BUT 2 / R3.05
PROGRAMMATION SYSTÈME

INTRODUCTION: RÔLE DU SE, ASPECTS MATÉRIELS ET LOGICIELS, MODES NOYAU ET UTILISATEUR

ORGANISATION ET ÉVALUATION DE LA RESSOURCE

? ? ?

- **8 séances en groupe, 6 en demi-groupe, 1 DS final**

? ? ?

- Évaluation :
  - Note de TP (implication, avancement)
  - Un DS intermédiaire **? ? ?**
  - Un DS final

SOURCES

- Cours de P-A Champin / X. Merrheim (IUT)
  www.champin.net > enseignement > Système d'exploitation

- M. Kanté (Université Clermont Auvergne)
  https://perso.isima.fr/~makante/cours/cours-os.pdf

- Linux - 4e éd - Programmation système et réseau - Cours et exercices corrigés, Joëlle Delacroix

TENTATIVE DE PLANNING

Partie “cours”

- Système d’exploitation : intro
- Ordonnancement, priorités
- Création et gestion des processus sous Linux
- Entrées-sorties
- Signaux
- Threads
- Programmation réseau

Partie TP

- Ordonnancement
- Création et gestion des processus sous Linux
- Entrées-sorties, implémentation d’un mini shell
- Threads (Java)
- Programmation réseau (Java) :
  - Client de messagerie
  - Serveur web

DEFINITION : SYSTÈME D'EXPLOITATION

- Logiciel permettant d'exploiter un système informatique
- Les 5 A: Aide, Abstraction, Augmentation, Autorisation, Arbitrage
  - Aide: simplifier l'utilisation du système informatique (ex. l'accès aux périphériques)
  - Abstraction: exposer au programme une abstraction des périphériques (même appel/fonction pour tous les périphériques de même nature)
  - Augmentation: des ressources non-disponibles peuvent être émulées (multi-tâches, mémoire virtuelle)
  - Autorisation: limiter l'accès aux ressources (droits des fichiers, privilèges, etc.)
  - Arbitrage: arbitrer l'accès aux ressources (ordonnancement, etc.)

→ aujourd'hui, on aborde surtout l'aspect Aide et Abstraction

QUESTIONS

- Quels problèmes liés à l’abstraction se poseraient si les programmes implémentaient directement l’accès aux ressources matérielles ?

- Quels problèmes liés à l’arbitrage se poseraient si les programmes accédaient directement aux ressources ?

TÂCHES OPÉRÉES PAR UN SE

- Interaction avec les périphériques (souris, clavier, écran, réseau, clé USB, etc.)
- Gestion des utilisateurs
- Gestion du/des systèmes de fichiers
- Gestion des droits/groupes
- Gestion des processus (création lourd/léger, ordonnancement, communication)
- Synchronisation des processus (sémaphores, etc.)
- Gestion du réseau (réception/transmission, TCP/IP, etc.)
- IHM utilisateur (commandes ou graphiques)
- Etc., etc.

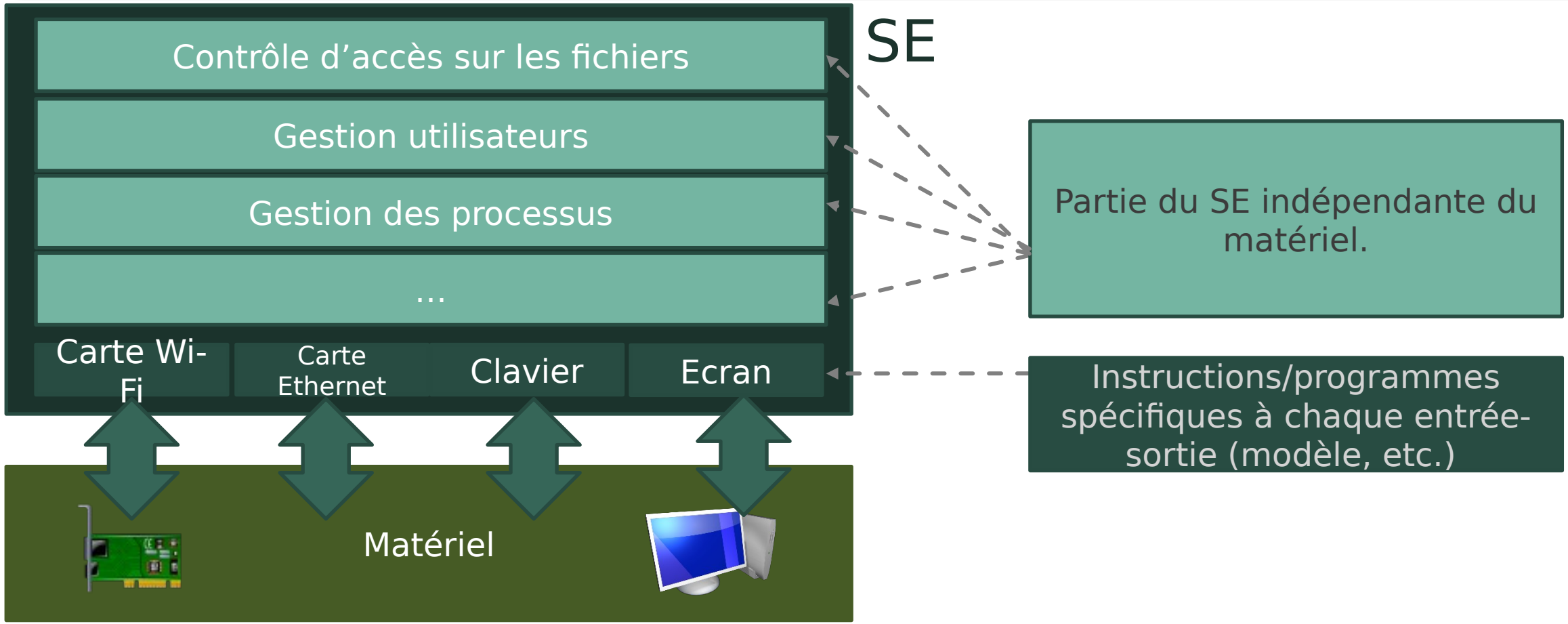
FONCTIONNEMENT EN COUCHE ET PILOTE

(ABSTRACTION)

DEFINITION

- Les périphériques d’entrée-sortie peuvent être différents même s’ils assurent la même fonction
  - Souris, écrans, clavier, etc.
- Les instructions à exécuter pour piloter ces périphériques peuvent être différentes d’un matériel à l’autre
- L’abstraction offre aux programmes une interface unique, indépendante du matériel, pour interagir avec un périphérique
- Le SE doit connaître le type de matériel, gérer les détails et spécificités de celui-ci.

FONCTIONNEMENT EN COUCHES

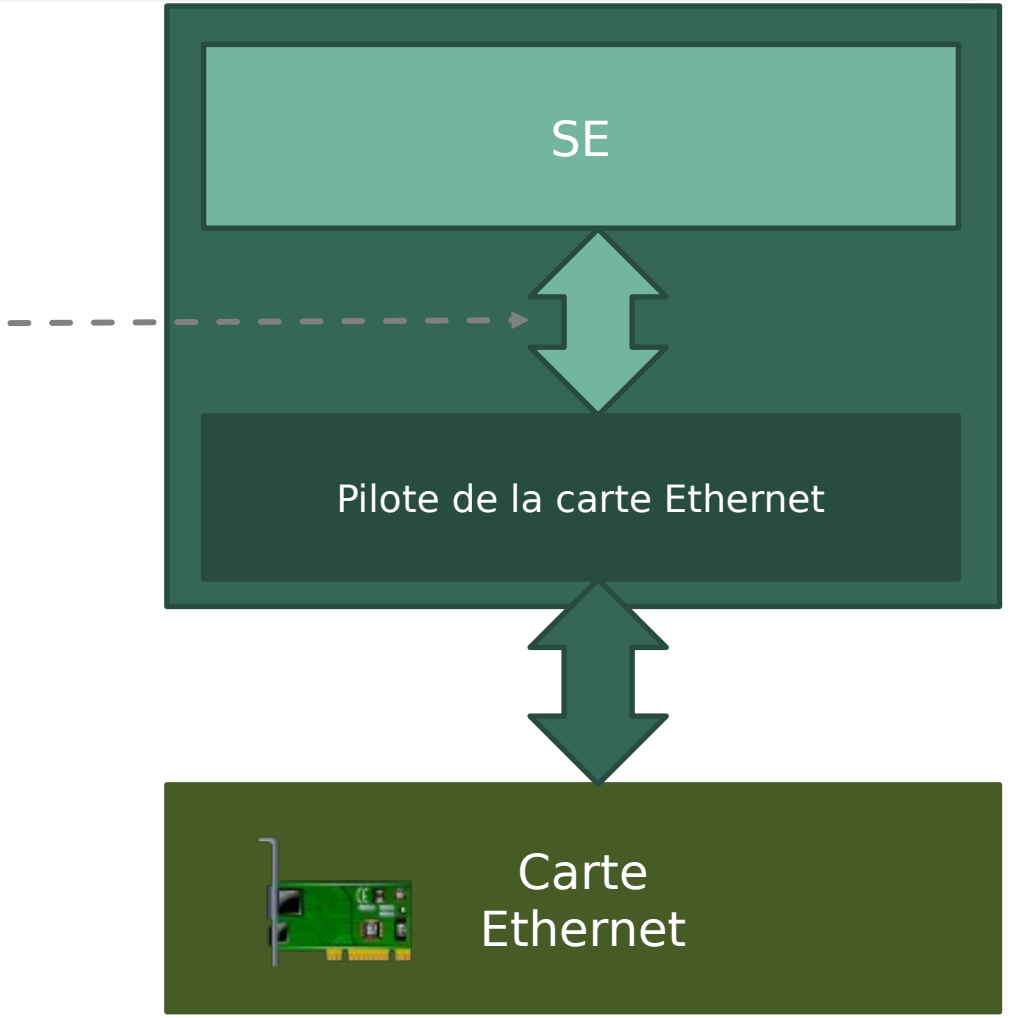


PILOTE DE PÉRIPHÉRIQUES

- Programme qui gère l’interaction avec un périphérique
- Très souvent implémenté par les constructeurs (de périphériques)
- Spécifique au matériel
- Spécifique au SE

PILOTE DE PÉRIPHÉRIQUES

Les interactions entre les pilotes sont en partie fixées pour chaque SE.
Le SE n'a donc pas besoin d'implémenter des appels spécifiques à chaque périphérique.



Exemple:

Tous les pilotes de carte réseau doivent implémenter la fonction `hard_start_xmit()` qui demande au pilote d'envoyer un paquet.

QUESTIONS

1. Les pilotes sont toujours développés par les constructeurs de périphériques ?
   - VRAI / FAUX
2. Que doit mettre un pilote à disposition du SE ?
   a. Rien. Le pilote n'a pas d'interaction avec le SE.
   b. Une liste des fonctions que le SE peut utiliser pour interagir avec le pilote.
   c. Des fonctions qui ont déjà été fixées par le SE (quel que soit le pilote).
3. Quelle proposition ci-dessous est vrai.
   a. Les couches basses du SE sont plus dépendantes du matériel.
   b. Les couches hautes du SE interagissent avec les pilotes.
   c. Il n'y a pas de notion de couches pour les SE.

INTERRUPTIONS

INTERRUPTIONS / MODE NOYAU ET UTILISATEUR / APPELS SYSTÈME

MODE NOYAU / UTILISATEUR

- Le processeur comporte au moins deux « modes » :

MODE NOYAU (ou privilégié) **MODE UTILISATEUR**

- En mode noyau, toutes les instructions assembleur sont autorisées
  - On peut invoquer des *appels système*
- En mode utilisateur, il y a des restrictions :
  - Ne peut pas « accéder » au matériel
  - Plages de mémoire limitées à l'espace du processus
- Une instruction assembleur voulant accéder au matériel en mode utilisateur déclenche une erreur
- Les processus lancés par les utilisateurs démarrent en mode utilisateur
- Le processeur peut passer en mode noyau pour deux raisons :
  - Le processus fait un appel système
  - Il y a une interruption

INTERRUPTION MATÉRIELLE

- Le SE peut demander régulièrement à un périphérique si un événement est survenu (mouvement de souris, réception de données, etc.): polling
  - Pas toujours pratique car oblige à interroger régulièrement le pilote
  - Consomme des ressources inutilement / ralenti les performances
- Les périphériques peuvent notifier d'un événement au travers d'une **interruption matérielle** (c'est le contraire du polling)

  → On appelle cela une **IRQ (Interruption ReQuest)**
  - Réception sur la carte réseau
  - Données lues sur un disque dur
  - Mise à jour de l'horloge (souvent 100 ou 1000Hz)
  - Tape sur le clavier / mouvement de souris
  - Etc.

INTERRUPTION MATÉRIELLE

- Ce sont des circuits dédiés.
- Leur nombre est généralement de 16 (plusieurs périphériques peuvent se partager le même IRQ).
  - Il y a la possibilité de passer des paramètres grâce à des registres
  - Certains processeurs ont des systèmes plus complexes (ex : APIC chez Intel : 256 IRQ) quelques exemples :
    - Horloge système
    - Clavier
    - Disque
    - Interruptions logicielles
    - Réseau

INTERRUPTION MATÉRIELLE

Lorsqu'une interruption se produit, le processeur :

- *passe en mode noyau*
- sauvegarde le contexte (état des registres) du programme courant
- exécute le gestionnaire d'interruption (un circuit dédié)
- restaure le contexte sauvegardé et reprend le processus courant (dans le bon mode d'exécution) ou un autre processus

INTERRUPTION MATÉRIELLE

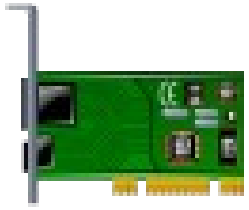
Contrôleur d'interruption

CPU

2. Emission d'un IRQ (5)

3. Emission d'un IRQ (5)

4. Interrompt le processus courant (changement de contexte)



5. Recherche de l'adresse mémoire correspondant à l'interruption 5 (0xaff134).

1. Réception d'un paquet IP.

IRQ	Adresse mémoire
1	0x56453
2	0xaff134
5	0xdd6a3

6. Exécute les instructions à l'adresse indiquée (en mode noyau)

7. Revient au processus interrompu (changement de contexte) ou à un autre

AUTRES INTERRUPTIONS (INTERRUPTIONS LOGICIELLES)

- Il existe d'autres interruptions dites **logicielles**

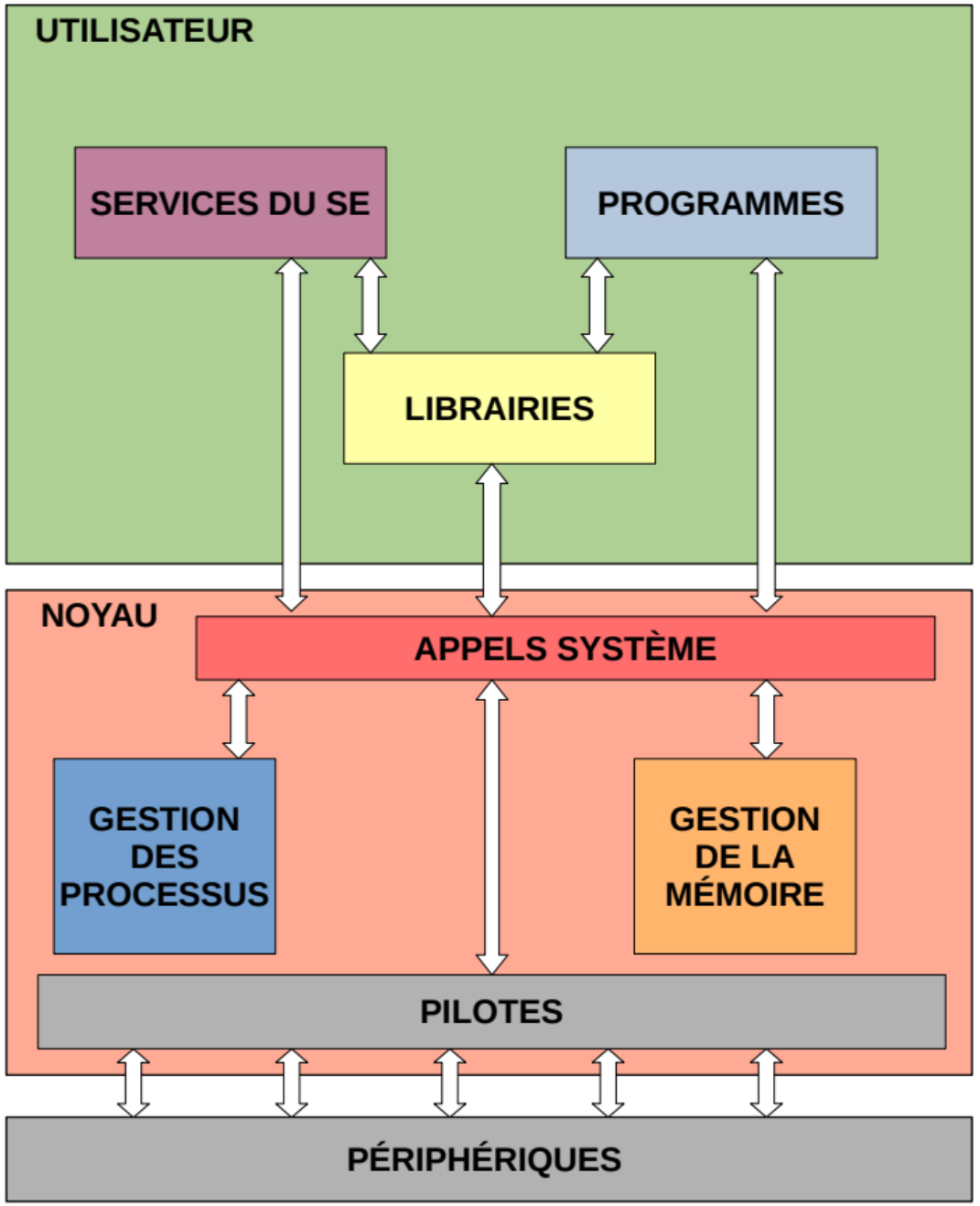
1. Elles sont déclenchées par le microprocesseur (exceptions)
   - Instruction invalide
   - Erreur de calcul (ex. division par 0, dépassement float/int)
   - Accès à une adresse mémoire erroné
   - ...
2. Appels systèmes (voir plus loin)

- Dans les deux cas une routine du SE est lancée à partir d'un tableau/vecteur (valeur de l'interruption – routine)
- Même comportement que pour une interruption matériel (changement de contexte, passage en mode noyau, etc.).

APPELS SYSTÈMES

- Les appels systèmes sont des fonctions qui peuvent être appelées dans le code des applications.
- **Les appels systèmes sont des fonctions du noyau.**
- Ils permettent aux applications de demander des services au SE:
  - Matériel
    - Afficher quelque chose
    - Lire/écrire dans un fichier
    - Recevoir/envoyer des données sur le réseau
    - Etc.
  - Ou logiciel
    - Créer un processus ou un thread
    - Envoyer un signal à un processus
    - Réserver de la mémoire
    - Etc.

RÉSUMÉ


UTILISATEUR
SERVICES DU SE
PROGRAMMES
LIBRAIRIES
NOYAU
APPELS SYSTÈME
GESTION DES PROCESSUS
GESTION DE LA MÉMOIRE
PILOTES
PÉRIPHÉRIQUES

POUR LES PROCHAINS TP

- Compiler du C sous Linux
- Utilisation du « man » sous Linux
- Gestion des erreurs sous Linux

LE MANUEL SOUS LINUX

- Fonctionnement du manuel
  - En ligne de commande :

    `man uneFonction`

    `uneFonction` peut être
    - Une commande de l'invite de commande (ex : cd)
    - Un appel système (ex : write)
    - Une fonction d'une bibliothèque (ex : fopen)
    - Un programme de gestion du système (ex : ifconfig)

Pour connaître le fonctionnement du man :

`man man`

  - Disponible aussi sur internet depuis n'importe quel moteur de recherche

APPEL SYSTÈME OU FONCTION D'UNE LIBRAIRIE? LES MAN

- Fonctionnement du manuel/man (ligne de commande ou web)

SLEEP(3) **Linux Programmer's Manual** **SLEEP(3)**

NAME top

sleep - sleep for a specified number of seconds

SYNOPSIS top

```
#include <unistd.h>

unsigned int sleep(unsigned int seconds);
```

DESCRIPTION top

sleep() causes the calling thread to sleep either until the number of real-time seconds specified in *seconds* have elapsed or until a signal arrives which is not ignored.

RETURN VALUE top

Zero if the requested time has elapsed, or the number of seconds left to sleep, if the call was interrupted by a signal handler.

ATTRIBUTES top

For an explanation of the terms used in this section, see

1 - commandes
2 – appels systèmes
3 – fonction de la libc
8 – gestion du système

Librairies à inclure.
Prototype de la fonction.

Ce qui est retourné par la fonction.

Important: pour les appels systèmes il y a une section ERROR. Elle indique les valeurs de la variable **errno** mis à jour par le SE pour chaque appel système.

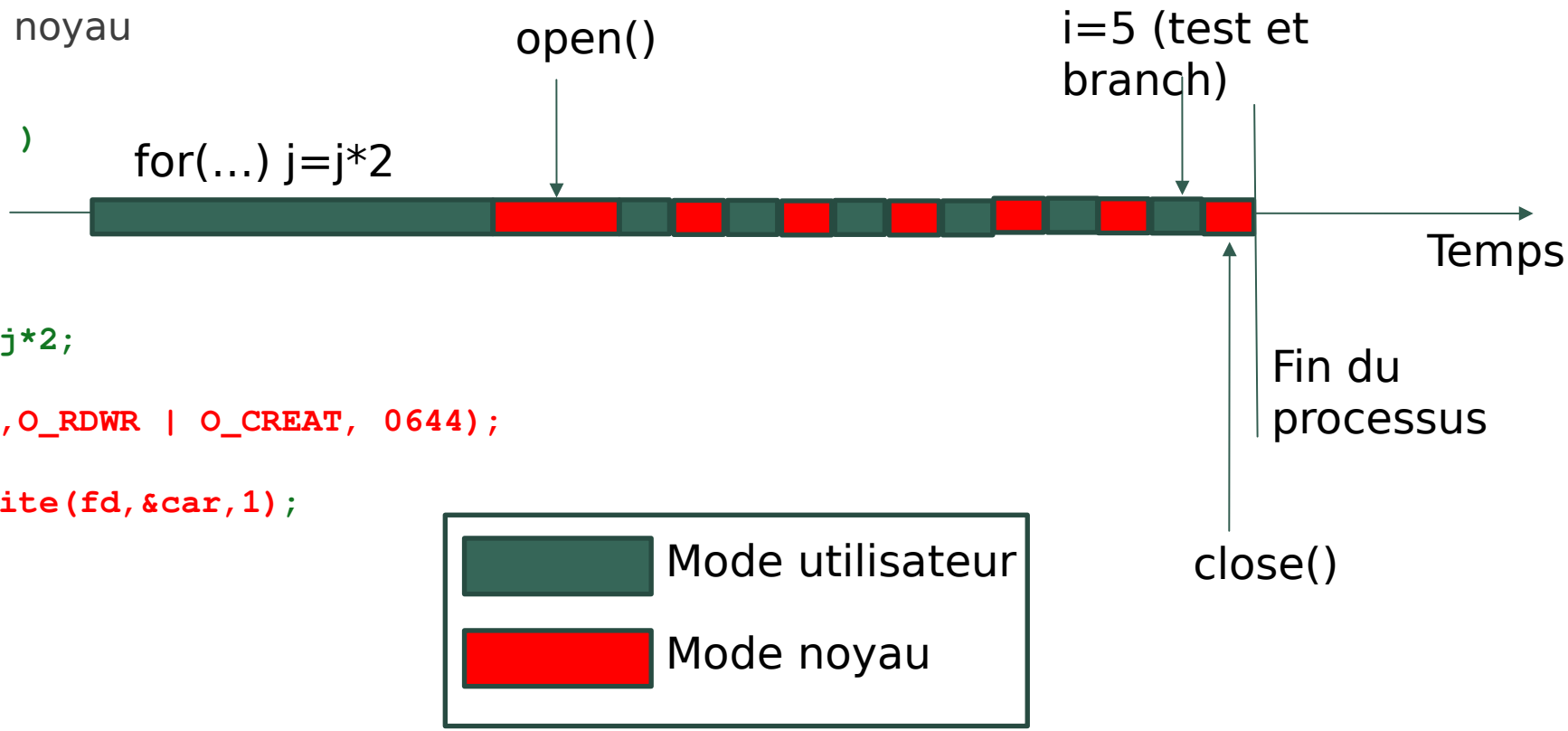
EXERCICE 1

- Consulter le man de l'appel système write() (→ et non pas la *commande* write !)
  - Est-ce un appel système ou une fonction de la libc?
  - Que fait la fonction?
  - Quelles sont les librairies à inclure?
  - Quelle est le type et la sémantique de la valeur renvoyée ?
  - Erreur:
    - Les erreurs sont elles indiquées par des chaînes de caractères ou des entiers?
    - Où (dans quel fichier) sont déclarées ces valeurs ?
    - Donnez le nom et la valeur numérique pour une des constantes.

DÉROULEMENT D'UN PROCESSUS

- Un processus passe du mode utilisateur au mode noyau en fonction de ses appels
  - Fonctions/instructions classiques : mode utilisateur
  - Appels systèmes: mode noyau

```
int main( int argc, char** argv )
{
    int j=1;
        char car = '0';

    for(int i=0; i<100; i++) j=j*2;

    int fd = open("fichier.txt",O_RDWR | O_CREAT, 0644);

    for(int i=0; i<500; i++) write(fd,&car,1);

    close(fd);

        return 0 ;
}
```



QUESTIONS

- Lorsque l’un processus est en cours d’exécution, comment le SE peut reprendre la main pour exécuter un autre processus dans un contexte multi-tâche?
- Que se passe t-il lors d’une interruption (rappeler les étapes) ?
- Quels sont les deux types d’interruption logicielles?
- Quelles sont les étapes lors de l’appel d’un appel-système par un processus?
- Quelle est la différence entre sudo/root et le mode noyau sur le microprocesseur?

LA COMMANDE TIME

```
TIME(1)                    General Commands Manual                    TIME(1)

NAME
       time - run programs and summarize system resource usage

SYNOPSIS
       time   [ -apqvV ] [ -f FORMAT ] [ -o FILE ]
              [ --append ] [ --verbose ] [ --quiet ] [ --portability ]
              [ --format=FORMAT ] [ --output=FILE ] [ --version ]
              [ --help ] COMMAND [ ARGS ]

DESCRIPTION
       time run the program COMMAND with any given arguments ARG....  When COMMAND finishes, time
       displays information about resources used by COMMAND (on the standard error output, by
       default).  If COMMAND exits with non-zero status, time displays a warning message and the
       exit status.
```

- Permet de connaître le temps passé en mode noyau et utilisateur
- Par défaut, 3 « temps » :
  - real : chronomètre (en secondes) du temps passé pour exécuter le programme
  - user : temp CPU passé en mode utilisateur
  - sys : temps CPU passé en mode noyau
- Attention : d'autres processus peuvent s'exécuter en même temps (cf cours sur l'ordonnancement)
  → ce temps est compté dans « real », mais pas dans « user » ni « sys »
- Attention : votre système peut comporter plusieurs processeur
  → ce temps est compté dans « user » et « sys », mais pas vraiment dans « real »

EXERCICE 2

```
int main()
{
  double reel=1.0;

  for(int i=0;i<99999999;i++)
     reel*=1.0;

  return 0;
}
```

- Recopiez le code suivant dans un fichier appelé **exo2.c**

- Rendez-vous en ligne de commande (commande **cd**) dans le répertoire où vous avez placé le fichier, puis :

```
gcc -o exo2 exo2.c
```

- → si la compilation a fonctionné, cela a créé l'exécutable **exo2**, que l'on exécute en faisant :

```
./exo2
```

- Observez les temps passés en mode utilisateur et mode noyau :

```
time ./exo2
```

- expliquez

EXERCICE 3

```
#include <unistd.h>
#include <fcntl.h>

int main()
{
   double reel = 1.0;

   for (int i = 0 ; i < 99999999 ; i++)
      reel *= 1.0;

   char car = '0';
   int fd = open("fichier.txt", O_RDWR | O_CREAT , 0644);
   for (int i = 0 ; i < 50000 ; i++)
      write(fd, &car, 1);

   close(fd);

   return 0 ;
}
```

- Complétez le code précédent comme suit dans un fichier appelé **exo3.c**

- Rendez-vous en ligne de commande (commande **cd**) dans le répertoire où vous avez placé le fichier, puis :

```
gcc -o exo3 exo3.c
```

- → si la compilation a fonctionné, cela a créé l'exécutable **exo3**, que l'on exécute en faisant :

```
./exo3
```

- Observez les temps passés en mode utilisateur et mode noyau :

```
time ./exo3
```

- expliquez

LA COMMANDE TIME

- Dans quels cas peut on avoir :
  - real = sys + user
  - real ≠ sys + user
  - real > sys + user
  - real < sys + user

EXERCICE 4

- Sous Windows
  - Allez sur le poste de travail / clic droit / propriétés / gestionnaire de périphérique
  - Allez sur la carte réseau (Ethernet) et le clavier et observez l'IRQ et l'adresse affectée à cette IRQ
- Sous Linux
  - Tapez la commande suivante: `lspci -v`
  - Observez pour certain des périphériques les numéros des IRQ