Spécification des fonctions d'un pilote de périphérique de type capteur

Paul Adenot Étienne Brodu

Table des matières

1 Documentation de l'API

open

Synopsis

```
int open(const char* filename, int flags, int perms)
```

Description

Ouvre le capteur désigné par `filename`, et renvoie une descripteur de fichier (*file descriptor*), qui l'identifie au sein du programme. `flags` indique le mode d'ouverture, et doit être fixé à `O_RDONLY`, les capteurs étants en lecture seule. D'autres valeurs, possiblement passées par l'utilisteur, provoque une erreur, et errno est fixé à `EARG`. L'argument `perms` dénote les permission qui seront utilisée sur le fichier. Plusieurs capteurs peuvent être ouvert au sein du même programme. Si un même capteur est ouvert plusieurs fois au sein du même programme, alors plusieurs descripteurs de fichiers seront disponibles pour lire sur un même capteur. Si le fichier précisé dans le premier paramêtre (`filename`) n'existe pas, l'appel échoue, et `open` retourne immédiatement, avec la valeur -1.

Valeur de retour

Si l'appel reussi, un descripteur de fichier (entier positif). Sinon, -1, et `errno` est fixé à l'une des valeurs suivantes :

`EARG :` L'appel a été effectué avec de mauvais arguments, avec une valeur autre que `O_RDONLY` pour `flags`.

`ENEXIST :` Premier argument invalide, le fichier n'existe pas.

`EALREADYOPENED :` Le périphérique est déjà ouvert.

creat

Synopsis

```
int creat(const char *pathname, int mode);
```

Description

Le comportement de cette fonction est similaire à celui de la fonction `open`

Valeur de retour

Les valeurs de retours sont les mêmes que celles de la fonction `open`.

close

Synopsis

```
int close(int fd);
```

Description

Ferme le capteur désigné par le descripteur de fichier `fd`. Celui-ci ne sera plus utilisable dans le programme. Si le paramètre `fd` est invalide (i.e. négatif ou ne correspondant pas à un descripteur de fichier valide), `close` retourne -1, et `errno` est positionné à `EARG`.
Si le capteur est en cours d'utilisation, l'appel échoue en renvoyant -1, et `errno` est positionné à `EBUSY`.

Valeur de retour

Si l'appel réussi, 0 est renvoyé, -1 sinon, et `errno` est positionné aux valeurs suivantes :

`EARG :` L'appel a été effectué avec de mauvais arguments, le `fd` spécifié est invalide.

`EBUSY :` Le capteur est en cours d'utilisation.

remove

Synopsis

```
int remove(const char *pathname);
```

Description

Ferme le capteur désigné par `pathname`. Il ne sera plus utilisable au sein du programme. Si `pathname` est invalide (le fichier n'existe pas, ou n'est pas ouvert au sein du programme), alors l'appel échoue en renvoyant -1, et `errno` est positionné) `ENEXIST`. Si le capteur est en cours d'utilisation, l'appel échoue en renvoyant -1, et `errno` est positionné à `EBUSY`.

Valeur de retour

Si l'appel réussi, 0 est renvoyé, -1 sinon, `errno` est positionné à l'une des valeurs suivantes :

`EARG :` Le fichier précisé n'existe pas.

`EBUSY :` Le capteur est en cours d'utilisation.

read

Synopsis

`int read (int fd, char *buffer, size_t maxbytes);`

Description

Lit un message d'un capteur désigné par `fd`, et le place dans l'adresse pointé par `buffer`. Si un message est disponible, alors il est placé dans à l'adresse `buffer`, mais n'est pas *consommé*, la lecture étant non destructive. Un message lu sur un capteur est du type `capt_msg`, qui est défini de la manière suivante :

```
1  struct capt_msg
2  {
3          unsigned ID;
4          timestamp date;
5          char msg[TAILLE_MAX];
6  };
```

L'entier `ID` est commun à tous les capteurs, et est incrémenté à chaque message. Lors de l'initialisation du driver, il est fixé à zéro. En cas de dépassement de capacité, la valeur de `ID` redeviendra 0, et continuera normalement.
La date `date` est un entier, qui correspond au nombre de périodes de 20ms qui se sont écoulés depuis le démarrage du système. Il sert donc à ordonner temporellement les message, et non à déterminer leur date d'arrivée.

Valeur de retour

Un entier positif, correspondant à la taille lue (`TAILLE_MSG`) est renvoyée. En cas d'erreur, -1 est renvoyé, et `errno` est positionné aux valeurs suivantes :

`EARG :` L'appel a été effectué avec de mauvais arguments, le `fd` spécifié est invalide.

`ENOAVAIL :` Aucun message n'est disponible.

write

Synopsis

`int write (int fd, char *buffer, size_t maxbytes);`

Description

Appel non supporté, les capteurs sont en lecture seule. Pour faire une opération sur un capteur, utiliser `ioctl`.

Valeur de retour

N.A.

ioctl

Synopsis

`int ioctl(int fd, int request, int value);`

Description

Configuration du pilote. Si `request` est inférieur ou égal à 255, `ioctl` comprend qu'il s'agit du numéro du périphérique à remplacer. L'adresse du nouveau périphérique branché doit être passé en `value`. Cette partie sera spécifiée ultérieurement. Les valeurs de `request` plus élevées sont réservées, et pourront correspondre à d'autres fonctionnalités, dans le futur.

Valeur de retour

`ioctl` renvoie 0 en cas de succès, -1 sinon, et `errno` est alors positionné à l'une des valeurs suivantes :

`EARG :` L'appel a été effectué avec de mauvais arguments.

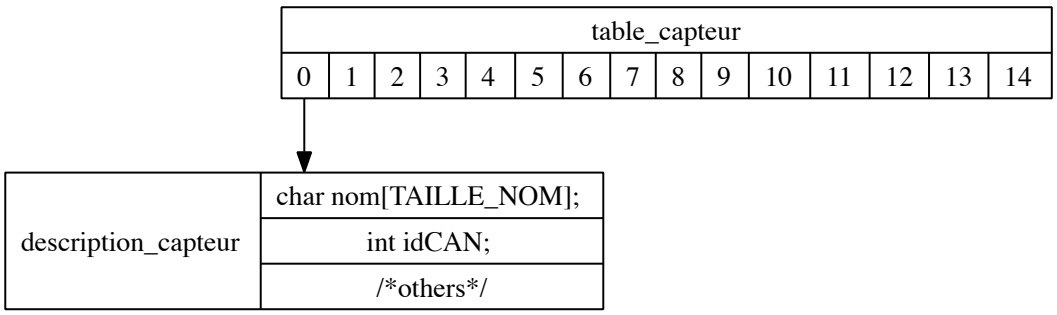
`EBUSY :` Le capteur est occupé.

2 Structure des données

2.1 table_capteur

Ce tableau contient 15 pointeurs vers des structures decrivant chaque capteur. L'index du tableau servant d'identifiant logique au sein du driver. Structure décrivant un capteur :

```
1  struct description_capteur
2  {
3          char nom[TAILLE_NOM];
4          int idCAN;
5          /* others */
6  };
```

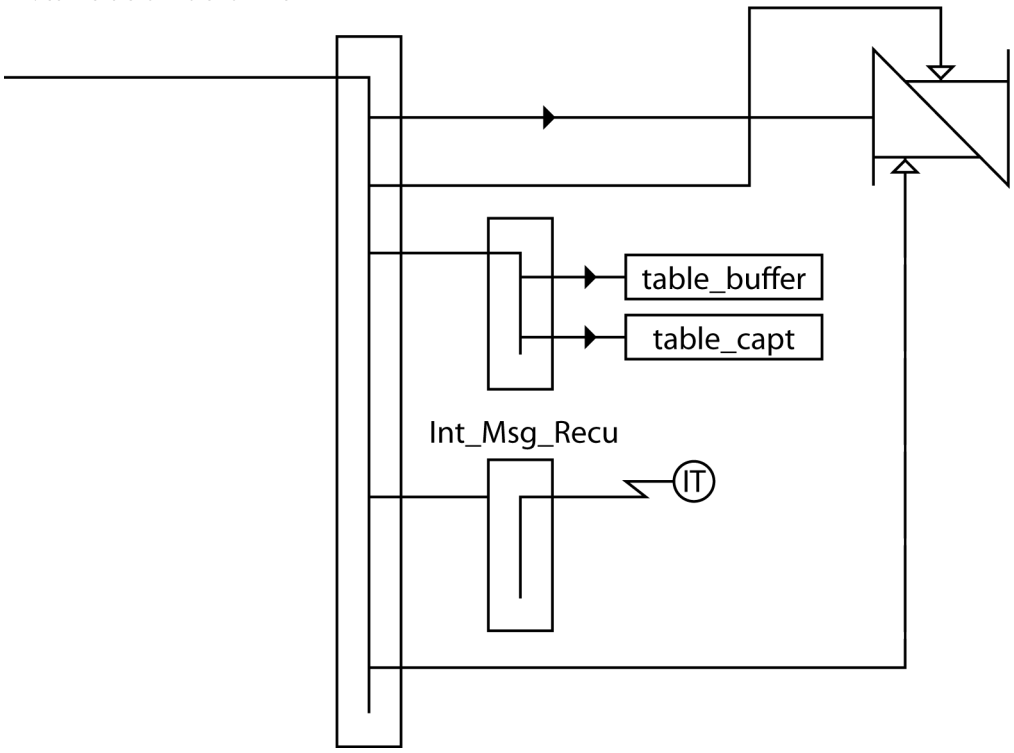


2.2 table_buffer

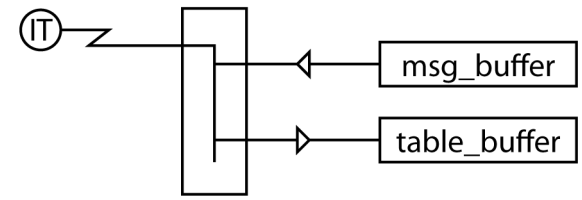
Ce tableau contient 15 pointeurs vers le dernier message du capteur dont l'index du tableau est l'identifiant logique.

3 Conception graphique

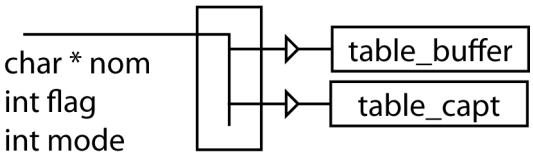
Instanciation du driver



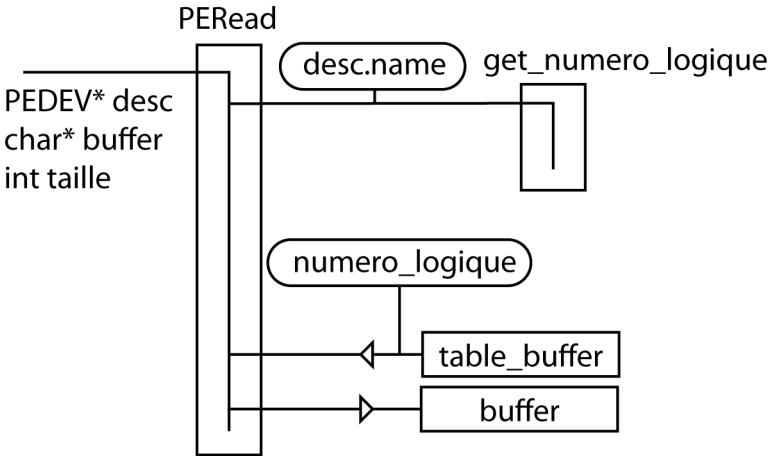
Int_Msg_Recu

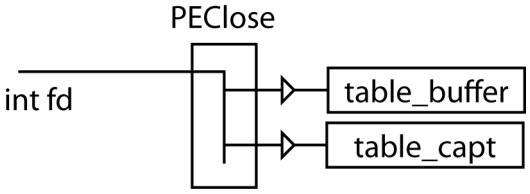


PEOpen



PERead





4 Plan de test

Test 1 – Installation d'un driver

Description

Installer le driver alors qu'il n'est pas installé

Resultat attendu

La valeur de retour doit être positive, et correspond au numéro du driver. Il doit être possible de le retrouver en utilisant la fonction `iosDrvShow`.

Test 2 – Installation d'un driver déjà installé

Description

Installer le driver alors qu'il est déjà installé : appeler deux fois `iosDrvInstall`.

Resultat attendu

La première installation doit bien se passer (valeur de retour positive). Le second appel de `iosDrvInstall` doit renvoyer `ERROR`.

Test 3 – Retrait d'un driver

Description

Utilisation de la fonction `iosDrvRemove`, alors que le pilote est installé sur le système, et qu'il n'est pas utilisé.

Resultat attendu

La valeur de retour doit être égale à `OK`.

Test 4 – Ajout d'un périphérique

Description

Utilisation de la fonction `iosDevAdd`, une seule fois, avec des paramètre valides.

Resultat attendu

La valeur de retour doit être `OK`, le périphérique doit être trouvable en utilisant `iosDevFind`, qui ne doit pas renvoyer `NULL`.

Test 5 – Retrait d'un périphérique

Description

Utilisation de la fonction `iosDevDelete`, avec des paramètres valides.

Resultat attendu

Il ne doit plus être possible d'ouvrir le périphérique : un appel à `open` sur ce périphérique doit échouer (il doit retourner `ERROR`), et `errno` doit être positionné à `ENEXIST`.

Test 6 – Ajout d'un périphérique alors que 15 périphériques ont été ajoutés.

Description

Utilisation de la fonction `iosDevAdd`, 16 fois, avec des paramètres valides.

Resultat attendu

Le 16^{e}appel à `iosDevAdd` doit provoquer une erreur, et renvoyer `ERROR`.

Test 7 – Ouverture d'un capteur

Description

Appeler `open` sur un capteur valide (le fichier existe et est accessible en écriture), avec des options valide (`O_RDONLY`), une seule fois.

Resultat attendu

La valeur de retour doit être un entier positif.

Test 8 – Ouverture d'un capteur déjà ouvert

Description

Appeller `open` sur un capteur valide (le fichier existe, et est accessible en lecture), alors qu'il vient d'être ouvert avec succès.

Resultat attendu

`open` doit renvoyer `error`, et `errno` doit être positionné à `EALREADYOPENED`.

Test 9 – Fermeture d'un capteur

Description

Appeler `close` sur un descripteur de fichier valide (qui a été ouvert avec succès précédemment).

Resultat attendu

La valeur de retour doit être égale à `OK`

Test 10 – Lecture d'une valeur dans un capteur

Description

Utiliser `read` sur un capteur ouvert avec succès

Resultat attendu

La valeur de retour doit être un nombre positif, et doit être cohérente par rapport aux paramètre d'appel de `read`.

Test 11 – Utilisation de `read` avec une taille de lecture invalide

Description

Utilisation de l'appel système `read` avec une descripteur de fichier valide, mais avec une taille de lecture négative.

Resultat attendu

Test 12 – Utilisation de `ioctl` avec des paramètres corrects

Description

Utilisation de `ioctl` avec des paramètres corrects : un descripteur de fichier valide, une valeur pour `request` inférieur où égale à 255, et une adresse valide pour le nouveau périphérique en 3^{e}argument.

Resultat attendu

La valeur de retour doit être égale à `OK`

Test 13 – Utilisation de `ioctl` avec de mauvais arguments

Description

Utilisation de la fonction `ioctl` avec un second paramètre correspondant à une fonction non-implémenté. Le descripteur de fichier passé en tant que premier paramètre doit être valide, et correspondre à un périphérique géré par le driver.

Resultat attendu

L'appel doit échouer (la valeur de retour doit être égale à `ERROR`), et `errno` doit être positionné à `EARG`.

Test 14 – Utilisation de `write`

Description

Appel de write sur un capteur ouvert avec succès.

Resultat attendu

L'appel doit échouer, et `errno` doit être positionné à `ENOTSUP`.