Systèmes d'exploitation

Chapitre 1 : Généralités

Chargé de cours :
Emery Kouassi Assogba
Tél : 95 22 20 73
Emery.assogba@uac.bj
Emery.assogba@uclouvain.be

Objectifs du cours

Objectifs :

- Comparer différentes réalisations pour les systèmes d'exploitation et mettre en avant les avantages et inconvénients de ces réalisations
- Comprendre et expliquer ce que sont les principaux problèmes à résoudre par un système d'exploitation et présenter les différentes solutions qui y sont apportées avec leurs avantages et leurs inconvénients
- Comprendre et mettre en œuvre les mécanismes assurant la qualité des services
- Identifier les choix importants en matière de configuration et de gestion de systèmes, ainsi que les critères de décision pour effectuer ces choix.

Objectifs du cours

Objectifs

- Compréhension détaillée (théorique et pratique) du fonctionnement des systèmes d'exploitation
- Cas d'étude : Famille Unix
- Linux pour les travaux pratiques
- Principaux problèmes abordés
  - Processus et threads: concepts, problèmes et solutions
  - Communication entre processus
  - Gestion de la mémoire
  - Entrées-sorties
  - Systèmes de fichiers

Travaux pratiques

Exercices de base

- Petits programmes de base en C sous LINUX
  - A faire individuellement chaque semaine
  - S’inscrire sur www.inginious.uac.bj au cours Introduction à la programmation C
- Projet
  - Programme à faire par groupe de deux étudiants
  - Enoncé d’ici deux semaines
  - Projet à remettre en deux phases :
- Validation de l'architecture mi décembre
  - Rapport final et code source Linux mi-mars
  - Programmation kernel
  - Ajout d'une nouvelle fonctionnalité dans le kernel

Modalités d'évaluation

- Partie théorique
  - Examen oral à livre fermé portant sur toute la matière couverte au cours théorique et TPs
  - Pondération
    - 50% de la note finale pour l'examen oral
  - Travaux pratiques
    - Exercices individuels en C
    - Pénalités si les exercices ne sont pas rendus
  - Projet à faire par groupes de deux
    - 30% des points
  - Modification au kernel MINIX
    - 20% des points

Systèmes d'exploitation : bases

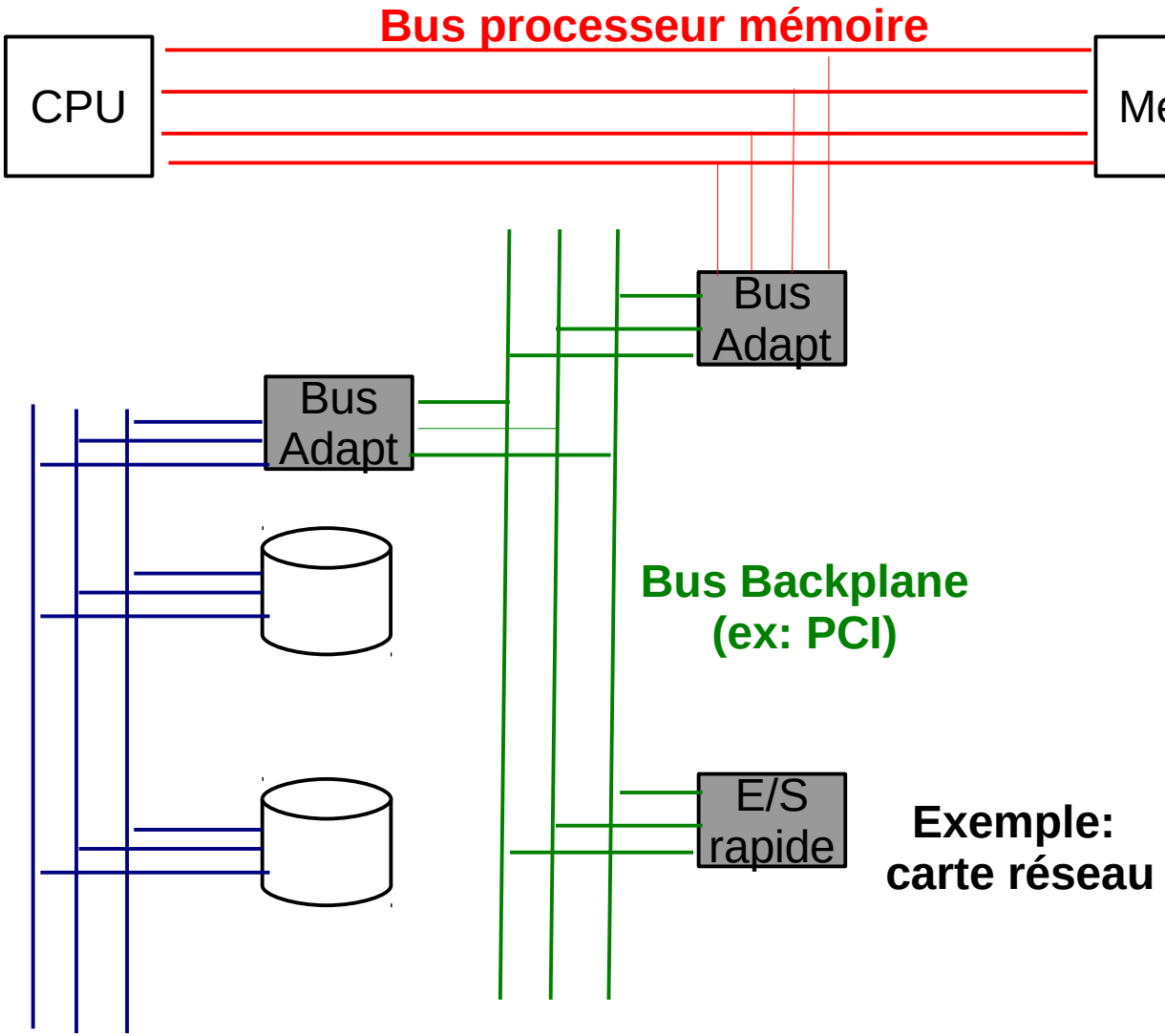
Fonctionnement des ordinateurs

- Microprocesseur
- Hiérarchie de mémoires
- Entrées/sorties

Le Système d'exploitation

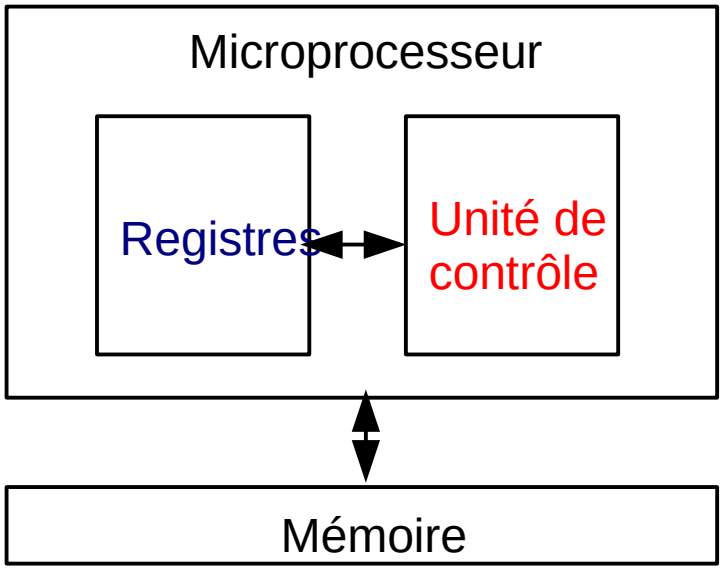
- Rôles
- Appels système
- Gestion des fichiers
- Gestion des processus

Organisation d'un ordinateur



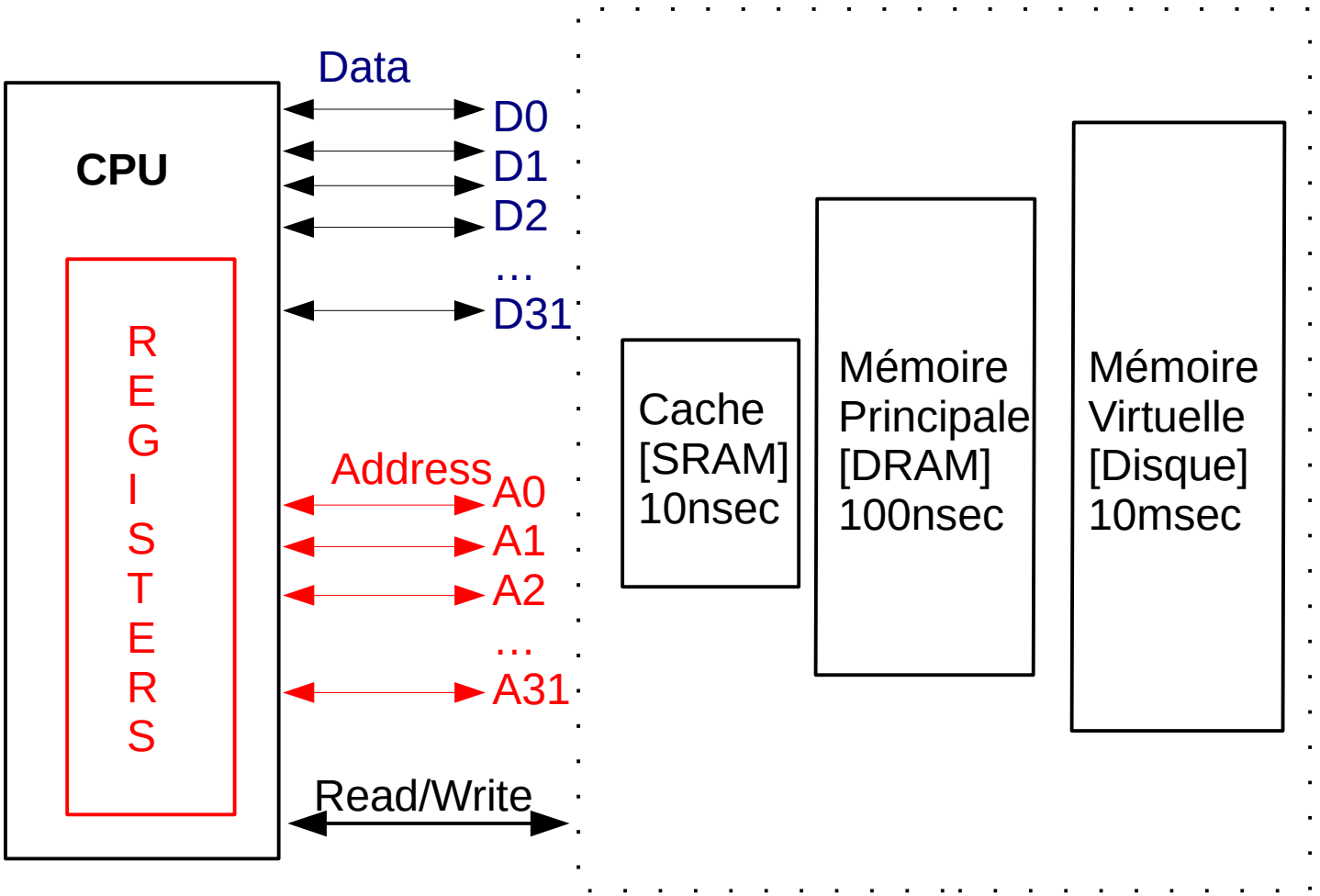
Microprocesseur

- Registres de calcul
- PC Program Counter
- SP Stack Pointer
- PSW Program Status Word



- Mode de fonctionnement
  - Mode kernel/protégé
    - Toutes les instructions sont utilisables
    - Le CPU peut accéder à toute la mémoire
  - Mode user
    - Les instructions de manipulation du matériel ne sont pas utilisables
    - Seule une partie de la mémoire est accessible

Hiérarchie de mémoires



- Objectif
  - L'accès aux différents niveaux de la hiérarchie doit
  - être le plus transparent possible pour le CPU

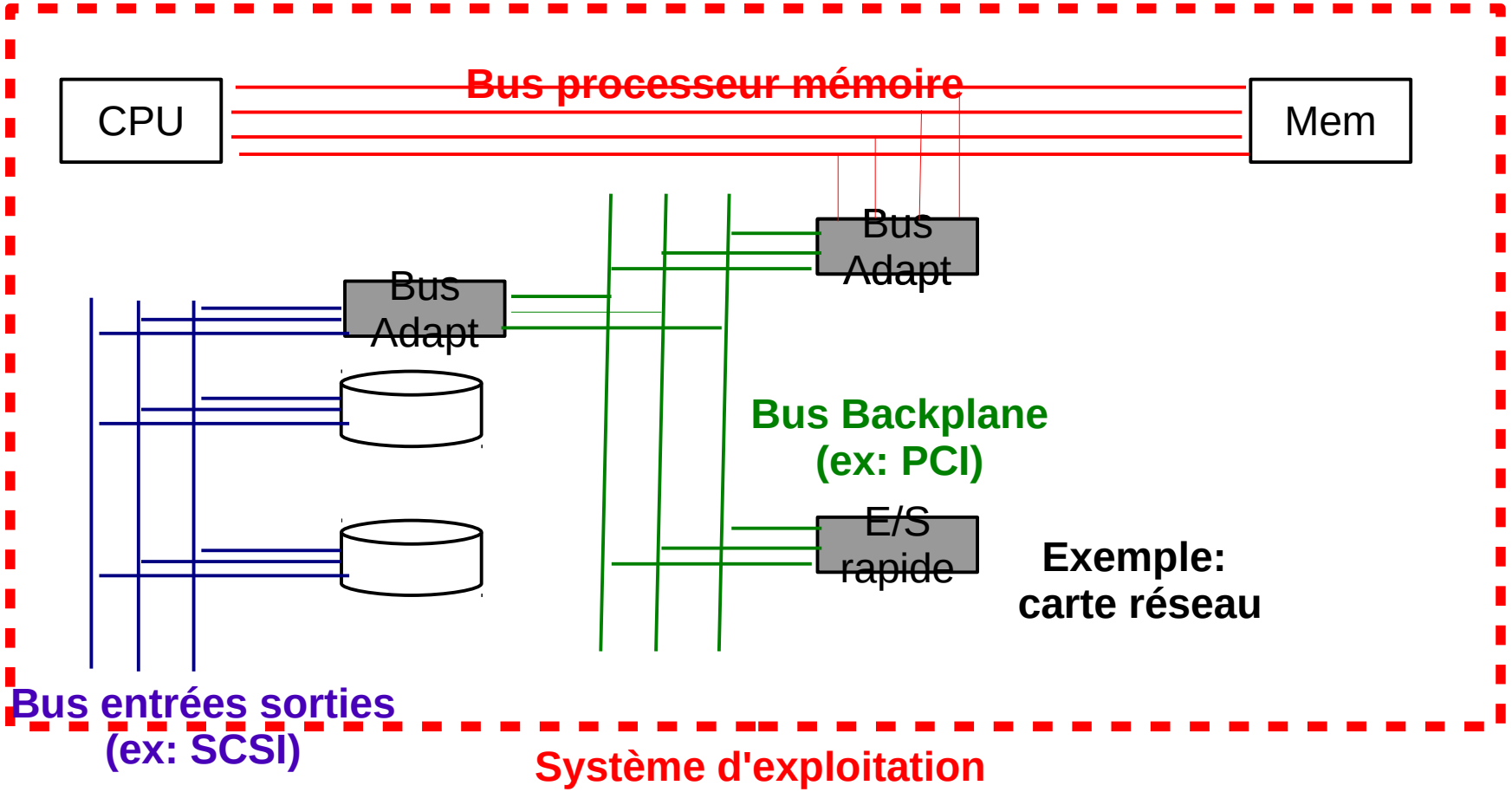
Entrées/Sorties

Dispositif	I/O	Utilisateur	Débit (KB/sec)
Clavier	I	Humain	0.01
Souris	I	Humain	0.02
Voix	I	Humain	0.02
Scanner	I	Humain	400
audio	O	Humain	0.6
Impr. laser	O	Humain	200
écran	O	Humain	60000
Modem	I/O	Machine	8
Réseau Lan	I/O	Machine	500-6000
floppy	stockage	Machine	100
Disque opt.	Stockage	Machine	1000
Disque dur	Stockage	Machine	2000-10000
Lect. Bandes	Stockage	Machine	2000-10000

- Nombreux dispositifs avec des caractéristiques souvent très différentes

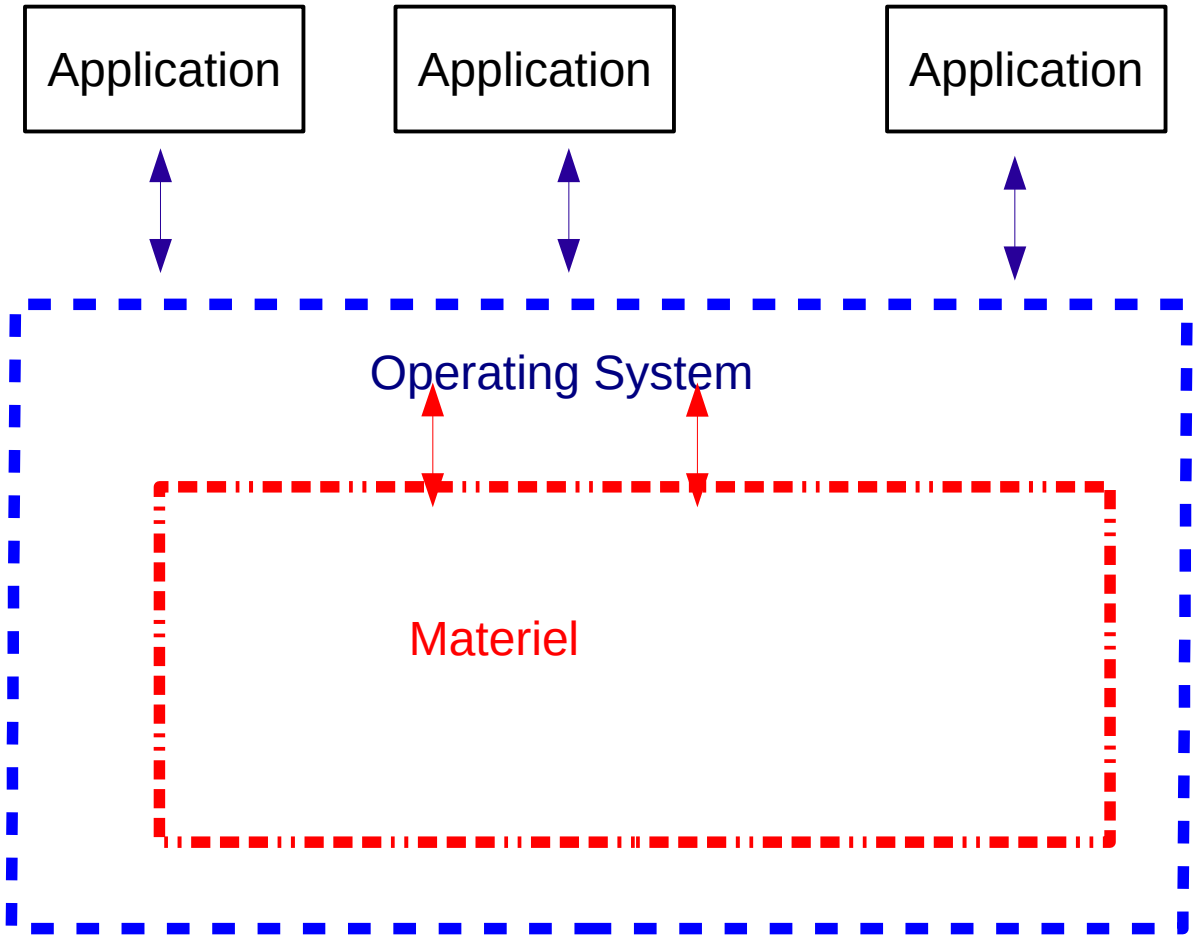
Rôle du système d'exploitation

- 1 : Gérer l'ensemble du matériel
  - Permettre aux applications d'y accéder sans en
  - connaître les détails technologiques



Rôle du système d'exploitation (2)

- 2 : Gestion des processus



Gestion des utilisateurs (2)

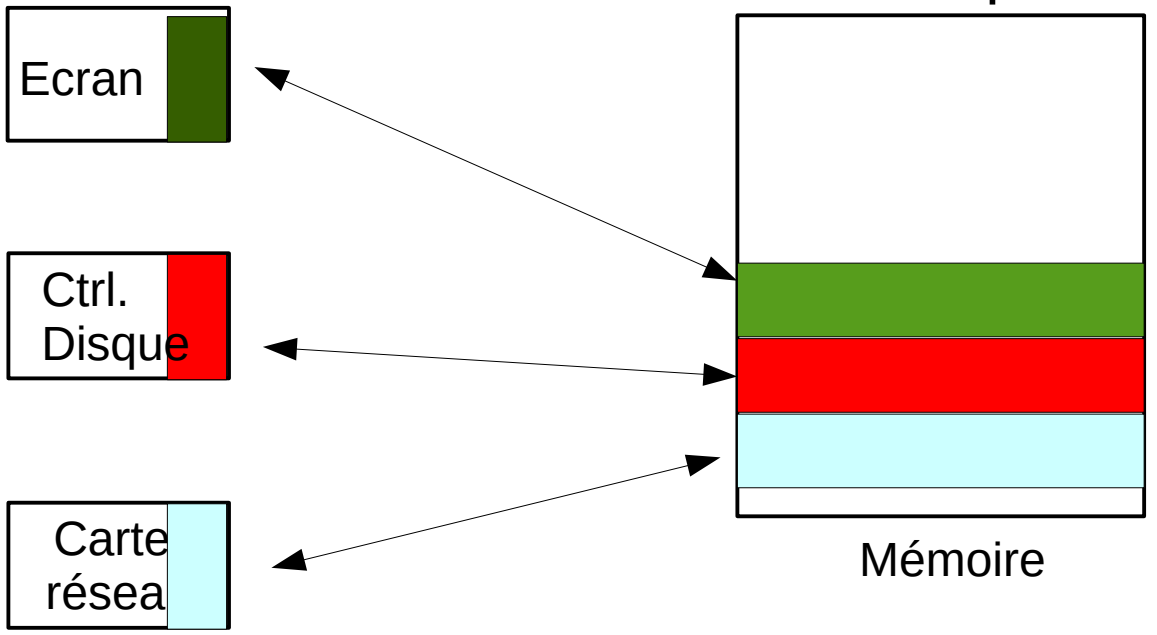
L'OS maintient une liste d'utilisateurs

- Utilisateurs !
- Administrateur (root sous Unix)

- Pour chaque ressource importante
  - Dispositif d'entrée sortie, fichiers, répertoires, ...
- Il faut définir quelles opérations chaque utilisateur/groupe d'utilisateurs peut effectuer
  - Lire
  - Ecrire
  - Exécuter
  - ....

Interactions OS->Matériel

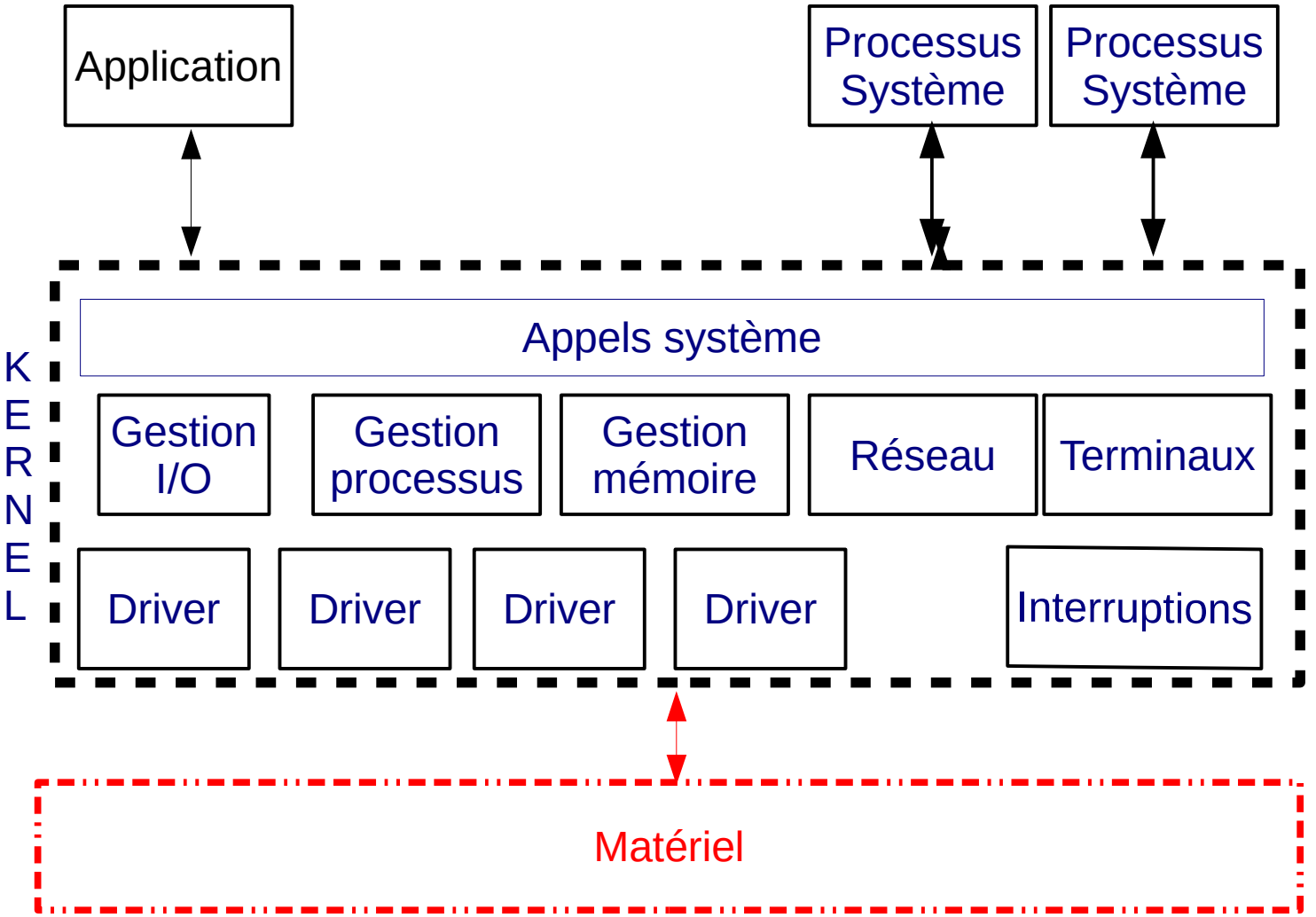
- Instructions spéciales
  - IN port value
  - OUT port value
  - Halt, break, ...
- Memory-mapped I/O
  - OS écrit commandes à des adresses spéciales



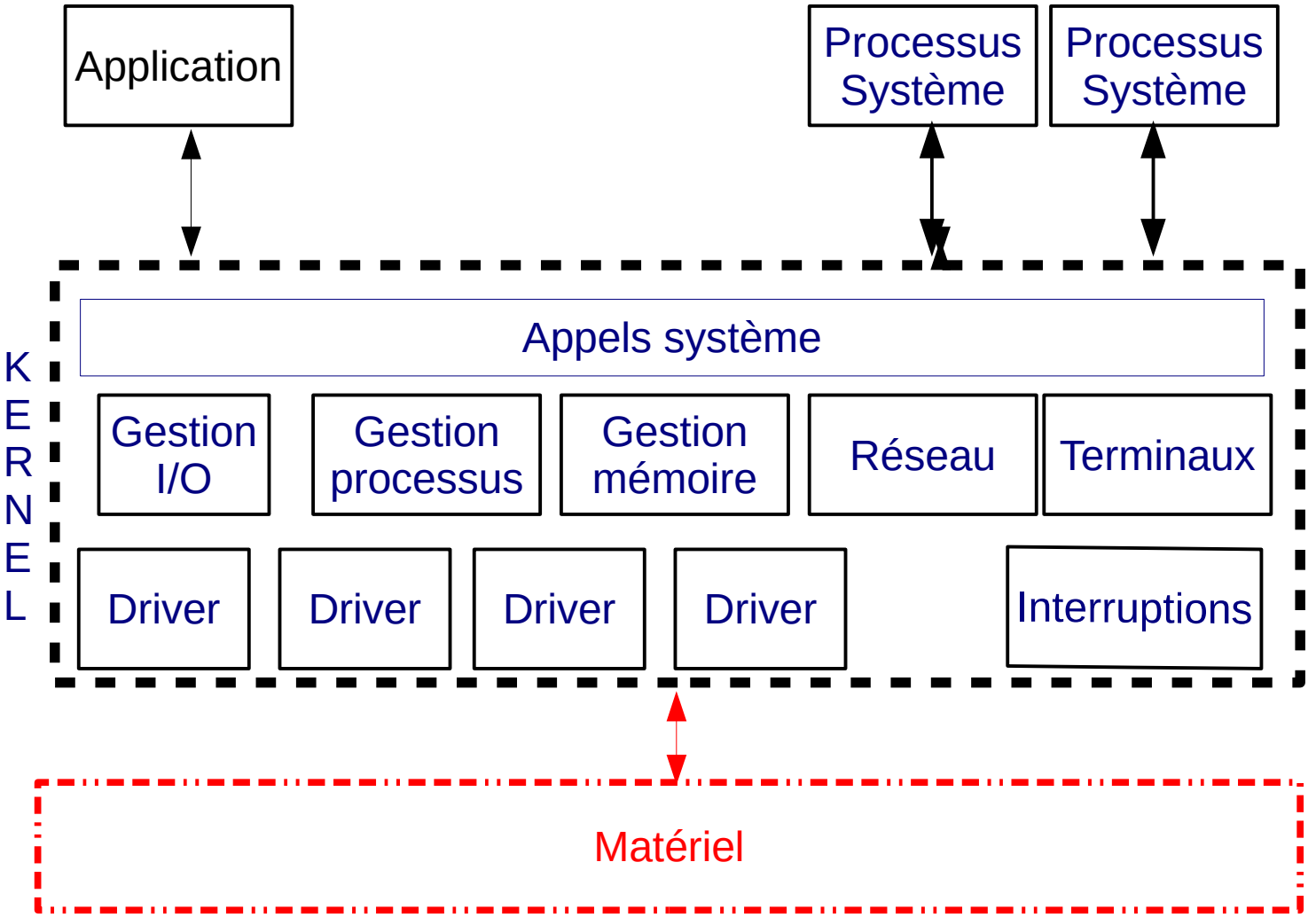
Interactions OS->Matériel

- Memory-mapped I/O
  - OS lit les réponses des dispositifs d'I/O à des adresses spéciales
- Interruptions
  - "Signal" produit par le processeur ou les dispositifs d'I/O
  - Division by zero
  - Segmentation fault
  - Data ready sur I/O
  - Horloge
  - ...
  - Provoque l'exécution de code placé par l'OS à une adresse spéciale
    - Interrupt vector

Organisation de l'OS

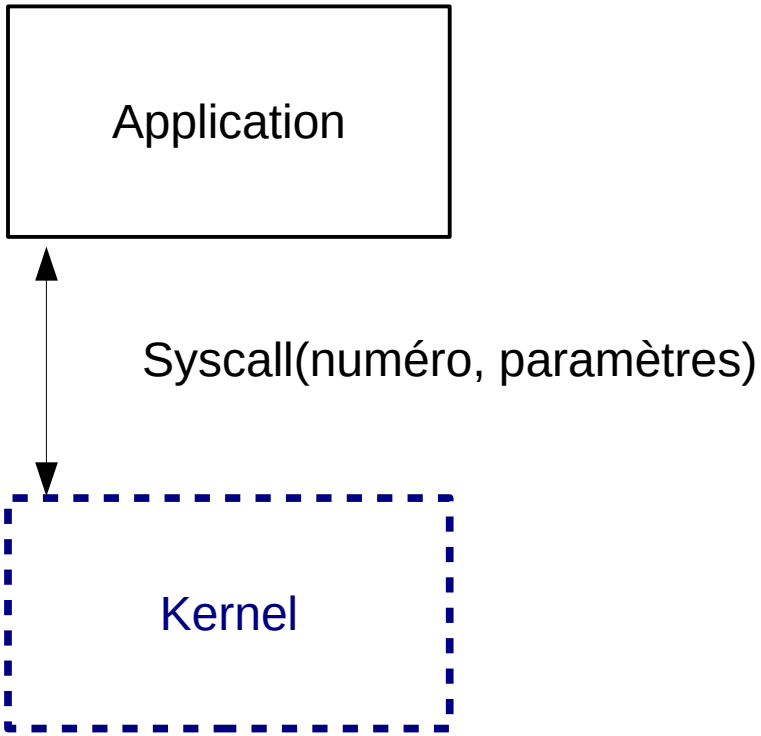


Organisation de l'OS



Appels système

- Interactions entre les applications et l'OS
  - Application spécifie
    - Identification de l'appel système
    - Paramètres éventuels
      - Registres ou en mémoire
  - Kernel analyse l'appel
    - CPU passe en mode kernel
    - Traitement des paramètres
    - Effectue l'appel système
  - Kernel
  - Kernel retourne à l'application en mode user
    - Résultats en registres ou en mémoire



L'appel système est la seule façon pour une application de faire exécuter (par le kernel) du code en mode protégé

Appels système(2)

- Exemples
  - exit(status)

    Un processus a terminé et retourne la valeur status au processus qui l'a créé
  - sleep(sec)

    Le processus entre en hibernation sec secondes
  - fd=fopen(file, params, ...)

    Ouverture du fichier file qui sera identifié après par fd
  - n=read(fd, buffer, nbytes)

    Lit N octets dans le fichier identifié par fd
  - s=close(fd)

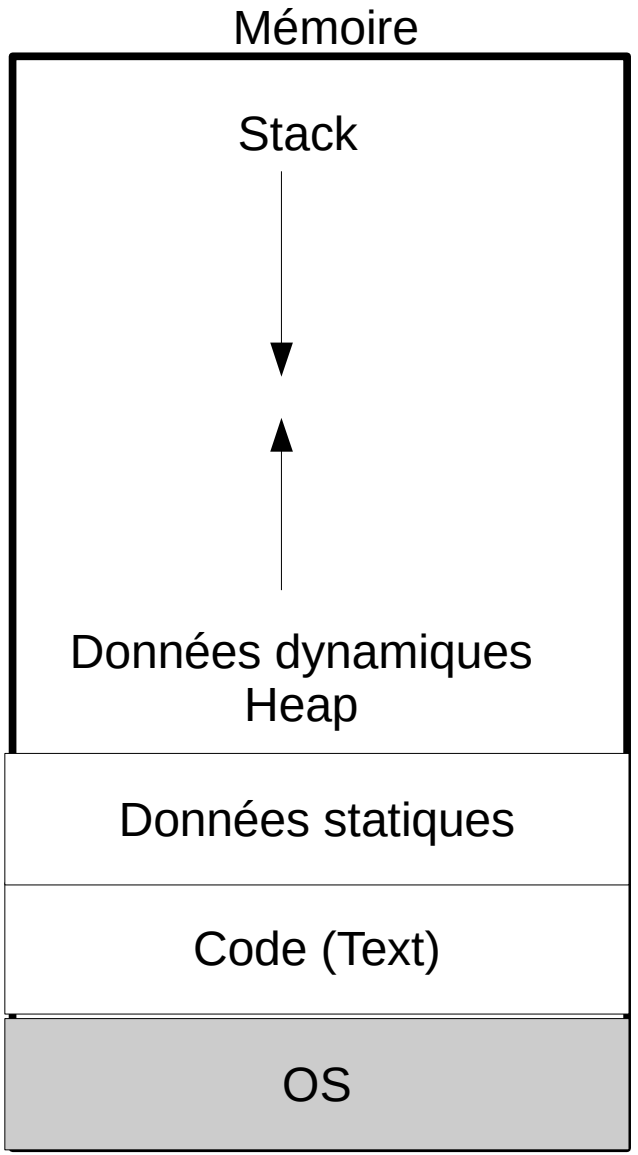
    Ferme le fichier identifié par fd
  - s=kill(pid,signal)

    Envoie un signal au processus pid

Les applications utilisent également des librairies où une fonction peut faire plusieurs appels système

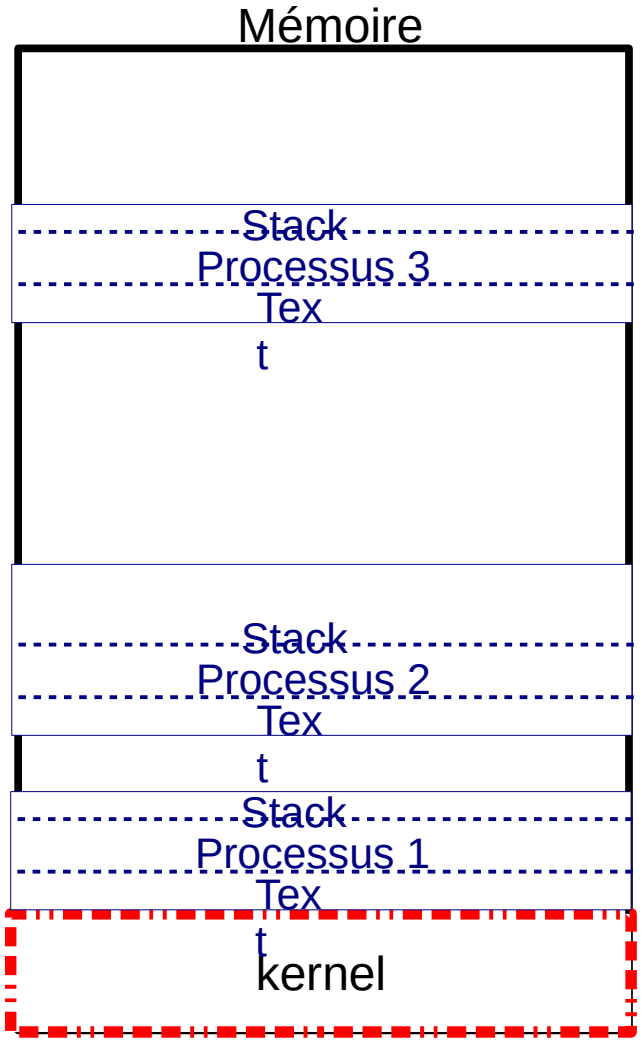
Organisation d'un processus

- Sur une machine simple monotâche



Organisation d'un processus

- Sur une machine multitâches...



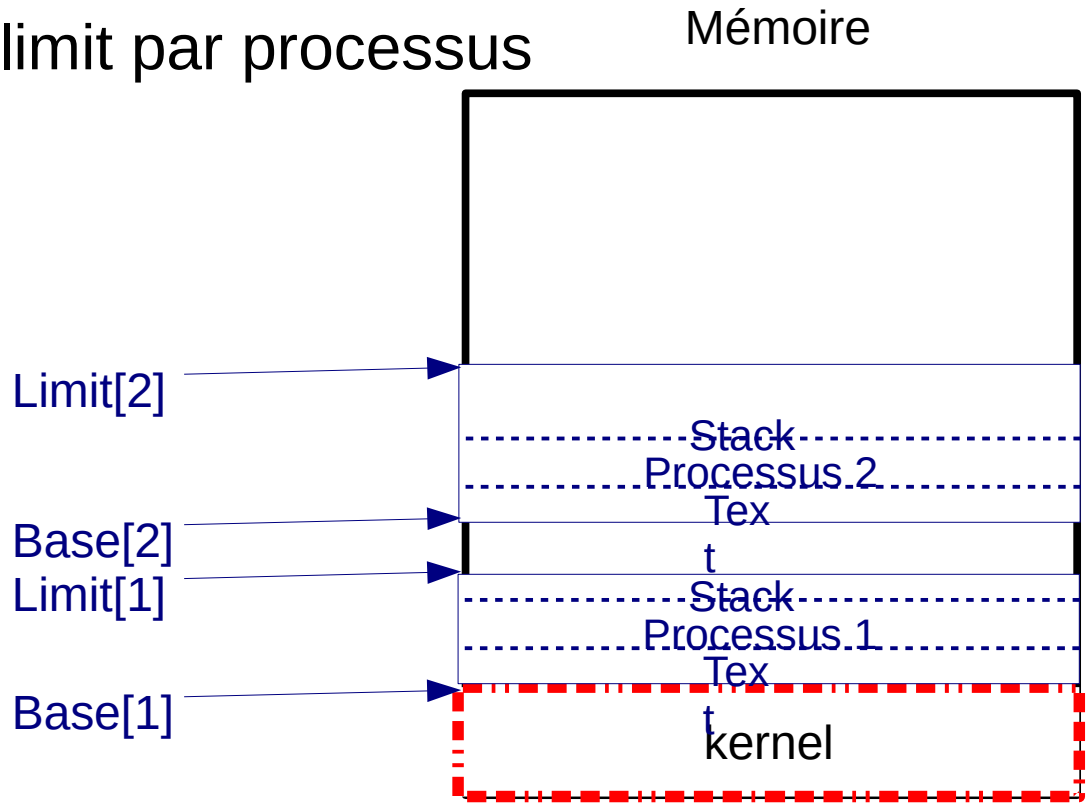
Gestion des processus

Problèmes à résoudre

- Partage de la mémoire entre l'OS et les processus utilisateurs ?
  - L'OS et les processus ont un stack, une partie text, ...
- Partage du CPU entre les processus et l'OS ?
  - Comment éviter qu'un processus qui boucle ne puisse bloquer l'OS et les autres processus ?
- Partage des accès aux dispositifs d'I/O ?
  - Accès exclusif à certains dispositifs
    - Ex: imprimante, graveur de CD, ...
  - Respect des permissions associées aux dispositifs

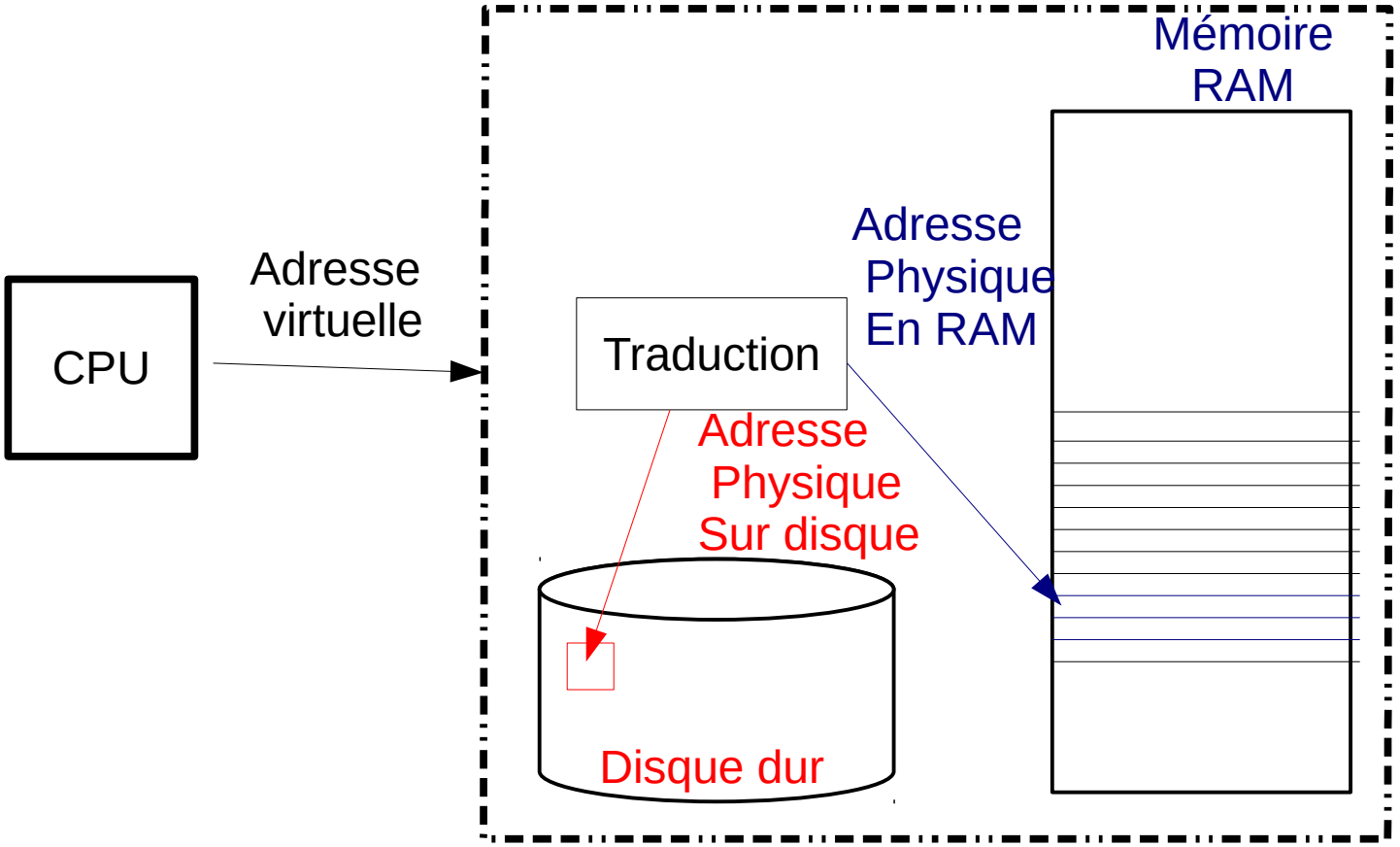
Partage de la mémoire

- Solution simple
  - Registres base et limit par processus



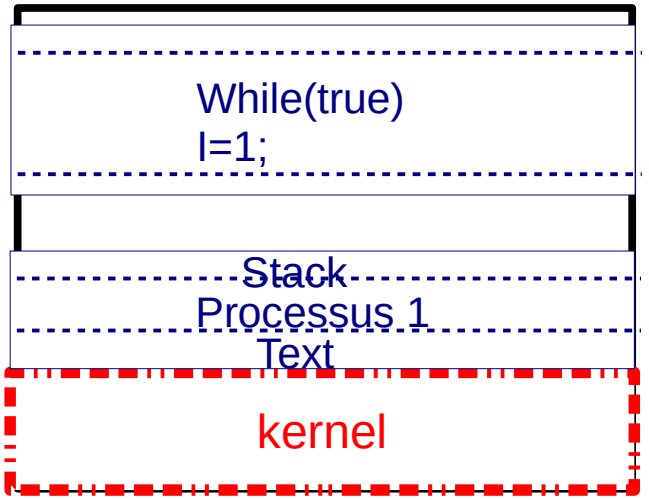
  - Solution actuelle
    - Mémoire virtuelle
    - Une table des pages pour chaque processus

Mémoire virtuelle



- Principe
  - l'information peut se trouver en RAM ou sur disque
  - le CPU y accède avec des adresses virtuelles

Partage du CPU

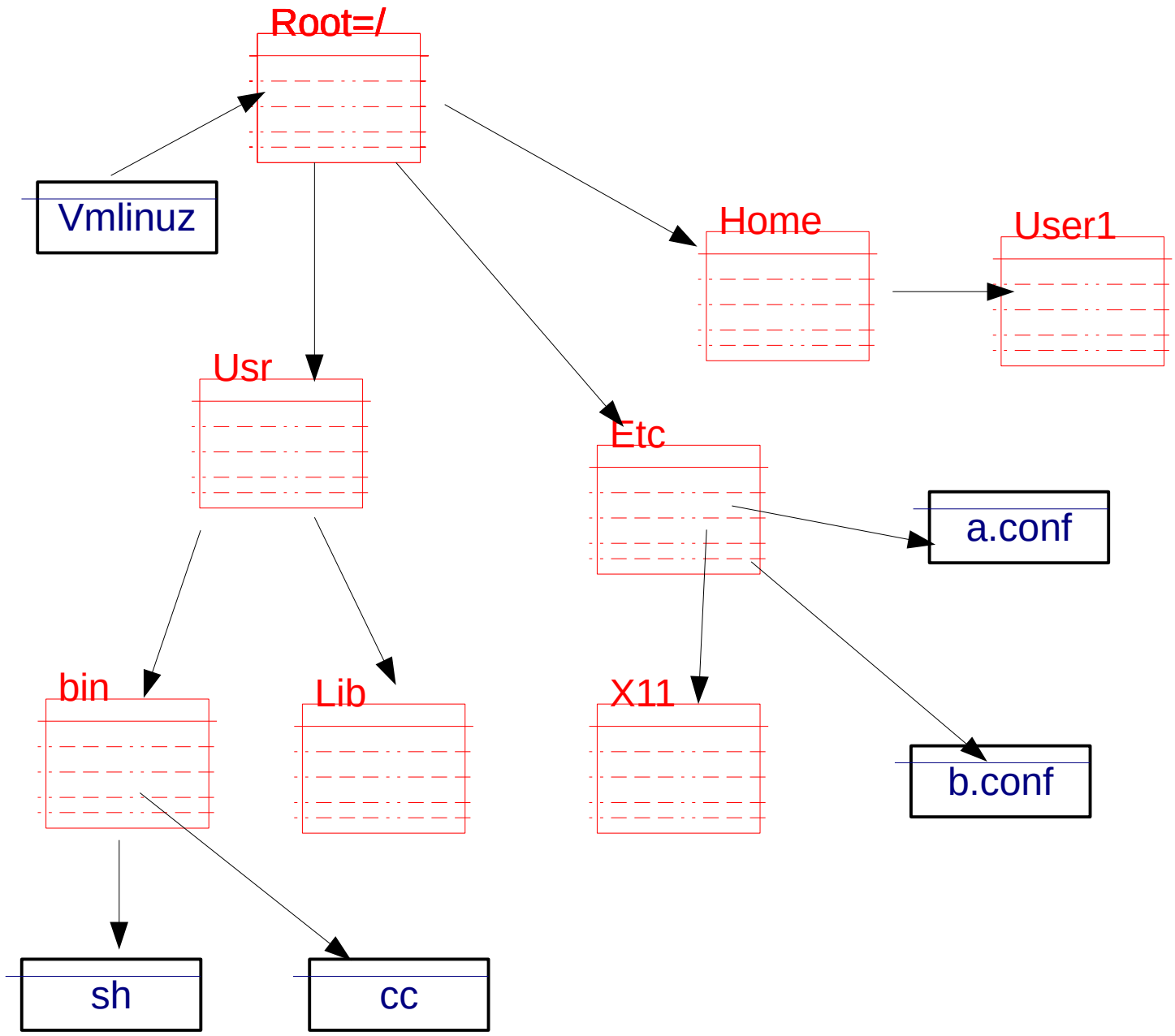


- Exécutions de code du kernel
  - Processus X effectue un appel système
  - Le kernel peut bloquer X durant l'exécution de l'appel et en profiter pour exécuter un autre processus
  - Hardware génère une interruption
    - Toutes les t millisecondes, interruption horloge
    - Le kernel peut en profiter pour arrêter le processus courant et exécuter un autre processus à la place
  - Conséquence
    - Un processus peut être interrompu à tout instant par le kernel !

Contexte d'un processus

- Ensemble des informations nécessaires pour poursuivre ultérieurement l'exécution du processus
  - Contenu des registres de données
  - Contenu du registre de sommet de pile
  - Contenu du Program Counter
- Passage du processus X au kernel
  - Kernel sauvegarde le contexte de X
- Passage du kernel au processus X
  - Kernel réinitialise les registres avec contexte de X

Système de fichiers



Contenu du Système de fichiers

- Informations stockées dans le filesystem
  - Pour chaque fichier
    - Nom du fichier
    - Date de création, dernière modification, dernier accès
    - Taille du fichier en octets
    - Nombre de liens vers le fichier
    - UID/GID du possesseur du fichier
    - Mode d'accès au fichier
      - Lecture pour le propriétaire, le groupe, n'importe qui
      - Ecriture pour le propriétaire, le groupe, n'importe qui
      - Exécution pour le propriétaire, le groupe, n'importe qui
    - Positions des données du fichier sur disque
    - Un répertoire est un fichier avec une structure spéciale

Appel système de manipulation du Système de fichiers

- Informations stockées dans le filesystem
  - Pour chaque fichier
    - Nom du fichier
    - Date de création, dernière modification, dernier accès
    - Taille du fichier en octets
    - Nombre de liens vers le fichier
    - UID/GID du possesseur du fichier
    - Mode d'accès au fichier
      - Lecture pour le propriétaire, le groupe, n'importe qui
      - Ecriture pour le propriétaire, le groupe, n'importe qui
      - Exécution pour le propriétaire, le groupe, n'importe qui
    - Positions des données du fichier sur disque
    - Un répertoire est un fichier avec une structure spéciale

Appel système de manipulation du Système de fichiers

- fd=open(file,params,...)
  - Ouverture du fichier file en écriture/lecture/...
    - Ce fichier sera ensuite associé au descripteur fd
- s=close(fd)
  - Fermeture du fichier référencé par le descripteur fd
- n=read(fd,buffer,nbytes)
  - Lecture de nbytes octets du fichier fd dans buffer
  - buffer doit exister et être alloué par malloc
  - retourne le nombre d'octets lus
- n=write(fd,buffer,nbytes)
  - Ecriture de nbytes octets de buffer dans fd
- position=lseek(fd,offset,...)
  - Déplacement dans le fichier référencé par fd

Appel système de manipulation du Système de fichiers

Processus Utilisateur

- Placer file et mode en mémoire accessible au kernel Appeler open

CPU en Mode protégé

Kernel

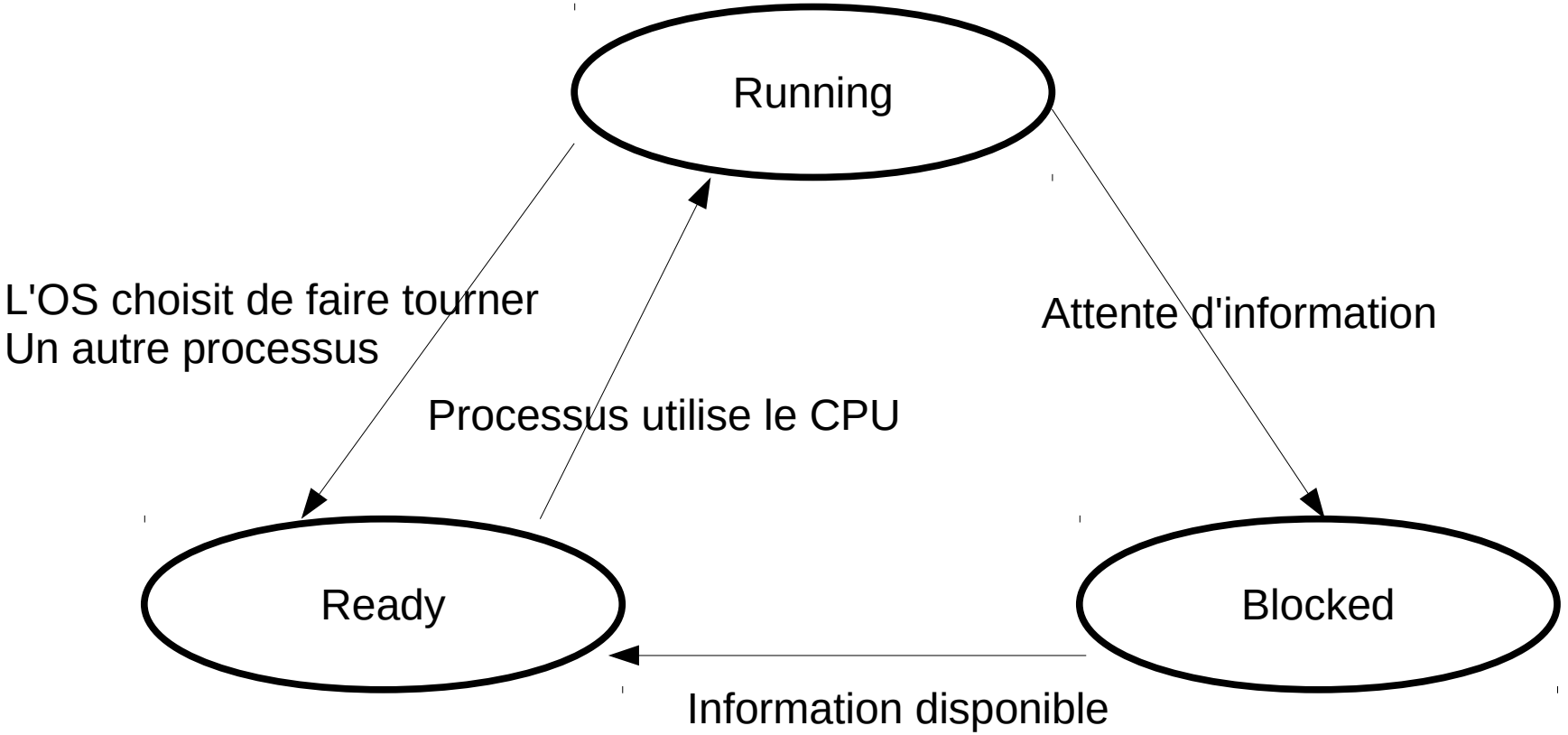
- Sauvegarde du contexte
- Récupérer file et mode
- Vérifier les droits d'accès
  - UID du processus
  - Permissions du fichier
- Prendre le premier fd libre
- Associer à fd open file object
  - Pointe vers file
  - offset=0
- Retour au processus
  - Placer fd à un endroit où
  - le processus peut le récupérer
  - Récupération contexte aller à l'adresse suivant open

Suite du code

Appel système de manipulation des processus

- pid=fork()
  - Crée un processus fils identique au processus père
    - Retourne pid=0 dans le fils
    - Retourne pid=id du processus fils dans le père
- s=execve(name,argv,envp)
  - Remplace le processus courant par l'exécutable name
  - avec arguments=argv et environnement=envp
- exit(status)
  - Fin du processus et retour de status au père
- pid=waitpid(p,&status,opts)
  - Attend la fin d'un processus fils et récupère status
  - Attente de n'importe quel fils si p=-1
  - Attente d'un fils dont le pid est p si p>0

Etats d'un processus



Running
Le procesus tourne sur un CPU
Ready
Le processus est prêt à tourner mais n'utilise pas de CPU pour le moment
Blocked
Le processus attend le résultat d'un appel système

Systèmes d'exploitation

Chapitre 1 : Généralités

Chargé de cours :
Emery Kouassi Assogba
Tél : 95 22 20 73
Emery.assogba@uac.bj
Emery.assogba@uclouvain.be

Objectifs du cours

Objectifs :

- Comparer différentes réalisations pour les systèmes d'exploitation et mettre en avant les avantages et inconvénients de ces réalisations
- Comprendre et expliquer ce que sont les principaux problèmes à résoudre par un système d'exploitation et présenter les différentes solutions qui y sont apportées avec leurs avantages et leurs inconvénients
- Comprendre et mettre en œuvre les mécanismes assurant la qualité des services
- Identifier les choix importants en matière de configuration et de gestion de systèmes, ainsi que les critères de décision pour effectuer ces choix.

Objectifs du cours

Objectifs

- Compréhension détaillée (théorique et pratique) du fonctionnement des systèmes d'exploitation
- Cas d'étude : Famille Unix
- Linux pour les travaux pratiques
- Principaux problèmes abordés
  - Processus et threads: concepts, problèmes et solutions
  - Communication entre processus
  - Gestion de la mémoire
  - Entrées-sorties
  - Systèmes de fichiers

Travaux pratiques

Exercices de base

- Petits programmes de base en C sous LINUX
  - A faire individuellement chaque semaine
  - S’inscrire sur www.inginious.uac.bj au cours Introduction à la programmation C
- Projet
  - Programme à faire par groupe de deux étudiants
  - Enoncé d’ici deux semaines
  - Projet à remettre en deux phases :
- Validation de l'architecture mi décembre
  - Rapport final et code source Linux mi-mars
  - Programmation kernel
  - Ajout d'une nouvelle fonctionnalité dans le kernel

Modalités d'évaluation

- Partie théorique
  - Examen oral à livre fermé portant sur toute la matière couverte au cours théorique et TPs
  - Pondération
    - 50% de la note finale pour l'examen oral
  - Travaux pratiques
    - Exercices individuels en C
    - Pénalités si les exercices ne sont pas rendus
  - Projet à faire par groupes de deux
    - 30% des points
  - Modification au kernel MINIX
    - 20% des points

Systèmes d'exploitation : bases

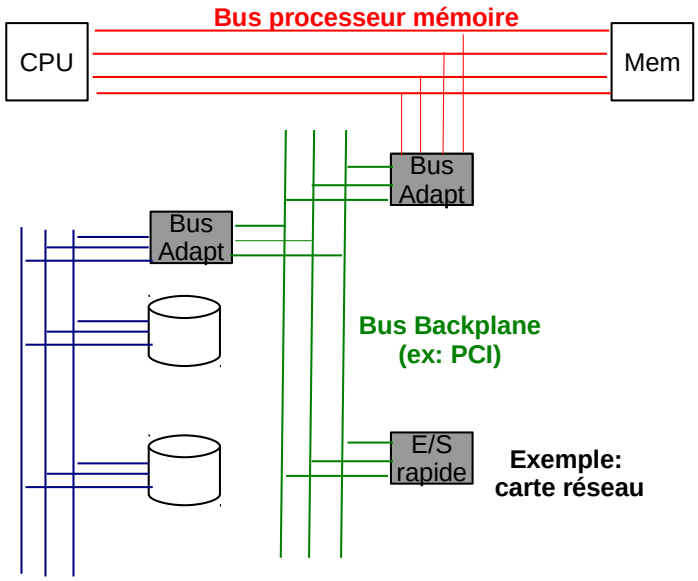
Fonctionnement des ordinateurs

- Microprocesseur
- Hiérarchie de mémoires
- Entrées/sorties

Le Système d'exploitation

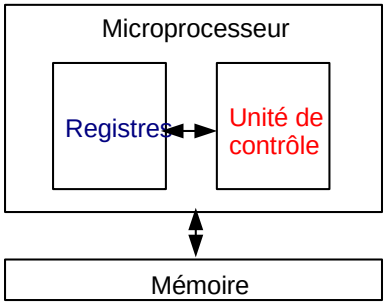
- Rôles
- Appels système
- Gestion des fichiers
- Gestion des processus

Organisation d'un ordinateur



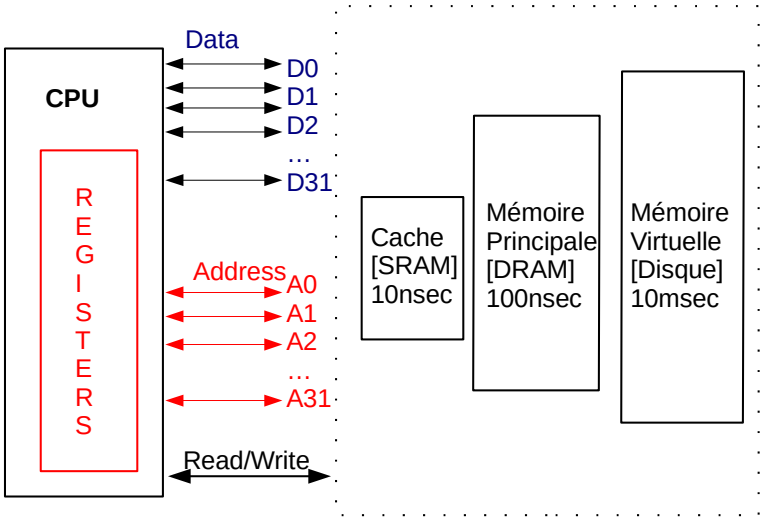
Microprocesseur

- Registres de calcul
- PC Program Counter
- SP Stack Pointer
- PSW Program Status Word



- Mode de fonctionnement
  - Mode kernel/protégé
    - Toutes les instructions sont utilisables
    - Le CPU peut accéder à toute la mémoire
  - Mode user
    - Les instructions de manipulation du matériel ne sont pas utilisables
    - Seule une partie de la mémoire est accessible

Hiérarchie de mémoires



- Objectif
  - L'accès aux différents niveaux de la hiérarchie doit
  - être le plus transparent possible pour le CPU

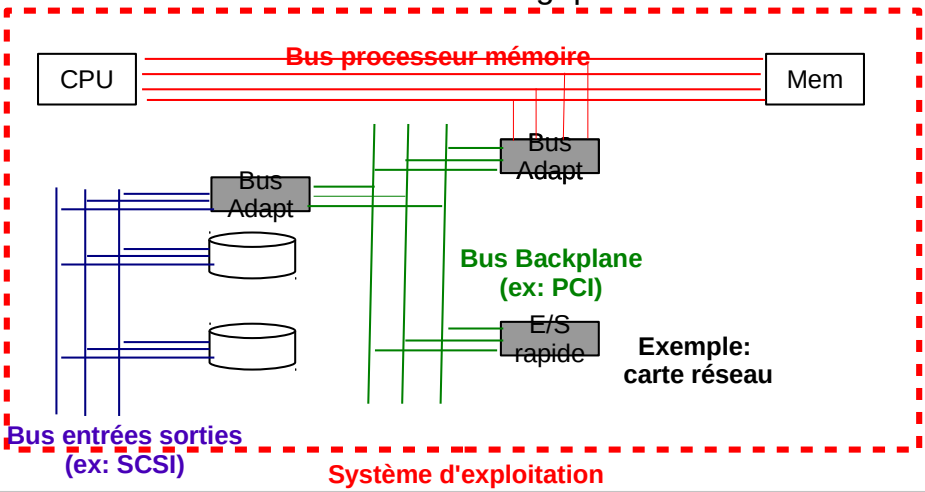
Entrées/Sorties

Dispositif	I/O	Utilisateur	Débit (KB/sec)
Clavier	I	Humain	0.01
Souris	I	Humain	0.02
Voix	I	Humain	0.02
Scanner	I	Humain	400
audio	O	Humain	0.6
Impr. laser	O	Humain	200
écran	O	Humain	60000
Modem	I/O	Machine	8
Réseau Lan	I/O	Machine	500-6000
floppy	stockage	Machine	100
Disque opt.	Stockage	Machine	1000
Disque dur	Stockage	Machine	2000-10000
Lect. Bandes	Stockage	Machine	2000-10000

- Nombreux dispositifs avec des caractéristiques souvent très différentes

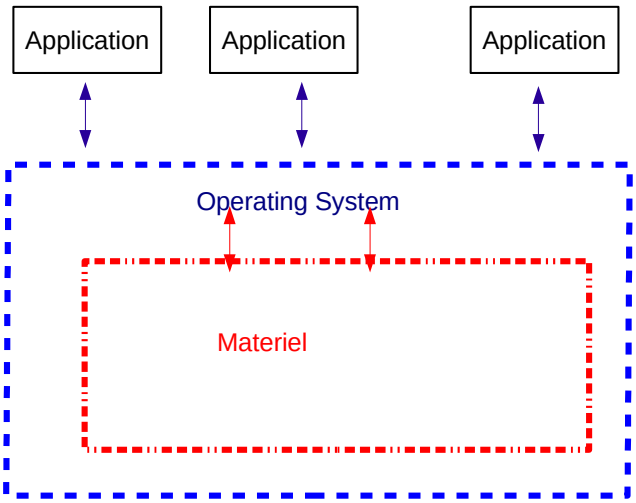
Rôle du système d'exploitation

- 1 : Gérer l'ensemble du matériel
  - Permettre aux applications d'y accéder sans en
  - connaître les détails technologiques



Rôle du système d'exploitation (2)

- 2 : Gestion des processus



Gestion des utilisateurs (2)

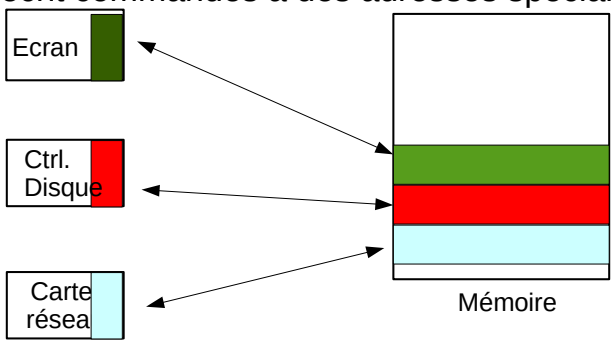
L'OS maintient une liste d'utilisateurs

- Utilisateurs !
- Administrateur (root sous Unix)

- Pour chaque ressource importante
  - Dispositif d'entrée sortie, fichiers, répertoires, ...
- Il faut définir quelles opérations chaque utilisateur/groupe d'utilisateurs peut effectuer
  - Lire
  - Ecrire
  - Exécuter
  - ....

Interactions OS->Matériel

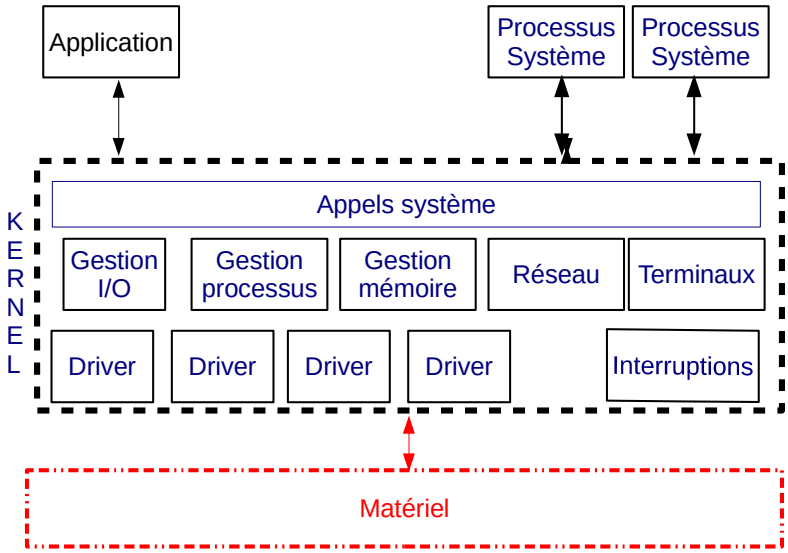
- Instructions spéciales
  - IN port value
  - OUT port value
  - Halt, break, ...
- Memory-mapped I/O
  - OS écrit commandes à des adresses spéciales



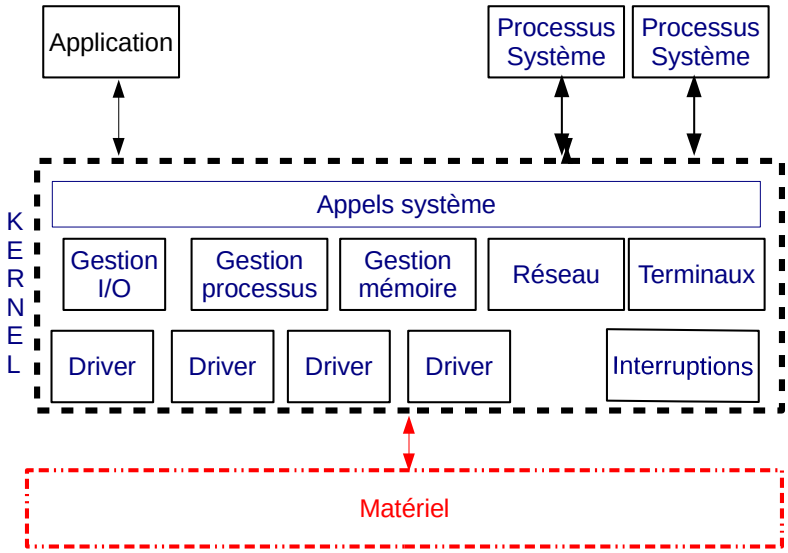
Interactions OS->Matériel

- Memory-mapped I/O
  - OS lit les réponses des dispositifs d'I/O à des adresses spéciales
- Interruptions
  - "Signal" produit par le processeur ou les dispositifs d'I/O
  - Division by zero
  - Segmentation fault
  - Data ready sur I/O
  - Horloge
  - ...
  - Provoque l'exécution de code placé par l'OS à une adresse spéciale
    - Interrupt vector

Organisation de l'OS

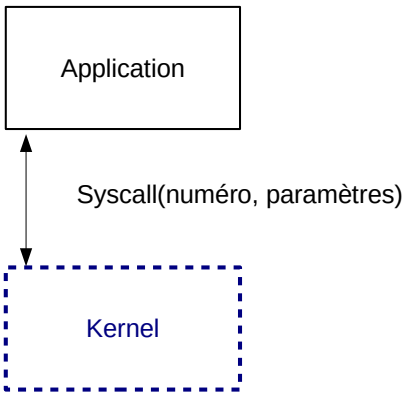


Organisation de l'OS



Appels système

- Interactions entre les applications et l'OS
  - Application spécifie
    - Identification de l'appel système
    - Paramètres éventuels
      - Registres ou en mémoire
  - Kernel analyse l'appel
    - CPU passe en mode kernel
    - Traitement des paramètres
    - Effectue l'appel système
  - Kernel
  - Kernel retourne à l'application en mode user
    - Résultats en registres ou en mémoire



L'appel système est la seule façon pour une application de faire exécuter (par le kernel) du code en mode protégé

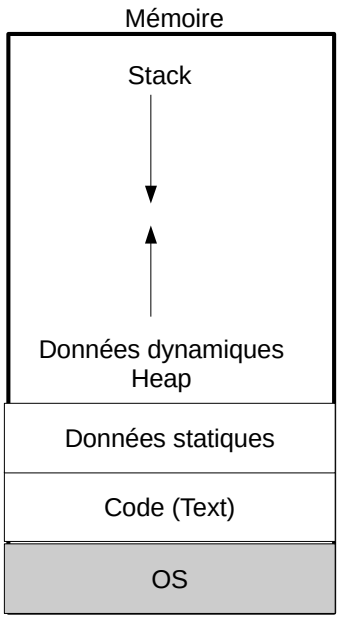
Appels système(2)

- Exemples
  - exit(status)
    Un processus a terminé et retourne la valeur status au processus qui l'a créé
  - sleep(sec)
    Le processus entre en hibernation sec secondes
  - fd=fopen(file, params, ...)
    Ouverture du fichier file qui sera identifié après par fd
  - n=read(fd, buffer, nbytes)
    Lit N octets dans le fichier identifié par fd
  - s=close(fd)
    Ferme le fichier identifié par fd
  - s=kill(pid,signal)
    Envoie un signal au processus pid

Les applications utilisent également des librairies où une fonction peut faire plusieurs appels système

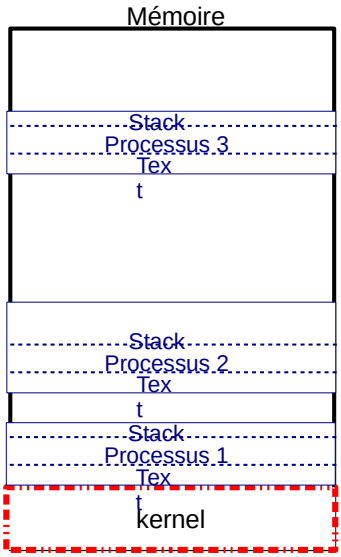
Organisation d'un processus

- Sur une machine simple monotâche



Organisation d'un processus

- Sur une machine multitâches...



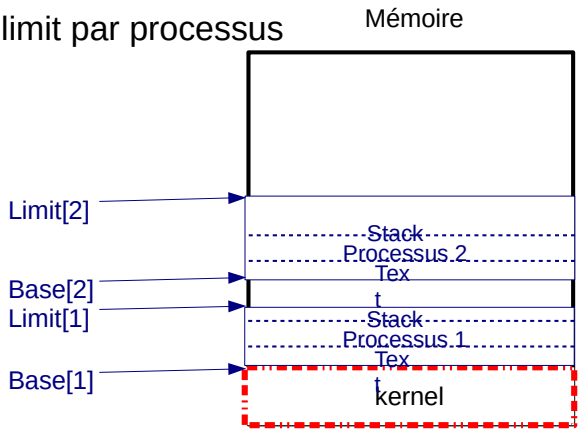
Gestion des processus

Problèmes à résoudre

- Partage de la mémoire entre l'OS et les processus utilisateurs ?
  - L'OS et les processus ont un stack, une partie text, ...
- Partage du CPU entre les processus et l'OS ?
  - Comment éviter qu'un processus qui boucle ne puisse bloquer l'OS et les autres processus ?
- Partage des accès aux dispositifs d'I/O ?
  - Accès exclusif à certains dispositifs
    - Ex: imprimante, graveur de CD, ...
  - Respect des permissions associées aux dispositifs

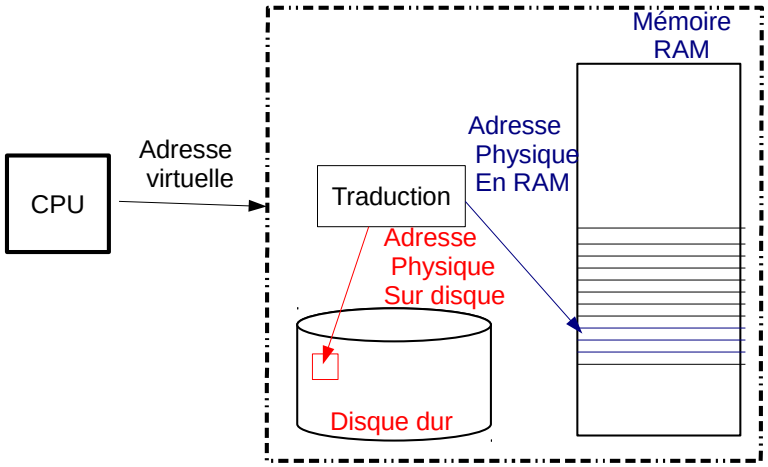
Partage de la mémoire

- Solution simple
  - Registres base et limit par processus



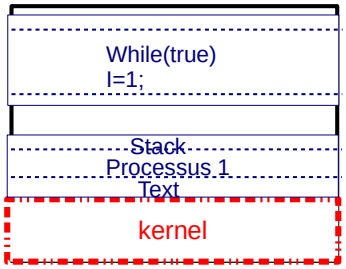
  - Solution actuelle
    - Mémoire virtuelle
    - Une table des pages pour chaque processus

Mémoire virtuelle



- Principe
  - l'information peut se trouver en RAM ou sur disque
  - le CPU y accède avec des adresses virtuelles

Partage du CPU

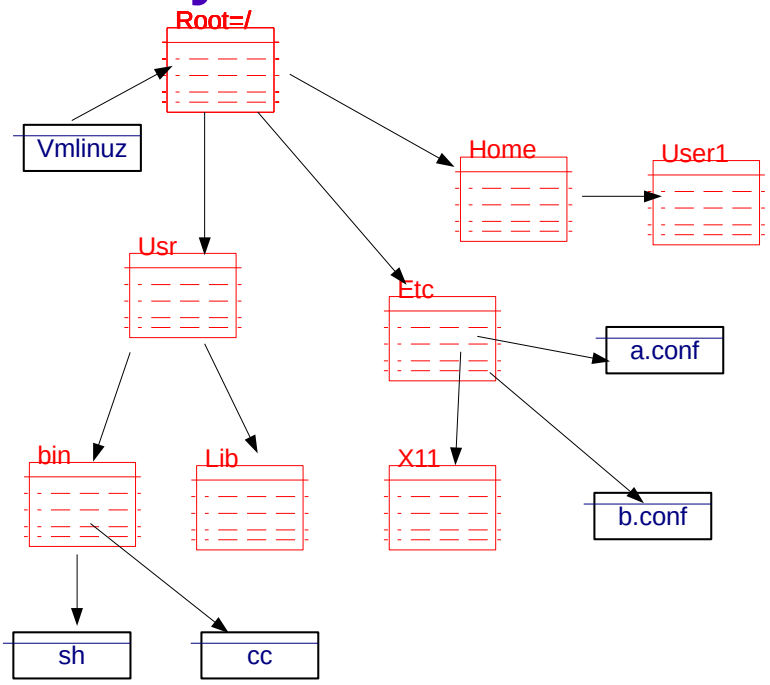


- Exécutions de code du kernel
  - Processus X effectue un appel système
  - Le kernel peut bloquer X durant l'exécution de l'appel et en profiter pour exécuter un autre processus
  - Hardware génère une interruption
    - Toutes les t millisecondes, interruption horloge
    - Le kernel peut en profiter pour arrêter le processus courant et exécuter un autre processus à la place
  - Conséquence
    - Un processus peut être interrompu à tout instant par le kernel !

Contexte d'un processus

- Ensemble des informations nécessaires pour poursuivre ultérieurement l'exécution du processus
  - Contenu des registres de données
  - Contenu du registre de sommet de pile
  - Contenu du Program Counter
- Passage du processus X au kernel
  - Kernel sauvegarde le contexte de X
- Passage du kernel au processus X
  - Kernel réinitialise les registres avec contexte de X

Système de fichiers



Contenu du Système de fichiers

- Informations stockées dans le filesystem
  - Pour chaque fichier
    - Nom du fichier
    - Date de création, dernière modification, dernier accès
    - Taille du fichier en octets
    - Nombre de liens vers le fichier
    - UID/GID du possesseur du fichier
    - Mode d'accès au fichier
      - Lecture pour le propriétaire, le groupe, n'importe qui
      - Ecriture pour le propriétaire, le groupe, n'importe qui
      - Exécution pour le propriétaire, le groupe, n'importe qui
    - Positions des données du fichier sur disque
    - Un répertoire est un fichier avec une structure spéciale

Appel système de manipulation du Système de fichiers

- Informations stockées dans le filesystem
  - Pour chaque fichier
    - Nom du fichier
    - Date de création, dernière modification, dernier accès
    - Taille du fichier en octets
    - Nombre de liens vers le fichier
    - UID/GID du possesseur du fichier
    - Mode d'accès au fichier
      - Lecture pour le propriétaire, le groupe, n'importe qui
      - Ecriture pour le propriétaire, le groupe, n'importe qui
      - Exécution pour le propriétaire, le groupe, n'importe qui
    - Positions des données du fichier sur disque
    - Un répertoire est un fichier avec une structure spéciale

Appel système de manipulation du Système de fichiers

- fd=open(file,params,...)
  - Ouverture du fichier file en écriture/lecture/...
    - Ce fichier sera ensuite associé au descripteur fd
- s=close(fd)
  - Fermeture du fichier référencé par le descripteur fd
- n=read(fd,buffer,nbytes)
  - Lecture de nbytes octets du fichier fd dans buffer
  - buffer doit exister et être alloué par malloc
  - retourne le nombre d'octets lus
- n=write(fd,buffer,nbytes)
  - Ecriture de nbytes octets de buffer dans fd
- position=lseek(fd,offset,...)
  - Déplacement dans le fichier référencé par fd

Appel système de manipulation du Système de fichiers

Processus Utilisateur

- Placer file et mode en mémoire accessible au kernel Appeler open

CPU en Mode protégé

Kernel

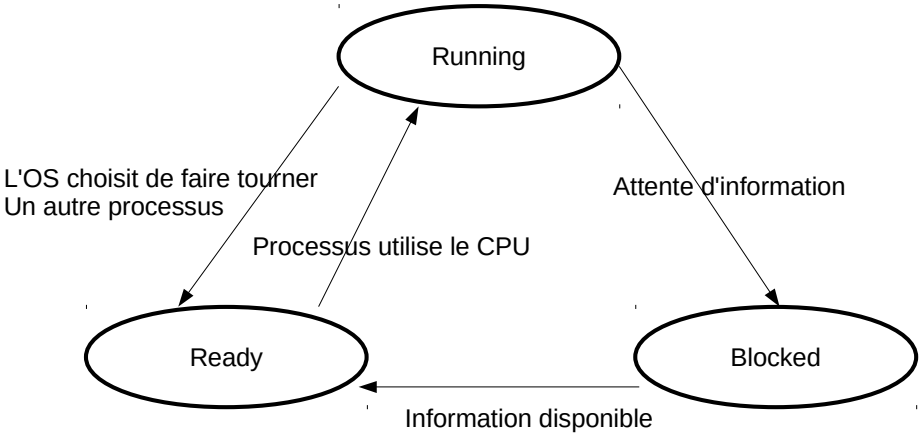
- Sauvegarde du contexte
- Récupérer file et mode
- Vérifier les droits d'accès
  - UID du processus
  - Permissions du fichier
- Prendre le premier fd libre
- Associer à fd open file object
  - Pointe vers file
  - offset=0
- Retour au processus
  - Placer fd à un endroit où
  - le processus peut le récupérer
  - Récupération contexte aller à l'adresse suivant open

Suite du code

Appel système de manipulation des processus

- pid=fork()
  - Crée un processus fils identique au processus père
    - Retourne pid=0 dans le fils
    - Retourne pid=id du processus fils dans le père
- s=execve(name,argv,envp)
  - Remplace le processus courant par l'exécutable name
  - avec arguments=argv et environnement=envp
- exit(status)
  - Fin du processus et retour de status au père
- pid=waitpid(p,&status,opts)
  - Attend la fin d'un processus fils et récupère status
  - Attente de n'importe quel fils si p=-1
  - Attente d'un fils dont le pid est p si p>0

Etats d'un processus



Running
Le procesus tourne sur un CPU
Ready
Le processus est prêt à tourner mais n'utilise pas de CPU pour le moment
Blocked
Le processus attend le résultat d'un appel système