# Programming HW4

System Programming’15 Fall

12302015

## Introduction

- The goal of this homework is to sharpen your thread programming proficiency.
- You are asked to parallel a two-pass merge sort algorithm by POSIX thread.

## Pass I

- Given an array of size `n` and a segment size `s`, divide the input array into several segments of size `s`. Note that if `n` is not divisible by `s`, the size of the last segment will be less than `s`.
- For each array segment, create a `POSIX` thread and run a sorting algorithm on it. You do not have to implement the sorting algorithm on your own, that is, sorting function like `qsort()` in `<stdlib.h>` can be called based on your choice.

## Pass I

- The following components should be printed on the screen:

(1) The content of segment handled by each thread.

(2) The length of segment handled by each thread.

| | | | | |
|---|---|---|---|---|
| 68 42 32 16 | 42 99 100 5 | 33 65 16 68 | 12 2 76 50 | 99 38 |
| `sort()` | `sort()` | `sort()` | `sort()` | `sort()` |

| | | | | |
|---|---|---|---|---|
| 16 32 42 68 | 5 42 99 100 | 16 33 65 68 | 2 12 50 76 | 38 99 |

## Pass II

- The merging steps will be done in several rounds. For each round, launch **`[#segs/2]`** threads to merge adjoining segments in parallel. If there is an odd number of segments in a round, simply advance the last segment to the next round.
- Each round should be blocked until all threads finish merging.
- The merging of $2$ sorted segments should be done in **`O(len(seg1)+len(seg2))`**.

## Pass II

- The following components should be printed on the screen:

(1) The content of segment handled by each thread.

(2) The # of duplicates found on each merging task.

- Note that you should pick the element of the **previous** segment when a duplicate take place.

| | | | | |
|---|---|---|---|---|
| 16 32 42 68 | 5 42 99 100 | 16 33 65 68 | 2 12 50 76 | 38 99 |

| | | |
|---|---|---|
| 5 16 32 42 42 68 99 100 | 2 12 16 33 50 65 68 76 | 38 99 |

| | |
|---|---|
| 2 5 12 16 16 32 33 42 42 50 65 68 68 76 99 100 | 38 99 |

| |
|---|
| 2 5 12 16 16 32 33 38 42 42 50 65 68 68 76 99 99 100 |

## Sample Execution

- `$ ./merger [segment_len] < testdata`
- testdata format:

(1) 1st line: # of integers to be sorted

(2) 2nd line: the content of the array

- The input data is an integer array.

## Report

- Generate random test data of size `n={100, 10000,1000000,10000000}`. For each of the data, use `segment_size={n/100,n/25,n/10,n/5,n/2,n}`. accordingly. Thus there will be `24` combination in total.
- Use certain time measurement methods to measure the execution time for each combination.
- Briefly state your finding in report, that is, how’s the (real time/user time) affected by number of segments, and why?

## Submission

- Pack the following components in `.tar.gz` and submit to CEIBA.

(1) `merger.c` (or `cpp`)

(2) `Makefile`

(3) `Report.pdf`

- The deadline will be 23:59:59 on 1/18, 2016.

## Scoring

(1) Completeness (1 point)

(2) Correctness (5 points)

(3) Output format (1 point)

- Note that the messages printed by each thread should not be interrupted by other threads.

(4) Report (2points)

## Punishments

(1) Plagarism

(2) Late Submission

**(No late submission allowed this time!)**

(3) File format

# Q and A