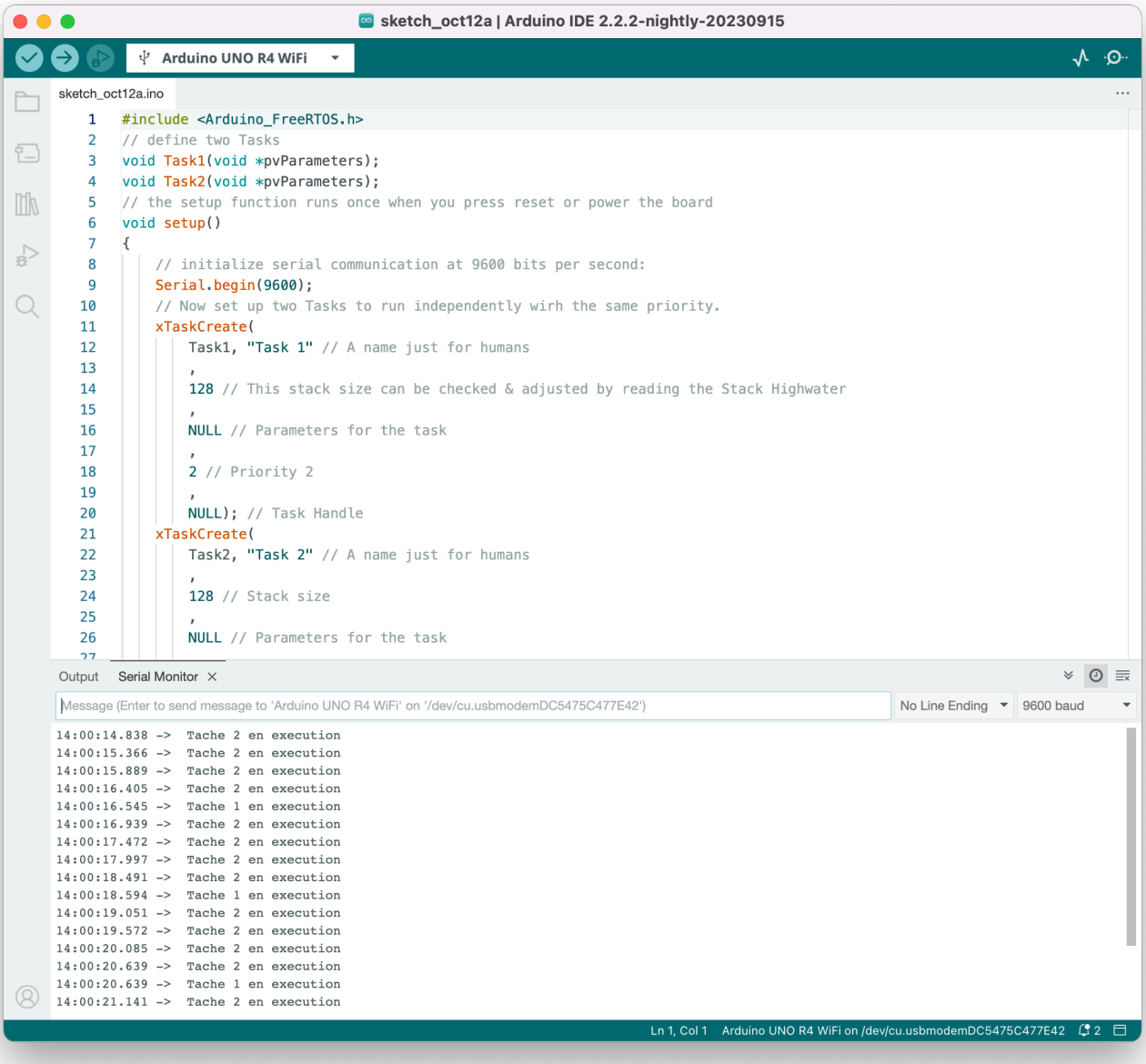
Vincent LAGOGUÉ – David TEJEDA - Thomas PEUGNET
M1 Réseaux et Sécurité – RS3

Programmation Multitâches – Rendu TP03

Exercice 3.1

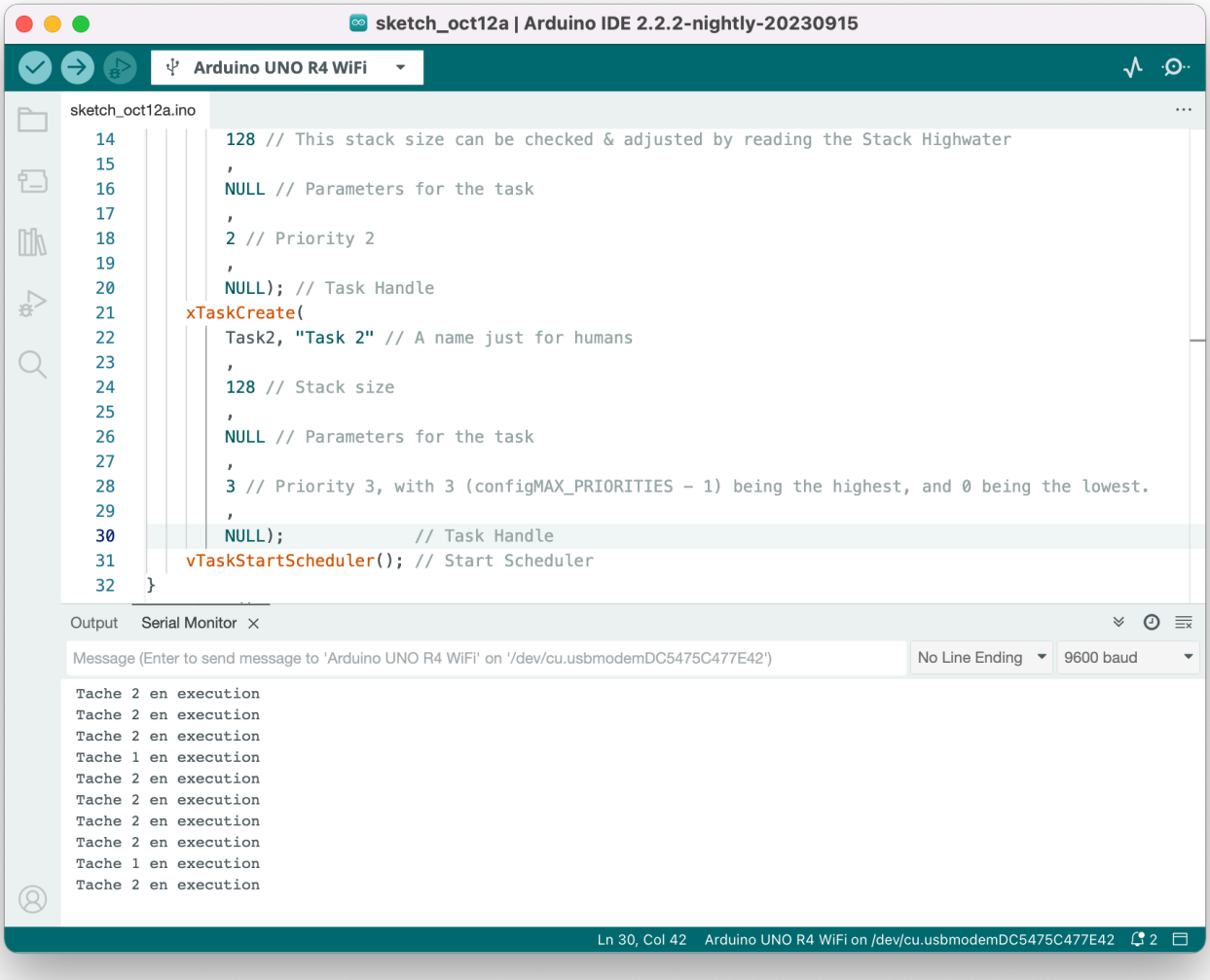
Le code donné dans l'énoncé nous permet d'obtenir le résultat suivant.



Nous pouvons constater que la tâche 1 va afficher une fois « Tâche 1 en exécution » toutes les 2000ms puis se mettra en attente pendant cette durée. Il en est de même pour la tâche n°2, qui elle sera mise en attente (`Blocked`) toutes les 500ms avant d'être mise en état `Ready`, soit 4 fois plus souvent que la tâche n°1. C'est pourquoi nous pouvons voir « Tâche 2 en exécution » 4 fois plus souvent sur notre `Serial Monitor`.

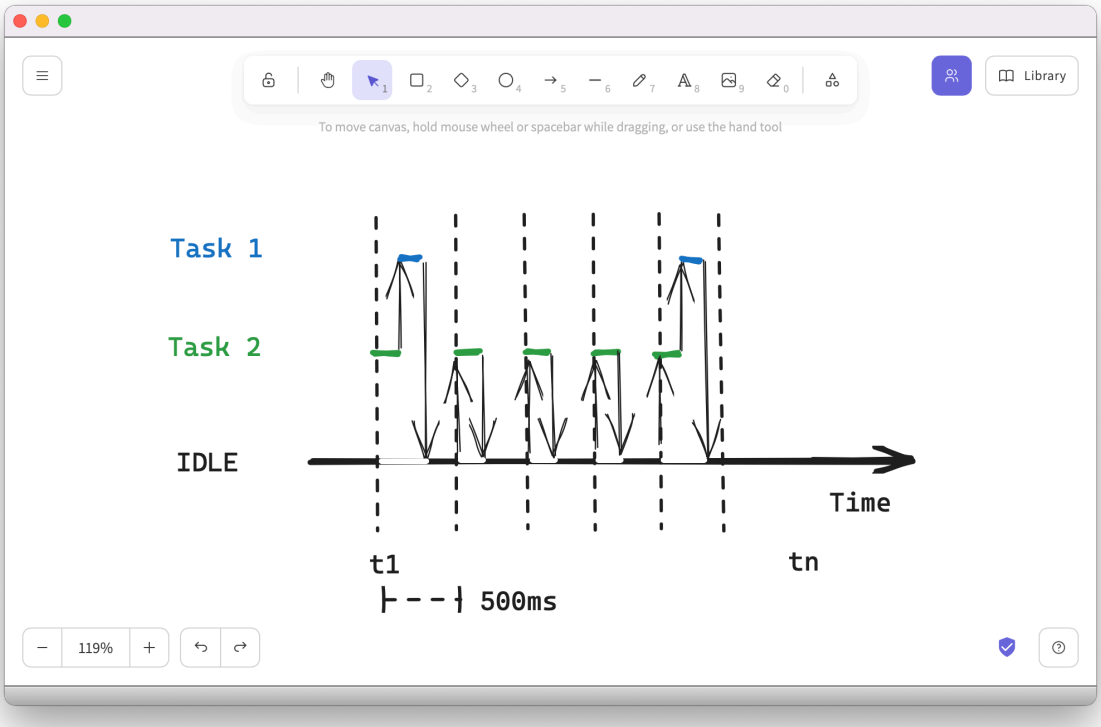
Exercice 3.2

La première tâche réalisée est la tâche n°2. Ce comportement n'est pas étonnant dans la mesure ou cette tâche possède la priorité la plus élevée.



Exercice 3.3

D'après le code donné en énoncé, la séquence d'exécution de ce dernier serait la suivante.



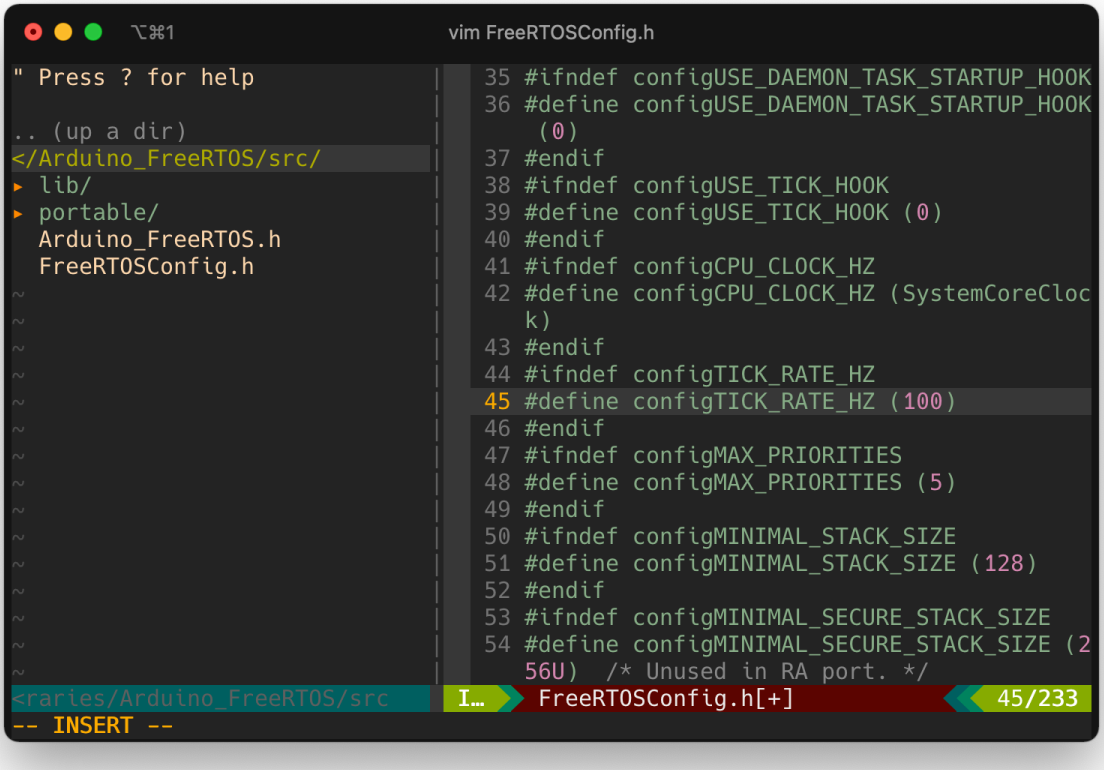
Exercice 3.4

En effet, en commentant ces lignes, nous obtenons un résultat similaire.

La tâche 2 ayant une priorité plus haute, et l'utilisation de la boucle `for` n'étant pas aussi précis (de loin) que l'utilisation de la fonction `vTaskDelay()`, à certains moments, la boucle `for` va s'exécuter de façon sensiblement plus rapide ou plus lente, perturbant donc l'affichage des chaînes de caractères.

Exercice 3.5

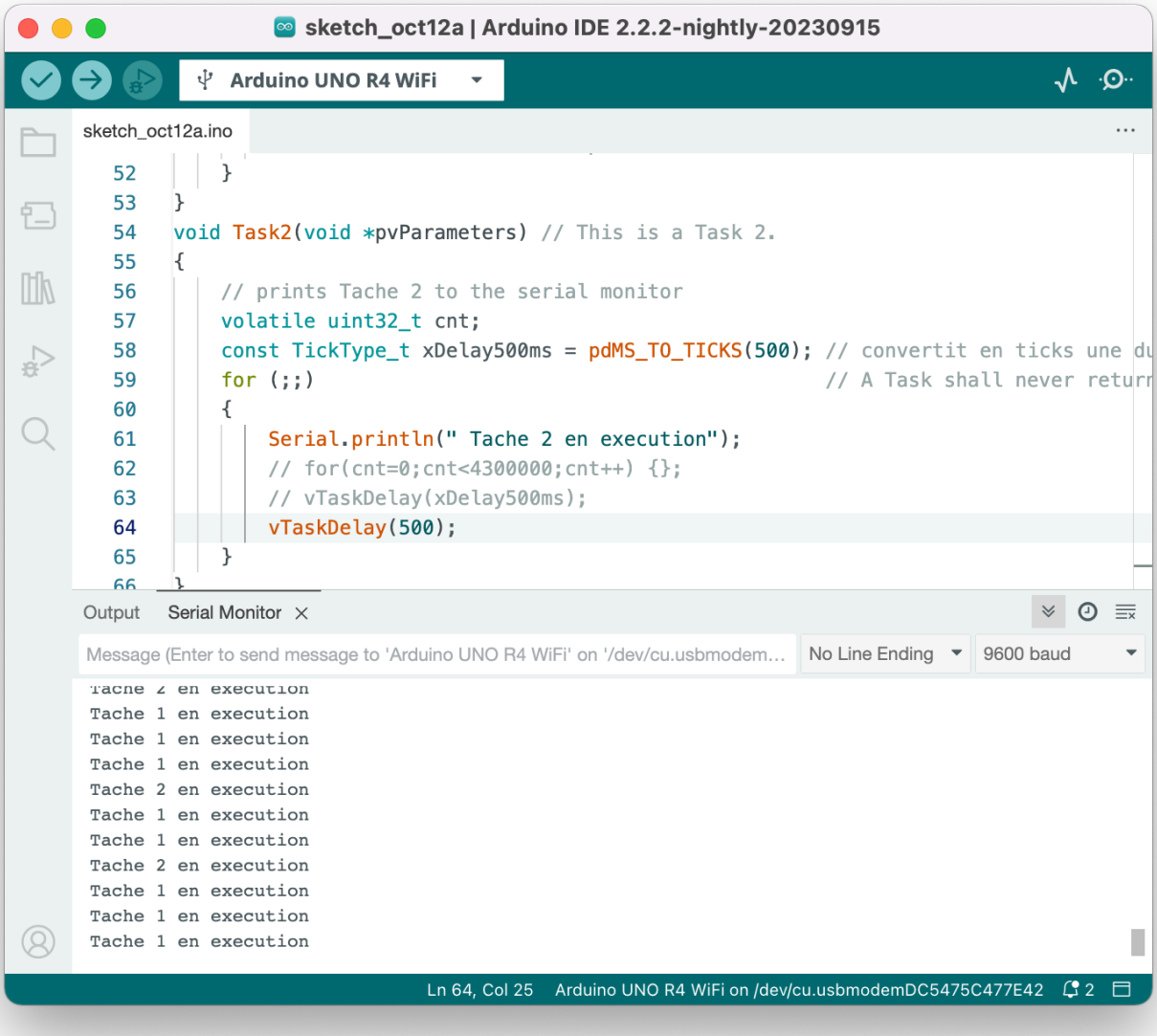
Le fichier édité est le suivant.



Le problème ne semble plus apparaître.

Exercice 3.6

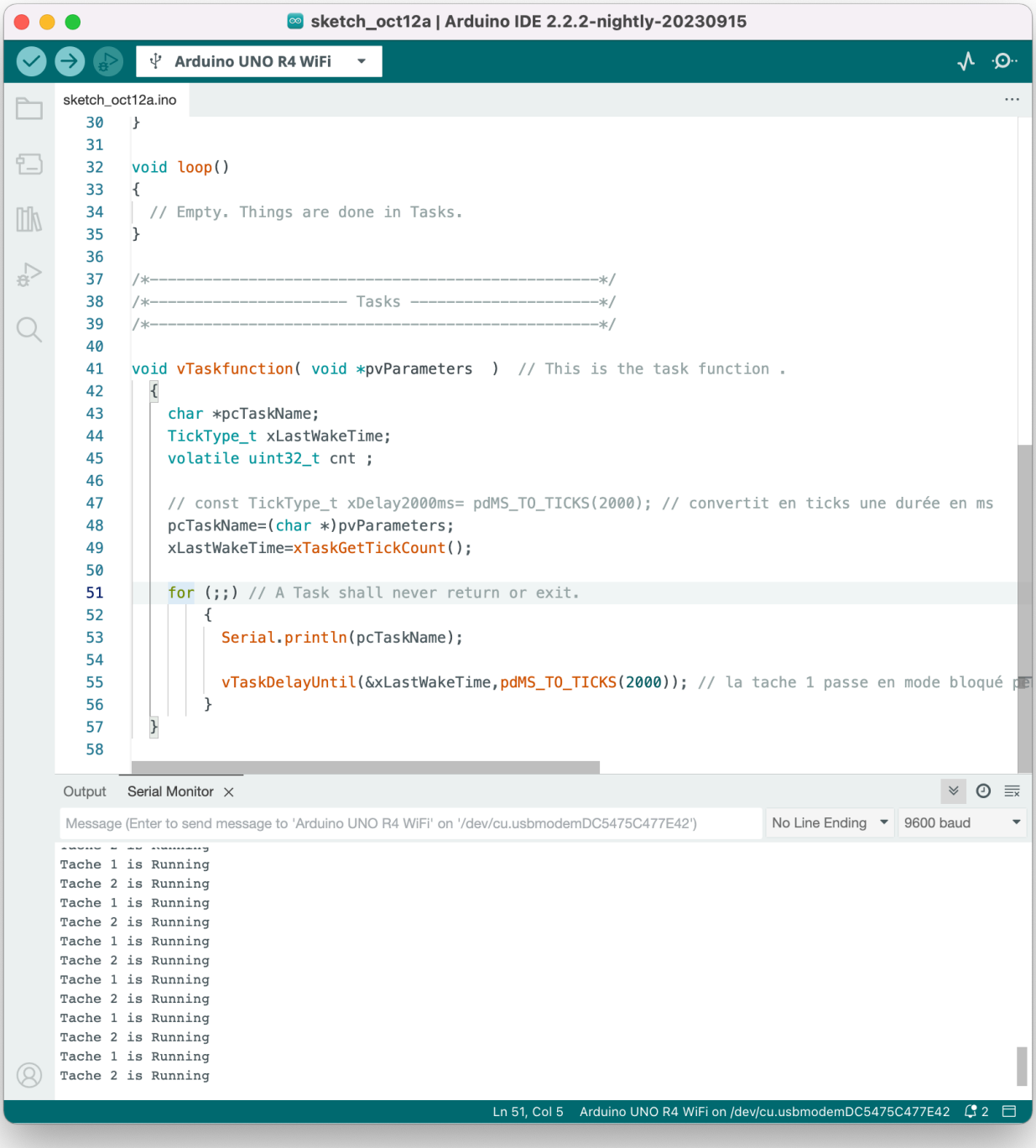
Après modification de la ligne, nous observons le résultat suivant.



La séquence d'exécution est devenue plus lente, et on observe des irrégularités dans le comportement du programme. C'est normal, le `vTaskDelay(500)` attend la durée de `500 Tick`, et non `500ms`.

Exercice 3.7 & 3.8

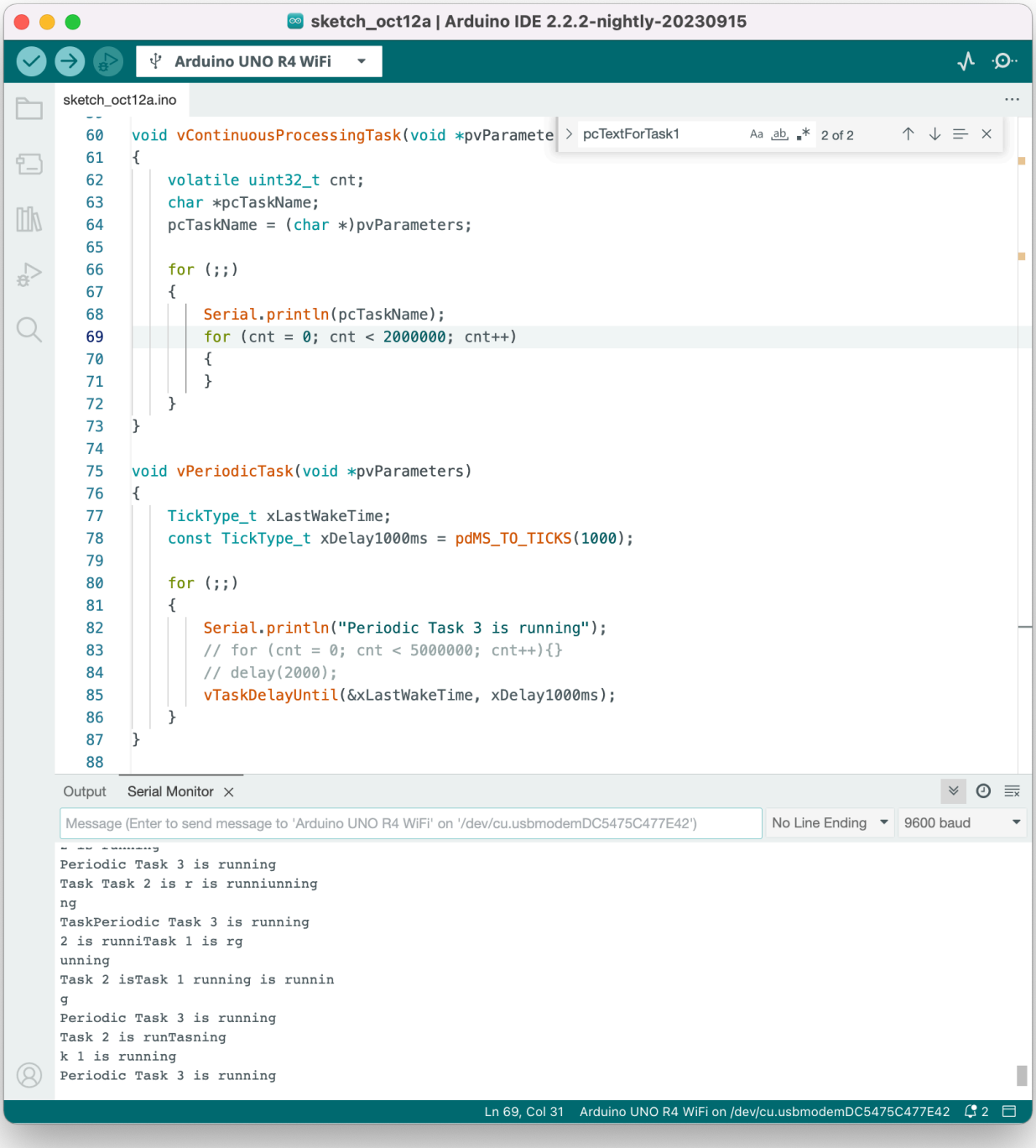
Nous obtenons le résultat suivant.



Nous pouvons observer une parfaite régularité d'exécution, contrairement à ce que nous pouvions constater dans l'exemple 6.

Exercice 3.9

Après avoir exécuté la tâche, nous obtenons le résultat suivant.

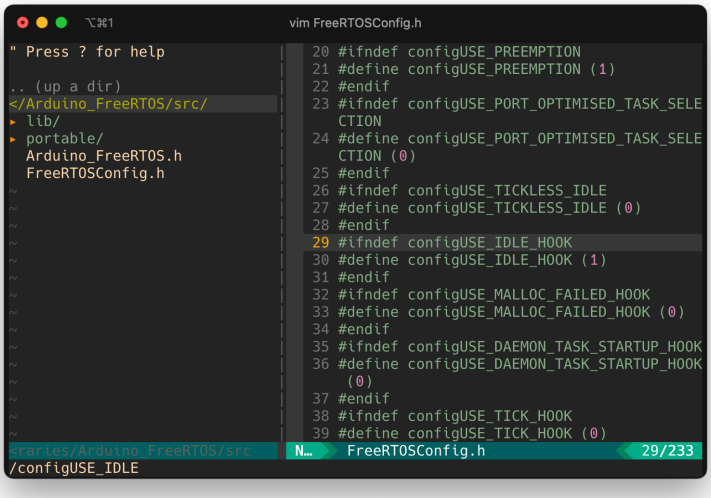


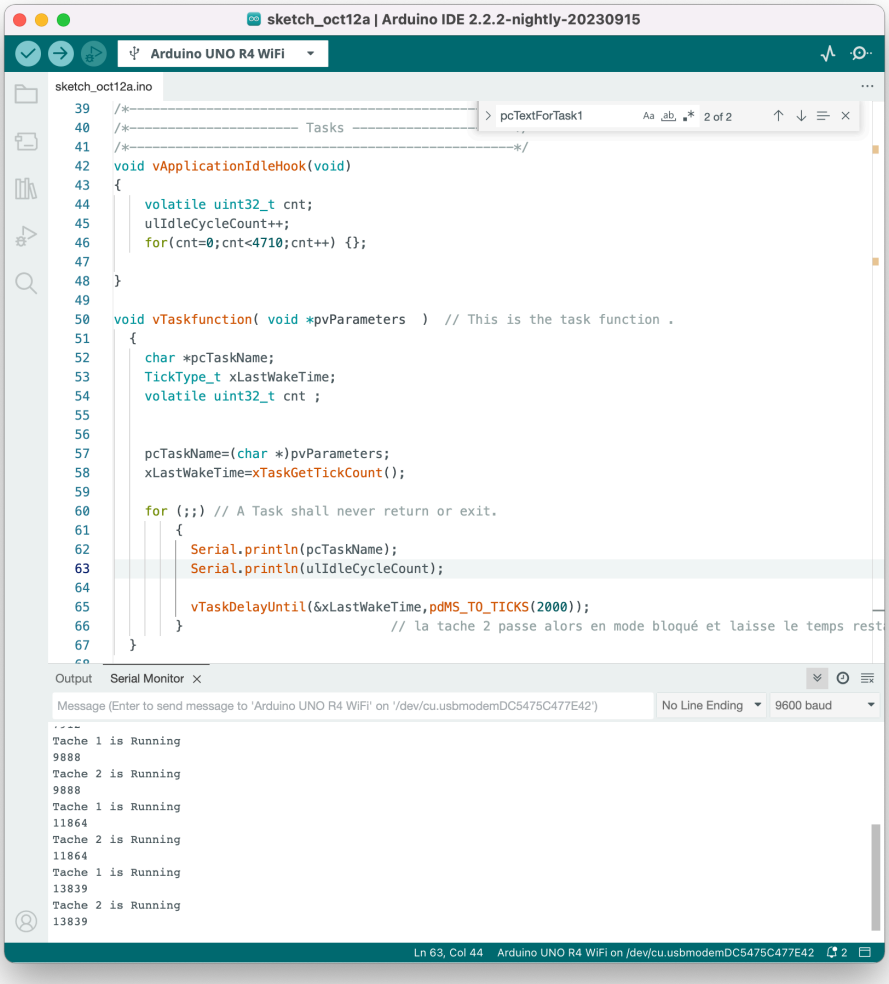
Il n'est pas étonnant. La tâche 3 est exécutée de façon périodique en suivant le nombre de `ticks` absolu. Elle s'exécute donc de façon régulière. Cependant, les deux autres tâches ne sont pas mises en état `Blocked`, et son déclenchées en décalé lorsque la tâche 3 devait s'exécuter en même temps qu'eux. (La priorité de la tâche 3 étant plus élevée, elle passe en première.)

Le modèle d'exécution présenté ensuite dans l'énoncé illustre parfaitement cette hypothèse.

Exercice 3.10

Il semble que la constante `configUSE_IDLE_HOOK` soit déjà mise à 1 dans le fichier `FreeRTOSConfig.h`.





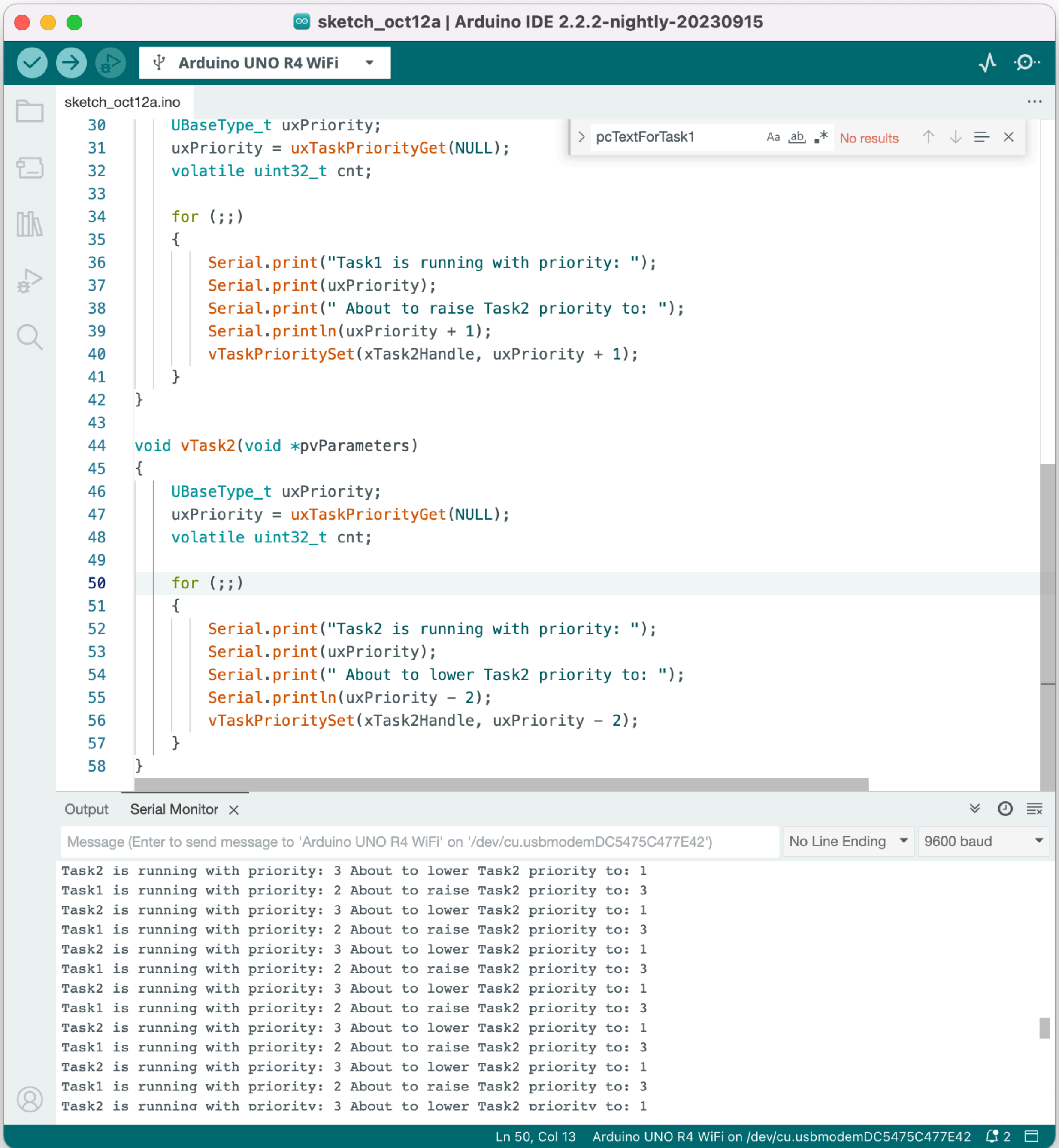
Nous pouvons constater que le compteur incrémenté dans `vApplicationIdleHook()` indiquant le nombre de ms écoulées en `Idle` est en constant augmentation entre l'exécution de la tâche 2 et la tâche 1, mais pas entre la tâche 1 et la tâche 2, indiquant donc que les deux tâches sont exécutées à la suite.

Vincent LAGOGUÉ – David TEJEDA - Thomas PEUGNET
M1 Réseaux et Sécurité – RS3

Exercice 3.11

Partie A

Nous obtenons le résultat suivant :

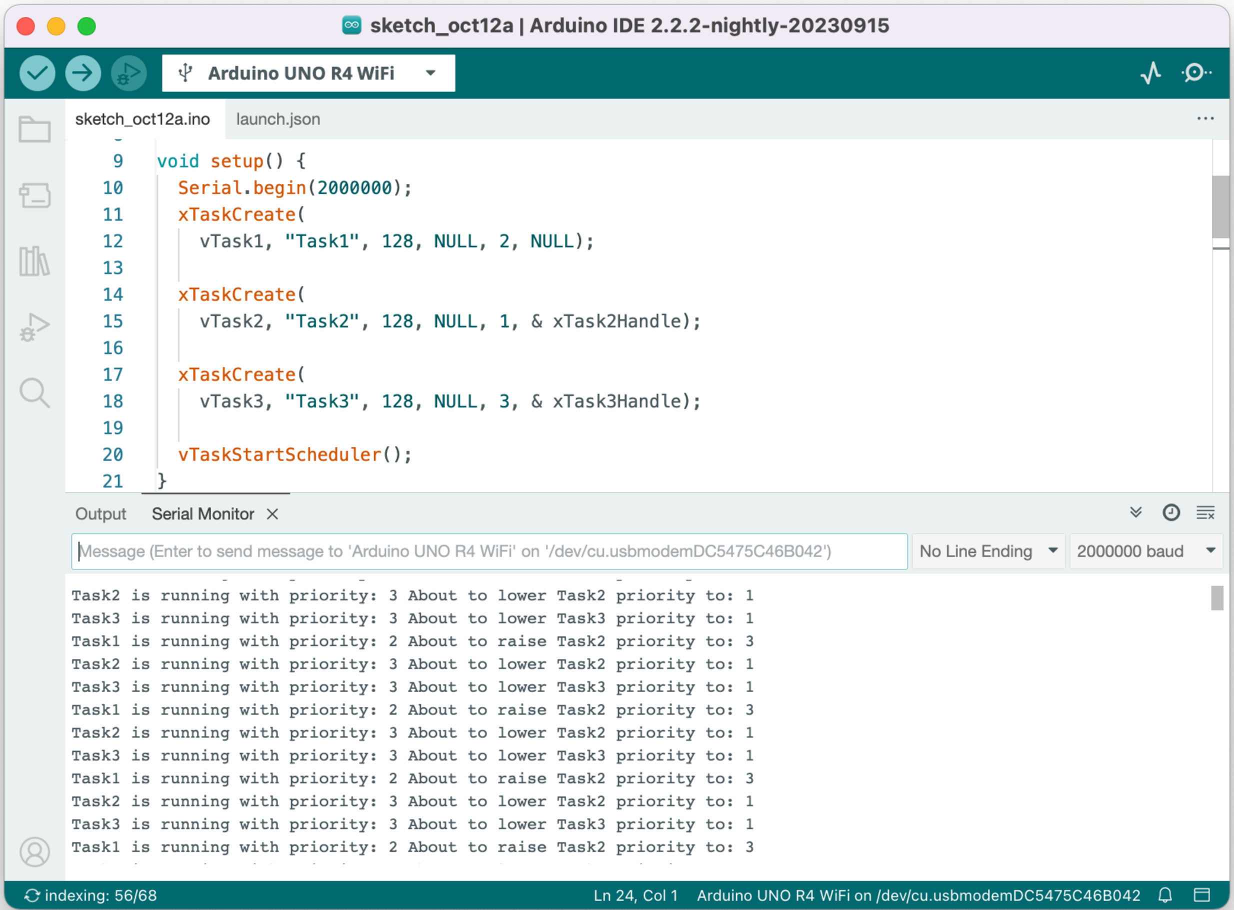


Nous pouvons constater que le déroulement du programme s'effectue de façon régulière. La tâche 2 passe de la priorité 3 à la priorité 1, puis de 1 à 3 et ainsi de suite. La priorité de la tâche 1 n'étant pas changé, cette dernière s'exécute avant la tâche 2 quand cette dernière possède une priorité à 1.

Vincent LAGOGUÉ – David TEJEDA - Thomas PEUGNET
M1 Réseaux et Sécurité – RS3

Partie B

Pour obtenir l'ordre des tâches Task1, Task2, Task3, il faut définir une nouvelle tâche et jouer un peu sur les priorités.
On obtient le résultat suivant :



Le code des différentes tâches est le suivant :

Vincent LAGOGUÉ – David TEJEDA - Thomas PEUGNET
M1 Réseaux et Sécurité – RS3

Exercice 3.12