SUTD 2022 Term 7 50.046 Homework 3

Hand-out: 15 March

Hand-in: 31 March (Thursday 23:59)

Name: James Raphael Tiovalen Student ID: 1004555

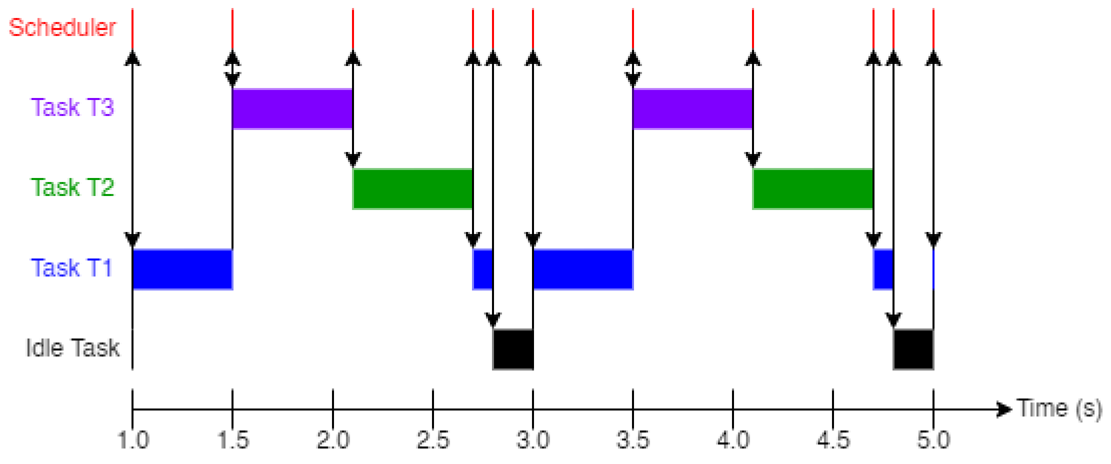
Question 1. [10pt] Consider three FreeRTOS tasks T1, T2, T3, with priority numbers of 1, 2, 3 respectively. The larger the number, the higher the corresponding task's priority.

- Task T1 goes into the ready state at every odd second (i.e., at 1.0s, 3.0s, 5.0s, …).
- Task T2 goes into the ready state at every even second (i.e., at 2.0s, 4.0s, 6.0s, ….).
- Task T3 goes into the ready state 0.5s after Task T1 goes into its ready state (i.e., Task T3 becomes ready at 1.5s, 3.5s, 5.5s ….)
- Each of the three tasks needs 0.6 seconds of dedicated CPU time before it finishes its processing and goes back to the blocked state.

Assume there is only one CPU, and there are no other tasks except for the idle task. Also, assume that a tick interrupt happens every 1 millisecond and the time taken by the FreeRTOS scheduler itself can be ignored.

1) First, assume that there is no inter-dependency among the three tasks. Draw a diagram to show which task is in the running state in which sub-periods, over the time period from 1.0s to 5.0s. **[3pt]**

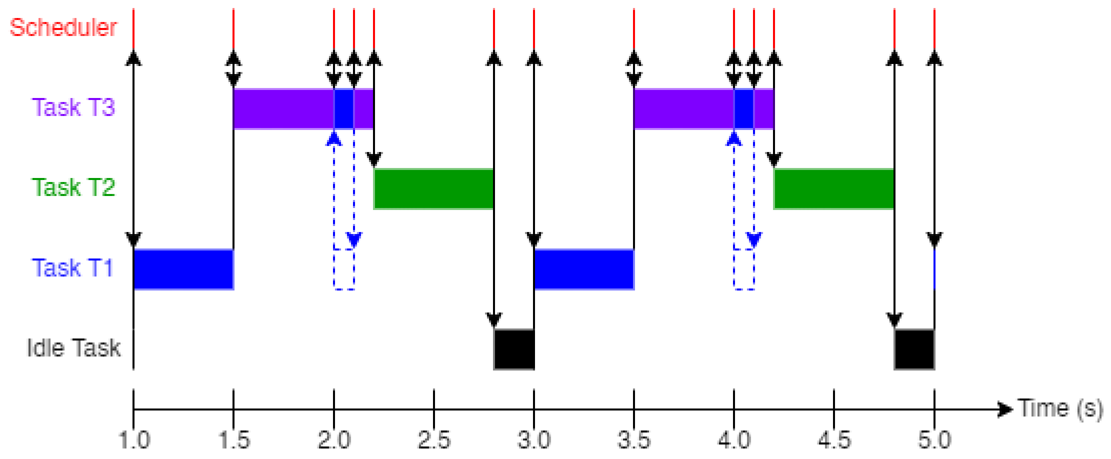
Answer: Ignoring trivial tick interrupts since the time taken by the FreeRTOS scheduler can be ignored, we have this diagram (to scale):



2) Now assume task T1 takes a mutex the moment it goes into the running state and gives back the mutex right before finishing its whole 0.6s of execution (and returning to the blocked state). Task T3 needs to take the same mutex (as task T1) 0.5s after it enters the

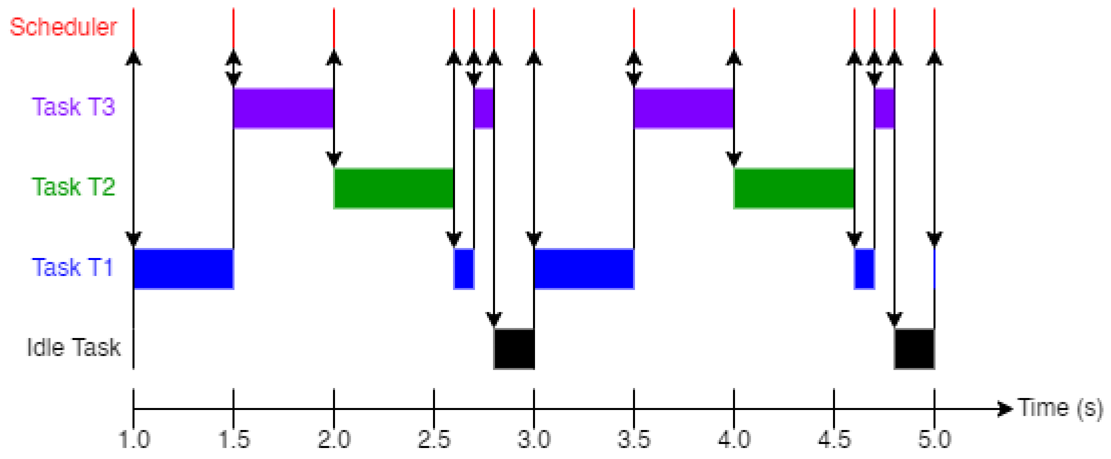
running state and gives it back right before finishing its whole 0.6s of execution (and returning to the blocked state). Draw a diagram to show which task is in the running state in which sub-periods, over the time period from 1.0s to 5.0s. **[4pt]**

Answer: Ignoring trivial tick interrupts as well, then since mutex includes a priority inheritance mechanism, the diagram would be as follows (also to scale):



3) Consider the same setting as setting 2 above. Now assume the embedded device developer (mistakenly) uses a binary semaphore with an initial value of 1 (instead of the mutex) to guard the resource needed by both tasks T1 and T3. Draw a diagram to show which task is in the running state in which sub-periods, over the time period from 1.0s to 5.0s. **[3pt]**

Answer: Ignoring trivial tick interrupts again, then since a binary semaphore does not include a priority inheritance mechanism, priority inversion would occur, and the diagram would be as such (also to scale):



Question 2. [10pt] You are tasked to design an IoT system that collects sensor readings from a remote wind farm that consists of hundreds of wind turbines. The IoT system should be able to actuate an emergency disconnection operation automatically, whenever an anomaly is detected based on the sensed data. In order to minimize the negative physical damage, a wind turbine must be physically disconnected within 10 milliseconds upon an anomaly that happens in the physical world. The control center (as well as the nearest data center that provides cloud

service) is situated 600 km away from the wind farm. The operators in the control center need to be notified about any state change of the wind farm, and they also need to monitor the overall performance of the wind farm and issue various commands (e.g., including remote disconnection commands for maintenance purposes).

1) Assume the actuator device for a wind turbine needs 5 milliseconds to disconnect the turbine upon receiving the disconnect command, and the highest sampling rate of the sensors attached to the turbines is 500Hz, calculate how much time is left for the communication and computation tasks to carry out the automatic disconnection operation. **[2pt]**

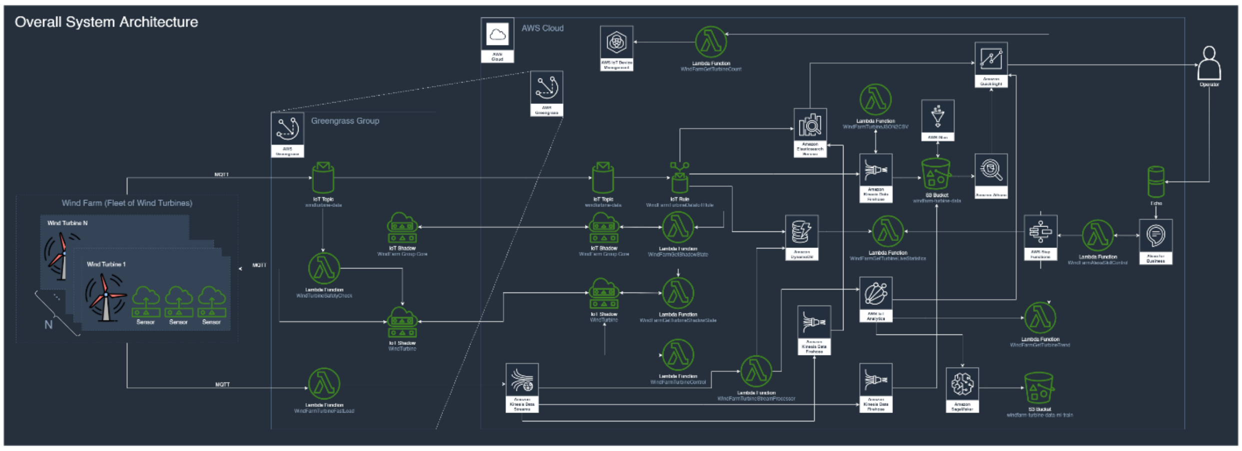
Answer: Time left:

$$\text{Worst case} = 10 - 5 - \left(\frac{1}{500} \times 1000\right) = \mathbf{3\ milliseconds}$$

$$\text{Best case} = 10 - 5 = \mathbf{5\ milliseconds}$$

2) Propose a design to meet the various functional and non-functional requirements mentioned above. Draw an architectural diagram to illustrate your design, including key software, cloud services, and networking technologies (including protocols) that you may use at/between different components. Briefly explain how your design meets the various requirements as in the question description. (You can make additional assumptions in places needed.) **[6pt]**

Answer: My proposed system architectural diagram is as such (also available as a separate full image file included in the submission):


Overall System Architecture
AWS Cloud
AWS Cloud
AWS IoT Device Management
Lambda Function WindFarmGetTurbineCount
AWS Greengrass
Greengrass Group
AWS Greengrass
MQTT
IoT Topic windturbine-data
IoT Topic windturbine-data
IoT Rule WindFarmTurbineDataIoTRule
Amazon Elasticsearch Service
Lambda Function WindFarmTurbineJSON2CSV
AWS Glue
Amazon QuickSight
Operator
Amazon Kinesis Data Firehose
S3 Bucket windfarm-turbine-data
Amazon Athena
Echo
Wind Farm (Fleet of Wind Turbines)
Wind Turbine N
Wind Turbine 1
N
Sensor
Sensor
Sensor
MQTT
IoT Shadow WindFarm Group Core
IoT Shadow WindFarm Group Core
Lambda Function WindFarmGetShadowState
Amazon DynamoDB
Lambda Function WindFarmGetTurbineLiveStatistics
AWS Step Functions
Lambda Function WindFarmAlexaSkillControl
Alexa for Business
Lambda Function WindTurbineSafetyCheck
IoT Shadow WindTurbine
IoT Shadow WindTurbine
Lambda Function WindFarmGetTurbineShadowState
AWS IoT Analytics
Amazon Kinesis Data Firehose
Lambda Function WindFarmGetTurbineTrend
Lambda Function WindFarmTurbineControl
MQTT
Lambda Function WindFarmTurbineFastLoad
Amazon Kinesis Data Streams
Lambda Function WindFarmTurbineStreamProcessor
Amazon Kinesis Data Firehose
Amazon SageMaker
S3 Bucket windfarm-turbine-data-ml-train

To start, each wind turbine would be attached with several sensors that will collect the relevant data and report it to an ESP32 microcontroller, as well as some actuators. Then, we install the AWS IoT Greengrass Core software onto this fleet of ESP32s to ensure edge and near real-time inference capabilities. We use the MQTT protocol to ensure a sufficiently high throughput between the sensors attached to the wind turbines and the Greengrass Group service. We use ESP32s since they support FreeRTOS to ensure determinism. The IoT sensor devices would continuously read and publish sensor values to a local IoT Gateway using AWS Greengrass. There will be an AWS Lambda Function at the edge that would perform local safety checks and automatically actuate an emergency disconnection operation if an anomaly is detected. The gateway would perform a local inference to evaluate turbine safety based on the various data collected, such as rotation speed and vibrations. Another AWS

Lambda Function would stream the collected data through Amazon Kinesis Data Streams to AWS IoT Analytics and Amazon SageMaker on AWS Cloud, which would be used to build and train the machine learning models. Those data would be stored for analytical purposes and visualized over time on a dashboard as well. Device states are managed through the usage of IoT Device Shadows to request changes from the cloud and the edge.

3) The designed IoT system is required to operate with high reliability too. Based on your architectural diagram, describe two different causes that may affect the system's reliability. For each cause, describe how your designed solution can mitigate the reliability risk due to the corresponding cause. **[2pt]**

Answer: Two possible causes of reliability issues include:

A. Individual turbine components might fail due to wear and tear or storms. To mitigate the effects of this issue, the Amazon SageMaker machine learning model included in my designed solution can be used to conduct asset life and health predictions by using the power curve data and energy analysis to predict battery lifetime, and the AWS Step Functions together with Alexa can be used to conduct real-time remote monitoring, detection, diagnosis, assessment, and troubleshooting, as well as to perform energy flow and wind load redirection between the wind turbines in the wind farm if needed.
B. A wind farm can span an enormous geographical land area and local weather conditions might vary between one area and another. To mitigate this issue, weather sensors can be bootstrapped to each wind turbine as well. Since my architecture is edge-oriented due to the presence of IoT Shadows and since the ESP32s are running FreeRTOS which ensures determinism as well as using the MQTT protocol to communicate with Greengrass, it allows for local inference and control, which is much faster as compared to contacting the cloud's side first. In the event of anomalies, emergency disconnection operations can be activated in near real-time and would more likely satisfy the 3 milliseconds time window/interval condition compared to a more cloud-oriented solution. In the case of dangerous weather that might damage the wind turbines, brakes can be activated when the weather sensors detect certain anomalies to pause the operations of the wind turbines, thus preventing any further potential damage to the wind turbines.

-- END --