H4202 – PLANCHE Benjamin & PHAN Nicolas

TP Système d'Exploitation Avancée

Conception d'un pilote

Document de spécification

Contenu

Introduction

Ce TP, qui consiste en la conception d'un pilote de périphérique, permet de comprendre l'utilité de l'**Input/Output System**, ou **IOS**. Pour des raisons pédagogiques, ce projet est réalisé sous VxWorks. En effet, cette couche d'abstraction qu'est l'IOS est bien plus indépendante et simple à comprendre que dans des systèmes d'exploitation comme Linux ou Windows.

La programmation de ce pilote se fera en C dans un contexte multitâche.

Le cahier des charges de ce TP exige la réalisation du pilote, mais aussi la simulation des périphériques associés à notre pilote correspond, et de tâches applicatives les consultant.

Composition

Dans ce TP, les périphériques, considérés fictifs, devront simuler le fonctionnement de capteurs ayant pour vocation de détecter certains événements et le signaler au système. Cette simulation nous permet de faire abstraction des aspects de bas de niveau de l'interface matérielle avec les périphériques, car ce n'est pas l'objectif principal de l'exercice.

Plus spécifiquement nous avons décidé de prendre l'exemple d'un lecteur de badge situé devant une porte à accès restreint. Ce lecteur doit permettre à un utilisateur de s'authentifier à l'aide d'un badge, de lui donner accès à la porte si ses autorisations lui permettent, et enfin d'envoyer cet événement sous forme de message à notre système.

Pour rester dans le cadre du TP, nous ne traiterons que la partie d'envoi de messages au système.

Le pilote que nous développerons devra donc pouvoir gérer un ensemble de capteurs, émettant ponctuellement des messages, consistant en un numéro de badge. Les tâches applicatives du système devront pouvoir, une fois le pilote en place et les périphériques installés, consulter à leur souhait ces messages, préalablement horodatés avec précision. Le tout devra fonctionner dans un contexte multitâche sans conflit.

Notre travail consistera à développer :

- **Une interface avec le matériel** : couche basse du pilote, elle assurera la réception des messages émis par les périphériques et leur sauvegarde.
- **Une interface avec l'IOS et les applications** : couche supérieure qui contiendra les fonctions principales de l'IOS à implémenter (open, read, close, …). Ce sont ces fonctions qui seront appelées par les applications au travers de l'IOS.
- **Une interface de service** : elle est destinée à l'administrateur du système. Il pourra ainsi gérer le pilote et les périphériques par rapport au système (ajout d'un périphérique, installation du driver, …).

Ces trois interfaces permettent de comprendre la place de l'IOS au sein d'un système d'exploitation. Nous détaillerons les explications au fil des spécifications.

Interface avec le matériel

Ce pilote est dédié à des périphériques de type capteurs, employés ici à la surveillance des entrées et sorties par des porte dotées de lecteurs de badge.

Chaque badge est identifié par un numéro à deux chiffres.

A chaque passage, ce numéro est relevé par le capteur, qui le signe et l'enregistre temporairement dans un registre matériel commun, avant de prévenir le système de l'émission. Cet appel se fait à l'aide d'une interruption matérielle.

Notre pilote possède donc un handler sur cette IT, une procédure ISR bloquante et non-interruptible chargée de récupérer le message dans le registre, l'horodater à 5 millisecondes près, et le transmettre aux structures de données adaptées dans le pilote.

Remarque :

Dans la simulation que nous devons effectuer, les périphériques étant fictifs, le registre matériel sera représenté par une variable globale, et l'appel de l'ISR se fera « manuellement » (i.e comme un appel normal de procédure) par les tâches simulant les périphériques.

Les périphériques sont donc en lecture seule, et la communication n'est initiée que côté capteur.

Plusieurs de ces périphériques peuvent être pris en charge par le pilote. Celui-ci ne fait pas la différence entre un périphérique muet et un périphérique non-présent.

Enfin, nous avons souhaité qu'une relative sauvegarde des passages soit offerte au niveau même du pilote. Nous nous engageons donc à conserver plusieurs messages par capteur, afin de pouvoir connaître les allers et venues des utilisateurs sur une porte à accès restreint. Afin de ne pas ralentir les consultations des données reçues, le nombre de message maximal (stocké dans le pilote) par capteur sera initialement défini à dix messages (constante définie au sein du programme). Une fois ce nombre maximum atteint pour un périphérique, les plus anciens messages seront supprimés à chaque nouvel ajout.

Remarque :

L'implémentation d'une fonction *ioctl* d'édition de cette valeur est à l'étude. En effet, de par la structure des données employée, une telle fonctionnalité ne poserait aucun problème de mise en place.

Interface avec l'IOS et les applications

L'IOS permet d'offrir aux applications une interface uniforme pour communiquer avec les périphériques en donnant accès aux ceux-ci sous forme de fichier. Tout cela se fait au travers d'une API standardisée, au travers de sept fonctions sous VxWorks : *open, close, read, write, creat, remove et ioctl.*

Pour notre pilote nous n'utiliserons pas les fonctions *write*, *creat*, *remove* et *ioctl*. En effet l'utilisateur ne peut consulter que les messages délivrés par les périphériques, l'écriture de messages (*read*) par celui-ci n'a pas de sens dans notre contexte (périphérique en lecture seule). Les fonctions *creat* et *remove* n'en ont pas non plus, car ce n'est pas à l'application d'ajouter ou de retirer un lecteur de badge.

Enfin la primitive *ioctl*, qui permet de réaliser des opérations spécifiques que l'on ne peut réaliser par un appel système classique, n'a pour l'instant aucune raison d'être, faute de fonctionnalités autres à offrir.

La possibilité est donc donnée aux tâches applicatives d'ouvrir un fichier sur n'importe quel périphérique installé, de consulter le ou les X derniers messages émis par

celui-ci (X étant inférieur ou égal à 10, comme expliqué auparavant), et de refermer ce fichier.

Chaque périphérique possède deux identifiants :

- Le premier consiste en son **numéro matériel** (2 chiffres). Il n'est connu que de l'administrateur et employé dans les couches basses du pilote pour reconnaître la signature des messages.
- Le second consiste en un **nom** (chaîne de caractères) donné par l'administrateur à l'installation du périphérique.

Nous considérons que les tâches applicatives ne connaissent que ce nom. L'IOS est ainsi en charge de faire l'association, transmettant au pilote les données du périphérique appelé par tel nom.

Chaque fonction mise à la disposition des applications doit pouvoir couvrir tous les cas d'utilisations possibles, renvoyant les codes de retour adaptés aux situations rencontrées.

Enfin, la structure des messages remis est la suivante :

hh:mm:ss-xxx|NN\n

avec *hh* nombre d'heures, *mm* de minutes, *ss* de secondes, *mm* de millisecondes, et *NN* numéro du badge.

Ce temps retourné correspond au temps écoulé depuis l'installation du pilote.

Remarque :

A noter qu'avec la configuration actuelle, lors d'utilisation prolongée de plus de 99h, le compteur des heures reviendra à zéro.

Interface de service

Cette interface offerte par le pilote est destinée à l'administrateur du système. Dans notre cas nous serons amenés à utiliser les fonctions de cette interface pour pouvoir mener à bien nos simulations.

Cette API nous proposera 4 fonctions qui permettent :

- **L'installation du pilote** : s'occupe des diverses initialisations nécessaires au bon fonctionnement du pilote pour pouvoir gérer un ensemble de capteur. Cette fonction n'entraînera aucune conséquence si elle est appelée plus d'une fois. Seule la première installation compte.

- **La désinstallation du pilote** : au contraire de la précédente fonction, celle-ci s'occupera de supprimer le pilote et son environnement utilisé pour la gestion des périphériques.

- **L'ajout d'un périphérique** : permet au pilote d'associer le périphérique à un fichier afin de pouvoir exploiter les informations émises par le périphérique. Le pilote doit être au préalable installé. Un périphérique peut être ajouté deux fois sans conséquences, seul le premier ajout sera pris en compte.

- **Le retrait d'un périphérique** : permet au pilote de retirer du système le périphérique correspondant. Il ne se passe rien si le périphérique n'existe pas au niveau du système (pas d'association à un fichier).

L'administrateur est supposé fiable. Il est le seul individu à connaître les numéros matériel des capteurs, et est responsable de leur nommage, afin de permettre leur visibilité par les tâches applicatives.

Spécification détaillée du pilote

Fonctions réservées à l'administrateur système

- Installation d'un pilote : **DrvInstall**
  DrvInstall() renvoie entier

Paramètres d'entrée	----------
Paramètre de sortie	Entier : Numéro du driver dans la table de driver.
Retour d'erreur	Retour = -60 : Le driver a déjà été installé. Retour = -1 : Plus de place disponible dans la table de driver.

L'administrateur doit lancer cette fonction afin d'installer le pilote.

- Désinstallation d'un pilote : **DrvRemove**
  DrvRemove () renvoie entier

Paramètres d'entrée	----------
Paramètre de sortie	Entier : Numéro du driver dans la table de driver.
Retour d'erreur	Retour = -1 : Le driver a encore des fichiers ouverts. Retour = -20 : Le driver n'est pas installé.
Contrat	Tous les périph. gérés par ce pilote doivent être retirés au préalable.

A la fin de l'utilisation de l'application et après avoir fermé tous les fichiers (à travers l'appel close), l'administrateur doit appeler cette fonction pour désinstaller correctement le pilote et toutes les structures qui ont été créés. Si le pilote n'est pas installé il ne se passera rien.

- Ajout d'un périphérique : **DevAdd**
  DevAdd(char* nomPeriph, char[2] numeroCapteur, entier maxMsg) renvoie entier

Paramètres d'entrée	Char* : Nom du périphérique a ajouter. Char[2] : Numéro du capteur de badge. Entier : Nombre maximal de msg sauvegardables pour ce périph.
Paramètre de sortie	Entier : Nombre de périphériques gérés par le pilote.
Retour d'erreur	Retour = -1 : Le driver n'a pas été installé. Retour = -50 : Le périphérique a déjà été ajouté.
Contrat	Nous ne vérifions pas si le périphérique existe réellement.

Permet à l'administrateur d'ajouter un périphérique dans la gestion du pilote.

- Retrait d'un périphérique : **DevDel**

DevDel (char* nomPeriph) renvoie entier

Paramètres d'entrée	Char* : Nom du périphérique à supprimer.
Paramètre de sortie	Retour = 0 : Tout s'est bien passé.
Retour d'erreur	Retour = -10 : Le périphérique n'a pas été trouvé dans la table de correspondance.

Permet à l'administrateur de supprimer un périphérique dans la gestion du pilote. La suppression d'un périphérique déjà supprimé n'aura aucune conséquence.

Fonction destiné au programmeur d'application

- Ouverture d'un périphérique : **open**

  open (constant char* nomPeriph, entier typeAcces, entier mode) renvoie entier

Paramètres d'entrée	Char * : Nom du périphérique Entier : Type d'accès du périphérique : 0 (en lecture, O_RDONLY) Entier : Mode - Non utilisé, valeur à 0.
Paramètre de sortie	Entier : Descripteur de fichier
Retour d'erreur	Retour = -1 : Erreur dans la recherche du périphérique.

  - Avant de pouvoir utiliser des applications travaillant avec un périphérique (avec par exemple la fonction *read*), il est nécessaire d'ouvrir le fichier correspondant au périphérique avec l'appel système *open*. L'utilisateur appelle cette fonction avec comme paramètres le nom du périphérique, le type d'accès et le mode.
  - Le Système d'Entrée/Sortie cherche alors dans la table de périphérique celui qui a le nom donné en paramètre. Si un périphérique est trouvé l'IOS garde un emplacement disponible dans la table de descripteur de fichier (*TDF*), va chercher dans la table de driver l'adresse de la fonction *monOpen* (que nous allons implémenter) pour l'appeler.
  - Après l'exécution de *monOpen*, et un enregistrement du périphérique dans l'emplacement que nous avons gardé disponible dans la table de descripteur de fichier, la fonction *open* que nous avons appelée nous renverra l'index (descripteur de fichier) de cet emplacement dans la table.
  - Il est très important d'invoquer cet appel système pour pouvoir interagir avec un périphérique, et il est aussi essentiel de fermer le fichier lorsque l'utilisateur a fini ses interactions.

- Consultation des messages : **read**

  read (entier DescripteurFichier, char* buffer, entier longueur) renvoie entier

Paramètres d'entrée	Entier : Descripteur de fichier, valeur renvoyée par open(). Tableau de caractère : buffer Entier longueur: Nombre d'octets que veut récupérer l'utilisateur.
Paramètre de sortie	Entier : Nombre de messages récupérés.
Contrat	• Buffer initialisé. • La longueur doit valoir (X*16+1), X étant le nombre de messages que l'utilisateur veut récupérer. Les 16 octets correspondent à la longueur d'un message, et le dernier octet au caractère de fin de chaîne. • Périphérique ouvert au préalable (par la fonction open()).

Retour d'erreur	• -20 : Périphérique désinstallé entre temps. • -40 : Passage de buffer de taille incorrecte.

- La lecture ne sera pas bloquante, dans le sens où s'il n'y a pas de messages stockés au moment de celle-ci, la fonction indiquera qu'aucun message n'a été récupéré.
- Les messages les plus récents seront considérés plus prioritaires que les plus anciens, ainsi l'utilisateur récupèrera en premier les messages dernièrement arrivés.
- La lecture simultanée de messages d'un périphérique par plusieurs lecteurs est permise, sans risque d'altération des données.
- Pour en savoir plus sur la structure du message renvoyé, voir précédemment.
- La fonction de lecture renverra un entier correspondant au nombre de messages récupérés. Ce nombre pourra aussi correspondre à une erreur lorsque celui-ci sera négatif.
- Si l'utilisateur souhaite lire X messages tel que X soit supérieur au nombre maximal de messages sauvegardables pour un périphérique, ou supérieur au nombre de message émis par celui-ci, la fonction de lecture renverra tous les messages existants encore dans la liste en indiquant le nombre de messages récupérés.

- Fermeture d'un périphérique : **Close**
  close (char* nomPeripherique) renvoie entier.

Paramètres d'entrée	Chaîne de caractère : Nom du périphérique à fermer.
Paramètre de sortie	Entier à 0 : Tout s'est bien passé.

Lorsqu'un périphérique est ouvert avec l'appel *open*, un descripteur de fichier est conservé dans la *TDF* jusqu'à ce que la fonction close soit appelé par l'utilisateur.

A ce moment le descripteur de fichier ne peut plus être utilisé par aucune tâche.