Operating Systems Assignment
COMP2006

Multitasked Sudoku Solution Validator

by

Buddhika Jayanarth
Benthara Poramba Badalge

Curtin ID 19201973

Signature
08 May 2017

Contents

ProcessProgram.c

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <pthread.h>
#include "FunctionsP.h"


int main(int argc, char *argv[])
{
        int pid;
        int shmidb1;
        int shmidb2;
        int shmidc;
        int *Buffer1;
        int *Buffer2;
        int *counter;
        int keyb1 = 11111;
        int keyb2 = 22222;
        int keyc = 33333;
        int sizeb1 = 81;
        int sizeb2 = 11;
        int sizec = 1;
        int position;

        //(re)create log file

        FILE *logf = fopen("ProcessLogFile.txt", "w");
        fprintf(logf, "-----||Log file for Process Program||----- \n \n");
        fclose(logf);

        //initialising semaphores
        sem_init(&Buffer2Sem, 1, 1);
        sem_init(&CounterSem, 1, 1);

        int timedelay = atoi(argv[2]);

        //creating shared memories
        shmidb1 = shmget(keyb1, sizeb1, IPC_CREAT | 0775);
        shmidb2 = shmget(keyb2, sizeb2, IPC_CREAT | 0775);
        shmidc = shmget(keyc, sizec, IPC_CREAT | 0775);

        //attaching Buffer 1, Buffer 2, Counter pointers to reference shared
        //memory
        Buffer1 = (int *) shmat(shmidb1, NULL, 0);
        Buffer2 = (int *) shmat(shmidb2, NULL, 0);
        counter = (int *) shmat(shmidc, NULL, 0);

        *counter = 0;

        //reading and assign to buffer1
        populateBuffer1(argv[1], keyb1, sizeb1);
```

```
//create 11 child processes for 3 groups of tasks
for (position = 0; position <91; position = position +9)
{

        if (position<73)
        {
                //create process for group 1
                pid = fork();
                if(pid == -1)
                {
                        printf("Group1 child failed");
                }
                else if(pid == 0)
                {
                        //a child process for Group 1 task
                        Group1task(position, keyb1, sizeb1, keyb2,
                     sizeb2, keyc, sizec, timedelay);

                        exit(0);
                }
                else
                {
                        //parent process

                }

        }
        else
        {
                if (position==81)
                {
                        //create process for group 2
                        pid = fork();
                        if(pid == -1)
                        {
                                printf("Group2 child failed");
                        }
                        else if(pid == 0)
                        {
                                //a child process for Group 2 task
                                Group2task(keyb1, sizeb1, keyb2,
                           sizeb2, keyc, sizec, timedelay);

                                exit(0);
                        }
                        else
                        {
                                //parent process

                        }
                }

                else
                {
                        //create process for group 3
                        pid = fork();
                        if(pid == -1)
```

```
                        {
                                printf("Group3 child failed");
                        }
                        else if(pid == 0)
                        {
                                //a child process for Group 3 task
                                Group3task(keyb1, sizeb1, keyb2,
                           sizeb2, keyc, sizec, timedelay);

                                exit(0);
                        }
                        else
                        {
                                //parent process

                        }
                }
        }
}

//Wait for all child processes to terminate before waking parent
for (int i = 0; i<11; i++)
{
wait(NULL);
}

//Display Solution Summary
printf("\nCount of valid rows, columns and subgrids: %d ",
    *counter);

if (*counter == 27)
{
        printf("and thus solution is valid\n");
}
else{
        printf("and thus solution is invalid\n");
}

//detach shared memories
shmdt((void*) counter);
shmdt(Buffer1);
shmdt(Buffer2);

//deallocate shared memory space
shmctl(shmidb1,IPC_RMID, NULL);
shmctl(shmidb2,IPC_RMID, NULL);
shmctl(shmidc,IPC_RMID, NULL);

return 0;
}
```

FunctionsP.c

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <pthread.h>
#include "FunctionsP.h"


void Group1task(int position, int keyb1, int sizeb1, int keyb2, int sizeb2,
int keyc, int sizec, int timedelay)
{

        int shmidb2 = shmget(keyb2, sizeb2, IPC_CREAT | 0775);
        int *Buffer2 = (int *) shmat(shmidb2, NULL, 0);

        int shmidc = shmget(keyc, sizec, IPC_CREAT | 0775);
        int *counter = (int *) shmat(shmidc, NULL, 0);

        int validrow = 0;

        //verify row
        validrow = verifyrow(position, keyb1, sizeb1);

        //display result on screen
        printf("Validation result from PID-%u Row %d is valid \n",(unsigned
int)pthread_self(), (position/9)+1);

        //timedelay
        sleep(timedelay);

        //write to buffer2
        sem_wait(&Buffer2Sem);
        Buffer2[position/9] = validrow;
        sem_post(&Buffer2Sem);

        //update counter
        sem_wait(&CounterSem);
        *counter = *counter + validrow;
        sem_post(&CounterSem);
        shmdt(Buffer2);

        shmdt((void*) counter);
}


void Group2task(int keyb1, int sizeb1, int keyb2, int sizeb2, int keyc, int
sizec, int timedelay)
{

        int shmidb2 = shmget(keyb2, sizeb2, IPC_CREAT | 0775);
        int *Buffer2 = (int *) shmat(shmidb2, NULL, 0);
```

```
        int shmidc = shmget(keyc, sizec, IPC_CREAT | 0775);
        int *counter = (int *) shmat(shmidc, NULL, 0);

        int validcol =0;

        //verify all columns
        validcol = verifycol(keyb1, sizeb1);

        //display result on screen
        printf("Validation result from PID-%u %d of 9 columns are valid \n",
(unsigned int)pthread_self(), validcol);
        //timedelay
        sleep(timedelay);

        //update buffer2
        sem_wait(&Buffer2Sem);
        Buffer2[9] = validcol;
        sem_post(&Buffer2Sem);

        //update counter
        sem_wait(&CounterSem);
        *counter = *counter + validcol;
        sem_post(&CounterSem);

        shmdt(Buffer2);

        shmdt((void*) counter);

}

void Group3task(int keyb1, int sizeb1, int keyb2, int sizeb2, int keyc, int
sizec, int timedelay)
{

        int shmidb2 = shmget(keyb2, sizeb2, IPC_CREAT | 0775);
        int *Buffer2 = (int *) shmat(shmidb2, NULL, 0);

        int shmidc = shmget(keyc, sizec, IPC_CREAT | 0775);
        int *counter = (int *) shmat(shmidc, NULL, 0);

        int validgrid =0;

        //verify all subgrids
        validgrid = verify3x3subgrid(keyb1, sizeb1);

        //display result on screen
        printf("Validation result from PID-%u %d of 9 3x3 subgrids are valid
\n",(unsigned int)pthread_self(), validgrid);
        //timedelay
        sleep(timedelay);

        //update buffer2
        sem_wait(&Buffer2Sem);
        Buffer2[10] = validgrid;
        sem_post(&Buffer2Sem);

        //update counter
        sem_wait(&CounterSem);
        *counter = *counter + validgrid;
```

```
        sem_post(&CounterSem);

        shmdt(Buffer2);

        shmdt((void*) counter);

}

void populateBuffer1(char *filename, int keyb1, int sizeb1)
{
        int *Buffer1;
        int shmidb1 = shmget(keyb1, sizeb1, IPC_CREAT | 0775);
        Buffer1 = (int *) shmat(shmidb1, NULL, 0);
        //opening sudoku solution file for reading
        FILE *f;
        f = fopen(filename, "r");

        if(f == NULL)
        {
                printf("Input file error");
        }
        else
        {
                printf("Input file succesfully read \n");
                //entering solution into Buffer1 array
                for (int i = 0; i<81; i++)
                {
                        fscanf(f, "%d", &Buffer1[i]);

                }

                fclose(f);
        }
        shmdt(Buffer1);
}

//takes starting position of row and compares the 9 numbers from there.
//Return 1 if all numbers are unique and only comprises of 1-9 digits.
int verifyrow(int position, int keyb1, int sizeb1)
{

        int *Buffer1;
        int shmidb1 = shmget(11111, 81, IPC_CREAT | 0775);
        Buffer1 = (int *) shmat(shmidb1, NULL, 0);

        int valid;
        int numofInvalid=0;
        int tempbuf[9];


        for (int k = position; k < position+9; k++)
        {

                if (Buffer1[k] > 9)
                {
                        numofInvalid++;

                }
                else if (Buffer1[k] < 1)
```

```
                {
                        numofInvalid++;
                }

        }

        if (numofInvalid == 0)
        {
                for (int i = position; i < position+8; i++)
                {
                        for (int j = i + 1; j< position+9; j++)
                        {
                                if (Buffer1[i] == Buffer1[j])
                                {
                                        numofInvalid++;

                                }
                        }
                }
        }

        if(numofInvalid>0)
        {
                valid=0;
                //append log file
                FILE *logf = fopen("ProcessLogFile.txt", "a");
                fprintf(logf, "PID-%u Row %d is invalid \n",(unsigned
int)pthread_self(), (position/9)+1);
                fclose(logf);
        }
        else
        {
                valid=1;
                //append log file
                FILE *logf = fopen("ProcessLogFile.txt", "a");
                fprintf(logf, "PID-%u Row %d is valid \n",(unsigned
int)pthread_self(), (position/9)+1);
                fclose(logf);
        }
        shmdt(Buffer1);

        return valid;
}

//check if all the numbers in a row are unique if exists within range 1-9
int verifycol(int keyb1, int sizeb1)
{

        int *Buffer1;
        int shmidb1 = shmget(keyb1, sizeb1, IPC_CREAT | 0775);
        Buffer1 = (int *) shmat(shmidb1, NULL, 0);

        int valid = 0;
        int numofInvalidpercol;

        for (int m = 0; m < 9; m++)
        {
                int tempcolbuffer[9];
                int tempbuffercount = 0;
```

```
                numofInvalidpercol = 0;
                for (int n = m; n < m+73; n = n+9)
                {

                        tempcolbuffer[tempbuffercount] = Buffer1[n];
                        tempbuffercount++;
                }

                for (int k = 0; k < 9; k++)
                {
                        if (tempcolbuffer[k] > 9)
                        {
                                numofInvalidpercol++;
                        }
                        else if (tempcolbuffer[k] < 1)
                        {
                                numofInvalidpercol++;

                        }
                }

                for (int i = 0; i <  9; i++)
                {
                        for (int j = i + 1; j< 10; j++)
                        {
                                if (tempcolbuffer[i] == tempcolbuffer[j])
                                {
                                        numofInvalidpercol++;

                                }
                        }
                }

                if(numofInvalidpercol == 0)
                {
                        valid ++;
                        //append log file
                        FILE *logf = fopen("ProcessLogFile.txt", "a");
                        fprintf(logf, "PID-%u Column %d is valid \n",
(unsigned int)pthread_self(), m+1);
                        fclose(logf);
                }
                else
                {
                        //append log file
                        FILE *logf = fopen("ProcessLogFile.txt", "a");
                        fprintf(logf, "PID-%u Column %d is invalid \n",
(unsigned int)pthread_self(), m+1);
                        fclose(logf);
                }

        }
        shmdt(Buffer1);
        return valid;
}

//checks if all number in subgrids one to nine are unique and exists within
//range 0-9
int verify3x3subgrid(int keyb1, int sizeb1)
```

```
{

        int *Buffer1;
        int shmidb1 = shmget(keyb1, sizeb1, IPC_CREAT | 0775);
        Buffer1 = (int *) shmat(shmidb1, NULL, 0);

        int numinvalidpersg;
        int valid = 0;
        int outercounter = 0;
        for(int s=0; s < 61 ;s = s +3)
        {
                outercounter ++;
                int innercounter=0;
                int tempsgbuffer[9];
                numinvalidpersg=0;

                tempsgbuffer[0] = Buffer1[s];
                tempsgbuffer[1] = Buffer1[s+1];
                tempsgbuffer[2] = Buffer1[s+2];
                tempsgbuffer[3] = Buffer1[s+9];
                tempsgbuffer[4] = Buffer1[s+10];
                tempsgbuffer[5] = Buffer1[s+11];
                tempsgbuffer[6] = Buffer1[s+18];
                tempsgbuffer[7] = Buffer1[s+19];
                tempsgbuffer[8] = Buffer1[s+20];

                for (int k = 0; k < 9; k++)
                {
                        if (tempsgbuffer[k] > 9)
                        {
                                numinvalidpersg++;
                        }
                        else if (tempsgbuffer[k] < 1)
                        {
                                numinvalidpersg++;
                        }

                }

                for (int i = 0; i <  9; i++)
                {
                        for (int j = i + 1; j< 10; j++)
                        {
                                if (tempsgbuffer[i] == tempsgbuffer[j])
                                {
                                        numinvalidpersg++;
                                }
                        }
                }

                if(numinvalidpersg == 0)
                {
                        valid ++;

                        //append log file
                        FILE *logf = fopen("ProcessLogFile.txt", "a");
                        fprintf(logf, "PID-%u 3x3 SubGrid start at spot %d
(counting left to right, top to bottom) is valid \n",(unsigned
int)pthread_self(), s+1);
```

```
                        fclose(logf);
                }
                else
                {
                        //append log file
                        FILE *logf = fopen("ProcessLogFile.txt", "a");
                        fprintf(logf, "PID-%u 3x3 SubGrid start at spot %d
(counting left to right, top to bottom)is invalid \n",(unsigned
int)pthread_self(), s+1);
                        fclose(logf);
                }

                switch(outercounter)
                {
                        case 3:
                        case 6:
                                s = s + 18;
                                break;
                }

        }
        shmdt(Buffer1);
        return valid;
}
```

FunctionsP.h

```
#ifndef FUNCTIONSP_H_INCLUDED
#define FUNCTIONSP_H_INCLUDED

sem_t Buffer2Sem;
sem_t CounterSem;

void Group1task(int position, int keyb1, int sizeb1, int keyb2, int sizeb2,
int keyc, int sizec, int timedelay);

void Group2task(int keyb1, int sizeb1, int keyb2, int sizeb2, int keyc, int
sizec, int timedelay);

void Group3task(int keyb1, int sizeb1, int keyb2, int sizeb2, int keyc, int
sizec, int timedelay);

void populateBuffer1(char *filename, int keyb1, int sizeb1);

int verifyrow(int position, int keyb1, int sizeb1);

int verifycol(int keyb1, int sizeb1);

int verify3x3subgrid(int keyb1, int sizeb1);


#endif
```

ThreadProgram.c

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

int Buffer1[81]; //array to hold the incoming sudoku solution
int Buffer2[11]; //array to hold results for 9 individual row tests, 1 all-
columns test, 1 all-subgrids test
int counter;    //counts the sum of valid rows, columns, subgrids
int timedelay; //user specified time delay for threads


int main(int argc, char *argv[])
{
        //(re)create log file

        FILE *logf = fopen("ThreadLogFile.txt", "w");
        if(logf == NULL)
        {
        perror("Error");
        }
        fprintf(logf, "-----||Log file for Thread Program||----- \n \n");
        fclose(logf);

        long int position;
        counter = 0;
        pthread_t threads[11];
        timedelay = atoi(argv[2]);

        //reading and assign to buffer1
        populateBuffer1(argv[1]);

        if (pthread_mutex_init(&buffer2mutex, NULL) ==0)
        {
        //      printf("\n bmutex success");
        }

        if (pthread_mutex_init(&countermutex, NULL) ==0)
        {
        //      printf("\n cmutex success");
        }

        //create 11 threads for 3 groups of tasks
        for (position = 0; position <91; position = position +9)
        {

                if (position<73)
                {
                        //create thread for group 1
                        pthread_create(&threads[position/9],NULL,Group1task,
(void *)position);

                }
                else
                {
```

```
                        if (position==81)
                        {
                                //create thread for group 2

        pthread_create(&threads[position/9],NULL,Group2task,NULL);
                        }

                        else
                        {
                                //create thread for group 3

        pthread_create(&threads[position/9],NULL,Group3task,NULL);
                        }
                }
          }

          //wait for all child threads to exit before resuming parent
          for (int i = 0; i<11; i++)
          {
                pthread_join(threads[i],NULL);
          }

          //Display Solution Summary
          printf("\nCount of valid rows, columns and subgrids: %d ", counter);

          if (counter == 27)
          {
                printf("and thus solution is valid\n");
          }
          else{
                printf("and thus solution is invalid\n");
          }
          return 0;
}
```

Functions.c

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


void *Group1task(void *arg){

        long int position = (long int)arg;

        int validrow = 0;
        //verify row
        validrow = verifyrow(position);

        //display result on screen
        printf("Validation result from PID-%u Row %ld is valid \n",(unsigned
int)pthread_self(), (position/9)+1);

        //timedelay
        sleep(timedelay);

        //write to buffer2
        pthread_mutex_lock(&buffer2mutex);
        Buffer2[position/9] = validrow;
        pthread_mutex_unlock(&buffer2mutex);

        //update counter
        pthread_mutex_lock(&countermutex);
        counter = counter + validrow;
        pthread_mutex_unlock(&countermutex);

        pthread_exit(NULL);
}


void *Group2task(void *arg)
{

        int validcol =0;

        //verify all columns
        validcol = verifycol();

        //display result on screen
        printf("Validation result from PID-%u %d of 9 columns are valid \n",
(unsigned int)pthread_self(), validcol);

        //timedelay
        sleep(timedelay);

        //update buffer2
        pthread_mutex_lock(&buffer2mutex);
        Buffer2[9] = validcol;
        pthread_mutex_unlock(&buffer2mutex);
```

```
        //update counter
        pthread_mutex_lock(&countermutex);
        counter = counter + validcol;
        pthread_mutex_unlock(&countermutex);

        pthread_exit(NULL);

}

void *Group3task(void *arg)
{

        int validgrid =0;

        //verify all subgrids
        validgrid = verify3x3subgrid();

        //display result on screen
        printf("Validation result from PID-%u %d of 9 3x3 subgrids are valid
\n",(unsigned int)pthread_self(), validgrid);
        //timedelay
        sleep(timedelay);

        //update buffer2
        pthread_mutex_lock(&buffer2mutex);
        Buffer2[10] = validgrid;
        pthread_mutex_unlock(&buffer2mutex);

        //update counter
        pthread_mutex_lock(&countermutex);
        counter = counter + validgrid;
        pthread_mutex_unlock(&countermutex);

        pthread_exit(NULL);

}

void populateBuffer1(char *filename)
{

        //opening sudoku solution file for reading
        FILE *f;
        f = fopen(filename, "r");

        if(f == NULL)
        {
                printf("Input file error");
        }
        else
        {
                printf("Input file succesfully read \n");
                //entering solution into Buffer1 array
                for (int i = 0; i<81; i++)
                {
                        fscanf(f, "%d", &Buffer1[i]);

                }

                fclose(f);
```

```
	}
}

//takes starting position of row and compares the 9 numbers from there.
//Return 1 if all numbers are unique and only comprises of 1-9 digits.
int verifyrow(long int position)
{
	int valid;
	int numofInvalid=0;
	int tempbuf[9];


	for (int k = position; k < position+9; k++)
	{
		if (Buffer1[k] > 9)
		{
			numofInvalid++;
		}
		else if (Buffer1[k] < 1)
		{
			numofInvalid++;
		}

	}

	if (numofInvalid == 0)
	{
		for (int i = position; i < position+8; i++)
		{
			for (int j = i + 1; j< position+9; j++)
			{
				if (Buffer1[i] == Buffer1[j])
				{
					numofInvalid++;
				}
			}
		}
	}

	if(numofInvalid>0)
	{
		valid=0;
		//append log file
		FILE *logf = fopen("ThreadLogFile.txt", "a");
		fprintf(logf, "PID-%u Row %ld is invalid \n",(unsigned
int)pthread_self(), (position/9)+1);
		fclose(logf);
	}
	else
	{
		valid=1;
		//append log file
		FILE *logf = fopen("ThreadLogFile.txt", "a");
		fprintf(logf, "PID-%u Row %ld is valid \n",(unsigned
int)pthread_self(), (position/9)+1);
		fclose(logf);
	}
	return valid;
}
```

```
//check if all the numbers in a row are unique if exists within range 1-9
int verifycol()
{
	int valid = 0;
	int numofInvalidpercol;

	for (int m = 0; m < 9; m++)
	{
		int tempcolbuffer[9];
		int tempbuffercount = 0;
		numofInvalidpercol = 0;
		for (int n = m; n < m+73; n = n+9)
		{

			tempcolbuffer[tempbuffercount] = Buffer1[n];
			tempbuffercount++;
		}

		for (int k = 0; k < 9; k++)
		{
			if (tempcolbuffer[k] > 9)
			{
				numofInvalidpercol++;
			}
			else if (tempcolbuffer[k] < 1)
			{
				numofInvalidpercol++;

			}
		}

		for (int i = 0; i <  9; i++)
		{
			for (int j = i + 1; j< 10; j++)
			{
				if (tempcolbuffer[i] == tempcolbuffer[j])
				{
					numofInvalidpercol++;

				}
			}
		}

		if(numofInvalidpercol == 0)
		{
			valid ++;
			//append log file
			FILE *logf = fopen("ThreadLogFile.txt", "a");
			fprintf(logf, "PID-%u Column %d is valid \n",
(unsigned int)pthread_self(), m+1);
			fclose(logf);
		}
		else
		{
			//append log file
			FILE *logf = fopen("ThreadLogFile.txt", "a");
			fprintf(logf, "PID-%u Column %d is invalid \n",
(unsigned int)pthread_self(), m+1);
```

```
                        fclose(logf);
                }

        }
return valid;
}

//checks if all number in subgrids one to nine are unique and exists within
//range 0-9
int verify3x3subgrid()
{
int numinvalidpersg;
int valid = 0;
int outercounter = 0;
        for(int s=0; s < 61 ;s = s +3)
        {
                outercounter ++;
                int innercounter=0;
                int tempsgbuffer[9];
                numinvalidpersg=0;

                tempsgbuffer[0] = Buffer1[s];
                tempsgbuffer[1] = Buffer1[s+1];
                tempsgbuffer[2] = Buffer1[s+2];
                tempsgbuffer[3] = Buffer1[s+9];
                tempsgbuffer[4] = Buffer1[s+10];
                tempsgbuffer[5] = Buffer1[s+11];
                tempsgbuffer[6] = Buffer1[s+18];
                tempsgbuffer[7] = Buffer1[s+19];
                tempsgbuffer[8] = Buffer1[s+20];

                for (int k = 0; k < 9; k++)
                {
                        if (tempsgbuffer[k] > 9)
                        {
                                numinvalidpersg++;
                        }
                        else if (tempsgbuffer[k] < 1)
                        {
                                numinvalidpersg++;
                        }

                }

                for (int i = 0; i <  9; i++)
                {
                        for (int j = i + 1; j< 10; j++)
                        {
                                if (tempsgbuffer[i] == tempsgbuffer[j])
                                {
                                        numinvalidpersg++;
                                }
                        }
                }

                if(numinvalidpersg == 0)
                {
                        valid ++;
```

```
                        //append log file
                        FILE *logf = fopen("ThreadLogFile.txt", "a");
                        fprintf(logf, "PID-%u 3x3 SubGrid start at spot %d
(counting left to right, top to bottom) is valid \n",(unsigned
int)pthread_self(), s+1);
                        fclose(logf);
                }
                else
                {
                        //append log file
                        FILE *logf = fopen("ThreadLogFile.txt", "a");
                        fprintf(logf, "PID-%u 3x3 SubGrid start at spot %d
(counting left to right, top to bottom)is invalid \n",(unsigned
int)pthread_self(), s+1);
                        fclose(logf);
                }

                switch(outercounter)
                {
                        case 3:
                        case 6:
                                s = s + 18;
                                break;
                }

        }

return valid;
}
```

Functions.h

```
#ifndef FUNCTIONS_H_INCLUDED
#define FUNCTIONS_H_INCLUDED


extern int Buffer1[];
extern int Buffer2[];
extern int counter;
extern int timedelay;

pthread_mutex_t buffer2mutex;
pthread_mutex_t countermutex;
pthread_mutex_t verifymutex;

void *Group1task(void *);

void *Group2task(void *arg);

void *Group3task(void *arg);

void populateBuffer1(char *filename);

int verifyrow(long int position);

int verifycol();

int verify3x3subgrid();


#endif
```

Compilation and Execution

For ProcessProgram.c:

--If necessary----------
sudo chmod u+x ProcessProgram.c
sudo chmod u+x FunctionsP.c
---

gcc -pthread -c FunctionsP.c
gcc -pthread -c ProcessProgram.c
gcc -pthread -c FunctionsP.o ProcessProgram.o -o ProcessExe
	//Either Incorrect Sudoku Solution Input File
./ProcessExe input.txt 5
	//Or Correct Sudoku Solution Input File
./ProcessExe correctinput.txt 5

For ThreadProgram.c:

--If necessary----------
sudo chmod u+x ThreadProgram.c
sudo chmod u+x Functions.c
---

gcc -pthread -c Functions.c
gcc -pthread -c ThreadProgram.c
gcc -pthread -c Functions.o ThreadProgram.o -o ThreadExe
	//Either Incorrect Sudoku Solution Input File
./ThreadExe input.txt 5
	//Or Correct Sudoku Solution Input File
./ThreadExe correctinput.txt 5

Test Input (Process Program)

input.txt:
2 6 5 9 4 8 3 1 7 1 8 3 5 2 7 6 4 9 9 4 7 6 3 1 8 5 2 5 1 4 7 6 2 9 3 8 8 9 2 1 5 3 4 7 6 7
13 6 8 9 4 1 2 5 3 2 8 4 7 6 5 9 1 6 7 9 3 1 5 2 8 4 4 55 1 2 8 9 7 3 13

timedelay:
5

Test Ouput (Process Program)

ProcessLogFile.txt:

-----||Log file for Process Program||-----

PID-2960451328 Row 1 is valid
PID-2960451328 Row 2 is valid
PID-2960451328 Row 7 is valid
PID-2960451328 Row 3 is valid
PID-2960451328 Row 9 is invalid
PID-2960451328 Row 4 is valid
PID-2960451328 Row 8 is valid
PID-2960451328 Column 1 is valid
PID-2960451328 Column 2 is invalid
PID-2960451328 Column 3 is valid
PID-2960451328 Column 4 is valid
PID-2960451328 Column 5 is valid
PID-2960451328 Column 6 is valid
PID-2960451328 Column 7 is valid
PID-2960451328 Row 5 is valid
PID-2960451328 Column 8 is invalid
PID-2960451328 Column 9 is invalid
PID-2960451328 3x3 SubGrid start at spot 1 (counting left to right, top to bottom) is valid
PID-2960451328 3x3 SubGrid start at spot 4 (counting left to right, top to bottom) is valid
PID-2960451328 3x3 SubGrid start at spot 7 (counting left to right, top to bottom) is valid
PID-2960451328 3x3 SubGrid start at spot 28 (counting left to right, top to bottom)is invalid
PID-2960451328 3x3 SubGrid start at spot 31 (counting left to right, top to bottom) is valid
PID-2960451328 3x3 SubGrid start at spot 34 (counting left to right, top to bottom) is valid
PID-2960451328 3x3 SubGrid start at spot 55 (counting left to right, top to bottom)is invalid
PID-2960451328 3x3 SubGrid start at spot 58 (counting left to right, top to bottom) is valid
PID-2960451328 3x3 SubGrid start at spot 61 (counting left to right, top to bottom)is invalid
PID-2960451328 Row 6 is invalid

Test Input (Thread Program)

input.txt:
2 6 5 9 4 8 3 1 7 1 8 3 5 2 7 6 4 9 9 4 7 6 3 1 8 5 2 5 1 4 7 6 2 9 3 8 8 9 2 1 5 3 4 7 6 7
13 6 8 9 4 1 2 5 3 2 8 4 7 6 5 9 1 6 7 9 3 1 5 2 8 4 4 55 1 2 8 9 7 3 13

timedelay:
5

Test Ouput (Thead Program)

ThreadLogFile.txt:

-----||Log file for Thread Program||-----

PID-902289152 Row 6 is invalid
PID-885503744 Row 8 is valid
PID-910681856 Row 5 is valid
PID-893896448 Row 7 is valid
PID-944252672 Row 1 is valid
PID-919074560 Row 4 is valid
PID-927467264 Row 3 is valid
PID-935859968 Row 2 is valid
PID-868718336 Column 1 is valid
PID-877111040 Row 9 is invalid
PID-868718336 Column 2 is invalid
PID-860325632 3x3 SubGrid start at spot 1 (counting left to right, top to bottom) is valid
PID-868718336 Column 3 is valid
PID-860325632 3x3 SubGrid start at spot 4 (counting left to right, top to bottom) is valid
PID-868718336 Column 4 is valid
PID-860325632 3x3 SubGrid start at spot 7 (counting left to right, top to bottom) is valid
PID-868718336 Column 5 is valid
PID-860325632 3x3 SubGrid start at spot 28 (counting left to right, top to bottom)is invalid
PID-868718336 Column 6 is valid
PID-860325632 3x3 SubGrid start at spot 31 (counting left to right, top to bottom) is valid
PID-868718336 Column 7 is valid
PID-860325632 3x3 SubGrid start at spot 34 (counting left to right, top to bottom) is valid
PID-868718336 Column 8 is invalid
PID-860325632 3x3 SubGrid start at spot 55 (counting left to right, top to bottom)is invalid
PID-868718336 Column 9 is invalid
PID-860325632 3x3 SubGrid start at spot 58 (counting left to right, top to bottom) is valid
PID-860325632 3x3 SubGrid start at spot 61 (counting left to right, top to bottom)is invalid

Critical Sections

In both programs the critical section occurs when the processes or threads are writing to Buffer2 and counter at the same time. To control access we used, in the case of processes, a POSIX semaphore for each critical section affected shared memory (Buffer2, counter). We then signal sem_wait(&semaphorename) when we want to access the shared memories and when access granted the semaphore is locked for other processes. Once done writing we use sem_post(&semaphorename) to release it for use from another process. Similarly for threads we use mutex to control the write access to shared variables affected by critical section. We use pthread_mutex_lock to signal for the use of the mutex. Once granted access mutex locked for other threads and mutex holding thread proceeds into critical section. After exiting critical section we use pthread_mutex_unlock to realease hold of the mutex.

Both these techniques provide mutual exclusion for a process/thread entering a critical section as no other process/thread is allowed concurrent write access.

Example (from FunctionsP.c) for Process:

```
//update counter
sem_wait(&CounterSem);
*counter = *counter + validcol;
sem_post(&CounterSem);
```

Example (from Functions.c) for Thread

```
//update buffer2
pthread_mutex_lock(&buffer2mutex);
Buffer2[10] = validgrid;
pthread_mutex_unlock(&buffer2mutex);
```

Testing

After testing both programs with various Sudoku solutions they both output the expected output again and again. Varying the time delay has no effect on the accuracy of the end result since true mutual exclusion is available at critical sections.
The only variance is the order in which the result for the process/threads is output to both on screen and in the log file since the process/threads most times finish in a different order than which they started in.