Aim

To identify and discuss the behaviors of the windows thread in multithreaded environment.

SCOPE

Windows threads are implemented for performing 2 different tasks (one thread for one task)

- Task 1: - CPU intensive task
- Task 2: - I/O (File) intensive task

Both the task is performed by thread under 6 different thread priority classes and 4 different priorities.

Thread Priority Class used for calculation are: -

1 IDLE_PRIORITY_CLASS
2 BELOW_NORMAL_PRIORITY_CLASS
3 NORMAL_PRIORITY_CLASS
4 ABOVE_NORMAL_PRIORITY_CLASS
5 HIGH_PRIORITY_CLASS
6 REALTIME_PRIORITY_CLASS

Thread Priorities used for calculation of in each Thread Priority Class: -

1. THREAD_PRIORITY_LOWEST
2. THREAD_PRIORITY_NORMAL
3. THREAD_PRIORITY_HIGHEST
4. THREAD_PRIORITY_TIME_CRITICAL

IMPLEMENTATION

CPU Intensive Task: -

We created 2 functions sorting and sorting1 for performing CPU intensive task. Both the functions sorting and sorting1 use swap and swap1 function respectively for performing bubble sort algorithm. Before performing bubble sort both the sorting functions generates the array random_value and random_value1 respectively of 5000 random numbers each. Bubble sort will loop for 500 times.

The reason for performing the sorting operation in a loop for 500 times is to obtain a measurable time (in microsecond) for which CPU is busy performing sorting operation.

I/O Intensive Task: -

We created 2 functions file_operation and file_operation1 for performing I/O intensive task. Both the functions file_operation and file_operation1 use file_read, file_write and file_read1, file_write1 function respectively for performing read and write operations. file_read and file_read1 function read the text form *file_for_reading.txt* and *file_for_reading1.txt* and store it in a vector array, then file_write and file_write1 function writes the text from the vector array to *file_for_wrtiting.txt* and *file_for_writing1.txt* files. Each file_operation execute file_read function before executing file_write function, and both file_read and file_write function are executed in a loop for 25 times.

The reason for performing the file_read and file_write operation in a loop for 25 times is to obtain a measurable time (in microsecond) for which CPU is busy performing I/O operation.

Thread Execution: -

We implement 4 Threads handler, two CPU Thread (cpuThread, cpuThread1) and two I/O Thread (fileThread, fileThread1). Table 1 shows the Thread and function implemented by the thread.

Thread HANDLER	***Function Name***
cpuThread	sorting ()
cpuThread1	sorting1()
fileThread	file_operation ()
fileThread1	file_operation1()

Table 1: - function implemented by the thread

Note: - 2 threads are executed at a time for creating multithreaded environment.

Thread Execution without Core Affinity

1. Two CPU Threads
2. Two I/O Threads
3. One CPU Thread and One I/O Thread.

Thread Execution with Core Affinity

1. Two CPU Threads Same Core Affinity
2. Two CPU Threads Different Core Affinity
3. Two I/O Threads Same Core Affinity
4. Two I/O Threads Different Core Affinity
5. One CPU Thread and One I/O Thread Same Core Affinity
6. One CPU Thread and One I/O Thread Different Core Affinity

RESULTS:

thread priorities

Thread priorities class

Table 1: TA C1 + TB C2

	idle		below normal		normal		above normal		high		realtime	
	cpu	file	cpu	file	cpu	file	cpu	file	cpu	file	cpu	file
lowest	17832176	14444289	17944233	13632499	16119293	13282033	15585159	12714946	14867610	12069323	14900970	12059200
normal	16474737	14087231	17415420	13237651	15919138	13026771	14919279	12067996	14874997	12095056	14930606	12119930
highest	17175456	12893848	15635718	13229917	14934634	12300499	14841454	12016058	14811321	11975349	14879596	12037187
timecritical	16217576	12366560	15427604	12240641	14953461	12329934	14922530	11953250	14784676	11976437	14775202	12021295

Table 2: TA C1 + TB C1

	idle		below normal		normal		above normal		high		realtime	
	cpu	file	cpu	file	cpu	file	cpu	file	cpu	file	cpu	file
lowest	22689527	13947635	22877271	16075789	22531462	16442095	22655727	15515441	21890287	15637425	22352673	15231854
normal	22499887	8384615	22888108	8430905	22248106	16631136	21960671	13174075	22540776	15672258	22036851	15780281
highest	14441732	8505575	13990895	8769840	13886748	8298593	13948415	8309682	13800699	8769729	14004917	8260235
timecritical	13947635	8423093	13924170	8517141	13899689	8261682	13800994	8619396	13801132	8281893	13892625	8748352

Table 3: TA C1 + TA C2

	idle		below normal		normal		above normal		high		realtime	
	cpu	cpu1	cpu	cpu1	cpu	cpu1	cpu	cpu1	cpu	cpu1	cpu	cpu1
lowest	20206523	20187002	19780929	20023356	20032676	20006462	19559305	19334616	19422352	19232062	19026531	19311097
normal	20178220	20138722	20047675	19970238	19717625	19761081	19559305	19016525	19718368	21436413	19248075	19076142
highest	19773330	20206523	19423297	19257006	18791279	18675069	19178305	19016525	21318442	19516750	18954276	19058403
timecritical	18629491	18657436	18640185	18524304	18632390	18641492	19231352	18579735	19113797	19222842	19196051	18854864

Table 4: TA C1 + TA C1

	idle		below normal		normal		above normal		high		realtime	
	cpu	cpu1	cpu	cpu1	cpu	cpu1	cpu	cpu1	cpu	cpu1	cpu	cpu1
lowest	28045581	28515121	32616531	32496550	34172204	31201019	28385482	28367774	27911824	28096272	28043623	28132903
normal	28326610	35583958	29384467	28869782	29916836	33568816	28006435	28044065	27956448	28056886	27970861	28568911
highest	14176089	14087105	14235995	14252077	14594733	14689287	14039858	14030230	14233714	14138587	13946607	14001527
timecritical	14045195	13919560	14041047	15386971	18239349	15313710	14087006	13985654	14259001	14102932	14196985	14070567

Table 5: TB C1 + TB C2

	idle		below normal		normal		above normal		high		realtime	
	file	file1	file	file1	file	file1	file	file1	file	file1	file	file1
lowest	14603742	15117713	14938883	14540151	14911693	14673986	14894960	14676578	13885854	14226320	13924175	13961652
normal	14624466	14880906	14690686	15107164	14289394	14744730	14151337	14014401	14230747	13988233	14243199	13973077
highest	14534878	14384543	14518420	14232703	13958206	13637009	14004772	14085803	13988193	14183269	13973070	14196718
timecritical	13839745	13675117	13677465	13760154	13699625	13748402	13900784	13928325	14226305	13860693	13961642	13897231

Table 6: TB C1 + TB C1

	idle		below normal		normal		above normal		high		realtime	
	file	file1	file	file1	file	file1	File	file1	file	file1	file	file1
lowest	16837447	17271279	16511829	17162011	17073705	16849208	17161057	18231717	18151007	17333973	19881950	16894755
normal	17003869	17379259	17348541	17227700	16898134	16758518	16928489	16725374	16550876	16796934	22861297	17709126
highest	8713592	8571329	8654811	8603134	8504893	8717601	8497603	8564439	8738825	8622777	9519391	11603236
timecritical	8472363	8742876	8731660	8440981	8630066	8836007	8739824	8789962	8510754	9850075	8860269	9033792

In all tables, Rows contains thread priorities class and columns are for thread priorities. With increment of thread priority class and thread priority, execution time reduces. As shown in Table-1, Thread A and thread B running on different cores. For thread priorities “High” and thread priority class “time critical” we are getting best performance for both Threads. Difference between Table-1 and Table-2 is one is running on different core and one in same cores respectively. Thread A and Thread B is doing different types of performance task. In this, case Result of Thread A and Thread B can’t be comparable. Both Thread A and Thread B is executed together with different priorities at same time.

As shown in Table-3, Two Thread A running on different cores. For thread priorities “High” and thread priority class “time critical” we are getting best performance for both Threads. Difference between Table-3 and Table-4 is one is running on different core and one in same cores respectively. For same thread running on different core gets good performance for lower priorities. But as priority increases both class priority and thread priority, same core performance is better than different core devices. Thread A is doing only CPU intensive task. Data for both Table-3 and Table-4 is calculated,

As shown in Table-5, Two Thread B running on different cores. For thread priorities “High” and thread priority class “time critical” we are getting best performance for both Threads. Difference between Table-5 and Table-6 is one is running on different core and one in same cores respectively. For same thread running on different core gets good performance for lower priorities. But as priority increases both class priority and thread priority, same core performance is better than different core devices. Thread B is doing memory intensive task.

CONCLUSION:

With this experiment, following things can be concluded:

- With increment of thread priority class and thread priority, execution time reduces.
- Execution time again increases at both realtime thread class and thread priorities.
- Task running at single core than different cores, will give less performance till thread class are low and moderate.
- For same type of task running on different core will give less performance than single core during thread class time critical and realtime priority. This might be different reasons like Cache coherency issue of different task, pipelining arch for scheduling and exchanging data between two cores real time its difficult for two different cores than one core.
- Selection of appropriate properties like core affinity, Thread priorities, and thread class improves thread and process performance.