DESIGN DOCUMENT

Overview

This MapReduce implementation in C is designed to perform a word count task by dividing the workload among multiple mapper and reducer processes using Unix pipes and `fork()` + `exec()` for process management.

The entire program consists of three components:

- `main.c`: The master process that coordinates mappers and reducers.
- `mapper.c`: Processes input text, outputs individual word counts.
- `reducer.c`: Aggregates intermediate word counts and outputs final results.

---

main.c

Purpose:

- Acts as the orchestrator for the MapReduce framework.
- Launches mapper and reducer processes.
- Sends input data to mappers and routes their output to reducers based on hashing.
- Collects reducer output and prints final result.

Key Functions:

`hash_word(const char* word, int num_reducers)`

Computes a simple hash to determine which reducer a word should go to. Ensures consistent word-to-reducer routing.

`write_all(int fd, const char* buf, size_t count)`

Guarantees all bytes are written to a file descriptor. Used to ensure complete writes to pipes.

`main()`

- **Pipe setup:** Creates `stdin` and `stdout` pipes for each mapper and reducer.

- **Fork mappers and reducers:**
  - Each mapper receives its own input pipe and output pipe.
  - `execlp("./mapper")` launches `mapper` binary.
  - Reducers are similarly launched using `execlp("./reducer")`.
- **Input distribution:**
  - Lines from `stdin` are round-robin sent to mappers.
- **Intermediate output routing:**
  - Reads mapper output ("word count" pairs).
  - Hashes words to determine reducer assignment.
  - Sends lines to corresponding reducer pipes.
- **Output collection:**
  - Monitors all reducer output pipes using `select()`.
  - Writes reducer output to `stdout`.

---

mapper.c

Purpose:

- Reads text input and emits each normalized word with count `1`.

Core Functions:

`normalize_string(char* str)`

Cleans and standardizes the input line by:

- Lowercasing
- Removing punctuation
- Replacing Unicode smart quotes and dashes
- Removing contractions/possessives

`extract_words(char* line)`

- Tokenizes normalized line.
- Outputs each word in `"word 1"` format.

`main()`

- Reads from `stdin`, accumulates text, and processes each full line.
- Calls `extract_words()` for each normalized line.

---

reducer.c

Purpose:

- Reads `(word count)` pairs from stdin.
- Aggregates total counts per word.
- Outputs sorted final word counts.

Core Structures:

`struct WordCount`

- Linked list node storing `word`, `count`, and `next` pointer.

Core Functions:

`add_word(const char* word, int count)`

- Adds or updates a word in the in-memory linked list.

`mergeSort()`, `merge()`

- Custom merge sort used to sort the word list alphabetically.

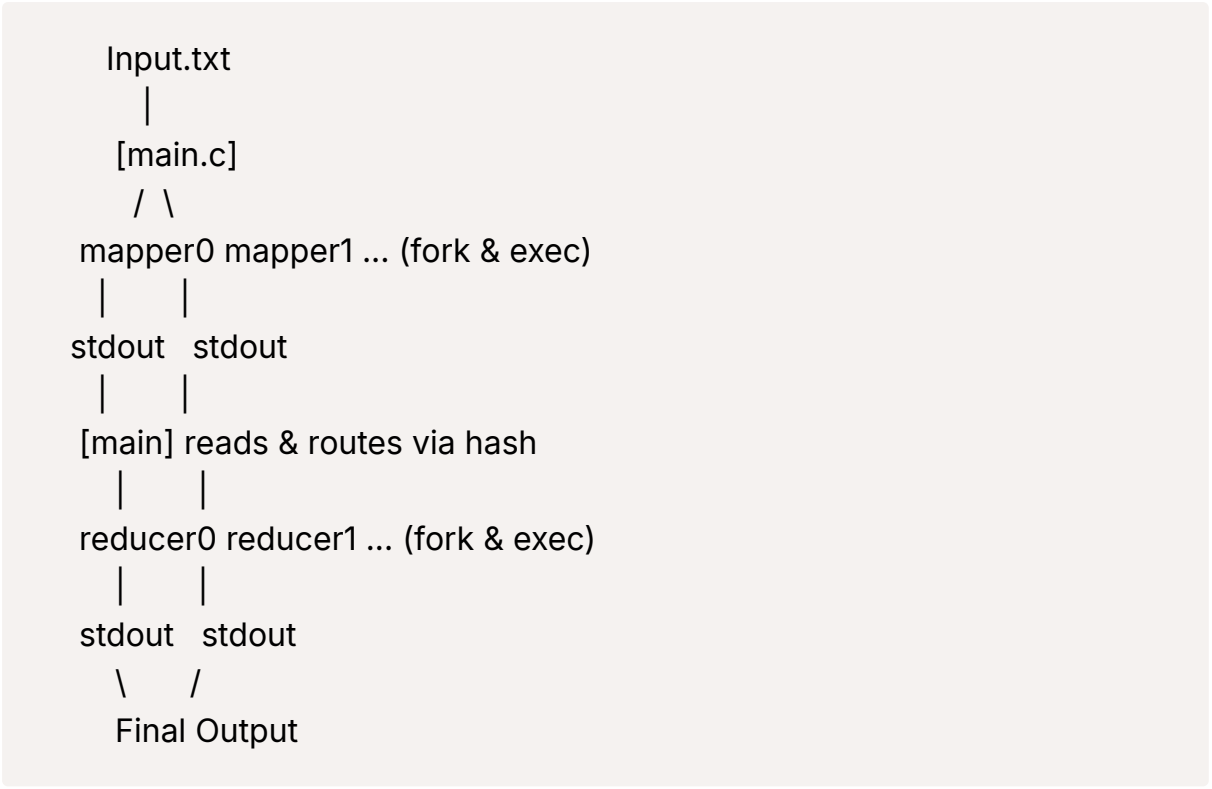
`output_results()`

- Outputs sorted `(word count)` pairs to `stdout`.

`main()`

- Reads `(word count)` pairs from stdin.
- Aggregates using `add_word()`.
- Outputs sorted results via `output_results()`.

Process Pipeline Summary



Notes & Extensions

- Currently uses fixed `NUM_MAPPERS` and `NUM_REDUCERS`, configurable via macro.
- Single-machine simulation; can be adapted to a distributed setup.
- Sorting in reducer can be extended to support `-sort_by_freq`.
- Punctuation filtering is tightly coupled with English; could add support for Unicode categories.