1. The difference between Mutex and Semaphore is the usage in MultiTasking in programming. Both Mutex and Semaphore is used to allocate resources to tasks, with the difference being that a tasks uses Mutex to lock the task, while Semaphore is used to signal the use of resources to tasks. [Mutex vs Semaphore - GeeksforGeeks](https://www.geeksforgeeks.org/mutex-vs-semaphore/)
2. The function of mutex is to make sure a resource is only used by one task at time. This is achieved by the task using Mutex to lock the resource.
3. Without a way to control tasks use of resources, several tasks could try accessing one resource at the same time. This will cause issues with tasks writing over memory before its used by another task. As well as Deadlocking. [Mutexes and Semaphores Demystified (barrgroup.com)](https://barrgroup.com/embedded-systems/how-to/rtos-mutex-semaphore)
4. The main difference between a Process, Threads and Task is their relation to memory. A Process uses a Memory Management unit for memory access, so does the Thread, but with the addition that it shares data with its process. A task runs within a Thread, and doesn’t utilize MMU. Lecture notes.

# Evaluation of a Multitasking System

* Shits fucked
* In Running Code 0, all the threads run in 137 iterations of a loop. Between each loop iteration, there is a delay in the threads, this is given from the parameters in the “MultiTasking and Realtime”-application. The Duration for each Thread is measured from the printed information and verified by multiplying number of loops by delay. Based on the output, its clear that all the threads start concurrently and finish in the order of the delay between the loops. With a noticeably difference in finish time, depending on the delay in the loops.
* For Running code 1, the threads go through 137 iterations. For this run, the delay in the loops of the threads are disregarded. This is because the threads run in an ascending queue. The total run time of all the threads will in this case have a minimal difference.
* Running code 2

#### Development of a multitasking system

|  |
| --- |
| Initialize static Sephamore;  ThreadClass  {  Initialize integer for loop count;  Initialize integer for delay;  Initialize thread class;  ThreadClass constructor(name, delay)  {  Initialize integer LoopCount = 0;  Sets delay equal to input delay;  Sephamore.WaitOne():  Initializes thread;  Sets name of thread;  }  Run Method()  {  Semaphore release;  Write to console starting thread;  Do While  {  Semaphore WaitOne;  Loop count ++;  Thread sleep with delay;  Semaphore release;  Write Thread name and loop;  }  Write ending of thread;  }  }  Class program  {  Static void main()  {  Write starting program;  Create array of thread classes;  Create string array of thread names;  Create int array of thread loop delay;  Loop that instantiate array of thread classes;  Loop through a count with Thread.Sleep(0);  Write end of program;  }  } |