Concurrency and Paralellism

GEI 2024

Lab Assignment #3 (Concurrent Compression using Message Passing)

We are going to develop a message passing version of the compression program from the previous lab assignment. As before, there is an existing implementation that does the compression using a single process. The API of the compression program is provided by the `comp` module through the following functions:

- `comp(File)` , that compresses `xxxx` into `xxxx.ch`.
- `comp(File, Chunk_Size)`. Same as the previous one, but setting the chunk size.
- `decomp(File)` descompresses `xxxx.ch` back into `xxxx`.
- `decomp(Input_File, Output_File)` descompresses `Input_File` into `Output_File`.

Compression

Compression uses two processes to read from the input file and to write into the compressed file.

The function `comp/2` starts the file reading process using `file_service:start_file_reader(File, Chunk_Size)`. If the reader starts successfully, `start_file_reader` returns the pid of the new process. The reader accepts the following two messages:

- `{get_chunk, From}`. The reader gets a chunk from the input file, and replies to `From` with one of the following messages:
  - `{chunk, Chunk_Number, Offset, Data}` if the data was read.
  - `eof` if the reader got to the end of the file.
  - `{error, Reason}` if there is any problem reading the file.
- `stop`. The reader process stops.

The `comp/2` function starts the compressed file writer in a similar way, using the function `archive:start_archive_writer(File)`. The writer accepts the following messages:

- `{add_chunk, Num, Offset, Data}`. Adds `Data` from chunk number `Num` into the output file.
- `stop`. The writer stops.

Decompression

Decompression also uses two processes, a reader of compressed files started using the function `archive:start_archive_reader(File)`, with the same API as the previous file reader:

- `{get_chunk, From}`, which replies:
  - `{chunk, Chunk_Number, Offset, Data}`
  - `eof`
  - `{error, Reason}`
- `stop`

And a file writer (`file_service:start_file_writer(File)`), that accepts the following messages:

- `{add_chunk, Offset, Data}`, adds a chunk of data in the current position.
- `stop`
- `abort`, stops the writer and removes the file.

Exercise 1 (**Write a concurrent compression function**) Compression uses a single process running in `comp_loop`. This function may be called from `comp(File)`, and `comp(File, Chunk_Size)`. Write two concurrent versions `comp_proc(File, Procs)` and `comp_proc(File, Chunk_Size, Procs)` that compress File using `Procs` processes. Make sure that all the processes created during the compression process stop when the compression ends.

Exercise 2 (**Write a concurrent decompression function**) Decompression also takes place in a single process in `decomp_loop`. This function is called from `decomp(Archive)` and `decomp(Archive, Output_File)`. Write two concurrent versions `decomp_proc(Archive, Procs)` and `decomp_proc(Archive, Output_file, Procs)` that decompress `Archive` using `Proc` processes. All the processes started should stop when the decompression ends.

Checking that all processes stop

Use the debugger (`debugger:start()` to check if all processes finish correctly. The debugger will list all processes running in the modules selected in the menu `modules=>interpret`. Use that option to check if all the processes have finished when the compression/decompression is done.

Submission Deadline

The submission deadline is March 17th. You can create your repository at https://classroom.github.com/a/uMxWXe9_.