LYCÉE ARISTIDE BERGÈS

SIO

401 cours planification (CRON)

Planification des actions

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision
21 février 2022	Création

Sommaire

A Introduction

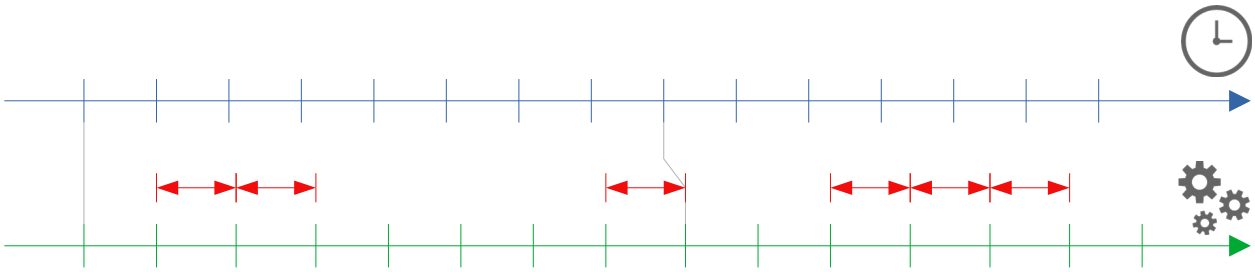
A.1 Description générale

La planification existe depuis très longtemps sur les systèmes. Elle répond à un besoin d'exécuter des scripts ou des codes, à intervalles réguliers. La planification répond aux besoins suivants :

- toutes les 5 minutes, j'aimerais…
- une fois par jour à 5h00, je voudrais…
- un jour sur deux, il faut…

Il est possible de développer soi-même une surveillance horaire, dans une boucle, mais c'est plus compliqué en programmation asynchrone (JS).

La solution serait de programmer des intervalles, mais nous avons déjà vu que les fonctions comme setTimeout et setIntervall sont dépendantes de la capacité du moteur à traiter les événements sans être bloqué. En clair, en programmant un intervalle de 60 secondes, l'exécution finira par se décaler par rapport à l'heure souhaitée. Dans le dessin ci-dessous, l'heure (en bleue) sert de référence, mais les commandes sur les timers sont dépendantes de l'exécution de codes. Lorsqu'un code prend trop de temps (en rouge) il décale la prochaine exécution, et finalement, la ligne de temps du programme (verte) n'est plus alignée avec l'horloge système.



La solution est de s'appuyer sur une fonction système, basée sur les interruptions processeurs et sur le composant RTC du matériel.

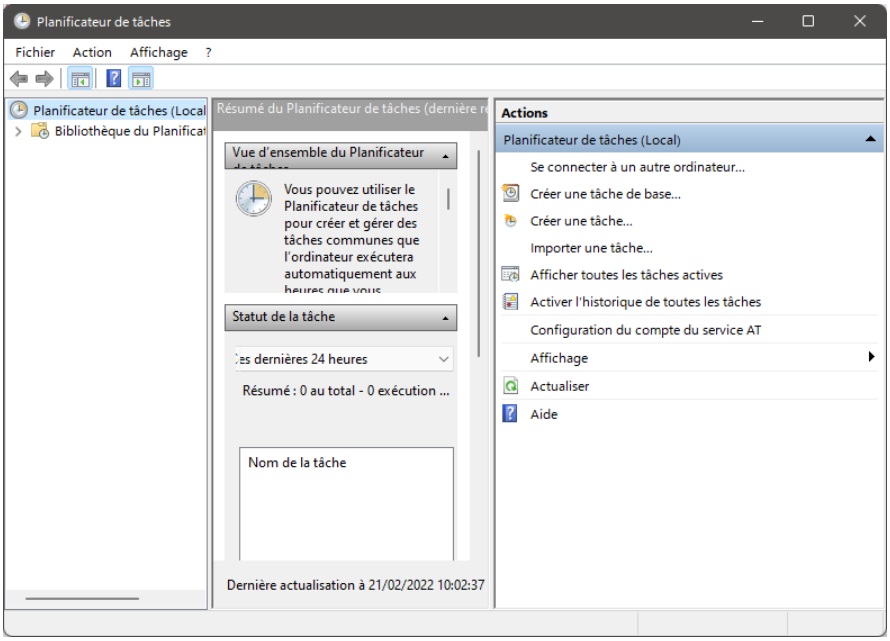
Les interruptions sont des événements généralement non-masquables que le microprocesseur reçoit et traite immédiatement. Les interruptions ont des priorités différentes, mais globalement, se lancent assez rapidement.

A.2 Les planifications de tâches

Dans tous les systèmes d'exploitation, on retrouve cette possibilité de programmer des tâches à intervalles réguliers.

A.2.1 Task Scheduler

Sous Microsoft Windows, on peut utiliser le planificateur de tâches (taskschd.msc), mais sa programmation n'est pas aussi facile, en ligne de commandes.



Exemple de commande en ligne, pour lancer RunMe.bat tous les jours à 09h00 :

```
SchTasks /Create /SC DAILY /TN “My Task” /TR “C:\RunMe.bat” /ST 09:00
```

De plus, la plupart des environnements de développement sont dans un environnement Linux.

A.2.2 Cron

Cron est un démon (un service) lancé avec le système d'exploitation au moment du démarrage. Il est capable de gérer des intervalles de temps allant de la minute (la seconde pour certains systèmes) à l'année.

Étant en ligne de commande, sa syntaxe est assez intuitive.

```
30 23 * * Mon, Tue, Wed, Thu, Fri /usr/local/bin/backup
```

La ligne ci-dessus, lance une sauvegarde à 23h30 les jours de semaine.

Cron propose 7 paramètres séparés par des espaces :

*	*	*	*	*
Minutes	Heures	Jours (1-31)	Mois	Jour (0 à 6e)
0	5	5,9,19	1-6	*

Ainsi que quelques caractères permettant d'exprimer des valeurs ou manières de fonctionner :

- * est un jocker pour toutes les valeurs
- ? est également un jocker
- */x signifie un intervalle de x (tous les x moments)
- , permet de placer plusieurs valeurs (exemple : 5,9, 19 exécutera au moment 5, 9 et 19)
- - permet de choisir un intervalle (exemple 1-6 exécutera au moment de 1 à 6)

A.2.3 Outils de vérification d'expression

Si la syntaxe de cron semble simple, il y a parfois quelques pièges. Je vous propose quelques outils en ligne, permettant de vérifier la syntaxe d'une expression existante, ou bien la création d'une expression à partir d'un besoin.

https://crontab.guru

http://crontab.cronhub.io

https://codebeautify.org/crontab-format

Prends en charge le format contenant les secondes et les années :

https://www.jstoolset.com/cron

B Programmation sous Node.JS

Puisque Node.JS s'exécute sur un serveur, il peut être intéressant de lancer des actions régulières. Nous allons donc créer une simple démonstration pour utiliser la bibliothèque 'node-scheduler' du MIT.

L'action à mener pourrait être la surveillance d'une messagerie instantanée, et des comptes inactifs depuis 5 minutes, un peu comme le bot "Clyde" dans Discord dans les conversations vocales.

B.1 Modélisation

Nous utiliserons une collection d'utilisateur, avec une propriété 'DerniereEmission' qui contiendra la date de la dernière transmission.

L'objectif sera de déconnecter les utilisateurs n'ayant rien émis depuis 5 minutes. Si un utilisateur émet un message, il suffirait de modifier la propriété en y plaçant la nouvelle date d'émission.

Notre jeu de test sera le suivant :

idConnexion	userName	Lastmessage
5612	Bob	date()-240000
5984	Anna	date()-150000
5985	Okko	date()-400000
6001	Eve	date()-8000000
6005	Aziza	date()-1000

Notre fonction de nettoyage s'exécutera toutes les minutes mais ne supprimera que les comptes ayant dépassé 5 minutes d'inactivité.

Rappel : le format date en JavaScript utilise le nombre de millisecondes écoulées depuis le 01/01/1970 00:00.

5 minutes	5 × 60 × 1000 millisecondes

Logiquement, les dates en rouge dans le tableau disparaîtront dès la première minute, puis celles en orange la minute suivante, etc.

B.2 Création du code

Créez un dossier "cron-test" et ouvrez votre éditeur dedans. Créez un nouveau projet avec la commande npm.

```
npm init
```

puis ouvrez le fichier 'index.js' et essayez de créer vous même la fonction de suppression de des utilisateurs.

La bonne méthode est certainement de travailler sur un tableau d'objets JSON et pour la suppression, de faire une boucle partant de la fin du tableau (le risque – en ne tenant pas compte de ce conseil – étant de ne pas permettre un parcours correct du tableau, la suppression d'un élément courant décalant les index et modifiant la longueur du tableau).

Pour mettre en œuvre le framework node-schedule vous pouvez vous référer à la documentation ici.

Enfin, si vous n'avez pas la patience de créer vous-même ce bout de code, vous trouverez la solution à la page suivante. Vous pouvez vous en inspirer, mais comme toujours, pour maîtriser la notion, il faudra étendre ce code.

- Modifier le cron pour effectuer le test toutes les 2 minutes
- Ajouter un cron pour afficher un avertissement aux utilisateurs ayant plus de 4 minutes d'inactivité
- Imaginer une situation pour afficher un graphique en temps réel du temps restant aux utilisateurs (axe Y = utilisateurs, axe X = durée inactivité)
- etc.

C Solution proposée

Le code qui suit n'est pas le meilleur, mais est opérationnel.

```
/**
 * @author "David ROUMANET" <david.roumanet@ac-grenoble.fr>
 * @version 1.0
 * @description exemple d'utilisation du framework node-schedule du MIT
 */

const { Console } = require('console');

// cron example: https://crontab.guru)
// Cron format : SS MM HH Day Month weekday
let schedule = require('node-schedule');
let d = new Date()
let table = [{idConnexion:5612  userName:"Bob",  lastMessage: d-240000},
             {idConnexion:5984  userName:"Anna",  lastMessage: d-150000},
             {idConnexion:5985  userName:"Okko",  lastMessage: d-400000},
             {idConnexion:6001  userName:"Eve",  lastMessage: d-8000000},
             {idConnexion:6005  userName:"Aziza",  lastMessage: d-1000}
]

let checkInactive = () => {
    let d = new Date()
    let index = table.length-1
    while (index >= 0) {
        if (d-table[index].lastMessage > 300000) {
            console.log(table[index].userName, "REMOVED")
            table.splice(index, 1)
        }
        index -= 1
    }
    afficherActif()
}

let afficherActif = () => {
    console.log("---------------------------------")
    table.forEach(element => {
        let d = new Date()
        console.log(element.idConnexion, element.userName, (d-
element.lastMessage)/60000)
    });
}

let every1m = schedule.scheduleJob('* */1 * * *', () => {
    // Annonce du lancement de la tâche
    console.log("Lancement du nettoyage des inactifs...")
    checkInactive()
})
console.log("Planification OK (si non nul)", every1m)
afficherActif()
```