Project: Memory-Mapped UNSAT

CSCE 311 - Operating Systems

April 28, 2025

Purpose

In this project, you will explore memory-mapped files and threaded programming by solving the n-sat problem in-place in file.

Task

Your task is to accept the command-line input described below to parse lines from a given file and decide whether they are SAT or UNSAT. I have provided the libraries `util/include/bool_expr_parser.h`, `sync/include/thread_mutex.h`, and their source files for your convenience. I would recommend using `SatSolver` from the util files.

You have three issues to address:

- Use memory-mapping to open and read a file —

  You must include the following comments such that a text-search function will find them:

  `// OPENING FILE`
  On the line above where you open file to acquire file descriptor.

  `// MEMORY MAPPING FILE`
  On the line above where you memory map the file from the above file descriptor.

  `// CLOSING FILE`
  On the line above where you close the file from above.

  `// READING FILE`
  On the line above where you read data from the file memory-mapped above.

  `// UNMAPPING FILE`
  On the line above where you unmap the file you mapped above.

- Threaded access to shared file —

  You must include the following comments where you use create, execute, and wait for threads to finish. YOU MUST USE `pthread`s.

  `// CREATING PTHREADS`
  On the line above where you create you pthreads.

  `// START ROUTINE`
  On the line above where you declare the function/method which you will pass to the thread creation function.

  `// JOINING PTHREADS`
  On the line above where you call function to block waiting for thread completion.

- Synchronizing access to the shared file —

  You must include the following comments where you use synchronization. If you do not use synchronization, see below:

  `// ENTERING CRITICAL SECTION`
  On the line above where you enter the critical section.

  `// EXITING CRITICAL SECTION`
  On the line above where you exit the critical section.

  `// ENTERING CRITICAL SECTION EXITING CRITICAL SECTION` on the line above where you explain in comments why you did not need synchronization. If you explanation is invalid, you will receive zero points for correctness.

Execution:

An example of execution and output from my solution:

```
me@mymachine~/311/proj4$ ./n-sat-solver 4 dat/dnf_exprs_16_10.txt 16
Thread  Sat  Unsat
   0    1      2
   1    1      1
   2    1      1
   3    2      1
Total   5      5

me@mymachine~/311/proj4$ ./n-sat-solver 4 dat/dnf_exprs_16_50.txt 16
Thread  Sat  Unsat
   0    4      8
   1    7      6
   2    2      6
   3   12      5
Total  25     25
```

Deliverables

You must provide a zipped directory containing at least the following files:

```
proj4/
+-- include/
|   +-- n_sat_solver.h
|
+-- src/
|   +-- n_sat_solver.cc
|
+-- README.md
```

Note that any other files will be ignored. The libraries I have included for your convenience will be available for your code to compile against, but you will not be able to change it because we will not copy any files other than those mentioned above. Note that I will add logging info to the synchronization class and function to ensure you used them.

Grading Criteria

Build (20%)

`n_sat_solver.cc]` successfully build with `n_sat_solver.h` and provided libraries to make `n-sat-solver`.

Correctness (30%)

In addition to running without crashing on a file 1 line long, you must demonstrate (as stated above). Any section missing any comments will lose 10% per section with a missing comment.[1]

Memory-mapping,

Threading (with POSIX threads), and

Synchronization.

Correctness (40%)

I have provided the code to correctly assess SAT/UNSAT. Therefore I will use four tests files. You will get 10% for each perfectly correct text output, i.e., YOUR PRINT MATTERS.

README.md (10%)

- Provide a reasonable `README.md` file, including build and execution instructions, file structure (only files you provide), and file contents for a free 10%.

Academic Integrity

By now, you know my stance on generative A.I.—it is a tool, not a panacea for all your C++ woes. I will ask three Generative A.I. for solutions and will include those in the stack of programs submitted for comparison. Given the breadth of freedom you have with solving this problem, I do not expect a great deal of code overlap.

[1] Try `cat src/n_sat_solver.cc | grep "// OPENING FILE"` for example to check yourself.