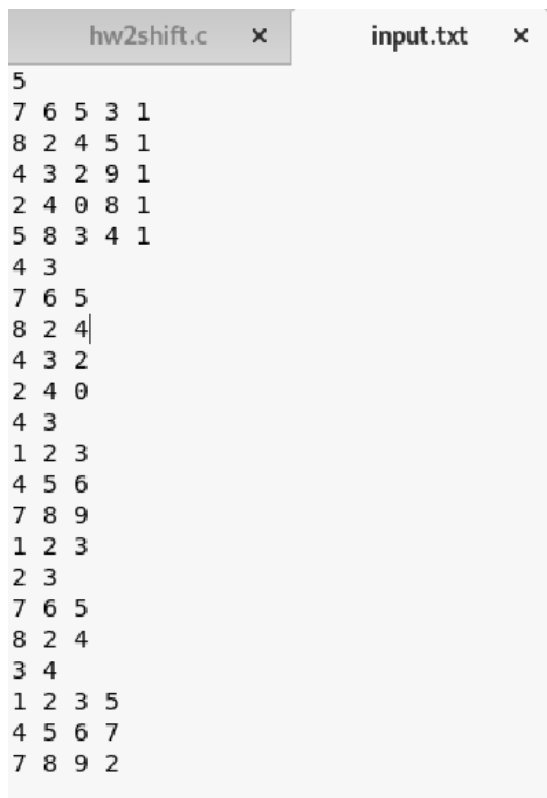


Advanced Unix

Assignment 2

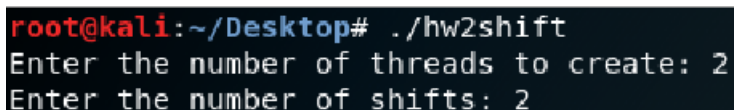
1. In order to shift matrix elements from left to right and bottom to top with "d" number of threads, firstly, an input file containing the size and the matrix is given. The below picture shows the "input.txt". Also, the input file "input.txt" is going to be used as an example in the screenshots for explaining the operations in the assignment. Later, execution times of different number of threads are going to be shown.



```
hw2shift.c x input.txt x
5
7 6 5 3 1
8 2 4 5 1
4 3 2 9 1
2 4 0 8 1
5 8 3 4 1
4 3
7 6 5
8 2 4
4 3 2
2 4 0
4 3
1 2 3
4 5 6
7 8 9
1 2 3
2 3
7 6 5
8 2 4
3 4
1 2 3 5
4 5 6 7
7 8 9 2
```

In the "input.txt", firstly, the file starts with the size of the first matrix that is going to be used for shifting. In the above picture, a 5x5 matrix is given for the first part of the assignment. Secondly, two matrixes of size 4x3 are given for matrix summation for the second part of the assignment. Later, two matrixes of size 2x3 and 3x4 are given for matrix multiplication for the third part of the assignment.

In the first part, we are getting the number of threads to be created and number of shifts to be performed from the keyboard. In the example, we are going to create 2 threads and perform 2 shifts.



```
root@kali:~/Desktop# ./hw2shift
Enter the number of threads to create: 2
Enter the number of shifts: 2
```

Next, the contents of the input file and the size of the matrix is printed to the screen:

```
Contents of the input file:
m:5
matrix from parent before shifting:
7 6 5 3 1
8 2 4 5 1
4 3 2 9 1
2 4 0 8 1
5 8 3 4 1
```

Then, the 2 threads are created and start to run the “shiftMatrix” function. We are printing the id of the thread and the row/columns that they are responsible for to the screen. In the below picture, because we are having 2 threads, the first thread with id 2373...20 is responsible for shifting the 0th and 1st row and column of the given matrix. The second thread is going to shift more rows and columns because $d \neq m$. Second thread with id 2289...16 is responsible for shifting the 2nd, 3rd and 4th row and column of the given matrix. All threads print the matrix before and after shifting.

```
inside shiftMatrix I am 237336320 and start: 0
inside shiftMatrix I am 237336320 and end: 1
I am 237336320 and before shifting rows:
7 6 5 3 1
8 2 4 5 1
4 3 2 9 1
2 4 0 8 1
5 8 3 4 1
I am 237336320 and after shifting rows
3 1 7 6 5
5 1 8 2 4
4 3 2 9 1
2 4 0 8 1
5 8 3 4 1
inside shiftMatrix I am 228943616 and start: 2
inside shiftMatrix I am 228943616 and end: 4
I am 228943616 and before shifting rows:
3 1 7 6 5
5 1 8 2 4
4 3 2 9 1
2 4 0 8 1
5 8 3 4 1
```

Later, after the shifting of rows are finished, the threads wait for each other in order to start column shifting. After all threads are done shifting the rows, the column shifting starts. Again, threads print the resultant matrix before and after shifting.

```
I am 228943616 and after shifting rows
3 1 7 6 5
5 1 8 2 4
9 1 4 3 2
8 1 2 4 0
4 1 5 8 3
I am 228943616 and before shifting columns
3 1 7 6 5
5 1 8 2 4
9 1 4 3 2
8 1 2 4 0
4 1 5 8 3
I am 228943616 and after shifting columns:
3 1 4 3 2
5 1 2 4 0
9 1 5 8 3
8 1 7 6 5
4 1 8 2 4
I am 237336320 and before shifting columns
3 1 4 3 2
5 1 2 4 0
9 1 5 8 3
8 1 7 6 5
```

After column shifting is performed, the parent thread prints the resultant matrix to the screen with the execution time. The resultant matrix is also written to the “output.txt” file that is shown below. The “output.txt” file also contains the other resultant matrixes that are calculated in the second and third parts of the assignment. The details of summation and multiplication of matrixes are going to be explained in the next parts.

```
I am 237336320 and before shifting columns
3 1 4 3 2
5 1 2 4 0
9 1 5 8 3
8 1 7 6 5
4 1 8 2 4
I am 237336320 and after shifting columns:
9 1 4 3 2
8 1 2 4 0
4 1 5 8 3
3 1 7 6 5
5 1 8 2 4
matrix from parent after shifting:
9 1 4 3 2
8 1 2 4 0
4 1 5 8 3
3 1 7 6 5
5 1 8 2 4
exec time of shifting with 2 threads:0.000253
```

```
output.txt x
5
9 1 4 3 2
8 1 2 4 0
4 1 5 8 3
3 1 7 6 5
5 1 8 2 4
4 3
8 8 8 |
12 7 10
11 11 11
3 6 3
2 4
66 84 102 87
44 58 72 62
```

The same matrix is going to be used with different number of threads in order to measure execution times. Again, 2 number of shifts are performed. The execution times increased because when increasing the number of threads, we are doing more context switch and this degrades performance. The table below shows the execution times using different number of threads:

d	Execution Time
1	0.000247
2	0.000253
3	0.000531
4	0.000754
5	0.001062

2. In order to sum two matrixes that are given in the "input.txt" file, because we are having 4x3 matrixes, we are having 4 threads that are responsible for each row. Firstly, the matrixes that are given in the input file are printed to the screen by the main thread. The matrixes are named as "matrix1" and "matrix2" for the second part of the assignment. The threads that are created for the matrix summation is different from the first part of the assignment and it starts after shifting the matrix of the first part.

```
matrix1 from parent before summation:
7 6 5
8 2 4
4 3 2
2 4 0
matrix2 from parent before summation:
1 2 3
4 5 6
7 8 9
1 2 3
```

After creating threads, the threads start to run “sumMatrix” function and print their ids. After making summation of their responsible rows, each matrix prints the resultant matrix to the screen.

```
inside sumMatrix I am -881727744:  
I am -881727744 and after summation:  
8 8 8  
0 0 0  
0 0 0  
0 0 0  
inside sumMatrix I am -873335040:  
I am -873335040 and after summation:  
8 8 8  
12 7 10  
0 0 0  
0 0 0  
inside sumMatrix I am -890226944:  
I am -890226944 and after summation:  
8 8 8  
12 7 10  
11 11 11  
0 0 0  
inside sumMatrix I am -898619648:  
I am -898619648 and after summation:  
8 8 8  
12 7 10  
11 11 11  
3 6 3
```

Later, the main thread prints the resultant matrix to the screen with the execution time.

```
result from parent after summation:  
8 8 8  
12 7 10  
11 11 11  
3 6 3  
exec time of summation with 4 threads:0.005836
```

Also, the resultant matrix and its size are written to the output file named “output.txt” which is shown in the first part.

The below table shows execution times of summation of different sized matrixes. Again, the number of rows show the number of threads that are created make the summation. Again, the execution times increased because when increasing the number of threads, we are doing more context switch and this degrades performance. The number of columns are given as 3 in order to be consistent with the above example given.

Number of Rows	Number of Threads	Execution Time
1	1	0.001402
2	2	0.001543
3	3	0.002926
4	4	0.005836
5	5	0.005987

3. In order to multiply two matrixes that are given in the “input.txt” file, because we are having 2x3 and 3x4 matrixes, we are having 2 threads that are responsible for each row and column. Firstly, the matrixes that are given in the input file are printed to the screen by the main thread. The matrixes are named as “matrix3” and “matrix4” for the third part of the assignment. The threads that are created for the matrix multiplication are different from the first and second parts of the assignment and it starts after summation of the matrixes of the second part.

```
matrix3 from parent before multiplication:
7 6 5
8 2 4
matrix4 from parent before multiplication:
1 2 3 5
4 5 6 7
7 8 9 2
```

After creating threads, the threads start to run “multMatrix” function and print their ids. After making multiplication of their responsible rows and columns, each matrix prints the resultant matrix to the screen.

```
inside multMatrix I am 587192064:
I am 587192064 and after multiplication:
66 84 102 87
0 0 0 0
inside multMatrix I am 595584768:
I am 595584768 and after multiplication:
66 84 102 87
44 58 72 62
```

Later, the main thread prints the resultant matrix to the screen with the execution time.

```
result from parent after multiplication:
66 84 102 87
44 58 72 62
exec time of multiplication with 2 threads:0.003467
```

Also, the resultant matrix and its size are written to the output file named “output.txt” which is shown in the first part.

The below table shows execution times of multiplication of different sized matrixes. Again, the number of rows show the number of threads that are created make the multiplication. Again, the execution times increased because when increasing the number of threads, we are doing more context switch and this degrades performance.

Number of Rows	Number of Threads	Execution Time
1	1	0.000168
2	2	0.003467
3	3	0.003326
4	4	0.004179
5	5	0.007131