Pale INF422 – Composants d'un système informatique
INF422 Final Exam – Components of a Computing System

2009–2010

- L'examen dure 3 heures.
  The exam lasts 3 hours.
- Tous documents autorisés.
  All documents are authorized.
- Il est impératif de commenter les programmes et de justifier vos réponses.
  Programs must be commented and every answer must be justified.

Sujet en français

On étudie la conception de la composante informatique d'un système d'affichage électronique du type de celui présenté sur la Figure 1.

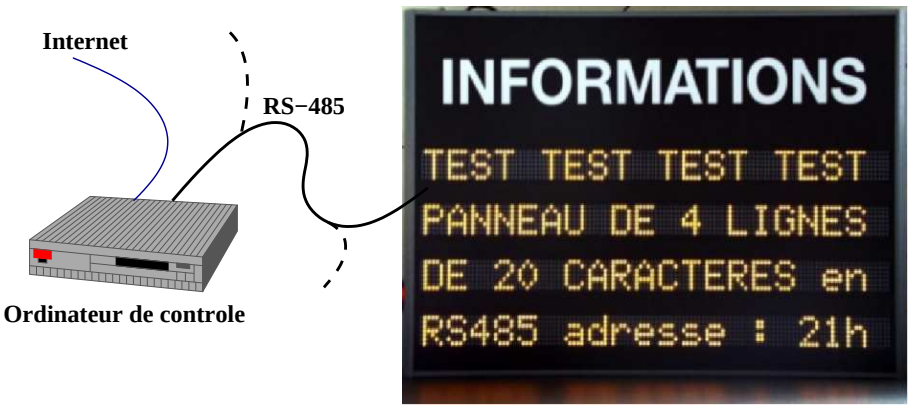

FIG. 1 – Afficheur électronique

Cet afficheur dispose de 4 lignes de 20 caractères. Il est piloté depuis un ordinateur de contrôle embarqué dans un boitier relié au réseau Internet. L'interface entre l'ordinateur de contrôle et l'afficheur utilise une liaison série à la norme RS-485, que nous étudierons ultérieurement. Cette norme autorise la connection de plusieurs dispositifs électroniques sur la même liaison : dans notre exemple, il s'agit d'un ensemble panneaux d'affichage (par exemple dans une gare). L'ordinateur de contrôle qui pilote tous ces afficheurs est basé sur un processeur ARM et utilise le noyau Linux.

1 Représentation des caractères

Le codage des caractères le plus fréquent à longtemps été le code ASCII, adapté aux caractères les plus courants de la lange anglaise. Les caractères ASCII sont représentés sur les 7 bits de poids faible d'un octet — donc leur valeur est comprise entre 0 et 127.

La norme UTF-8, beaucoup plus récente, est la plus couramment utilisée pour représenter les caractères internationaux (accents, symboles spéciaux, autres alphabets, caractères non alphabétiques). Les caractères ASCII

sont codés de la mème manière en UTF-8, afin de garantir la compatibilité avec les programmes plus anciens. Les *autres* caractères sont codés sur *plusieurs* octets, 4 au maximum.

Nous allons étudier les principes de codage mise en oeuvre dans la norme UTF-8. Les caractères non-ASCII sont partitionnés en 3 classes : les plus fréquents (par exemple les lettres accentuées) représentés sur 2 octets, les idéogrammes courants sur 3 octets, et les symboles rares sur 4 octets.

Question 1.1

Proposez un schéma de codage pour les caractères non-ASCII sur 2 à 4 octets respectant ce partitionnement. Certains bits seront utilisés pour identifier la classe de caractères (2, 3 ou 4 octets) et d'autres pour numéroter les caractères eux mêmes ; pour chaque classe, indiquez le nombre de bits disponibles pour la numérotation des caractères.

Justifiez informellement l'absence d'ambiguïté (l'unicité) du principe codage proposé.

Question 1.2

Pour fiabiliser la lecture d'un texte UTF-8, ses concepteurs ont eu l'idée de renforcer les propriétés du codage en le rendant *auto-synchronisant* : la lecture de n'importe quel octet d'un texte permet immédiatement de savoir si celui-ci est le premier octet codant un caractère ou non. Ainsi, toute position dans une chaîne de caractères UTF-8 définit une sous-chaîne de caractères (en sautant éventuellement les 1, 2 ou 3 premiers octets). Modifiez votre principe de codage pour le rendre auto-synchronisant.

Question 1.3

L'ordinateur de contrôle range les octets composant les nombres entiers selon l'ordre *little-endian* (convention pour Linux sur processeur ARM) alors que le circuit spécialisé de l'afficheur utilise le codage *big-endian*. Faut il prévoir une conversion des codes de caractères multi-octets, et si oui, comment ?

2 Liaison série

La liaison série RS-485 est une couche physique pour communiquer entre dispositifs distants. Elle est très peu couteuse, ne nécessitant que 2 fils en mode half duplex — pas de communications à double-sens simultanées. Le signal est défini par la différence de potentiel entre les deux fils. Elle permet de connecter jusqu'à 32 dispositifs électroniques sur des distances importantes.

Dans notre installation, tous les dispositifs ont été configurés pour communiquer des octets (bit par bit) à un rythme compatible avec les caractéristiques électriques de la liaison série (quelques kilo-octets par seconde). La norme RS-485 ne définit pas de protocole particulier pour l'arbitrage entre les dispositifs électroniques désirant transmettre un signal. Nous allons donc définir notre propre protocole, implémenté sur tous les dispositifs connectés.

Notre protocole utilise un "jeton" unique circulant entre les dispositifs. Ce jeton pourra être implémenté par une variable booléenne sur chaque dispositif : il indique quel dispositif est responsable de l'émission de données sur la liaison série. Le dispositif responsable — ordinateur ou afficheur — émet en permanence. S'il n'a rien à émettre de particulier, il émet l'octet 0. L'ordinateur possède le jeton d'émission au démarrage.

Chaque afficheur dispose d'un identifiant unique entre 1 et 30 (au maximum 30 afficheurs). L'ordinateur est associé à l'identifiant 0.

Question 2.1

Pour indiquer qu'il va envoyer un message, le dispositif responsable de l'émission commence par émettre la série d'octets correspondant à la chaîne de caractères `BEGIN`, codée en UTF-8 (équivalent à ASCII pour ces caractères). Il émet ensuite une séquence correspondant au message lui même, puis termine par la chaîne `END`. Proposez un format de message permettant de réaliser les commandes suivantes :

- transférer le jeton d'émission à un autre dispositif — ordinateur ou afficheur ;
- envoyer une suite de n octets à un autre dispositif.

Question 2.2

Étendez le format précédent pour inclure une commande permettant au dispositif responsable k de tester la connexion avec un (autre) dispositif k' et le bon fonctionnement de ce dispositif.

Indiquez la liste des messages circulant sur la liaison série au cours d'un tel test.

Question 2.3

Que se passe-t-il si le dispositif responsable tombe en panne ? Distinguez le cas où la panne survient lorsqu'il est entrain d'envoyer une commande du cas où il est simplement entrain d'émettre 0 en continu.

Question 2.4

En supposant que l'ordinateur de contrôle ne tombe pas en panne (sinon il n'y a plus rien à faire...), modifiez le protocole pour garantir les communications avec les afficheurs restant en fonctionnement. Vous pourrez utiliser une constante de temps prédéterminée (*timeout*) pour détecter une panne.

On supposera pour simplifier que les afficheurs ne tombent pas en panne pas la suite.

3 Pilote pour la liaison série

Au niveau de l'ordinateur de contrôle, on souhaite abstraire cette liaison série et le protocole de communication associé en définissant un pilote périphérique dédié. Ce pilote implémente les fonctionnalités d'un périphérique de type caractère, et en particulier les fonctions suivantes exécutées par le noyau lors des appels système `read()` et `write()` :

- `rs485read(`*`buf, n`*`)` : lit au plus *n* octets depuis la liaison série et les range dans le buffer *buf*, puis renvoie le nombre d'octets effectivement lus ;
- `rs485write(`*`buf, n, id`*`)` : écrit au plus *n* octets sur la liaison série depuis le buffer *buf*, à destination du dispositif *id*, puis renvoie le nombre d'octets effectivement écrits.

Pour lire ou écrire un octet sur la liaison série, ces deux fonctions lisent ou écrivent cet octet à une adresse mémoire déterminée lors du démarrage du périphérique (*memory-mapped I/O*). On simule ce comportement dans un programme Java (une aberration pour du code s'exécutant en mode privilégié dans le noyau, mais nécessaire dans le contexte du cours INF422) : on suppose que tout lecture ou écriture du champ statique `rs485link` correspond respectivement à la récupération ou à l'émission d'un octet sur la liaison série.

La lecture est "destructrice" : à chaque lecture un octet est retiré du buffer de lecture du contrôleur de la liaison série (UART). De même, chaque écriture ajoute un octet dans le buffer d'écriture du contrôleur.

Question 3.1

Que manque-t-il pour pouvoir écrire un programme effectuant des communications fiables sur la liaison série ? Indication : les buffers du contrôleur de la liaison série sont bornés, et il faut respecter très précisément un certain taux d'octets par seconde à transmettre pour assurer une communication fiable.

On ignorera cette difficulté par la suite, et on supposera que l'on peut lire ou écrire aussi vite que possible dans le champ statique `rs485link` pour traiter les octets les uns à la suite des autres.

Question 3.2

La lecture est bloquante : l'appel à la fonction `rs485read()` ne retourne que lorsque un message est parvenu. Écrivez un programme Java implémentant cette fonction. Il attend la réception d'un message destiné à l'ordinateur de contrôle, puis stocke les *n* premiers octets de son contenu (ou l'intégralité s'il est plus court) dans le tableau d'octets *buf* et retourne ne nombre d'octets effectivement lus.

Question 3.3

La fonction d'écriture teste que l'ordinateur est bien le dispositif responsable de l'émission sur la liaison série, et si ce n'est pas le cas attend d'obtenir le jeton (ne pas oublier de traiter la réception du message correspondant). Ensuite elle écrit les *n* premiers octets de du tableau *buf* sur la liaison série. Pour simplifier, on suppose qu'il n'y a pas de panne ou d'erreur lors de la transmission, donc la fonction retourne toujours *n*. Implémentez cette fonction en Java.

Question 3.4

Commentez les qualités de vos programmes, en termes de réactivité du noyau du système (qui implémente cette fonction) et en termes de dépense énergétique de l'ordinateur de contrôle. Sans modifier le code lui même, proposez des pistes pour l'améliorer, en utilisant éventuellement un support matériel additionnel.

4 Pilote pour la gestion des afficheurs

On définit à présent le format des commandes reconnues par les afficheurs. Chaque afficheur comporte 4 lignes de 20 caractères.

- La commande `CLEAR` est de taille $n = 1$, codée par un unique octet nul. Elle efface entièrement l'afficheur.
- La commande `MOVE` est de taille $n = 2$, codée par l'octet 1 suivi de la position du caractère sur lequel positionner le curseur. La position 0 correspond au premier caractère en haut à gauche, la position 79 au dernier caractère en bas à droite.

– La commande `PRINT` est composée de l'octet 2 suivi de $n-1$ octets codant une chaîne de caractères UTF-8. Elle affiche cette chaîne (en passant à la ligne si nécessaire) à partir de la position courante, sans effacer les caractères précédant ou suivant la chaîne ainsi insérée.

Question 4.1

Les échanges de messages sont toujours initiés par l'ordinateur de contrôle. Les afficheurs sont amenés à émettre des messages en réaction à une commande, notamment pour indiquer la fin de traitement d'une opération d'affichage. Définissez un format de message de fin de traitement unique pour les trois commandes précédentes.

Question 4.2

Programmez l'affichage du message `Troisième !` sur la troisième ligne de l'afficheur numéro 17, à l'aide des fonctions `rs485read()` et `rs485write()`.

Question 4.3

Indiquez la liste d'octets circulant sur la liaison série lors de l'affichage du message. Vous représenterez les caractères ASCII par leur lettre et le caractère non-ASCII è par un code de votre choix reprenant la représentation définie au premier exercice.

Question 4.4

On suppose que l'on a implémenté un pilote de périphérique de type caractère implémentant implémentant les trois opérations d'affichage `CLEAR`, `MOVE` et `PRINT`. Proposez une affectation raisonnable d'un appel système à associer à chacune de ces opérations, parmi `read()`, `write()` et `ioctl()`.

Question 4.5

Par l'intermédiaire de ce pilote, chaque afficheur est désigné par un fichier spécial de périphérique de type caractère `/dev/display`*`ID`* où *`ID`* est l'identifiant de l'afficheur (de 1 à 30). Pourquoi ne pas avoir créé un unique fichier spécial de périphérique pour tous les afficheurs ?

Que faut il encore spécifier pour que le pilote de gestion des afficheurs interprête les entrées-sorties (les appels systèmes) sur un de ces fichiers spéciaux comme des commandes à l'afficheur correspondant ?

Question 4.6

On suppose que la commande UNIX `ls -l /dev/cdrom` exécutée sur l'ordinateur de contrôle affiche :

```
brw-rw---- 1 root cdrom 11, 0 2009-11-06 09:04 /dev/cdrom
```

Proposez un affichage vraisemblable pour l'exécution de la commande `ls -l /dev/display17` en expliquant ce que veut dire cet affichage et pouquoi vous avez choisi ces méta-données pour le fichier spécial de périphérique `/dev/display17`.

5 Application

On suppose que toutes les applications Java sont exécutés avec les droits d'un simple utilisateur.

Question 5.1

On souhaite restreindre l'accès aux afficheurs à un certain groupe d'utilisateurs (dont fait partie le simple utilisateur exécutant les applications). En utilisant notamment les fonctionnalités du système de fichiers, expliquez comment mettre en oeuvre cette restriction.

Question 5.2

Écrivez un script shell (executé par le root) qui crée les 30 fichiers spéciaux de périphérique associés aux afficheurs avec les méta-données adéquates.

Question 5.3

Écrivez un programme Java faisant défiler une chaîne de caractères sur la première ligne de *chaque* les afficheurs (un *scrolling* en boucle). Vous utiliserez un thread et une chaine différente par afficheur (si vous manquez d'inspiration, le numéro du thread conviendra).