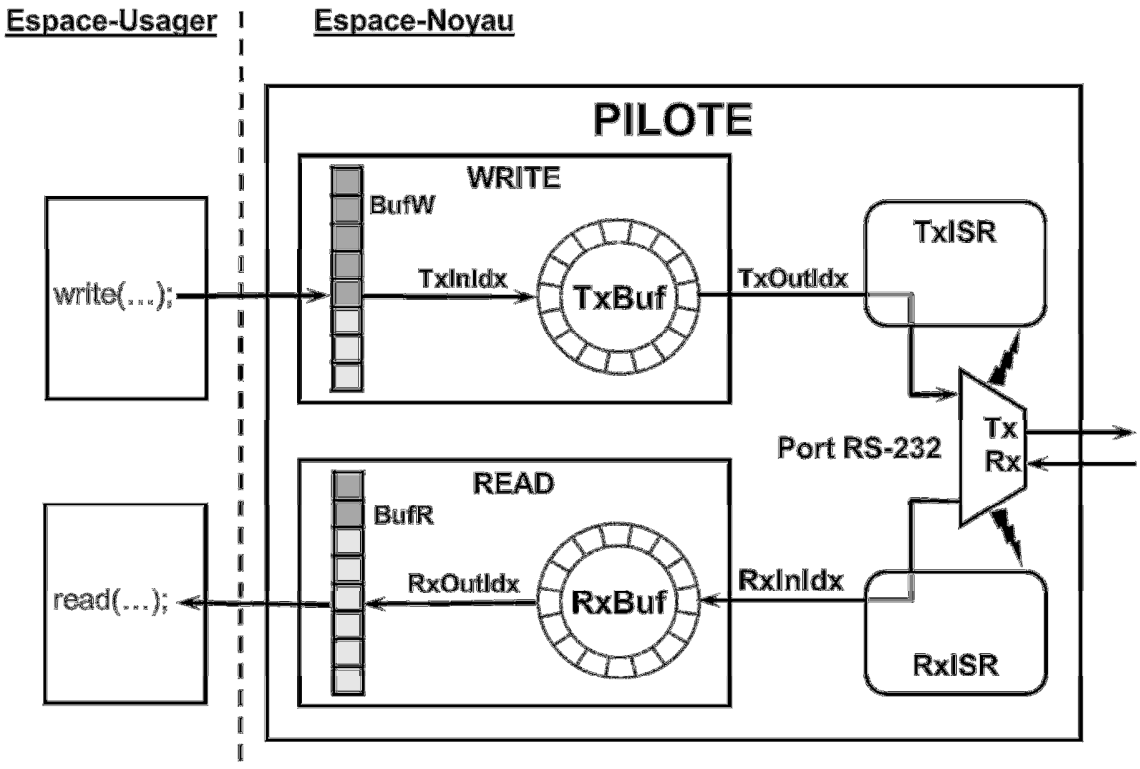
1. Objectif

L'objectif de ce laboratoire est d'initier l'étudiant à la programmation d'un pilote simple de type caractère et de l'amener à explorer et utiliser les différents mécanismes de gestion offert à l'intérieur du noyau de Linux.

2. Énoncé

2.1. Mise en contexte

On désire concevoir un pilote-caractère simple pour un Port Série standard (périphérique). La figure ci-dessous décrit grossièrement le principe de fonctionnement des lectures/écritures (Read/Write) du pilote :



Puisque la transmission (**Tx**) et la réception (**Rx**) du Port Série sont indépendantes, les services d'Écriture (**Write**) et de Lecture (**Read**) du Pilote doivent l'être aussi. Ainsi, ces deux services auront leur tampon individuel : **TxBuf** et **RxBuf**. Ces deux « tampons circulaires » permettent de synchroniser les échanges de données entre le Port Série, en tant que tel, et le Pilote, selon leur cadence d'échange respectives. C'est-à-dire que les échanges (**Tx** ou **Rx**) de données fait par le Port Série sont relativement lents (vitesse de communication du Port Série), mais constants. Tandis que les échanges (**Read** ou **Write**) de données du Pilote sont très rapides, mais par salves.

Les données reçues (**Rx**) par le Port Série sont placées une à la fois dans le tampon **RxBuf** par la routine d'interruption (**RxISR**) et réciproquement pour les données transmises qui sont retirées une à la fois du tampon **TxBuf** par la routine d'interruption (**TxISR**). Il en va de même du côté du Pilote; c'est-à-dire que le Pilote place ou retire les données des tampons **RxBuf** et **TxBuf** une à la fois.

De plus, afin de réduire les échanges de données entre le **Pilote** et l'**Usager**, chacun de ces deux services (**Read**, **Write**) utilisent un petit tampon de **8 bytes** : **BufR** et **BufW**. Ces petits tampons intermédiaires permettent de faire des échanges de données entre le Pilote et l'Usager par paquets de 8 bytes plutôt que de faire des échanges une donnée à la fois.

En ce qui concerne l'utilisation du **Pilote** par un **Usager**, puisque le Port Série ne peut accommoder les communications que d'un seul flux de données à la fois, le **Pilote** doit limiter l'accès à **un seul Usager à la fois**.

De plus, l'**Usager** unique ne peut « ouvrir » le **Pilote** plus d'une fois en **Lecture** ou en **Écriture**. C'est-à-dire que si l'**Usager** a ouvert le **Pilote** en **Lecture**, il ne peut l'ouvrir une deuxième fois en **Lecture** que s'il le ferme avant. Il en va de même pour l'ouverture en **Écriture**. Par contre, si l'**Usager** a ouvert le **Pilote** en **Lecture**, il peut ensuite l'ouvrir en **Écriture** et vice-versa. D'un autre côté, si l'**Usager** ouvre le **Pilote** en **Lecture-Écriture**, alors il ne pourra plus l'ouvrir de nouveau sans l'avoir fermé avant. Dans les cas de réouverture illégale, un code d'erreur approprié sera retourné à l'**Usager**.

Pour finir, les **Lectures**/**Écritures** peuvent être « **bloquantes** » ou « **non-bloquantes** », selon le choix de l'**Usager**. Dans le cas d'**Écriture Bloquante**, le **Pilote** « bloque » l'**Usager** tant qu'il n'a pas pu transférer toutes les données de l'**Usager** dans le tampon circulaire **TxBuf**. Tandis qu'en **Écriture Non-Bloquante**, le **Pilote** transfère les données de l'**Usager** dans le tampon circulaire **TxBuf** selon la place disponible dans ce dernier et informe l'**Usager** du nombre réel de données transférées.

Il en va de même pour la **Lecture Bloquante** et **Non-Bloquante**. C'est-à-dire que le **Pilote** « bloque » l'**Usager** tant qu'il n'a pas pu lui transférer toutes les données que l'**Usager** a demandées. Tandis qu'en **Non-Bloquant**, le **Pilote** transfère à l'**Usager** les données actuellement disponibles dans son tampon circulaire **RxBuf**, jusqu'à concurrence du nombre de données demandées par l'**Usager**, et informe l'**Usager** du nombre de données qui lui ont été réellement transférées.

2.2. Description des fonctions principales du pilote

À la fin de ce laboratoire, les fonctions principales de votre pilote devront minimalement respecter les spécifications suivantes :

Init : - En tout premier lieu, la fonction **INIT** du **Pilote** doit s'attacher au Port Série. C'est-à-dire que le Pilote doit réserver la plage d'adresses ainsi que l'interruption du Port Série. Si elle n'y parvient pas, elle doit obligatoirement retourner un code d'erreur et le Pilote ne pourra pas s'installer.

- La fonction **INIT** doit aussi faire toutes les initialisations requises, telles que les initialisations standards d'un Pilote ainsi que les mécanismes de synchronisation ou de protection d'accès et toutes les variables, drapeaux, allocations dynamiques, etc., dont le Pilote peut avoir besoin.

Exit : - La fonction **EXIT** doit essentiellement faire le chemin inverse de la fonction **INIT**. C'est-à-dire de libérer toutes allocations dynamiques, s'assurer que tous les mécanismes de synchronisation ou de protection d'accès ont été libérés et de se détacher de la plage d'adresses ainsi que de l'interruption du Port Série.

Open : - Vérifie le mode d'ouverture :
O_RDONLY, **O_WRONLY** ou **O_RDWR**

- Le Pilote ne peut être ouvert en « **lecture** » ou en « **écriture** » (**O_WRONLY** ou **O_RDONLY**) qu'une seule fois. Une ouverture en « lecture » **ET** « écriture » (**O_RDWR**) compte pour une ouverture dans chacun des deux modes.

- Toute nouvelle ouverture dans un mode déjà ouvert sera refusée et un code d'erreur (**ENOTTY**) sera retourné au demandeur.

- De plus, l'ouverture du Pilote est à usager unique. C'est-à-dire que dès qu'un usager ouvre le Pilote dans n'importe quel mode, lecture ou écriture ou les deux, le Pilote lui appartient et tous nouvel usager sera refusé avec un code d'erreur (**ENOTTY**).

- Aussi, si le mode d'ouverture **O_RDONLY** ou **O_RDWR** est choisi par l'usager, alors le Port Série doit être placé en mode **Réception** (active l'interruption de réception) afin de commencer à recevoir des données immédiatement.

Release : -Vérifie le mode d'ouverture (**O_RDONLY**, **O_WRONLY**, **O_RDWR**) et élimine ce mode de la liste des modes d'ouverture « actifs ».

- Si tous les modes d'ouverture ont été éliminés, alors le Pilote doit être considéré comme ayant été libéré de l'usager qui l'avait capturé.

- Et si le mode d'ouverture était **O_RDONLY** ou **O_RDWR**, la **Réception** du Port Série doit être interrompue afin d'arrêter de recevoir des données.

Read : - Doit supporter les lectures **bloquantes** et **non-bloquantes**.

- Les données sont retirées une à une du **RxBuf** et placées dans le tampon local **BufR**, avant d'être transmises en bloc à l'usager.

- Ce processus se répète autant de fois que nécessaire, tant qu'il y a des données disponibles et que la requête n'est pas entièrement satisfaite (nombre de données demandées).

- En **mode bloquant** : Le Pilote bloque l'usager jusqu'à ce que toutes les données demandées aient été transmises à l'usager.

- En **mode non-bloquant** : Le Pilote transmet immédiatement à l'usager les données disponibles au moment de la requête. S'il n'y a aucune donnée disponible, le Pilote retourne immédiatement en indiquant que zéro (0) données ont été fournies à l'usager.

Write : - Doit supporter les écritures **bloquantes** et **non-bloquantes**.

- Les données de l'usager sont placées dans le tampon local **BufW** par bloc et transférées une à une dans **TxBuf**.

- Ce processus se répète autant de fois que nécessaire, tant qu'il y a de la place dans le **TxBuf** et que la requête n'est pas entièrement satisfaite (nombre de données demandées).

- Dès que des données de l'usager sont placées dans le **TxBuf**, le Port Série est placé en mode **Transmission** afin de commencer la transmission des données immédiatement.

- En **mode bloquant** : Le Pilote bloque l'usager jusqu'à ce toutes les données demandées ont été placées dans le **TxBuf**.

- En **mode non-bloquant** : Le Pilote transfert immédiatement les données

de l'usager dans le **TxBuf** selon les places disponibles au moment de la requête. S'il n'y a aucune place disponible dans le **TxBuf**, le Pilote indique immédiatement à l'usager que zéro (0) données ont été transférées.

SerialISR : - Il s'agit de la fonction d'interruption liée au Port Série.

- Elle est responsable de gérer l'arrivée et l'envoi des données par le Port Série, lorsque déclenchée pour le signal d'interruption du Port Série.

- La routine vérifie la nature de l'événement (réception d'une donnée, prêt à transmettre) qui a déclenché l'interruption et traite tous les événements qui se sont produit.

- Si l'événement est une Réception, la routine récupère la donnée qui est arrivée, la transfert dans le **RxBuf** et avertit le Pilote (**Read**) qu'une donnée est arrivée.

- Si l'événement est « prêt pour une autre donnée à transmettre », la routine retire une donnée du **TxBuf**, la place dans le Port Série afin qu'elle soit transmise et avertit le Pilote (Write) qu'une place a été libérée dans le **TxBuf**.

- De plus, si après avoir retiré une donnée du **TxBuf**, le **TxBuf** est vide, alors la routine désactive la transmission dans le Port Série (désactive l'interruption de transmission).

La fonction **IOCTL** permet d'envoyer différentes commandes à votre pilote pour soit obtenir ou configurer certains paramètres de celui-ci. Elle devra minimalement supporter les commandes suivantes :

SetBaudRate : - Permet de changer la vitesse de communication du Port Série. La valeur permise se situe dans la plage 50 à 115200 Baud.

SetDataSize : - Permet de changer la taille des données de communication. La valeur permise se situe dans la plage 5 à 8 bits.

SetParity : - Permet de choisir le type de parité qui sera utilisée lors des communications. La valeur permise est :

0 : Pas de parité
1 : Parité impaire
2 : Parité paire

GetBufSize : - Retourne la taille des tampons **RxBuf** et **TxBuf** (note : les deux ont la même taille)

SetBufSize : - Redimensionne les tampons **RxBuf** et **TxBuf**, tout en s'assurant que les données qui s'y trouvent sont préservées et que les index (**TxInIdx**, **TxOutIdx**, **RxInIdx** et **RxOutIdx**) sont ajustés en conséquence.

- Un code d'erreur est retourné si les anciens **RxBuf** et **TxBuf** contiennent trop de données pour la nouvelle taille. L'opération n'est alors pas effectuée, c'est-à-dire que la taille n'est pas changée.

- Cette commande n'est accessible que pour un usager ayant des permissions d'administrateur (**CAP_SYS_ADMIN**).

3. Port Série

La présente section se veut un résumé, ciblé vers les besoins du lab, du document de référence (DataSheet) du module 16550 (voir fichier **pc16550d.pdf** fournit en guise d'annexe). Le Port Série qui sera utilisé dans ce lab est équipé de deux (2) modules de ce type, formant ainsi un Port Série Dual (2 ports).

Fréquence (f_{clk}) : **1.8432 MHz**	**Port 0**	**Port 1**
Adresse de base :	0x1100	0x1108
Numéro d'interruption :	20	21

Le module 16550 est composé de 12 registres de 8 bits chacun, répartit sur 8 adresses à partir de l'adresse de base, qui servent à configurer, contrôler et utiliser la communication série :

Adresse	Accès	DLAB	Nom	Fonction
0x00	R	0	RBR	Receiver Buffer Register (Read Only)
0x00	W	0	THR	Transmitter Holding Register (Write Only)
0x00	R/W	1	DLL	Divisor Latch LSB
0x01	R/W	1	DLM	Divisor Latch MSB
0x01	R/W	0	IER	Interrupt Enable Register
0x02	R	-	IIR	Interrupt Identification Register (Read Only)
0x02	W	-	FCR	FIFO Control Register (Write Only)
0x03	R/W	-	LCR	Line Control Register
0x04	R/W	-	MCR	MODEM Control Register
0x05	R/W	-	LSR	Line Status Register
0x06	R/W	-	MSR	MODEM Status Register
0x07	R/W	-	SCR	Scratch Register

Ce qui suit est une description plus détaillée des registres qui seront nécessaires dans le cadre du lab. Les autres registres font référence à des fonctionnalités du module 16550 que nous n'utiliserons pas.