Project 2: Kernel Data Structure

- **Handed out:** Tuesday, Sep 12, 2022
- **Due:** Friday, Sep 23, 2022

Introduction

The goal of this project is to develop your first kernel module, and to study the manipulation of the following frequently-used kernel data structures: a linked list, red-black tree, hash table, radix tree, and bitmap. The following concepts from the course will be put in practice in this project: (1) Kernel module development and (2) Kernel data structures.

Develop the kernel module in Linux v5.15. Use the provided `Makefile` and write your code in `kds.c`.

Make a folder named with your SBU ID (e.g., 112233445), put `Makefile` ,your `kds.c`, and screenshot `kds.png` files in the folder, create a single gzip-ed tarball named `[SBU ID].tar.gz`, and turn the gzip-ed tarball to Brightspace.

```
$  tar czvf 112233445.tar.gz 112233445/
112233445/
112233445/Makefile
112233445/kds.c
112233445/kds.png
```

Recommended Background Reading

- The Linux Kernel Module Programming Guide

Part 1. Write a kernel module

[4 points] Write a single Linux kernel module named `kds` in `kds.c`. The module takes one string parameter `int_str`, which is the arbitrary number of integers between 0 to 1000 (e.g., `insmod <module name> int_str='"11 44 22 33 5"'`). The module parses (tokenizes) the parameter `int_str` and print on the kernel log the input numbers using the `%d` format specifier.

Part 2. Add kernel data structures

Extend the module (`kds.c`) to include functions manipulating the following data structures:

- [7 points] Linked lists : 1) create a linked list containing the integers in `int_str`; 2) print on the kernel log the content of the list using the list iteration functions; and 3) destruct the list and free its content.
- [7 points] Red-black trees : 1) create a rbtree, which is indexed by integer numbers; 2) insert integer numbers in `int_str` to the rbtree; 3) look up the inserted numbers and print them out; and 4) remove all inserted numbers in the rbtree.
- [7 points] Hash table : 1) create a hash table, of which the number of buckets is 2^10; 2) insert integer numbers in `int_str` to the hash table; 3) iterate the entire hash table and print out all inserted integer numbers using `hash_for_each`; 4) look up the inserted numbers and print them out using `hash_for_each_possible`; 5) remove all inserted numbers in the hash table; and 6) destruct the hash table.
- [7 points] Radix tree : 1) create a radix tree, which is indexed by integer numbers; 2) insert integer numbers in `int_str` to the radix tree; 3) look up the inserted numbers and print them out; 4) tag all odd numbers in the radix tree; 5) look up all tagged odd number using `radix_tree_gang_lookup_tag`; and 6) remove all inserted numbers in the radix tree.

- [7 points] XArray : 1) create a xarray, which is indexed by integer numbers; 2) insert integer numbers in `int_str` to the XArray; 3) look up the inserted numbers and print them out; 4) tag all odd numbers in the radix tree; 5) look up all tagged odd number using `xa_for_each_marked`; and 6) remove all inserted numbers in the Xarray.
- [7 points] Bitmap : 1) create a bitmap, which is large enough to represent numbers between 0 to 1000; 2) set bits corresponding to integer numbers in `int_str`; 3) print all bits which are turned on; and 4) clear all bits in the bitmap.

Part 3. Test your module

[4 points] Take a screenshot of your kernel debug message using `dmesg` while running your module. Turn in the screenshot named `kds.png`.