Operating Systems CS3523
Programming Assignment 3
ES21BTECH11022

High Level Design of Code

---

1. Taking Input: The program uses command line arguments to take the input file containing values of n, k, c1 and c2. Here n represents the number of threads, and k represents the number of times each thread attempts to enter the critical section. Apart from n and k, we also inputc1 and c2. We use these constants of exponential time distribution to replicate the critical section code and the remainder section code, respectively. Here we also check for two errors an improper number of command line arguments and the case of fopen returning a NULL pointer. In both cases, the error is printed onto the output file.

2. CreatingThreads: For thread creation, we declare a vector of cpp threads. Then the program pushes back n threads using a for loop.The pushed back thread is initialized via the constructor calling the func function. We also pass an integer to the function as an input. This is the thread number for that thread. Then each thread runs the function func, and after completion, we join all threads using the join member function of cpp thread.

3. Function func: Each thread runs this function. We first initialize the variables required to calculate the delay times simulating the critical and remainder sections. Now for one request for entry into the CS we first note the time. Then entry section code is carried out, which is the mutual exclusion algorithms tas, cas, cas-bounded for each of the three programs. Once the thread enters CS, we note the time and sleep for t1 seconds (t1 calculated using c1). Now we exit the CS and note the exit time. Finally we sleep for t2 seconds (t2 calculated using c2) to simulate the remainder section. Now this was the procedure for one entry request into the CS. Each thread performs this task k times using a for loop.

4. Final Output File: We have only one main output file named "OutMain.txt" for this program. For each request of a thread to enter the CS, we note three times, request time, entry time and exit time. Now, these three quantities are outputted in the file as each thread requests entry into the CS k times, and there are n such threads.

Low-Level Design of Code

Considering low-level design, we first create a vector of threads. And we then push n threads, with each calling the thread constructor, which tells each thread to run the function func with argument as the thread number j.

In the function func we first declare the default random engine used to generate pseudo random numbers named generator. Then we declare two exponential distribution variables named distribution1 and distribution2, which are initialized using the constructor to their corresponding mean values. Then we can later use the operator () to generate the random numbers.

Now we come to requests to enter the CS. We output three time values: request time, entry time and exit time for each request. After outputting request time we move to the entry section, i.e., the mutual exclusion algorithm. Now this part is different for each of three programs as follows:

1. Test and Set: For implementing test and set, we use the atomic_flag struct and its inbuilt member function test_and_set(). We name the object of atomic_flag as lock_stream and initialize it using ATOMIC_FLAG_INIT. And then, we proceed normally using test and set in the while loop for waiting and then entry into CS. Once the thread exits from the CS, it clears the lock_stream via the clear member function.

2. Compare and Swap: For implementing Compare and Swap, we use the atomic_compare_exchange_weak function present in the atomic header file. This function takes three arguments, a pointer to an atomic int, a pointer to an integer (expected) and an integer (new value). With these values it executes Compare and Swap mutual exclusion algorithm. This is it for the entry section, and for the exit section, we just do my_lock = 0 so that another thread may enter CS.

3. Compare and Swap with Bounded Waiting: For implementing Compare and Swap with Bounded Waiting, we take the base model of CAS and implement a boolean array called waiting and an integer key. Once a thread requests for entry into CS, we do waiting[j] = true as in the thread, j is in contention for CS. Now we wait while waiting[j], and key is 1, and during this we assign the return value of the function atomic_compare_exchange to key. If any one of these becomes false, then we enter the CS. Now for exit section, we loop through the remaining threads to see which ones are waiting. If we find a waiting thread x, we do waiting[x] = false to allow it to enter CS. This ensures bounded waiting.

Once the entry and exit sections are modelled according to the mutual exclusion algorithm, we loop for k requests on each of the n threads. Once the thread returns from the function func, we join them using the join member function of teh thread class.

Analysis of Output:

1. Test and Set: Output file using n = 5, k = 5, c1 = 5 and c2 = 20

```
Request number 1 after 136 us by thread 1
Entry number 1 after 136 us by thread 1
Request number 1 after 200 us by thread 2
Request number 1 after 253 us by thread 3
Request number 1 after 280 us by thread 4
Exit number 1 after 136 us by thread 1
Entry number 1 after 280 us by thread 4
Request number 1 after 358 us by thread 5
Request number 2 after 386 us by thread 1
Exit number 1 after 280 us by thread 4
Entry number 1 after 200 us by thread 2
Request number 2 after 448 us by thread 4
Exit number 1 after 200 us by thread 2
Entry number 2 after 448 us by thread 4
Request number 2 after 512 us by thread 2
Exit number 2 after 448 us by thread 4
Entry number 1 after 253 us by thread 3
Request number 3 after 571 us by thread 4
Entry number 3 after 571 us by thread 4
Exit number 1 after 253 us by thread 3
Exit number 3 after 571 us by thread 4
Entry number 2 after 512 us by thread 2
Request number 2 after 638 us by thread 3
Request number 4 after 696 us by thread 4
Exit number 2 after 512 us by thread 2
Entry number 4 after 696 us by thread 4
Request number 3 after 756 us by thread 2
Exit number 4 after 696 us by thread 4
Entry number 3 after 756 us by thread 2
Request number 5 after 824 us by thread 4
Exit number 3 after 756 us by thread 2
Entry number 2 after 638 us by thread 3
Request number 4 after 883 us by thread 2
Exit number 2 after 638 us by thread 3
Entry number 1 after 358 us by thread 5
Request number 3 after 943 us by thread 3
Entry number 5 after 824 us by thread 4
Exit number 1 after 358 us by thread 5
```

```
Exit number 5 after 824 us by thread 4
Entry number 4 after 883 us by thread 2
Request number 2 after 1009 us by thread 5
Exit number 4 after 883 us by thread 2
Entry number 3 after 943 us by thread 3
Request number 5 after 1124 us by thread 2
Exit number 3 after 943 us by thread 3
Entry number 2 after 386 us by thread 1
Request number 4 after 1183 us by thread 3
Entry number 5 after 1124 us by thread 2
Exit number 2 after 386 us by thread 1
Exit number 5 after 1124 us by thread 2
Entry number 4 after 1183 us by thread 3
Request number 3 after 1251 us by thread 1
Exit number 4 after 1183 us by thread 3
Entry number 3 after 1251 us by thread 1
Request number 5 after 1366 us by thread 3
Entry number 5 after 1366 us by thread 3
Exit number 3 after 1251 us by thread 1
Exit number 5 after 1366 us by thread 3
Entry number 2 after 1009 us by thread 5
Request number 4 after 1427 us by thread 1
Entry number 4 after 1427 us by thread 1
Exit number 2 after 1009 us by thread 5
Exit number 4 after 1427 us by thread 1
Request number 3 after 1559 us by thread 5
Entry number 3 after 1559 us by thread 5
Request number 5 after 1611 us by thread 1
Exit number 3 after 1559 us by thread 5
Entry number 5 after 1611 us by thread 1
Request number 4 after 1679 us by thread 5
Entry number 4 after 1679 us by thread 5
Exit number 5 after 1611 us by thread 1
Exit number 4 after 1679 us by thread 5
Request number 5 after 1805 us by thread 5
Entry number 5 after 1805 us by thread 5
Exit number 5 after 1805 us by thread 5
```

2. Compare and Swap: Output file using n = 5, k = 5, c1 =2 and c2 = 2

```
Request number 1 after 213 us by thread 1
Entry number 1 after 297 us by thread 1
Request number 1 after 313 us by thread 2
Entry number 1 after 422 us by thread 2
Exit number 1 after 422 us by thread 1
Request number 1 after 441 us by thread 3
Entry number 1 after 445 us by thread 3
Request number 2 after 486 us by thread 1
Request number 1 after 487 us by thread 4
Entry number 1 after 497 us by thread 4
Entry number 2 after 497 us by thread 1
Exit number 1 after 497 us by thread 2
Exit number 1 after 527 us by thread 3
Exit number 2 after 561 us by thread 1
Request number 1 after 564 us by thread 5
Entry number 1 after 566 us by thread 5
Exit number 1 after 574 us by thread 4
Request number 2 after 579 us by thread 2
Entry number 2 after 580 us by thread 2
Request number 2 after 593 us by thread 3
Entry number 2 after 594 us by thread 3
Request number 3 after 622 us by thread 1
Exit number 1 after 628 us by thread 5
Entry number 3 after 628 us by thread 1
Request number 2 after 635 us by thread 4
Entry number 2 after 636 us by thread 4
Exit number 2 after 640 us by thread 2
Exit number 2 after 654 us by thread 3
Request number 2 after 687 us by thread 5
Entry number 2 after 688 us by thread 5
Exit number 3 after 689 us by thread 1
Exit number 2 after 696 us by thread 4
Request number 3 after 700 us by thread 2
Entry number 3 after 702 us by thread 2
Request number 3 after 714 us by thread 3
Exit number 2 after 746 us by thread 5
Entry number 3 after 746 us by thread 3
Request number 4 after 749 us by thread 1
Request number 4 after 749 us by thread 1
Entry number 4 after 750 us by thread 1
Request number 3 after 755 us by thread 4
Exit number 3 after 761 us by thread 2
Entry number 3 after 761 us by thread 4
Request number 3 after 804 us by thread 5
Entry number 3 after 805 us by thread 5
Exit number 3 after 808 us by thread 3
Exit number 4 after 810 us by thread 1
Request number 4 after 824 us by thread 2
Entry number 4 after 825 us by thread 2
Exit number 3 after 830 us by thread 4
Exit number 3 after 863 us by thread 5
Request number 4 after 868 us by thread 3
Request number 5 after 868 us by thread 1
Entry number 4 after 870 us by thread 3
Exit number 4 after 885 us by thread 2
Request number 4 after 889 us by thread 4
Entry number 4 after 890 us by thread 4
Entry number 5 after 885 us by thread 1
Request number 4 after 925 us by thread 5
Exit number 4 after 930 us by thread 3
Entry number 4 after 930 us by thread 5
Request number 5 after 946 us by thread 2
Entry number 5 after 948 us by thread 2
Exit number 4 after 955 us by thread 4
Exit number 5 after 955 us by thread 1
Exit number 4 after 960 us by thread 5
Request number 5 after 992 us by thread 3
Entry number 5 after 993 us by thread 3
Exit number 5 after 1007 us by thread 2
Request number 5 after 1016 us by thread 4
Entry number 5 after 1017 us by thread 4
Request number 5 after 1022 us by thread 5
Entry number 5 after 1053 us by thread 5
Exit number 5 after 1053 us by thread 3
Exit number 5 after 1078 us by thread 4
Exit number 5 after 1111 us by thread 5
```

3. Compare and Swap with Bounded Waiting: Output file using n = 5, k = 5, c1 =2 and c2 = 2

```
Request number 1 after 124 us by thread 1
Entry number 1 after 171 us by thread 1
Request number 1 after 191 us by thread 2
Exit number 1 after 275 us by thread 1
Entry number 1 after 275 us by thread 2
Request number 2 after 330 us by thread 1
Request number 1 after 342 us by thread 3
Exit number 1 after 344 us by thread 2
Entry number 2 after 344 us by thread 1
Request number 1 after 405 us by thread 4
Request number 2 after 414 us by thread 2
Exit number 2 after 407 us by thread 1
Entry number 1 after 407 us by thread 3
Request number 1 after 446 us by thread 5
Request number 3 after 472 us by thread 1
Exit number 1 after 503 us by thread 3
Entry number 1 after 503 us by thread 4
Request number 2 after 569 us by thread 3
Exit number 1 after 576 us by thread 4
Entry number 1 after 576 us by thread 5
Request number 2 after 633 us by thread 4
Exit number 1 after 640 us by thread 5
Entry number 3 after 641 us by thread 1
Request number 2 after 702 us by thread 5
Exit number 3 after 705 us by thread 1
Entry number 2 after 706 us by thread 2
Request number 4 after 778 us by thread 1
Exit number 2 after 782 us by thread 2
Entry number 2 after 783 us by thread 3
Request number 3 after 855 us by thread 2
Entry number 2 after 928 us by thread 4
Exit number 2 after 928 us by thread 3
Exit number 2 after 996 us by thread 4
Entry number 2 after 996 us by thread 5
Request number 3 after 1005 us by thread 3
Request number 3 after 1053 us by thread 4
Exit number 2 after 1060 us by thread 5
Entry number 4 after 1061 us by thread 1
Entry number 4 after 1061 us by thread 1
Request number 3 after 1123 us by thread 5
Exit number 4 after 1141 us by thread 1
Entry number 3 after 1143 us by thread 2
Request number 5 after 1203 us by thread 1
Entry number 3 after 1251 us by thread 3
Exit number 3 after 1250 us by thread 2
Exit number 3 after 1314 us by thread 3
Entry number 3 after 1314 us by thread 4
Request number 4 after 1323 us by thread 2
Request number 4 after 1379 us by thread 3
Exit number 3 after 1380 us by thread 4
Entry number 3 after 1380 us by thread 5
Request number 4 after 1436 us by thread 4
Exit number 3 after 1448 us by thread 5
Entry number 5 after 1449 us by thread 1
Request number 4 after 1509 us by thread 5
Exit number 5 after 1509 us by thread 1
Entry number 4 after 1510 us by thread 2
Exit number 4 after 1579 us by thread 2
Entry number 4 after 1579 us by thread 3
Exit number 4 after 1640 us by thread 3
Entry number 4 after 1640 us by thread 4
Request number 5 after 1647 us by thread 2
Request number 5 after 1696 us by thread 3
Exit number 4 after 1696 us by thread 4
Entry number 4 after 1697 us by thread 5
Request number 5 after 1751 us by thread 4
Exit number 4 after 1758 us by thread 5
Entry number 5 after 1759 us by thread 2
Request number 5 after 1820 us by thread 5
Entry number 5 after 1826 us by thread 3
Exit number 5 after 1825 us by thread 2
Exit number 5 after 1882 us by thread 3
Entry number 5 after 1882 us by thread 4
Exit number 5 after 1938 us by thread 4
Entry number 5 after 1938 us by thread 5
Exit number 5 after 1998 us by thread 5
```

Comparing the Algorithms

To compare the three mutual exclusion algorithms we plot two graphs :
1) Average waiting time vs Number of Threads
2) Worst waiting time vs Number of Threads.

Average Waiting Time vs Number of Threads

1. Test and Set:

Thread Number	Waiting time Readings (milliseconds)					Average Waiting Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
10	17.164	21.781	23.164	30.501	16.855	21.893
20	122.054	147.836	122.983	138.794	115.907	129.5148
30	213.256	212.446	197.603	204.683	228.436	211.2848
40	384	358.612	345.96	355.552	325.79	353.9828
50	453.114	439.064	451.202	423.682	388.12	431.0364

Thread Number	Worst WaitingTIme Readings (seconds)					Average Worst Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
10	0.324	0.178	0.138	0.13	0.456	0.2452
20	1.224	1.215	1.284	1.054	0.904	1.1362
30	1.883	1.882	1.982	1.667	2.108	1.9044
40	2.527	3.28	3.041	3.019	3.047	2.9828
50	3.156	3.583	3.013	3.735	3.25	3.3474

2. Compare and Swap:

Thread Number	Waiting time Readings (milliseconds)					Average Waiting Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
10	15.36	15.2	17.85	11.92	18.4	15.746
20	213.422	374.57	178.34	188.422	20.17	194.9848
30	432.356	334.263	391.98	304.18	387.88	370.1318
40	1406.46	464.69	634.305	464.69	531.692	700.3674
50	670.916	1849.612	741.912	468.8	819.084	910.0648

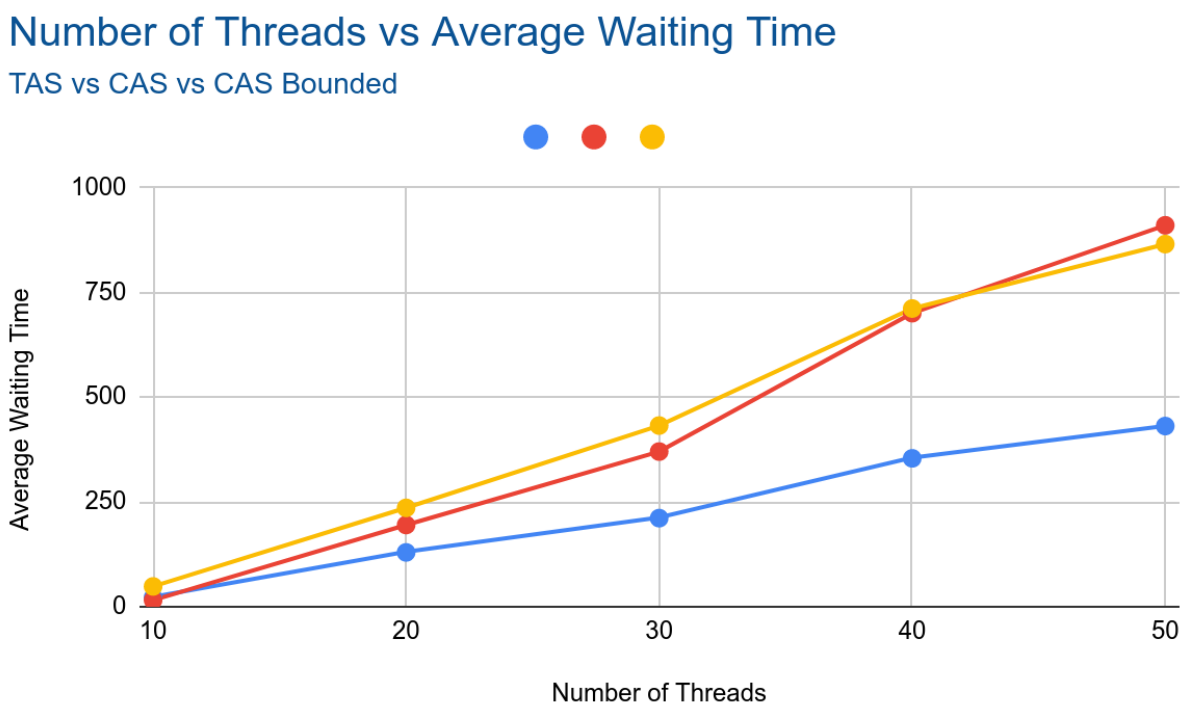
Thread Number	Worst WaitingTIme Readings (seconds)					Average Worst Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
10	0.181	0.102	0.14	0.097	0.17	0.138
20	1.543	0.741	0.176	0.213	0.219	0.5784
30	0.597	0.647	0.658	0.861	0.715	0.6956
40	1.01	1.563	0.794	1.86	1.093	1.264
50	1.185	3.13	1.185	1.595	1.594	1.7378

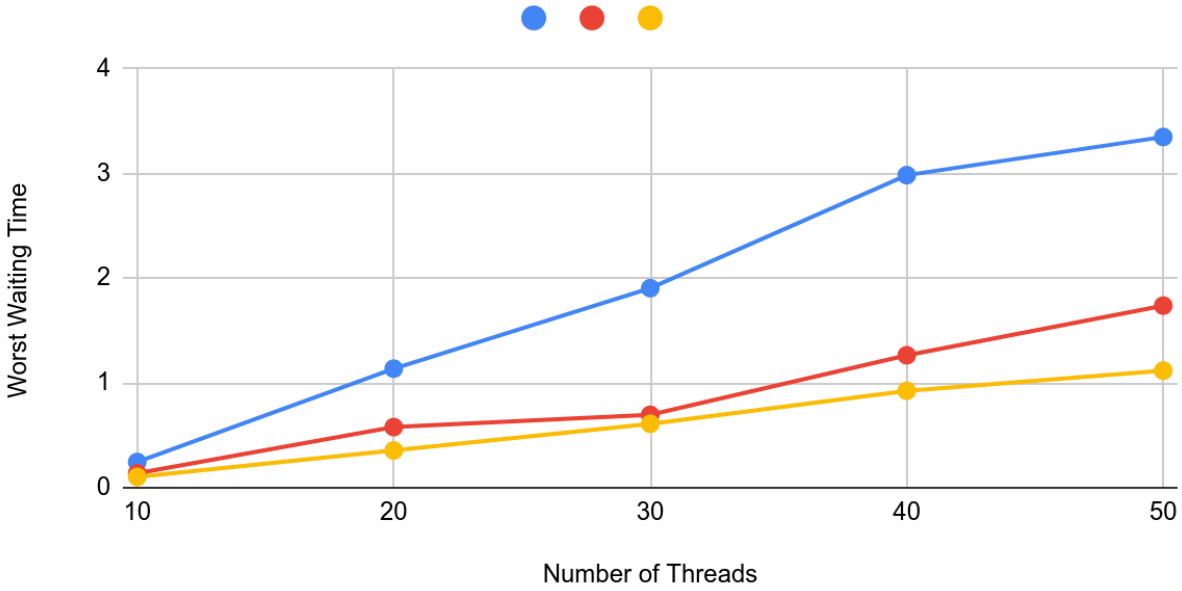
3. Compare and Swap with Bounded Waiting:

Thread Number	Waiting time Readings (milliseconds)					Average Waiting Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
10	56.17	48.74	47.06	42.37	47.24	48.316
20	221.715	245.2	254.405	236.375	221.445	235.828
30	462.286	421.606	401.493	450.153	425.692	432.246
40	743.745	672.83	703.612	705.192	731.647	711.4052
50	915.424	796.2	860.914	888.248	865.192	865.1956

Thread Number	Worst WaitingTIme Readings (seconds)					Average Worst Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
10	0.111	0.1	0.093	0.103	0.107	0.1028
20	0.323	0.399	0.399	0.339	0.318	0.3556
30	0.603	0.611	0.591	0.639	0.606	0.61
40	0.927	0.943	0.896	0.907	0.951	0.9248
50	1.22	1.009	1.139	1.135	1.084	1.1174

Graphs:




Number of Threads vs Worst Waiting Time
TAS vs CAS vs CAS Bounded
Worst Waiting Time
0
1
2
3
4
10
20
30
40
50
Number of Threads