CS 3113: Operating Systems

Spring 2022

Dr. Richard M. Veras

Lab 01: Bubble Sort on Linked Lists in Assembly

Goal: This lab is designed to give you a deep understanding of the low-level details of the machine model that we are targeting in the class. Specifically, I want this lab to help you understand the following: translating abstract data machine register state, addressing modes for accessing memory and memory segments. Knowledge of these topics will help give context as we progress through the rest of the three pieces in OS (virtualization, concurrency, and persistence). For example, the machine model, knowledge of the registers and address space are foundational when we discuss virtualization. Additionally, the problems in concurrency will make more sense when we understand our atomic machine instructions. Lastly, one of the biggest difficulties with persistence is in implementing low level file system data structures. This is a very address space and memory movement heavy assignment, that should make conceptualizing the more challenging aspects in C (pointers to the location of values, address of the location of a value) a bit easier.

Beyond the technical components, my hope is that you will gain extra experience with working on challenging problems as a group, refining specifications, translating specifications into designs, and iterating through designs. And most importantly you will get practice distilling, reporting, and communicating these technical processes through written and verbal communication.

Working in a group setting was cited as the number one concern in the autobiographies. That is okay. I understand that for most, if not everyone, that you are your most efficient when you are working alone. I want you to struggle collectively as a team because it will be through these inefficiencies, redundancies, conflict, etc. that you will gain the most from this experience.

Not every group will be perfect and conflict free. This is to be expected. A handful of safeguards are put in place in this assignment to help manage expectations (statement of work, schedule, meeting notes, etc.) and to minimize damage (breakdown of credit and justification). Beyond that it is up to you as individuals and as a team to work together through group issues. Finally, this will be one of multiple labs, and your teams will be randomized each time, so one bad lab will be diluted by the other labs.

Now for generic boiler plate lab stuff, then after that we will get to the specifics for this lab.

Deliverables: You will provide me with three classes of deliverables, which include written components, presentation, and implementations. These are further broken down into the following pieces:

1. **Written Components:**
    a. **README.md:** This file will contain the following **sections** and it is expected that this will be questioned, challenged, and updated continuously until the deadline. The first iteration of this should be written on day 1.

i. Tentative Plan – What is the problem? How will you solve it? What Ideas are you going to throw at the wall? How could these ideas break down? Why do you think these ideas will work?
ii. Statement of Work – How will the work be divided and why? **NOTE:** everyone is responsible for contributing to each of the three components (writing, coding, and presenting). Ideally, the answer to the why should not be because it is the individual's strengths, but because it is their blind spot. This way everyone maximizes the return on the delta of effort that they put into this. Further, as a team not leaning into your strengths, but instead pushing through your weaknesses, is a great way to ensure that everyone's contribution is based on effort and not on experience outside of class. Something to consider for a natural load-balancer.
iii. Schedule/timeline - When do you plan to meet as a team? When do you plan to work on these components? Consider making a Gantt Chart for this project.
iv. Meeting notes – Unchallenged ideas should be considered wrong at best and harmful at worst. All ideas in your plans should be challenged, and I want you to write down the arguments, details, drawings, screenshots, etc. These will be helpful in your writeup.
v. Self-Assessment – what was each individual team member's contribution to the lab (in terms of percentages). Provide justification for each contribution. This will be used in attributing grades for the lab.
vi. Reflections - If this lab could be done again what would you as an individual do differently? What would you, as a team, do differently? What should I, as the instructor, do differently?

b. **Report.pdf**: This writeup will give you an opportunity to demonstrate your understanding of the material in the programming assignments, reading assignments, the lectures, and the lab itself. You will do this by documenting the design, testing and refinement process behind each of your implementations, and relating that back to the content that you have covered in the class. The expectation is that this will be a technical report (see this, this and this), and that you will use a lot of figures, examples, pseudo code, GDB (or other tools') trace snippets to drive the conversation of your explanation. For example, look at the presentations used in class and use that as a template. Specific details -- for this specific lab -- of what you should include will be explained further in the **specifications section** of these instructions.

2. **Presentation**: This will be a ~5 minute talk that provides an overview of the report and a walk through of building and running your code. For a guide on writing presentations check this out. It might be helpful to start the presentation first to use it as a template for the writeup and the implementations. You can refine it as your plan changes.
    a. **Presentation.pdf**: These are the slides for your presentation
    b. **Recording.txt**: This will contain a link to your recording on OU My Media. You can use zoom to record your screen and save it to the cloud, which will upload it to My Media.
3. **Implementations**: This will be the meat and potatoes of the lab. You will have at least one implementation. You will be given a minimum specification that you must turn into a design, implement in code, and demonstrate its correctness. This single implementation (along with the

associated writeup and presentation) will be worth at most the minimum number of raw points for a passing grade. For additional points, you will provide additional implementations with different designs, algorithms, or other features and enhancements that will be specified (and subject to grow during the lab). Along with the associated writeup and addition into the presentation, these will add to your running. The raw value of these enhancements will be determined after the deadline of the lab and will be based on difficulty, uniqueness, the number of teams that attempted them, the uniqueness of the content covered by the enhancement, etc.

Source Control Management: All source code and applicable deliverable materials will be maintained using git in a **private** Gitlab.com repository. The name of the repository will be of the form: CS3113_SP22_LABYY_TEAMXX, where "YY" is the lab number and "TEAMXX" will be the team name assigned to you. All team members, the professor and the TA, must be included on the member list of the repository as **MAINTAINERS**. Finally, the repository must at least contain the following directory structure:

./README.md

./DOCS/

./DOCS/Report.pdf

./PRESENTATION/

./PRESENTATION/Presentation.pdf

./ PRESENTATION/Recording.txt

./IMPLEMENTATIONS/

./IMPLEMENTATIONS/IMPL01

./IMPLEMENTATIONS/IMPL{02-XX}

./REFERENCES/

Submission: We will grade the material that is in the MASTER branch of your team's GitLab repository (if you do not use or are unfamiliar with branches, this will be the default one). However, as a backup you will submit a tarball "**submission.tar.gz**" of the MASTER branch of your repository on canvas.

Grading: This lab is meant to have a minimum passing threshold and an unbounded ceiling. The minimum passing threshold is the maximum number of raw points that can be earned from the single specified implementation and its associated writeup and presentation. Additional raw points can be earned by attempting additional implementations that cover different approaches and/or achieve

additional features/enhancements (to be listed in the specifications section). The raw point value of these enhancements will be determined after the deadline of the lab by comparing the difficulty and content coverage of the enhancements between all the teams. A grand total of raw points will then be assigned to each team. The grade of an individual in a team will be a function of the team's grand total of points and the self-assessment (individual percentage of effort put into the lab).

This grading scheme is not meant to disadvantage you, but instead is designed to reward creativity, effort, productive failing (early and often), and throwing spaghetti at the wall. If the ceiling was explicitly set, then the lowest risk strategy would float to the top.

Now for details specific to this lab.

Due Date: March 1, 2022

Baseline (Minimum) Specification: In this lab I want you to implement a variety of sorting **operation** over a list **abstract data type (ADT**) The **input** will be an unsorted list of integers (64-bit machine integers) and the **output** will be a list of integers sorted in increasing order. The **algorithm** I want you to start with is bubble sort, and for additional **implementations** you can explore other algorithms. I want your **implementations** to be written in a subset of x86-64 assembly language (These restrictions will be explained in detail later). For your **concrete data type** you will use a linked-list, whose specific representation in your **implementation** will be determined by you (for example keeping the data and link next to each other in memory or as two separate lists).

Restrictions: Except for function calls, you can only use the registers **rax, rbx, rcx, rdx, rdi, rsi,** and their 32-bit subsets **eax, ebx, ecx ,edx, edi,** and **esi**. Note that you can use any of the special purpose registers without restriction.

Enhancements: In addition to the **implementation** of the minimum baseline specification you are encouraged to produce **additional implementations** that incorporate the listed enhancements.

The expectation is that this list contains far in excess of what can reasonably be done in this time frame. If this is not the case, then I will continue to add ideas until it does! This is the unbounded part of the lab. I want you to pick the things that interest you, that you will throw spaghetti at. For some of these enhancements it may make sense to do them as combinations. Choose carefully, I suggest picking things that cover blind spots in the content we have covered so far. Now for the list:

1. Using a different storage format for the linked lists (see this). Also look at Ch 6 of Programming from the Ground up.
2. Using several types of linked lists: doubly linked lists, circular linked, doubly circular linked list. Note that you can do variations of the storage format.
3. Using function calls to compartmentalize your implementations.
4. Creating a build system for your implementations with Make (here, here, and here can be adapted for assembly).
5. Scripting your GDB commands (see here, here and here). This would be useful for testing you implementations that lack input and output because you need to examine the memory locations of your list before and after you sort.
6. Using different loop-based sorting algorithms. For example: selection, insertion, or bucket sort.
7. Using different recursive sorting algorithms: Merge sort, Quick sort, Radix sort.
8. Using hybrids of algorithms. For example, bucket sort followed by selection sort within the buckets. Or a merge sort where a base case of size "k" is performed by an insertion sort.
9. Output results of the sort to a file or the terminal (a special case of a file "STDOUT").
10. Read a fixed size (pre-determined size) list from a file as the input (can use the .bss "buffers" memory segment). See Ch 5 of Programming from the Ground Up.
11. Read an arbitrary size list from file (size known when the file is being read, for example the first line says the number of elements). You will need to use the heap for this. See Ch 9 of Programming from the Ground Up for ideas. You will not need a memory manager, but you will need to use the system call "brk" and company to find the heap and request enough space.
12. Refine your implementations for performance. This will require timing your code.
13. Optimize your code using Single Instruction Multiple Data (SIMD) operations. See this and this.

Hint(s): Feel free to prototype your code in another language. If you use C, then you can tell the compiler to generate assembly. However, the assembly you provide must satisfy the explicitly stated restrictions. In fact the restrictions are in place so that you cannot directly use compiler generated assembly.

Report.pdf specific details: For each attempted implementation I want you to include the following (remember to use figures, drawings, tables, charts, pseudocode, debugger traces of memory/registers/state, etc. to get your point across):

1. What were your assumptions (this will be the hardest part)? All specifications written without the precision of mathematics will be up to interpretation. I want you to explicitly state your interpretation of the specification (and enhancement/feature if applicable) that your design and subsequent implementation depend on in order to be correct.
2. State the objective of your implementation. At a high level what are you trying to do?
3. What will the input and output look like? Explain at the high-level and relate back to the low-level. Where are you storing the input? Is it baked into the code, or will you load it in externally?

How will we see the input and output? Using “examine” in GDB before and after the algorithm runs? Or will values be read from a file and written to a file?

4. Relate what you are doing in your implementation back to the lectures, readings and programming assignments.
5. Explain the steps of the algorithm that your implementation uses. Why will this work?
6. How will you map this algorithm to your implementation? What registers will you use for what and why? What address modes will you use and why? How will your data be organized in memory and why? What does the control flow look like? How will you break up the code into blocks or functions?
7. How will you test your code for correctness? Will these tests cover every corner case in your implementation?
8. How do you build and run your code?
9. Show me that your algorithm is doing what you explained (trace through the code and show snippets that demonstrate the behavior of your algorithm, etc.).
10. If this implementation did not work, tell me why? Where did it break? What would you try differently to get it to work and why?
11. If you use any external source or reference, then cite it and explain clearly what work is your own and what is from the reference. Note: any external references (code, images, pdfs of web sites) should be included in the “./REFERENCES/” directory in your repository.
12. If you are wondering whether something should be included or not, include it. Show me that you know what you are talking about. Convince me that you understand the content.

Plagiarism and External Sources: You are welcome to use external sources for inspiration. However, you should not wholesale copy code. If you use external sources, cite them, include them in the reference section of your repository, and make it absolutely clear how your interpretation differs from the source. Anything short of that will be considered plagiarism.

Questions, Comments and Concerns: This is a deliberately vague open-ended project, and I want to see how you make and justify your decisions in response to this type of uncertainty. I will have a discussion board open for the lab, where you can direct your questions. My goal here is to respond to technical matters in a timely manner (also, feel free to respond to each other's questions), and at best respond with guiding questions for everything else.

Typos, Errors, and Issues: This is an imperfect document, so it is highly likely that there are issues. If you find something post it on the discussion board for the lab.

Good Luck!

- Richard