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

Paul Adenot Étienne Brodu

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.

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 paramêtre `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  };
```

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 nou-

veau 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.

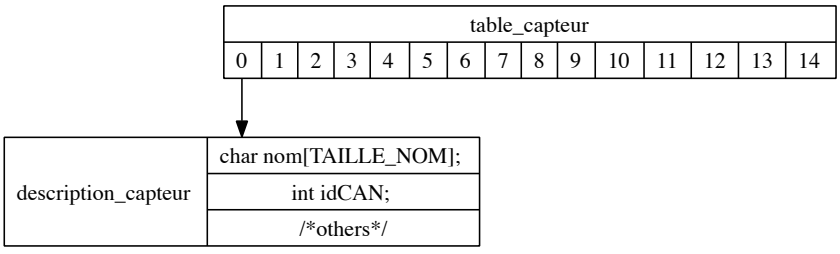
`EBUSY :` Le capteur est occupé.

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  };
```

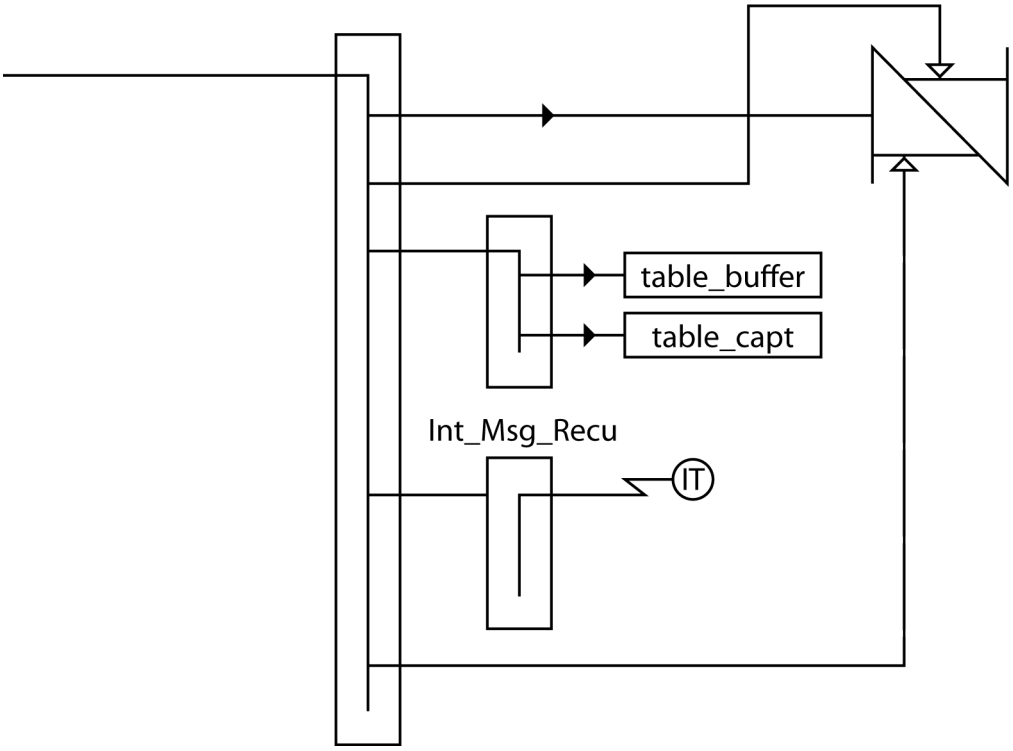


2.2 table_buffer

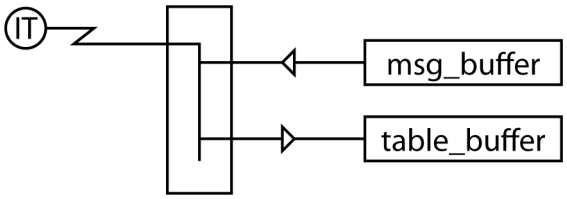
Ce tableau contient 15 pointeurs vers le dernier message du capteur dont l'index du tableau est l'identifiant logique.

3 Conception graphique

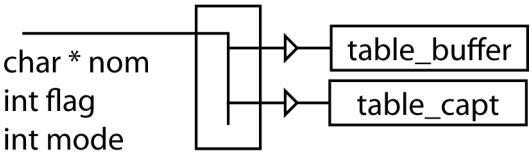
Instanciation du driver

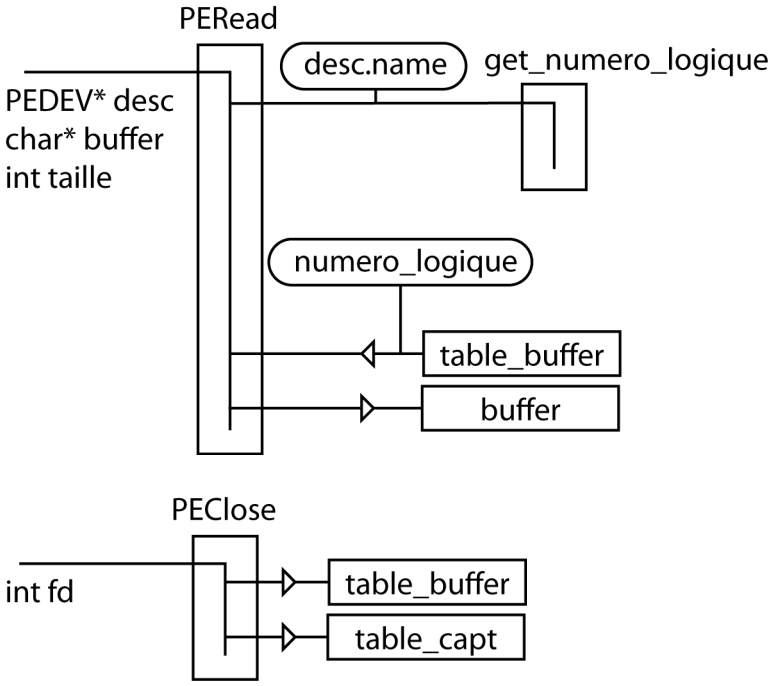


Int_Msg_Recu



PEOpen




PERead
desc.name
get_numero_logique
PEDEV* desc
char* buffer
int taille
numero_logique
table_buffer
buffer
PEClose
int fd
table_buffer
table_capt