Concurrent Programming
Lab 1 Write-up

Name – Atharva Nandanwar

Code Organization

There are three major components in the code –

1. Main Subroutine
2. Mergesort Function
3. Bucketsort Function

Main Subroutine handles the logic around argument parsing and files operations. It takes in the input, parses it using *getopt_long.* After all the arguments have been parsed – it reads the numbers from input file, sets up the array. Based on the arguments, it calls mergesort and bucketsoft functions. This main subroutine also handles few safety related adjustments to the number of threads.

Mergesort subroutine deals with mergesort implementation. Mergesort recursively splits up the array and merges it by passing it to *merge* function. *Merge* function takes the split and copies it into two temporary arrays. Three indices *indexl, indexp, indexh* keep track of elements from lower array, higher array, and original array. Addition to mergesort algorithm is that now, the array is split into NUM_THREADS parts, and mergesort is implemented on each of these individual array in separate threads.

Array → Split Array → Feed into threads → mergesort → Final merge after all threads terminate

Bucketsort subroutine implements bucketsort on the array. There are two parallel and independent strategies that I have used.
1. Parallel splitting of numbers into buckets
2. Parallel sorting of each bucket
Splitting takes thread_number as reference to jumps it makes in picking up elements to process them for adding into buckets. All the data is passed through to the split_arrays function through a struct. This design choice was made to not rely on global variables. I am using qsort from stdlib for sorting the buckets, and this was borrowed from https://www.tutorialspoint.com/c_standard_library/c_function_qsort.htm.

Array → Split into buckets (parallelised) → Sort each bucket (parallelised) → Final merge

Files

main.c – contains main subroutine, argument parsing, file logic
bucketsort.c – functions for bucketsort implementation
buckersort.h – header file for bucketsort
mergesort.c – functions for mergesort implementations
mergesort.h – header file for mergesort
Makefile – used for making the application and clean it

Compilation Instructions
Use `make` command to compile the mysort application
Use `make clean` to clean the executable

Execution Instructions
`./mysort <sourcefile> -t threads -o <outputfile> --alg=<fjmerge|lkbucket>`

-o <outputfile> provides the output file to write the output to. If not provided, output is piped to stdout.
--alg=<fjmerge|lkbucket> provides the algorithm to use. Not providing this argument would
Extant Bugs
Running `./mysort <sourcefile>`, it would give a SIGSEGV. You must always give --alg=fjmerge or --alg=lkbucket

Running the program with valgrind gives errors. I have not been able to take a look at them.

Having inputs larger than INTMAX would cause segmentation fault.