ESISAR

NE449 – Programmation répartie

TDM numéro 6

Table des matières

1 Présentation de la problématique....1
1.1 Présentation générale....1
1.2 Description détaillée pour la lecture sur la carte SD....2
1.3 Description détaillée du lien SPI vers la puce BROADCOM....3
2 Exercice 1 – Réalisation d'une solution « brutale »....3
2.1 Description de la solution attendue....3
2.2 Comment utiliser le fichier airbag.jar....4
2.3 Le travail à réaliser....5
3 Exercice 2 – Réalisation d'une solution optimale à l'aide d'un buffer tournant....6
3.1 Description de la solution attendue....6
3.2 Le travail à réaliser....6

1 Présentation de la problématique

1.1 Présentation générale

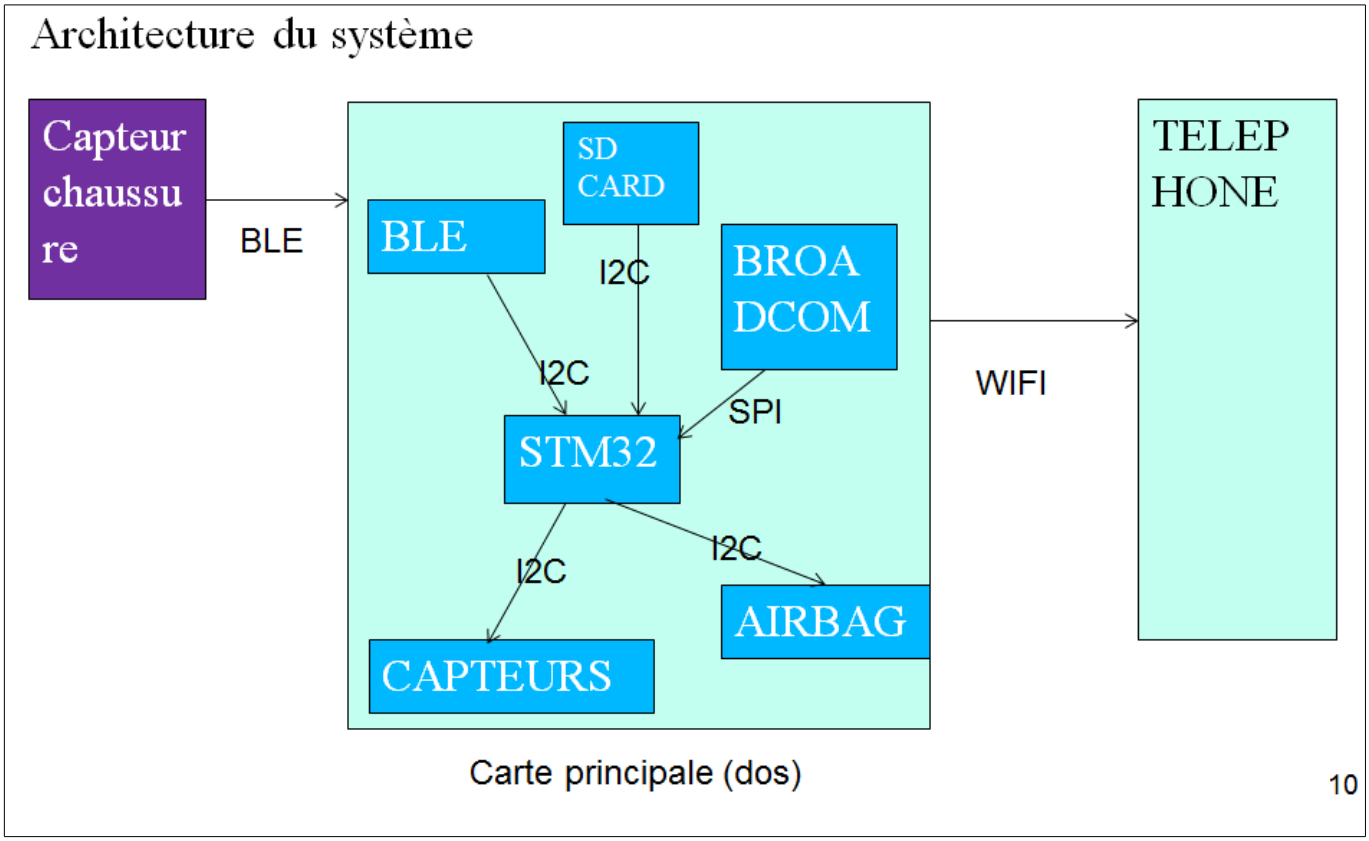
Pour ce TD, vous allez reprendre l'exemple « Airbag pour skieur » vu lors du premier cours.

Pour mémoire, voici deux liens avec des vidéos expliquant le fonctionnement du système :

https://www.youtube.com/watch?v=lnFnuFvXA6Y

https://www.youtube.com/watch?v=-R5JWthkX54

Voici le synoptique du système :



L'objectif de cet exercice est de développer un programme sur le micro contrôleur STM32 permettant l'envoi des données d'une course sur le téléphone du skieur.

Les données d'une course sont stockées sur une carte SD. Si le skieur demande l'envoi du fichier de la course, alors :

- le STM32 lit le fichier sur la carte SD et l'envoie à la puce BROADCOM
- la puce BROADCOM envoie le contenu de ce fichier au téléphone par WIFI

La mémoire RAM sur la puce STM32 est très limitée (environ 16 Ko octets pour tout le programme). Le fichier correspondant à une course fait une taille d'environ 50 Ko.

1.2 Description détaillée pour la lecture sur la carte SD

La puce STM32 possède une API pour la lecture de la carte SD. Cette API est similaire à l'API classique de lecture de fichier sur un PC. Cette API comprend une seule fonction

```
int readDataOnSDCard(byte[] buffer,int start,int len):
```

- les caractères lus sont positionnés dans le **buffer** à partir de la position **start**
- **len** correspond au nombre maximum de caractères qui seront lus
- cette fonction retourne le nombre d'octets réellement lus (donc un nombre compris entre 0 et len). Retourne -1 quand on a atteint la fin du fichier

1.3 Description détaillée du lien SPI vers la puce BROADCOM

La puce STM32 possède une API pour l'envoi des données sur SPI. Cette API comprend une seule fonction

```
int sendSpiData(byte[] buffer,int start,int len) :
```

- **buffer** contient les données à envoyer
- **start** correspond à la position du premier caractère à envoyer
- **len** correspond au nombre de caractères à envoyer
- cette fonction retourne le nombre d'octets envoyés (nombre compris entre 0 et len)
- cette fonction n'est pas bloquante, mais retourne 0 si elle ne peut pas envoyer les données tout de suite

Par construction, le fonctionnement du lien SPI est optimal si la taille des données envoyées est aux alentours de 128 octets. En cas d'envoi de petites trames, le lien SPI devient lent (le temps d'envoi d'une petite trame est quasi égal au temps d'envoi d'une longue trame, donc il est préférable d'envoyer des grandes trames). Par contre, le nombre d'octets envoyés est toujours inférieur à 256 (pas de trames supérieures à 256). **Le délai entre deux trames SPI est de 10 ms.**

2 Exercice 1 – Réalisation d'une solution « brutale »

2.1 Description de la solution attendue

Vous avez été missionné pour développer le programme de la puce STM32. Votre objectif est d'obtenir le transfert du fichier de la course dans le temps le plus bref possible (donc d'obtenir un débit maximum).

Etant donnée les caractéristiques mémoire de la puce BROACOM, vous disposez **d'un seul buffer interne** de 2 Ko.

Vous allez développer sur PC en JAVA un programme ayant le même comportement que celui de la puce STM32. Vous trouverez sur Chamilo un fichier airbag.jar permettant de simuler la liaison SPI et la liaison avec la carte SD.

Votre programme aura le comportement suivant :

```
création d'un buffer de 2 Ko
tant que la fin du fichier de la SDCard n'est pas atteinte
     lire x octets depuis la carte SD vers le buffer
     si x>0
     envoi des x octets vers la puce BROADCOM par SPI
fin tant que
```

Pour envoyer les x octets sur le lien SPI, le fonctionnement général sera le suivant :

```
faire un appel à la fonction sendSpiData()
la méthode retourne « y octets envoyés »
si x == y  (tout a été envoyé)
     fin de l'envoi
sinon
     refaire un appel à la fonction sendSpiData()
     et continuer tant que les x octets n'ont pas été envoyés
```

2.2 Comment utiliser le fichier airbag.jar

Voici la procédure pour utiliser le fichier airbag.jar :

1/ Télécharger le fichier **airbag.jar** depuis Chamilo avec le navigateur Chrome.

2//Créer votre projet sur Eclipse, par exemple avec le nom « td6_exo1 »

3/Copier ce fichier **airbag.jar** dans votre projet Eclipse, directement dans le répertoire td6_exo1 (en faisant par exemple un drap and drop depuis Chrome).

4/Vous faites ensuite « clic droit » sur le projet « td6_exo1 », puis « Properties » , puis « Java Build Path » puis onglet « Library »

5/Vous faites ensuite « Add Jars », vous sélectionnez le fichier « airbag.jar », puis « OK »

Maintenant, vous pouvez utiliser le code suivant pour démarrer :

```java
import tdm.tdm06.airbag.Airbag;

/**
 * Exemple de départ pour l'utilisation du fichier airbag.jar
 *
 */
public class Exo1Depart
{
      public static void main(String[] args)
      {
```

```
		Exo1Depart exo1 = new Exo1Depart();
		exo1.execute();
	}

	private void execute()
	{
		Airbag airbag = new Airbag();
		byte[] buf = new byte[1024];


		// Exemple pour lire sur la carte SD
		int nbRead = airbag.readDataOnSDCard(buf, 0, 1024);

		// Exemple pour envoyer les données par SPI
		int nbSent = airbag.sendSpiData(buf, 0 , nbRead);

	}

}
```

2.3 Le travail à réaliser

Question 0 : Faire d'abord un programme qui affiche à l'écran le contenu du fichier qui est sur la carte SD. Dans quelle station de ski se passe cette course ? Quelle est la vitesse (approximative) du skieur en fin de course ? Quelle est la taille du fichier ? A chaque appel à la fonction readDataOnSDCard, quel est le nombre d'octets lus en moyenne ?

Nota : le fichier est différent à chaque exécution du programme, ceci est normal car il est généré aléatoirement.

Question 1 :Ecrire sur papier le pseudo code associé au programme répondant aux exigences formulées au 2.1.

Question 2 : Réalisez maintenant ce programme sous Eclipse et testez le.

Question 3 : Mesurez le débit atteint (c'est à dire le nombre d'octets transférés par secondes) en ajoutant des traces dans votre programme.

Question 4 : Mesurez la consommation CPU de votre programme (avec l'outil top). Qu'en pensez vous ?

Question 5 : Corrigez votre programme pour obtenir une consommation de CPU moindre.
Indication : Il faut mettre une pause dans votre programme pour que votre programme ne fasse

pas en permanence des appels à la fonction sendSpiData. Il est en effet inutile d'appeler en boucle cette fonction si elle vient de retourner 0. Vérifiez que l'impact est nul sur le débit, et que la consommation CPU passe bien à 0.

3 Exercice 2 – Réalisation d'une solution optimale à l'aide d'un buffer tournant

3.1 Description de la solution attendue

Vous vous rendez compte que la solution précédente offre de faibles performances. En effet, à chaque lecture sur la carte SD, vous obtenez environ entre 10 et 30 octets, donc vous envoyez systématiquement des petites trames sur le lien SPI.

Il faut pouvoir envoyer de plus longues trames pour optimiser les performances. Pour cela, il faut mettre en place un **buffer tournant** , qui vous permettra de stocker les données avant de les envoyer.

Le fonctionnement général du programme sera le suivant :

```
création d'un buffer de 2 Ko
création de deux indexs :
     un index avec premier caractère valide dans le buffer
     un index avec le dernier caractère valide
tant que  la fin du fichier de la SDCard n'est pas atteinte
     lire x octets depuis la carte SD vers le buffer
     calcul de y =  le nombre de caractères valides dans le buffer
     si y>=128
          envoyer y octets ou moins sur le lien serie
fin tant que
```

Pour envoyer les **y ou moins octets** sur le lien SPI, le principe est le suivant :

```
vous faites un appel à la fonction sendSpiData()
la méthode retourne « z octets envoyés »
vous mettez à jour les indexs  premier et dernier caractère valide
fin de la méthode
```

3.2 Le travail à réaliser

Question 1 :Ecrire sur papier le pseudo code associé au programme répondant aux exigences formulées au 3.1.

Question 2 : Réalisez maintenant ce programme sous Eclipse et testez le.

Question 3 : Mesurez le débit atteint (c'est à dire le nombre d'octets transférés par secondes) en ajoutant des traces dans votre programme. Comparez avec le débit obtenu à l'exercice 1.

Question 4 : Mesurez la consommation CPU de votre programme (avec l'outil top).