Exercices multithread POSIX et driver Linux

Les exercices suivants ont pour objectif de faire des premiers développements multitâches sur POSIX et de driver Linux

Prise en main

Copier dans un répertoire les fichiers
SimulSensor.c
testDriver.c
unDriver.h
unDriver.c
Makefile
InstallDriver.sh

Rendre le fichier InstallDriver.sh exécutable (commande *chmod a+x InstallDriver.sh*)

Construire les exécutables et installer le driver, les périphériques
- Dans le répertoire où sont placés les fichiers
- Commande *make* (construction du driver)
- Commande *sudo ./InstallDriver.sh* (installation du driver et des devices)
- Commande *gcc –pthread SimulSensor.c –o SimulSensor* (construction de l'émulateur basé sur POSIX)
- Commande *gcc testDriver.c _o testDriver* (construction du programme de test)

Faire un test
- Ouvrir 2 terminaux
- 1er terminal, commande *./SimulSensor&* (lancement de l'émulateur)
- 2ème terminal, commande *./testDriver* (lancement de l'application, à faire plusieurs fois)
- 1er terminal, commande *pkill -9 SimulSensor*
- 2ème terminal, commande *./testDriver* (lancement de l'application, à faire plusieurs fois)

1er exercice : vérification de donnée

Ajoutez un fichier sensor.h qui contient les valeurs maximale (100) et minimale(0) possibles de la valeur capteur de température. Si lors de l'écriture la valeur est hors des plages, la valeur est incorrecte (isCorrect est à 0), sinon correcte (isCorrect est à 1). La lecture d'une donnée incorrecte ne renvoie rien (0 octets lus).

Cachez l'information time en l'enlevant de la structure sensorInfo. Et lors de la lecture, vérifiez l'age de la donnée, au delà d'un certain temps (spécifié dans sensor.h à 2 secondes), la donnée n'est plus considérée comme correcte.

2° exercice : ioctl

Ajoutez une commande de RAZ du compteur de ticks qui ne fonctionne que sur myDevice0.

3° exercice : simulation de plusieurs capteurs

On suppose que les capteurs communiquent avec le driver via un réseau. Un capteur est alors caractérisé par son adresse (type int) et sa valeur.

Le programme SimulSensor.c doit émuler les envois de message pour les deux capteurs myDevice1 et myDevice2. Dans sensor.h, construire un type de structure de message (adresse + valeur) et fixer les deux adresses adSensor1 et adSensor2 des deux capteurs, respectivement à 11 et 21.

La communication entre le simulateur et le driver se fait en utilisant myDevice0, utilisé comme périphérique de communication. Il ne correspond pas à un capteur réel.

Modifier SimulSensor.c pour que l'envoi de message sur myDevice0, correspondant à l'émulation de myDevice1, respectivement myDevice2, se fasse environ toutes les 1, respectivement 2, secondes.

La valeur envoyée pour myDevice1, resp myDevice2, suit la loi suivante : valeur initiale à -10, incrément de 10, resp. 20, à chaque envoi jusqu'à la valeur 120, puis passage à nouveau à la valeur -10.

Le driver doit mémoriser les informations par capteur, afin que l'opération read renvoie la valeur correspondante au périphérique myDevice1 ou myDevice2 (myDevice0 n'est considéré qu'en écriture).

Le programme de test lit toutes les secondes les valeurs correctes de myDevice1 et myDevice2.

Pensez à renvoyer une erreur pour une opération de lecture, resp. d'écriture, sur myDevice0, resp. myDevice1 ou myDevice2.

4° exercice : multithread POSIX

Dans le programme de test, créez un thread pollS2 activé toutes les 1 ms pour scruter myDevice1 et un thread pollS2 activé toutes les 2 ms pour scruter myDevice2. A chaque nouvelle acquisition correcte, chaque thread met à jour une donnée globale valMoyenne, calculée comme étant la moyenne des deux dernières valeurs lues (ou une seule si la deuxième est incorrecte). Si les deux dernières acquisitions n'ont pas récupéré de valeur correctes, valMoyenne est égal à -1. Enfin, un troisième thread Print affiche la valeur de valMoyenne après chaque modification d'un capteur, les deux données étant correctes.