

MSSV Documentation Report

Brief File Explanations:

solution.txt, solution2.txt, solution3.txt, solution4.txt

These files are the input files used in the command line arguments in MSSV. They are plain text files that contain the completed sudoku solutions that are checked by the program.

makefile

The makefile is a short script that compiles/decompiles the software.

README

A Short file that has instructions on how to compile/decompile and run the program.

main.c

The Entry point of the program where all the original data is initialized, and other functions are called during the beginning.

ThreadDataStruct.h

This file contains a struct with all the shared data the threads use throughout the software solution.

cmdlValidation.c, cmdlValidation.h

These files contain a function to validate the command line arguments which also includes error checking and graceful termination if invalid values occur.

fileIO.c, fileIO.h

These files contain a function that reads the inputted solution file, parses it, and adds it to the shared data struct where it is used by the child threads, fileIO.c also contains error checking in case an invalid file is inputted, or a file cannot be found.

threadding.c, threadding.h

These files contain the bulk of the program and include a variety of functions that are used by the child threads.

finalOutput.c, finalOutput.h

These files contain functions that turn the output data of all the threads into easy-to-read data.

How Synchronization is Achieved:

Synchronization is achieved with the use of 2 different mutex locks (mutex, mutex2).

2 mutex locks are used to prevent deadlocks. If a thread were to enter its critical section whilst the IDs of other threads were still being assigned, a deadlock would happen because mutex 1 is still being used to add values into the SimpleTID array. Having 2 mutex locks solved this problem. Once the threads have been assigned a task, they all use mutex2 when they enter their critical sections. Because of multiple mutex locks, a deadlock cannot occur anymore as the threads do not rely on each other but have their own local variables whilst acquiring the mutex lock only for write purposes. Below shows some screenshots of where the mutex locks are used in the explained instances.

```
pthread_mutex_lock(&mutex); // mutex lock acquired

if (shared_data->index < 4)
{
    shared_data->SimpleTID[shared_data->index] = tid;
    shared_data->index++;

    for(int i = 0; i < 4; i++)
    {
        printf("\nthread ID stored at index %d = %ld", i, shared_data->SimpleTID[i]);
    }
    printf("\n");

    assignTask(shared_data); // moving function call inside the lock prevents potential race conditions
}

pthread_mutex_unlock(&mutex); // mutex lock is released
```

Mutex Locks are used in further functions (Most often before the mutex lock is released)

Above: shows critical section code in buildThread function which shows an active lock after the thread moves on

```
/* critical section synchronization
Execution path of critical section
mutex lock acquired values from check are added to
Sub (if number != 0) i = 1
otherwise its 0
sub is gone over and numbers from it are added to counter
if Sub contains any 0s access Invalid Subsections and make index of subsection be 1 to indicate invalid subsection
reset sub back and check arrays back to 0
release mutex lock*/
pthread_mutex_lock(&mutex2); // mutex2 used to prevent deadlocks

index = 0;

for(int i = 0; i < 3; i++) // check array data is added to subsection array
{
    for(int j = 0; j < 3; j++)
    {
        shared_data->Sub[i][j] = check[index++];

        if(shared_data->Sub[i][j] != 0) // if not invalid data becomes 1
        {
            shared_data->Sub[i][j] = 1;
        }
        else // if Sub has 0s it is an invalid subsection
        {
            shared_data->InvalidSubsections[stage + invalidIndexHelper] = 1;
        }
    }
}

IncrementCounter = true; // set to true otherwise set to false later in the check
for(int i = 0; i < 3; i++) // this block checks the contents of Sub grid, if any 0s are found the eligibility of the incre
{
    for(int j = 0; j < 3; j++)
    {
        if(shared_data->Sub[i][j] != 1)
        {
            IncrementCounter = false;
        }
    }
}
```

Left:
Shows
where
mutex 2 is
used

Critical section of thread 1, 2
and 3 (similar in thread 4)

Contains steps of critical section
(uses mutex2)

Instances of Program not working correctly:

MSSV has been extensively tested to the point where there are no known faults. The reason for such robust software is because of the lack of input as well as error handling being used at the start during the command line input which exits the program if invalid inputs are used. If the input file is in a wrong format and contains characters instead of ints they are converted to 0 by default in fileIO.c therefore an expected result occurs.

During development, a tool called helgrind part of the valgrind suite was used to detect thread errors and potential race conditions. Some shared data is also reset after every step, which includes Row, Col, and Sub arrays. The contents of these arrays had to be printed out to stdout and analysed to ensure current output.

Finally, a partition check is done when all the threads are finished executing this checks the contents of the SimpleTID array and was used to compare the IDs of the threads whilst they were in execution to make sure each thread ID lines up with its index ID.

Sample Inputs:

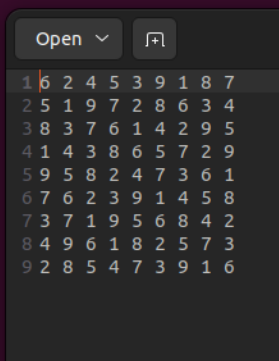
Below: input Solution.txt with a 1-second delay (showing a valid output)

```
Thread ID-4: true thread id ---> 126633505711680 is the last thread

PARTITION CHECK
tid (Thread ID) array thread at location 0 = 126633514104384
tid (Thread ID) array thread at location 1 = 126633497318976
tid (Thread ID) array thread at location 2 = 126633419732544
tid (Thread ID) array thread at location 3 = 126633505711680

Thread ID-1: valid
Thread ID-2: valid
Thread ID-3: valid
Thread ID-4: valid

There are 27 valid sub-grids, and thus the solution is valid.
```



	1	2	3	4	5	6	7	8	9
1	6	2	4	5	3	9	1	8	7
2	5	1	9	7	2	8	6	3	4
3	8	3	7	6	1	4	2	9	5
4	1	4	3	8	6	5	7	2	9
5	9	5	8	2	4	7	3	6	1
6	7	6	2	3	9	1	4	5	8
7	3	7	1	9	5	6	8	4	2
8	4	9	6	1	8	2	5	7	3
9	2	8	5	4	7	3	9	1	6

Below: input Solution2.txt with a 1-second delay (showing an invalid output with 18 valid subsections, rows, and columns)

```
administrator@administrator-virtual-machine: ~/Desktop/UNI/COMP2006/Assingment/MSSV8

Thread ID-4: true thread id ---> 125752548783680 is the last thread

PARTITION CHECK
tid (Thread ID) array thread at location 0 = 125752647349824
tid (Thread ID) array thread at location 1 = 125752638957120
tid (Thread ID) array thread at location 2 = 125752630564416
tid (Thread ID) array thread at location 3 = 125752548783680

Thread ID-1: Subgrid 2 Row 3 is/are invalid
Thread ID-2: Subgrid 4 Row 5 is/are invalid
Thread ID-3: Subgrid 9 Row 8 is/are invalid
Thread ID-4: Column 3 Column 5 Column 7 is/are invalid

There are in total 18 valid rows, columns, and sub-grids, and the solution is invalid.
administrator@administrator-virtual-machine:~/Desktop/UNI/COMP2006/Assingment/MSSV8$
```

Open	
1	6 2 4 5 3 9 1 8 7
2	5 1 9 7 2 8 6 3 4
3	8 3 7 6 5 4 2 9 5
4	1 4 3 8 6 5 7 2 9
5	9 5 2 2 4 7 3 6 1
6	7 6 2 3 9 1 4 5 8
7	3 7 1 9 5 6 8 4 2
8	4 9 6 1 8 2 1 7 3
9	2 8 5 4 7 3 9 1 6

Below: input Solution3.txt with a 1-second delay (invalid output with 0 valid parts)

```
administrator@administrator-virtual-machine: ~/Desktop/UNI/COMP2006/Assingment/MSSV8

Thread ID-4: true thread id ---> 138975410910784 is the last thread

PARTITION CHECK
tid (Thread ID) array thread at location 0 = 138975553517120
tid (Thread ID) array thread at location 1 = 138975536731712
tid (Thread ID) array thread at location 2 = 138975545124416
tid (Thread ID) array thread at location 3 = 138975410910784

Thread ID-1: Subgrid 1 Subgrid 2 Subgrid 3 Row 1 Row 2 Row 3 is/are invalid
Thread ID-2: Subgrid 4 Subgrid 5 Subgrid 6 Row 4 Row 5 Row 6 is/are invalid
Thread ID-3: Subgrid 7 Subgrid 8 Subgrid 9 Row 7 Row 8 Row 9 is/are invalid
Thread ID-4: Column 1 Column 2 Column 3 Column 4 Column 5 Column 6 Column 7 Column 8 Column 9 is/are invalid

There are in total 0 valid rows, columns, and sub-grids, and the solution is invalid.
```

Open	
1	5 5 7 5 5 9 7 5 5
2	8 5 5 8 6 5 7 5 4
3	5 9 5 5 5 8 5 5 5
4	7 5 4 2 5 6 5 3 5
5	5 5 2 5 5 5 7 5 4
6	5 8 5 2 5 8 5 5 5
7	7 3 9 5 7 5 5 1 5
8	5 5 5 5 8 7 5 5
9	5 1 1 5 9 5 5 5 8

Below: output Solution4.txt with a 1-second delay (invalid output with 0 valid parts as well as the solution file being filled with all 5s)

```
Open [v] [f]
1 5 5 5 5 5 5 5 5
2 5 5 5 5 5 5 5 5
3 5 5 5 5 5 5 5 5
4 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5
6 5 5 5 5 5 5 5 5
7 5 5 5 5 5 5 5 5
8 5 5 5 5 5 5 5 5
9 5 5 5 5 5 5 5 5

administrator@administrator-virtual-machine: ~/Desktop/UNI/COMP20

Thread ID-4: true thread id ----> 133741682878016 is the last thread

PARTITION CHECK
tid (Thread ID) array thread at location 0 = 133741674485312
tid (Thread ID) array thread at location 1 = 133741666092608
tid (Thread ID) array thread at location 2 = 133741691270720
tid (Thread ID) array thread at location 3 = 133741682878016

Thread ID-1: Subgrid 1 Subgrid 2 Subgrid 3 Row 1 Row 2 Row 3 is/are invalid
Thread ID-2: Subgrid 4 Subgrid 5 Subgrid 6 Row 4 Row 5 Row 6 is/are invalid
Thread ID-3: Subgrid 7 Subgrid 8 Subgrid 9 Row 7 Row 8 Row 9 is/are invalid
Thread ID-4: Column 1 Column 2 Column 3 Column 4 Column 5 Column 6 Column 7 Column 8 Column 9 is/are invalid

There are in total 0 valid rows, columns, and sub-grids, and the solution is invalid.
```

Below output Solution5.txt with a 1-second delay (invalid file format with letters, 0 valid parts)

```
Open [v] [f] solution
~/Desktop/UNI/COMP200
1 a b c d e f g h i
2 j k l m n o p q r
3 s t u v w x y z a
4 b c d e f g h i j
5 k l m n o p q r s
6 t u v w x y z a b
7 c d e f g h i j k
8 l m n o p q r s t
9 u v w x y z a b c

administrator@administrator-virtual-machine: ~/Desktop/UNI/COMP200

Thread ID-4: true thread id ----> 134035972019776 is the last thread

PARTITION CHECK
tid (Thread ID) array thread at location 0 = 134035988805184
tid (Thread ID) array thread at location 1 = 134035837802048
tid (Thread ID) array thread at location 2 = 134035980412480
tid (Thread ID) array thread at location 3 = 134035972019776

Thread ID-1: Subgrid 1 Subgrid 2 Subgrid 3 Row 1 Row 2 Row 3 is/are invalid
Thread ID-2: Subgrid 4 Subgrid 5 Subgrid 6 Row 4 Row 5 Row 6 is/are invalid
Thread ID-3: Subgrid 7 Subgrid 8 Subgrid 9 Row 7 Row 8 Row 9 is/are invalid
Thread ID-4: Column 1 Column 2 Column 3 Column 4 Column 5 Column 6 Column 7 Column 8 Column 9 is/are invalid

There are in total 0 valid rows, columns, and sub-grids, and the solution is invalid.
```