Multi-Process and IPC
Word Counting Program

Yewon Lee

1 Design of Multi-process Structure and IPC

The program is designed to efficiently process large files by employing a multi-process architecture. Its core functionality includes counting the number of lines, words, and characters in a parallelized manner. Initially, the file is partitioned into multiple segments, with each segment assigned to a specific child process. This approach enables concurrent processing of different file portions, optimizing overall performance.

Child processes are created to handle designated segments concurrently, and Inter-Process Communication (IPC) is established using pipes to facilitate data exchange between the parent and child processes. This ensures synchronized processing and accurate aggregation of results.

Error handling mechanisms are incorporated to address scenarios where child processes encounter issues during execution. Specifically, the program includes robust error-checking procedures to identify and respond to potential problems, ensuring smooth execution of the overall process.

2 Crash Handling

To effectively handle crashes, the program implements a strategic approach. First, the parent process monitors the termination status of child processes. In the event of a crash, the parent process initiates the creation of a new child process to replace the crashed process. This dynamic restart mechanism continues until a successful execution is achieved.

In addition, the program employs a recursive handling strategy. If a newly created child process also encounters a crash, the program recursively continues to create new child processes until a successful execution is achieved. This ensures that even in the presence of multiple consecutive crashes, the program can adapt and recover seamlessly.

3 Usage

Compiling the program can be done using the following command in the terminal:

```
make
```

Once the program is compiled, executing the program can be done using the following command:

```
./wc_multi ./<filename> <#processes> <crashrate>
```

`<filename>` Specifies the name of the file to be processed.

`<#processes>` Optional parameter indicating the number of child processes (default is 1). If less than 1, set as 1. If greater than 10, set as 10.

`<crashrate>` Optional parameter specifying the crash rate percentage (default is 0). If less than 0, set as 0. If greater than 50, set as 50.

4 Example Output

The following command processes the file `large.txt` using 10 child processes with a 30% crash rate:

```
./wc_multi ./large.txt 10 30
```

Upon completion, the program provides detailed output, including the total lines, words, and characters processed. Additionally, it reports the time taken for the execution of the entire process.

Output:

```
# of Child Processes: 10
crashRate RATE: 30%
[pid 16579] reading 10176325 bytes from offset 0
[pid 16580] reading 10176325 bytes from offset 10176325
[pid 16581] reading 10176325 bytes from offset 20352650
[pid 16582] reading 10176325 bytes from offset 30528975
[pid 16583] reading 10176325 bytes from offset 40705300
[pid 16584] reading 10176325 bytes from offset 50881625
[pid 16585] reading 10176325 bytes from offset 61057950
[pid 16586] reading 10176325 bytes from offset 71234275
[pid 16587] reading 10176325 bytes from offset 81410600
[pid 16588] reading 10176332 bytes from offset 91586925
[pid 16582] crashed.
[pid 16584] crashed.
[pid 16587] crashed.
```

```
[pid 16594] reading 10176325 bytes from offset 30528975
[pid 16595] reading 10176325 bytes from offset 50881625
[pid 16596] reading 10176325 bytes from offset 81410600

========= ./large.txt =========
Total Lines : 2000000
Total Words : 19082229
Total Characters : 82681028
======== Took 0.442 seconds ========
```