VersatileCodeLab

Konstantinos Pisimisis

March 25, 2024

1 Introduction

The aim of this project is to introduce students to bash scripting and the use of Python for simulating processes, including pipes and process control. This project was created for the Software Development Tools and Systems Programming course and serves as the second assignment. The initial section discusses NewsAnalyzer, which utilises the Jaccard Index and is also implemented in Python, but expands to include the sentiment of certain words. The remaining portion of the project focuses on pipes and process control.

2 Project Overview

2.1 NewsAnalyzer (Bash Script)

The NewsAnalyzer script is designed to analyze news articles using the Jaccard Index and sentiment analysis techniques.

- **Implementation**: The script utilizes Bash scripting to perform various operations such as file processing, Jaccard Index calculation, and sentiment analysis. It interacts with the user via the command-line interface and provides functionalities to analyze news articles based on categories and stems.

2.2 Sentiment Analysis

The sentiment analysis component analyzes the sentiment of news articles based on the presence of positive and negative stems.

- **Implementation**: Positive and negative stems are read from separate files, and stem-term pairs are processed to identify positive and negative terms. Each document's sentiment is determined by comparing the counts of positive and negative stems, and the result is stored in the output file `sentimentpernewsitem.txt`.

2.3 Process Simulation (Python Program)

The Python program simulates processes involving pipes and process control.

- **Implementation**: Written in Python, this program demonstrates the use of pipes and process control mechanisms. It includes functionalities to create and manage processes, establish communication between processes using pipes, and control the execution flow of processes.

2.4 Integration

While the NewsAnalyzer, Sentiment Analysis, and Process Simulation components are implemented independently, they collectively showcase the capabilities of Bash scripting and Python programming.

- **Implementation**: Each component is self-contained and can be executed separately. However, they are part of the same project repository and share common documentation and version control. The integration aspect is demonstrated through their cohesive presence within the project structure and their collective contribution to showcasing diverse programming skills.

2.5 Languages, Technologies, and Frameworks Used

- **Bash**: Used for implementing the NewsAnalyzer script, including file processing, command-line interaction, and sentiment analysis.
- **Python**: Utilized for developing the Process Simulation program, demonstrating processes, pipes, and process control mechanisms.
- **Version Control (Git)**: Employed for managing project files, tracking changes, and facilitating collaboration.
- **Documentation (Markdown/LaTeX)**: Markdown and LaTeX formats are used for documenting project details, including descriptions, usage instructions, and technical specifications.

3 Features

3.1 NewsAnalyzer(Bash Script)

- **Database and Table Creation:**The script initializes and populates several associative arrays to store data related to categories, documents, stems, and Jaccard indices.
- **File Processing:**Functions are provided to read categories, documents, and stem-term pairs from input files. Carriage return characters are removed from the input files to ensure compatibility across different platforms.
- **Jaccard Index Calculation:**The script calculates the Jaccard index for each stem-category pair based on the presence of terms in documents belonging to specific categories.Intersection and union operations are performed to determine the similarity between sets of terms.
- **Menu-Driven User Interface:**Users interact with the script through a menu-driven interface, where they can choose from various operations. Operations include retrieving relevant stems and categories, obtaining Jaccard indices, and displaying document details.
- **Operations Implementation:** Relevant functions are provided to perform operations such as retrieving most relevant stems for a category, fetching relevant categories for a stem, and calculating Jaccard indices for stem-category pairs. Additional operations include displaying associated categories and stems for a document, as well as counting unique terms and categories within a document.
- **Flexibility and Extensibility:** The script is designed to be flexible, allowing users to easily add new functionalities or modify existing ones to suit their requirements.Functions are modularized, enabling easy maintenance and extension of the script's capabilities.

3.2 Sentiment Analysis

- **Program Description:**This program analyzes each term in every document and determines whether it has a positive, negative, or neutral impact.
- **File Handling** The program begins by removing previous files, namely sentimentpernewsitem.txt and stem_term_unix.txt
- **Reading Stem Data:** It then reads the positive_stems and negative_stems files line by line, removing any blank spaces from each line and storing these items in associative arrays positive_stems and negative_stems respectively).
- **Processing Stem-Term Pairs:** The program reads the stem_term_pairs file and writes the data to a new file named stem_term_unix.txt. This is necessary because files created in a Windows environment have an added carriage return character which can cause issues in our program.

- **Document Analysis:** Next, the program reads the doc_term file line by line. The program initializes counters for positive and negative stems (positive_counter and negative_counter) for each line, which represents a document with its terms.For each term of the document, the program checks if it exists in the stem_term_pairs array and retrieves the corresponding stem if it does.
- **Determining Sentiment:** Then, it checks if the stem is in the positive_stems or negative_stems array. If the stem is in the positive_stems array, it increments the positive_counter. After checking the negative_stems table, the negative_counter is increased. This process is repeated for all terms in the current document to determine the document's impact based on the positive_counter and negative_counter.
- **Output:** The results are then written to the sentimentpernewsitem.txt file. The program terminates with the message 'Process ended' in the console.

3.3 Python Process Programming - Christmas Tree Process Generator

- The aim of this part is to create a hierarchical tree like the one shown below, using os.fork() appropriately. The program starts by calling create_tree(), which is the parent process of all subsequent processes. create_tree() calls the 'create_child()' function with the appropriate arguments, which will create the children recursively. Specifically, create_child() defines the behaviour of each process (level and index), prints out its own details and recursively creates child processes based on the defined hierarchical structure. The time.sleep() introduces an artificial delay, while the os.waitpid() prevents the creation of zombie processes. To complete the analysis, note that 'fork' returns 0 if we have a child, -1 if there is an error in the process and a positive value if we are in the parent process.

```
        A (Parent[0])
├── B (Child[1.1])
│   ├─ D (Child[2.1])
│   │   └─ W (Child[3.1])
│   │      └─ Q (Child[4.1])
│   └─ E (Child[2.2])
│      ├─ Z (Child[3.2])
│      │  ├─ V (Child[4.2])
│      │  └─ N (Child[4.3])
│      └─ X (Child[3.3])
│         └─ M (Child[4.4])
└─ C (Child[1.2])
   └─ F (Child[2.3])
      └─ T (Child[3.4])
         └─ L (Child[4.5])
```

Tree Process Generator - Expected Output

3.4 Python Pipes - Mixed Piping Implementation

- This section describes a Python script consisting of three functions: source_process, transformer_process, and output_process. The script uses anonymous and FIFO pipes to simulate a data processing pipeline.
- The source_process function writes data to the write end of an anonymous pipe. It handles possible 'OSError' exceptions during the write operation and sleeps for 1 second after writing each item.
- The function transformer_process reads data from the read end of an anonymous pipe, tokenizes it into words, and writes the tokens to a FIFO pipe. It also handles any 'OSError' exceptions that may occur during the reading and tokenizing processes.
- The function output_process reads data from the FIFO pipe, decodes it, and stores it in a result string. It also handles any OSError exceptions that may occur during the reading and closing actions.

- In the main function, an anonymous pipe and a FIFO pipe are created. The script creates a child process, with the parent process closing the read end of the anonymous pipe and invoking source_process to write data. The child process invokes transformer_process to read data from the anonymous pipe, tokenize it, and write tokens to the FIFO pipe. Finally, output_process is called to read data from the FIFO pipe and display the result on the screen.

4 Code Snippets

4.1 Introduction

The aim of this section is to highlight interesting parts of the project's code and explain their purpose. Our analysis will focus solely on the bash section due to its complexity compared to the developed Python code.

4.2 Calculation of intersection and union of term_docs and my_category_docs

```
# Calculation of intersection and union
intersection=($(printf "%s\n" "${term_docs[@]}" "${my_category_docs[@]}" | sort | uniq -d))
union=($(printf "%s\n" "${term_docs[@]}" "${my_category_docs[@]}" | sort | uniq))
```

Listing 1: Intersection and Union

This lines calculate the intersection and the union of the `term_docs` and `my_category_docs` arrays. Here's a breakdown:

1. **Array Expansions:**
   - `"${term_docs[@]}"` and `"${my_category_docs[@]}"` expand to all elements of the `term_docs` and `my_category_docs` arrays, respectively.
2. **Print and Sort:**
   - `printf "%s\n"` prints each element of the arrays on a new line.
   - `| sort` pipes the output to the `sort` command, sorting the lines in ascending order.
3. **Uniq with Duplicate Option:**
   - `| uniq -d` pipes the sorted output to the `uniq` command with the `-d` option, instructing it to print only duplicate lines (those present in both arrays).

Similar to the intersection, the union is performed using uniq without the argument -d to remove duplicate lines.

4.3 Extracting the stem and the category

Part of the exercise involves calculating associations that require multidimensional arrays. However, it is well known that bash scripts do not support multidimensional arrays. To overcome this obstacle, I often combine data into a new string to create a new key and use it to produce the desired associations. Although the task was easy, there is an issue when searching the array for a specific value. To overcome this, I utilised pattern matching techniques supported by bash. The following lines of code are of particular interest:

```
for key in "${!jaccard_index[@]}"
    do
        stem=${key%%_*}  # Extract the stem from the key
        key_category=${key#*_}  # Extract the category from the key
        if [[ $key_category == "$category" ]]
        then
            ToPrintStems["$key"]=${jaccard_index["$key"]}
        fi
    done
```

Listing 2: Part from the function get_most_relevant_stems_for_category()

```
        for ((i=0; i<k && i<${#sorted_keys[@]}; i++))
    do
        category=${sorted_keys[$i]#*_}  # Extract the category from the key
        echo -n "$category"
        if (( i < k-1 )); then
            echo -n ", "
        fi
    done
```

Listing 3: Part from the function get_most_relevant_categories_for_stem()

Starting from Listing 2, it is observed that in order to extract the stem from the key, which is in the form `stem_category`, the `key%%_*` is used. This is done to remove the longest match from the first underscore. Conversely, when extracting the category, the `key#*_` is used. This is done to remove the shortest pattern from the start of the variable.

4.4 Sorting keys by values

Another problem we faced was efficiently sorting keys based on their values. To solve this, I developed a complex bash command. To simplify, let's consider an associative array with the following data: ["fruit_apple"]=5 ["fruit_banana"]=7 ["fruit_cherry"]=2 and the code below:

```
IFS=$'\n' read -r -d '' -a sorted_keys < <(printf "%s\0" "${!ToPrintCategories[@]}" |
while IFS= read -r -d '' key; do echo "${ToPrintCategories[$key]} $key"; done | sort -gr
 | awk '{print $2}' && printf '\0')
```

Listing 4: Sorting keys by values

1. `IFS=$'\n' read -r -d '' -a sorted_keys`: Reads lines into the `sorted_keys` array. `IFS=$'\n'` sets the internal field separator to newline, `-r` prevents backslash escapes from being interpreted, `-d ''` sets the delimiter to null (which makes `read` continue until it encounters a null byte), and `-a sorted_keys` reads the lines into the `sorted_keys` array.The output in this case would be: `The output is fruit_apple \0fruit_banana\0fruit_cherry\0`.

2. `< <(...)`: Process substitution that allows the output of the command inside the parentheses to be read by the `read` command.

3. `printf "%s\0" "${!ToPrintStems[@]}"`: Prints each key of the `ToPrintStems` array followed by a null byte.

4. `| while IFS= read -r -d '' key; do echo "${ToPrintStems[$key]} $key"; done`: Pipe that sends the output of the `printf` command to a `while` loop. The `while` loop reads each key into the variable `key` and outputs the value corresponding to the key in the `ToPrintStems` array followed by the key.The output in this case would be: `5 fruit_apple\n7 fruit_banana\n2 fruit_cherry`.

5. `| sort -gr`: Another pipe that sends the output of the `while` loop to the `sort` command. The `-g` option tells `sort` to sort in general numerical order, and the `-r` option tells it to reverse the result of comparisons (which makes it sort in descending order).The output in this case would be: `7 fruit_banana \n5 fruit_apple\n2 fruit_cherry`.

6. `| awk '{print $2}'`: Another pipe that sends the output of the `sort` command to the `awk` command. `awk` prints the second field of each line, which is the key. The ouput in this case would be: `fruit_banana \nfruit_apple\nfruit_cherry`

7. `&& printf '\0'`: Command that prints a null byte if the previous command succeeded.This is necessary because the read command with -d " option reads input until it encounters a null byte.It's a way to signal to the read command that it has reached the end of the input. Without this, if the output of the process substitution doesn't end with a null byte, the read command would keep waiting for more input, and the script could hang.

5 Conclusion

In summary, this project provided a valuable opportunity to explore the functionality of bash scripts. We also gained insight into deeper programming subjects by working with processes and pipes in Python. While there are opportunities for further extensions, such as upgrading the sentiment analysis algorithm to determine word sentiment without relying on files, it is important to note that the limitations of bash may restrict our ability to extend the application. Finally, I hope this analysis helps to understand the choices and difficulties we faced during development. It should be sufficient to understand the code.