allocated region to the caller of `mymalloc()`. Space is allocated from the **end** of the selected free region to simplify the updating of the size field in the free list node. Each time a call to `myfree(addr, sz)` is made, the allocated region specified by `addr` and `sz` must be returned to the free list and amalgamated with any adjacent free blocks. This is the code you must extend! You may assume that arguments to `myfree()` are always valid. Note that you will need to **maintain** the free list in order **by address**. (This makes free space amalgamation easier and more efficient.)

To test your functions, a program that builds a linked list of positive integers typed at the command line (until zero is entered) is provided. This program reads the integers and stores them in a list of nodes created using `mymalloc()` and `myfree()`. It then prints the list nodes and deletes the nodes. Why is a minimum allocation size (`MINALLOCSIZE`) required?

2. Answer the following questions about the paged virtual memory system described below.

a) A paged virtual memory system has a page (and frame) size of 512 bytes and a physical memory consisting of 10 page **frames** numbered 0 to 9. The logical address space of the **single resident** process has 512 pages numbered 0 to 511. The current contents of **physical** memory (RAM) are as shown below. Addresses are shown in hex (0x) and decimal.



- **PHYSICAL MEMORY (RAM)**

[5] i. Draw a picture of the process' **page table** with gaps shown using ellipses ("…").

[1] ii. What happens when virtual address 0x7918(31000) is referenced? Explain briefly.

[4] iii. Draw a table showing which physical addresses (represented as page **frame** numbers and offsets) are referenced by the virtual addresses 0x1200(4608), 0x13FF(5119), 0x1400(5120), 0x7A08(31240)?

[10] b) Given the sequence of page references shown **below** and assuming that the available 5 frames of memory initially contain the pages 15, 7, 1, 11, and 8 (in that order, page 5 being most recently used though to page 8 being least recently used), illustrate the operation of the LRU replacement algorithm as the pages in the sequence below are referenced. Your answer should clearly show which pages are resident following **each** reference and should indicate **whenever** page faults occur. Also, be sure to clearly show what the "LRU order" of the resident pages is **before** each reference. Some sort of table with ordered entries is probably a good way to show the necessary information.

Page Reference Sequence: 7, 15, 10, 11, 14, 11, 9, 17, 9, 14

3. Answer the following questions about a filesystem similar to the Unix Fast File System that uses inode structures containing 8 direct block pointers as well as single, double and triple indirect pointers. Assume that disk **addresses** are 8 bytes and that disk blocks are 512 bytes.

[3] a) What is the largest file that can be stored in such a file system? Explain how you arrived at this maximum!

[5] b) How many disk blocks (data **and** index) are needed to store a file of the following sizes and how much **internal** fragmentation occurs in each case? (You can ignore space occupied by the inodes.) In each case, explain how you arrived at your answer.

[1] i. 512 bytes

[1] ii. 516 bytes

[1] iii. 10752 bytes

[1] iv. 2134016 bytes

[1] v. 8401408 bytes

[2] c) How many disk operations will it take to sequentially read an entire file of 77824 bytes? Explain how you arrived at your answer? You can ignore the access to the inode itself

Total: 40 marks

No Challenge Questions this time – you are better off focusing on core material in this and other courses you are taking and also starting to prepare for your final exams!