COMP2006 - Operating Systems

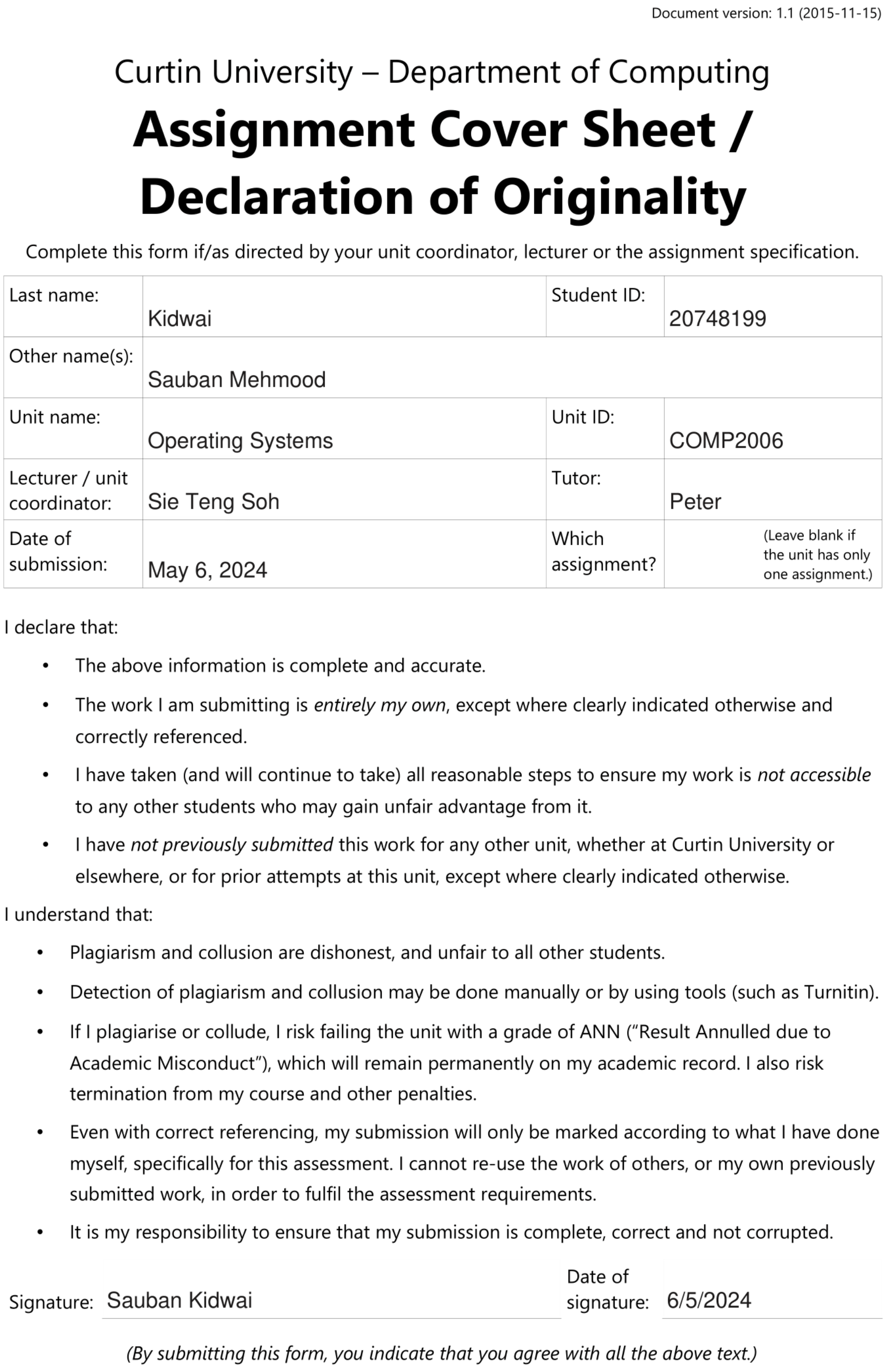
Final Assignment

Assignment Report

By: Sauban Kidwai

ID: 20748199

Tutorial Class (Building: 502C.101 | Time: 2pm - 3pm | Day: Every Wednesday)



Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Kidwai	Student ID:	20748199
Other name(s):	Sauban Mehmood		
Unit name:	Operating Systems	Unit ID:	COMP2006
Lecturer / unit coordinator:	Sie Teng Soh	Tutor:	Peter
Date of submission:	May 6, 2024	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: Sauban Kidwai Date of signature: 6/5/2024

(By submitting this form, you indicate that you agree with all the above text.)

Table of contents

How to Use

To try this program for yourself, please read the README File to compile and Run the Assignment. The README file has been provided as a pdf and a markdown file format for easy readability

File Descriptions

The Summary of all the files contained within the Assignemnt are as follows.

File	Description
makefile	Contains the code required to compile and run the program
mssv.c	Contains the Main function for the Assignment
validate.c	Contains the Validation functions for the Assignment
sudoku.h	Header File containing global variables to be shared between `validate.c` and `mssv.c`
sol1.txt	Contains one example of a correct Sudoku Solution
sol2.txt	Contains one example of a Correct Sudoku Solution
sol3.txt	Contains one example of an Incorrect Sudoku Solution
sol4.txt	Contains one example of an Incorrect Sudoku Solution
sol5.txt	Contains an example of an incorrect Sudoku Grid Size

Source Code

makefile

The makefile is crucial in this assignment as it allows for easy compilation of the program. For more information on how to use it, please refer to the README File in either pdf or markdown format

```
CC=gcc
CFLAGS=-Wall -g -s -fsanitize=address -pthread

mssv: mssv.o validate.o
    $(CC) $(CFLAGS) -o mssv mssv.o validate.o

mssv.o: mssv.c sudoku.h
    $(CC) $(CFLAGS) -c mssv.c

validate.o: validate.c sudoku.h
    $(CC) $(CFLAGS) -c validate.c

clean:
    rm -f *.o mssv
```

mssv.c

This c file contains the Main function of the Assignment. This is responsible for the operation of the program.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "sudoku.h"

#define GRID_SIZE 9

// Shared resources
int Sol[GRID_SIZE][GRID_SIZE];
int Row[GRID_SIZE], Col[GRID_SIZE], Sub[GRID_SIZE];
int Counter;
int threads_completed = 0;
pthread_t last_thread_id = 0;
pthread_mutex_t mutexCounter;
pthread_cond_t allDone;

int main(int argc, char *argv[]) {
    // Check for correct usage
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <filepath> <delay>\n", argv[0]);
        return EXIT_FAILURE;
    }

    // Open the input file
    FILE *file = fopen(argv[1], "r");
    if (!file) {
        perror("Failed to open file");
        return EXIT_FAILURE;
    }

    // Read the 9x9 grid from the input file
    for (int i = 0; i < GRID_SIZE; i++) {
        for (int j = 0; j < GRID_SIZE; j++) {
            if (fscanf(file, "%d", &Sol[i][j]) != 1) {
                fprintf(stderr, "Error: Incorrect input file, expected a 9x9
grid.\n");
                fclose(file);
                return EXIT_FAILURE;
            }
        }
    }
    fclose(file);

    // Initialize synchronization primitives
    pthread_mutex_init(&mutexCounter, NULL);
    pthread_cond_init(&allDone, NULL);
    printf("\nSudoku puzzle Validator:\n\n");

    pthread_t threads[4];
    // Get the delay value from the command line arguments and check it does not exceed 10
    int delay = atoi(argv[2]);
    if (delay > 10 || delay < 1) {
        fprintf(stderr, "Error: Delay value should be between 1 and 10\n");
        return EXIT_FAILURE;
    }
    int ranges[4][2] = {{0, 2}, {3, 5}, {6, 8}, {0, 8}};
    void** results = (void**)malloc(4 * sizeof(void*));

    // Initialize the results array
    for (int i = 0; i < 4; i++) {
        results[i] = NULL;
    }

    // Create 4 threads for validation
    for (int i = 0; i < 4; i++) {
        if (i < 3) {
            pthread_create(&threads[i], NULL, validate_rows_and_subgrids, &ranges[i]);
        } else {
            pthread_create(&threads[i], NULL, validate_columns, &delay);
        }
    }

    // Wait for all threads to complete
    pthread_mutex_lock(&mutexCounter);
    while (threads_completed < 4) {
        pthread_cond_wait(&allDone, &mutexCounter);
    }
    pthread_mutex_unlock(&mutexCounter);

    printf("\n");

    // Join the threads and display the results
    for (int i = 0; i < 4; i++) {
        if (pthread_join(threads[i], &results[i]) != 0) {
            fprintf(stderr, "Error joining thread %d\n", i);
            continue;
        }
        ThreadResult* res = (ThreadResult*)results[i];
        if (res == NULL) {
            fprintf(stderr, "Error: Thread %d did not return a valid result\n", i);
        } else if (res->valid) {
            printf("Thread ID-%d (%lu): valid\n", i + 1, (unsigned long)res->thread_id);
        } else {
            printf("Thread ID-%d (%lu): %sinvalid\n", i + 1, (unsigned long)res->thread_id, res->details);
        }
```

```c
        free(res);
    }

    // Display the final result
    printf("\nThere are %d valid rows, columns, and sub-grids, and thus the solution is %s.\n\n",
           Counter, Counter == 27 ? "valid" : "invalid");

    // Clean up
    free(results);
    pthread_mutex_destroy(&mutexCounter);
    pthread_cond_destroy(&allDone);

    return EXIT_SUCCESS;
}
```

validate.c

This c file contains the necessary validation functions for the Assignment. Tese functions are then called in `mssv.c`

```
#include "sudoku.h"
#include <stddef.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

// Function to check a row of the Sudoku grid
int check_row(int row) {
    int freq[GRID_SIZE + 1] = {0};
    for (int col = 0; col < GRID_SIZE; col++) {
        int num = Sol[row][col];
        if (num < 1 || num > 9 || freq[num] == 1) {
            return 0;  // Row is invalid
        }
        freq[num]++;
    }
    return 1;  // Row is valid
}

// Function to check a column of the Sudoku grid
int check_column(int col) {
    int freq[GRID_SIZE + 1] = {0};
    for (int row = 0; row < GRID_SIZE; row++) {
        int num = Sol[row][col];
        if (num < 1 || num > 9 || freq[num] == 1) {
            return 0;  // Column is invalid
        }
        freq[num]++;
    }
    return 1;  // Column is valid
}

// Function to check a 3x3 subgrid of the Sudoku grid
int check_subgrid(int startRow, int startCol) {
    int freq[GRID_SIZE + 1] = {0};
    for (int row = startRow; row < startRow + 3 && row < GRID_SIZE; row++) {
        for (int col = startCol; col < startCol + 3 && col < GRID_SIZE; col++) {
            int num = Sol[row][col];
            if (num < 1 || num > 9 || freq[num] == 1) {
                return 0;  // Subgrid is invalid
            }
            freq[num]++;
        }
    }
    return 1;  // Subgrid is valid
}

// Thread function to validate rows and subgrids
void* validate_rows_and_subgrids(void* arg) {
    int* range = (int*)arg;
    ThreadResult* result = (ThreadResult*)malloc(sizeof(ThreadResult));
    result->valid = 1;
    result->thread_id = pthread_self();
    result->details[0] = '\0';

    // Print which thread is working
    pthread_t self_id = pthread_self();
    printf("Thread ID: %lu is checking the rows and subgrids.....\n", (unsigned long)self_id);

    // Validate the rows and subgrids
    pthread_mutex_lock(&mutexCounter);
    for (int i = range[0]; i <= range[1] && i < GRID_SIZE; i++) {
        if (!check_row(i)) {
            sprintf(result->details + strlen(result->details), "row %d, ", i + 1);
            result->valid = 0;
        } else {
            Row[i] = 1;
            Counter++;
        }

        int subgridIndex = (i / 3) * 3 + (i % 3);
        if (!check_subgrid((i / 3) * 3, (i % 3) * 3)) {
            sprintf(result->details + strlen(result->details), "sub-grid %d, ", subgridIndex + 1);
            result->valid = 0;
        } else {
            Sub[subgridIndex] = 1;
            Counter++;
        }
    }
    // Update completion status
    threads_completed++;
    last_thread_id = self_id;  // Update last thread ID
    if (threads_completed == 4) {
        printf("\n\nThread ID-%lu is the last thread!\n\n", (unsigned long)self_id);
        pthread_cond_signal(&allDone);
    }
    pthread_mutex_unlock(&mutexCounter);

    return result;
}

// Thread function to validate columns
void* validate_columns(void* arg) {
    int delay = *(int*)arg;
    ThreadResult* result = (ThreadResult*)malloc(sizeof(ThreadResult));
    result->valid = 1;
    result->thread_id = pthread_self();
    result->details[0] = '\0';

    // Print which thread is working
    pthread_t self_id = pthread_self();
    printf("Thread ID: %lu is checking the columns......\n", (unsigned long)self_id);

    // Validate the columns
    pthread_mutex_lock(&mutexCounter);
    for (int col = 0; col < GRID_SIZE; col++) {
        sleep(delay);
        if (!check_column(col)) {
            sprintf(result->details + strlen(result->details), "column %d, ", col + 1);
            result->valid = 0;
            Col[col] = 0;
        } else {
            Col[col] = 1;
            Counter++;
        }
    }
    // Update completion status
    threads_completed++;
    last_thread_id = self_id;
    if (threads_completed == 4) {
        printf("\n\nThread ID-%lu is the last thread!\n\n", (unsigned long)self_id);
        pthread_cond_signal(&allDone);
    }
    pthread_mutex_unlock(&mutexCounter);

    return result;
}
```

sudoku.h

This header file contains the necessary global variables required for `validate.c` and `mssv.c` to share and use together. It also includes Function Declarations and a Thread Result Struct.

```
#ifndef SUDOKU_H
#define SUDOKU_H

#include <pthread.h>

#define GRID_SIZE 9

// Shared variables for the Sudoku grid and validation
extern int Sol[GRID_SIZE][GRID_SIZE];
extern int Row[GRID_SIZE], Col[GRID_SIZE], Sub[GRID_SIZE];
extern int Counter;
extern int threads_completed;
extern pthread_t last_thread_id;
extern pthread_mutex_t mutexCounter;
extern pthread_cond_t allDone;

// Structure to hold the result of a thread's validation
typedef struct {
    int valid;                   // Indicates if the validation was successful
    pthread_t thread_id;         // ID of the thread
    char details[256];           // Details of the validation result
} ThreadResult;

// Function declarations for validating rows, columns, and subgrids
int check_row(int row);
int check_column(int col);
int check_subgrid(int startRow, int startCol);
void* validate_rows_and_subgrids(void* arg);
void* validate_columns(void* arg);

#endif
```

Sudoku Solutions

The following 5 text files include the different Sudoku grids that are used for the testing of the program. Please read the README file to learn how to use these files with the assignment.

```


sol1.txt

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


sol2.txt

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


sol3.txt

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


sol4.txt

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

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

sol5.txt

5 3 4 6 0 8
6 7 2 1 9 5
1 9 8 3 4 2
8 5 9 7 6 1
4 2 6 8 5 3
7 1 3 9 2 4
```

Discussion of Assignment

Synchronisation

Synchronisation is an important aspect of this assignment because multiple threads will be validating different parts of the sudoku grid, while accessing the same shared variables. To avoid and race conditions or deadlocks, here is an overview of how synchronisation was attempted for this Assignment.

1. Shared Variables

    - The shared variables in this assignment were:
        - `sol` - Sudoku Grid Solution.
        - `Row` - Validation Result of a row.
        - `Col` - Validation Result of Columns.
        - `Sub` - Validation Result of Subgrids.
        - `Counter` - A Counter for valid Segments.
        - `threads_completed` - Thread Completion Status
    - Both thread functions (`validate_rows_and_subgrids` and `validate_columns`) access shared resources.
    - Each thread locks the mutex before accessing or modifying these resources and unlocks the mutex afterward.

2. Thread Functions

    - There are 2 main Thread Functions within this Assignment.
        - `validate_rows_and_subgrids` - This function is resposibele for checking rows, columns and subgrids and to determine whether the solution is valid or invalid.
        - `validate_columns` - This function is responsible to check and determine that all 9 columns are either valid or invalid.

3. Synchronization Mechanisms:

    - Mutex:

        - The `pthread_mutex_lock` and `pthread_mutex_unlock` functions are used to protect access to the shared variables.
        - This ensures that only one thread can modify the shared variables at a time, preventing race conditions.
        - The mutex is used when updating `Counter`, `threads_completed`, and other shared variables.

    - Condition Variable:

        - The condition variable (`allDone`) is used to signal the main thread that all threads have finished their tasks.
        - The main thread waits for all threads to complete using `pthread_cond_wait` inside a mutex lock (`pthread_mutex_lock`).
        - When the last thread completes, it signals the main thread using `pthread_cond_signal`.

4. Thread Completion:

    - The variable `threads_completed` is used to keep track of the number of threads that have finished.
    - When `threads_completed` reaches 4, the last thread signals the main thread.
    - The variable `last_thread_id` stores the ID of the last thread that finished, which is then printed by the main thread (in `mssv.c`).

Test Cases

These are some examples of Test cases that were tested to ensure good functionality of the Program

Test Case 1:

If there are invalid arguments at command line, program should exit gracefully with error

INPUT -- `./mssv`

OUTPUT -- `Usage: ./mssv <filepath> <delay>`

Test Case 2:

If delay value is not between 1 and 10, program should exit gracefully with error.

INPUT `./mssv sol1.txt 20`

OUTPUT `Error: Delay value should be between 1 and 10`

Test Case 3:

If the file passed into the argument is an invalid or non-existent file, the program should exit gracefully with error.

INPUT `./mssv nosol.txt 1`

OUTPUT `Failed to open file: No such file or directory`

Test Case 4:

When validating sol1.txt - Answer should be all columns Valid. (NOTE: The Thread ID will actually change each iteration, so for readability, this report will use Thread ID - 1 to 4)

INPUT `./mssv sol1.txt`

OUTPUT

```
Sudoku puzzle Validator:

Thread ID: 140266256201280 is checking the rows and subgrids.....
Thread ID: 140266247808576 is checking the rows and subgrids.....
Thread ID: 140266239415872 is checking the rows and subgrids.....
Thread ID: 140266231023168 is checking the columns......


Thread ID-140266231023168 is the last thread!


Thread ID-1 (140266256201280): valid
Thread ID-2 (140266247808576): valid
Thread ID-3 (140266239415872): valid
Thread ID-4 (140266231023168): valid

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

Test Case 5:

When validating sol4.txt - Answer should be invalid. (NOTE: The Thread ID will actually change each iteration, so for readability, this report will use Thread ID - 1 to 4)

INPUT `./mssv sol1.txt`

OUTPUT

```

Sudoku puzzle Validator:

Thread ID: 139627477333568 is checking the rows and subgrids.....
Thread ID: 139627468940864 is checking the rows and subgrids.....
Thread ID: 139627460548160 is checking the rows and subgrids.....
Thread ID: 139627452155456 is checking the columns......


Thread ID-139627452155456 is the last thread!


Thread ID-1 (139627477333568): sub-grid 1, sub-grid 2, sub-grid 3, invalid
Thread ID-2 (139627468940864): sub-grid 4, sub-grid 5, sub-grid 6, invalid
Thread ID-3 (139627460548160): sub-grid 7, sub-grid 8, sub-grid 9, invalid
Thread ID-4 (139627452155456): column 1, column 2, column 3, column 4, column 5,
column 6, column 7, column 8, column 9, invalid

There are 9 valid rows, columns, and sub-grids, and thus the solution is invalid.
```

Test Case 6:

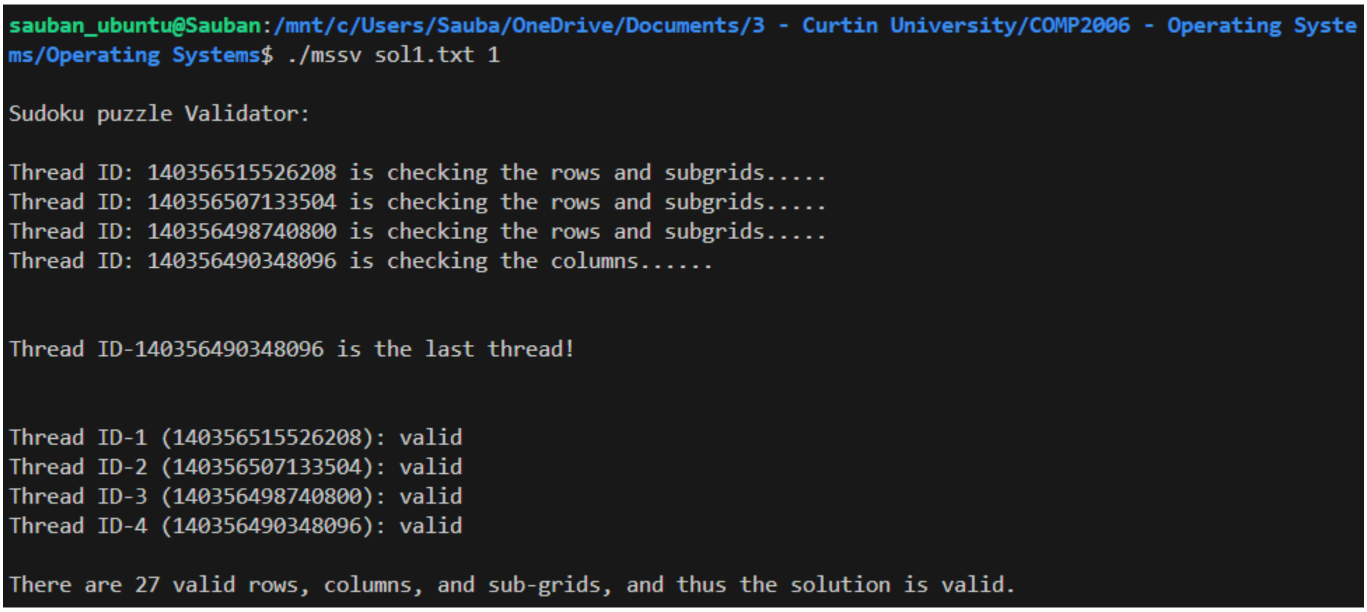
When validating sol5.txt - An Error should occur as the grid size is not valid

INPUT `./mssv sol1.txt`

OUTPUT `Error: Incorrect input file, expected a 9x9 grid.`

Sample Outputs

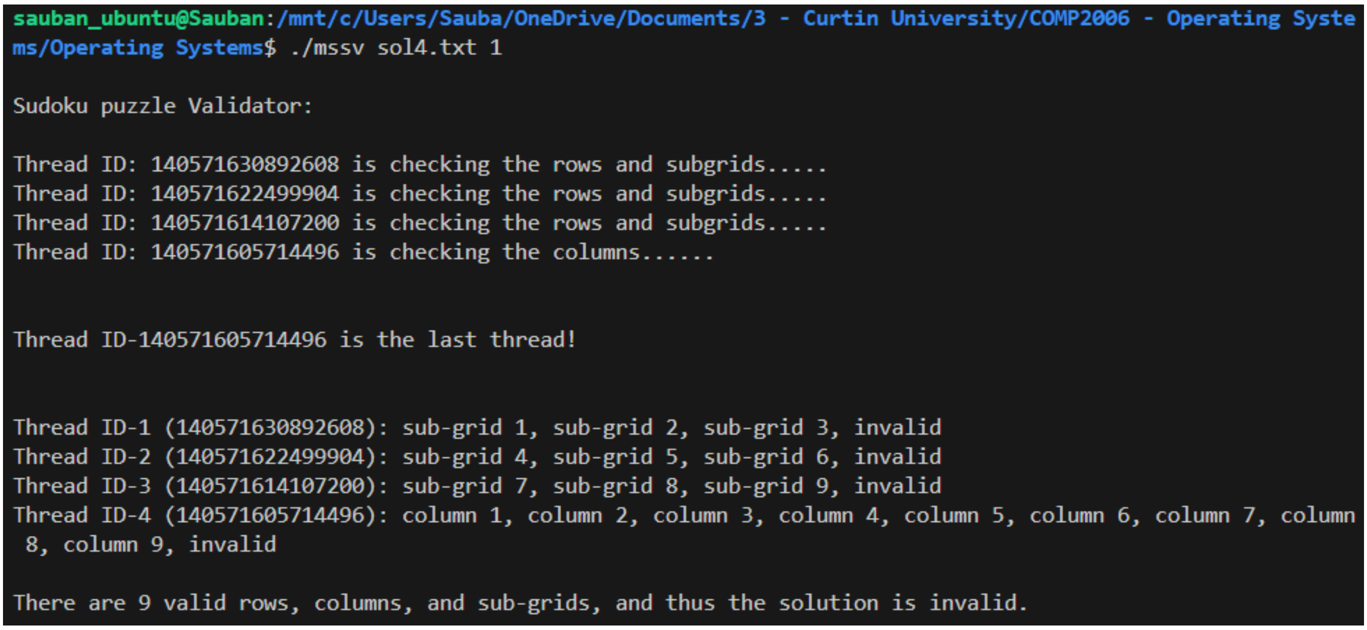
Correct Solution Output of sol1.txt



```
sauban_ubuntu@Sauban:/mnt/c/Users/Sauba/OneDrive/Documents/3 - Curtin University/COMP2006 - Operating Syste
ms/Operating Systems$ ./mssv sol1.txt 1

Sudoku puzzle Validator:

Thread ID: 140356515526208 is checking the rows and subgrids.....
Thread ID: 140356507133504 is checking the rows and subgrids.....
Thread ID: 140356498740800 is checking the rows and subgrids.....
Thread ID: 140356490348096 is checking the columns......


Thread ID-140356490348096 is the last thread!


Thread ID-1 (140356515526208): valid
Thread ID-2 (140356507133504): valid
Thread ID-3 (140356498740800): valid
Thread ID-4 (140356490348096): valid

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

This output is correct as all rows, subgrids and columns are valid

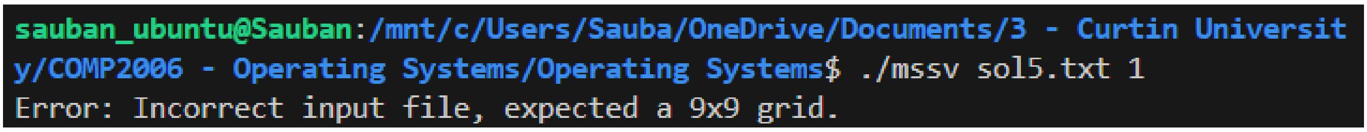
Correct Solution Output of sol4.txt



```
sauban_ubuntu@Sauban:/mnt/c/Users/Sauba/OneDrive/Documents/3 - Curtin University/COMP2006 - Operating Syste
ms/Operating Systems$ ./mssv sol4.txt 1

Sudoku puzzle Validator:

Thread ID: 140571630892608 is checking the rows and subgrids.....
Thread ID: 140571622499904 is checking the rows and subgrids.....
Thread ID: 140571614107200 is checking the rows and subgrids.....
Thread ID: 140571605714496 is checking the columns......


Thread ID-140571605714496 is the last thread!


Thread ID-1 (140571630892608): sub-grid 1, sub-grid 2, sub-grid 3, invalid
Thread ID-2 (140571622499904): sub-grid 4, sub-grid 5, sub-grid 6, invalid
Thread ID-3 (140571614107200): sub-grid 7, sub-grid 8, sub-grid 9, invalid
Thread ID-4 (140571605714496): column 1, column 2, column 3, column 4, column 5, column 6, column 7, column
 8, column 9, invalid

There are 9 valid rows, columns, and sub-grids, and thus the solution is invalid.
```

This output is correct as all rows, subgrids and columns are invalid

Correct Solution Output of sol5.txt



```
sauban_ubuntu@Sauban:/mnt/c/Users/Sauba/OneDrive/Documents/3 - Curtin Universit
y/COMP2006 - Operating Systems/Operating Systems$ ./mssv sol5.txt 1
Error: Incorrect input file, expected a 9x9 grid.
```

This output is correct as the grid size is invalid