

## Séance 9

# Gestion des entrées/sorties



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

# Rappels

- Abstraction **fichier** et organisation en répertoires
  - Opérations pour manipuler les fichiers et répertoires
  - Structures kernel organisant les fichiers
- Principes d'un **système de fichiers** et VFS
  - Structure et implémentation d'un système de fichiers
  - Méthodes d'allocation des fichiers sur le disque
- Le stockage de masse et la **structure d'un disque dur**
  - Algorithme d'ordonnancement du disque
  - Gestion de l'espace disque et formatage

# Objectifs

- Aspect **hardware** des entrées/sorties
  - Structure de bus
  - Contrôleur et driver de périphérique
  - Mécanisme d'interruption
- Aspect **software** des entrées/sorties

*Types d'entrées/sorties*
- **Sous-système** entrées/sorties du kernel

*Buffering, caching et spooling*

# Entrée/sortie (1)

- Contrôle des **périphériques** rattachés à la machine

*Une mission importante du système d'exploitation*

- Grande **variété** : souris, HDD, robot de bande magnétique...

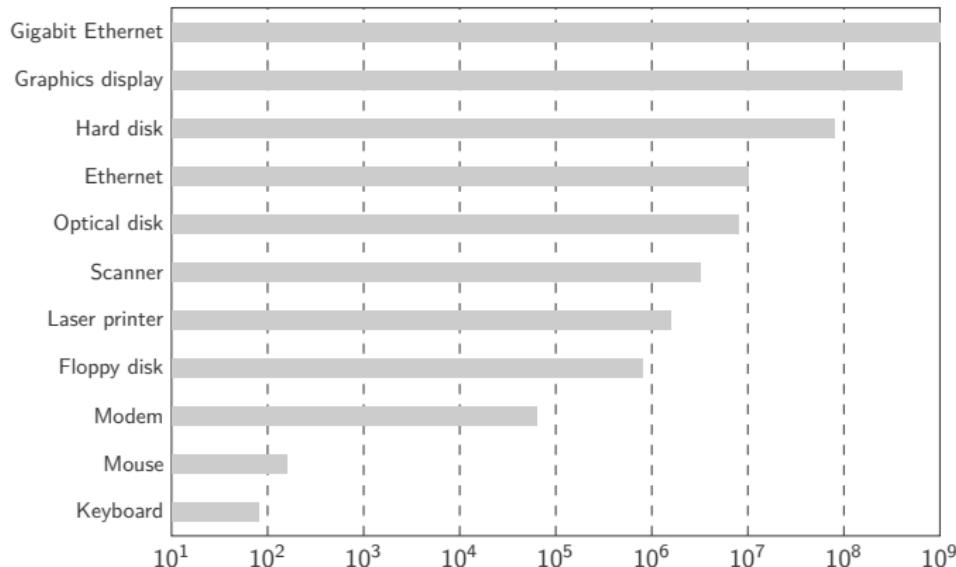
*Fonctions et vitesses très différentes*

- **Sous-système** d'entrée/sortie pour les gérer

- Méthodes spécifiques pour gérer et contrôler les E/S
- Séparation de la complexité de gestion E/S du reste du kernel

# Entrée/sortie (2)

- Taux de transfert en bps de différents périphériques E/S



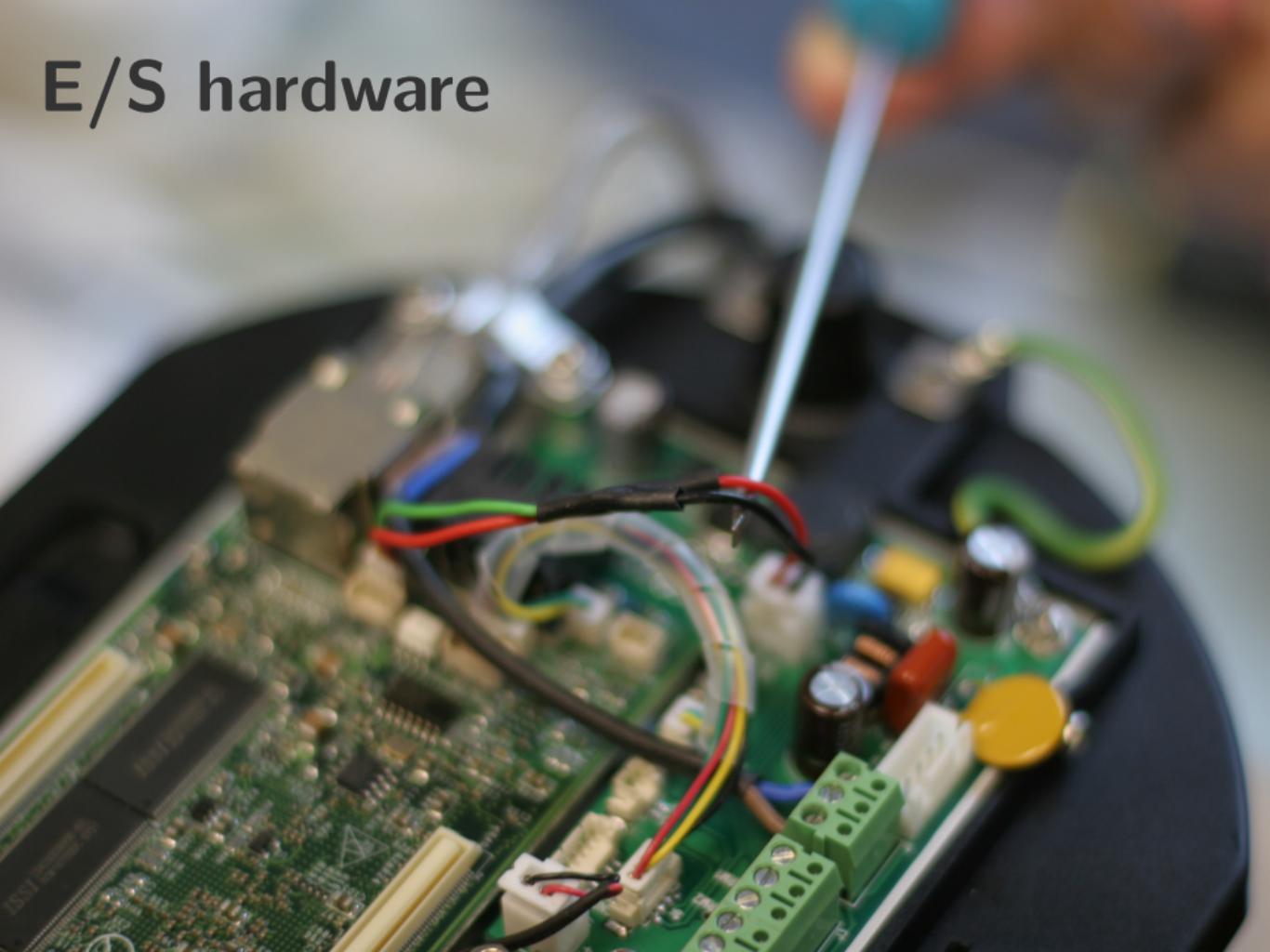
# Driver de périphérique

- Deux grandes **tendances** des E/S
  - Augmentation standardisation des interfaces software/hardware
  - Variété très large de dispositifs d'E/S
- Utilisation de **drivers de périphérique**

*Module utilisé par le kernel pour gérer le périphérique*
- Offre un **accès uniforme** vers le sous-système d'E/S

*Similaire aux appels système qui lient application et OS*

# E/S hardware



# Type d'E/S

- **Trois catégories** principales de périphériques d'E/S
  - **Stockage** : disque, bande magnétique...
  - **Transmission** : connexion réseau, Bluetooth...
  - **Interface humaine** : écran, clavier, souris, audio in/out...
- Existence de périphériques très **spécifiques**  
*Manche et pédales dans un avion*
- Quelques **principes clés** suffisent à comprendre  
*Comment le software peut prendre le contrôle du hardware*

# Communication

- **Envoi de signaux** du périphérique vers le système informatique

*Signaux transitent par un câble ou par l'air*

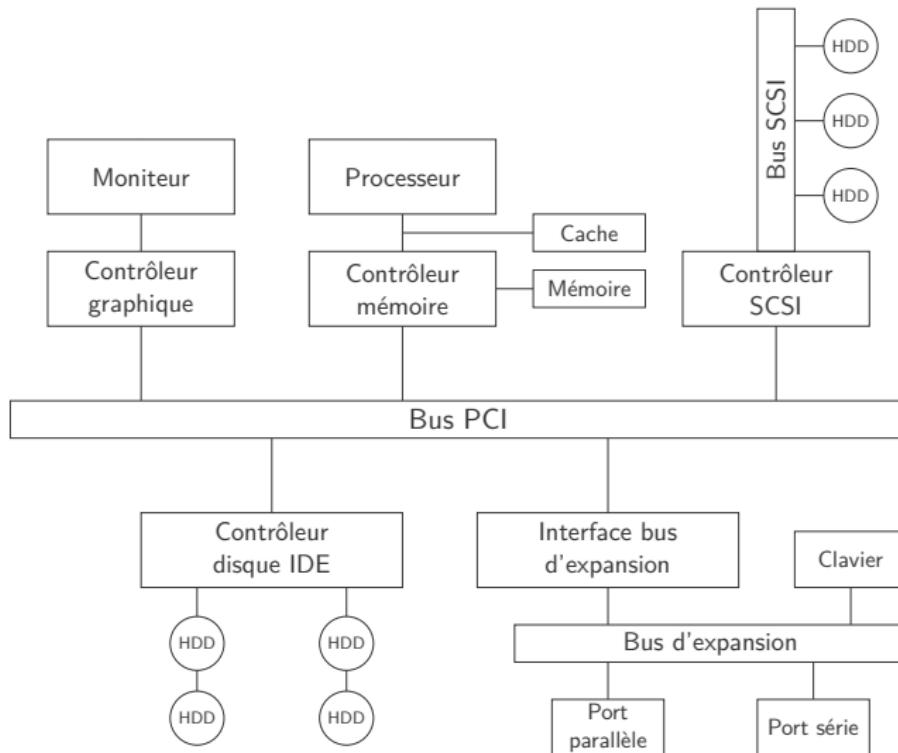
- Communication via un **point de connexion** appelé port

*Par exemple un port série, port parallèle ou bus*

- Un **bus** est un ensemble de fils avec un protocole

*Protocole rigide qui spécifie un ensemble de messages*

# Structure de bus (1)



# Structure de bus (2)

- Bus différent par signaux, vitesse, débit et connexion
- Plusieurs bus présents dans une architecture typique
  - PCI connecte processeur+mémoire aux périphériques rapides
  - Expansion connecte périphériques lents, série et USB
  - SCSI connecte des disques
- D'autres types de bus communs plus rapides

*PCIe jusqu'à 16 Go/s et HyperTransport jusqu'à 25 Go/s*

# Contrôleur (1)

- Collection d'électronique qui opère port, bus ou périphérique
  - Peuvent être de différents degrés de complexité*
- Deux exemples de contrôleurs
  - Port série : (portion de) puce qui contrôle signaux sur les fils
  - SCSI : carte électronique avec processeur, microcode, mémoire
- Contrôleur embarqué dans le périphérique
  - Par exemple implémentation de SCSI ou SATA sur un disque dur*

# Contrôleur (2)

- Contrôleur SATA sur un disque dur
- Exécution de **plusieurs tâches** à exécuter sur le disque
  - Mapping et gestion des secteurs défectueux
  - Prefetching des données, buffering et caching



# Communication processeur-contrôleur

- Plusieurs **registres** accessibles au processeur sur le contrôleur  
*Utilisables pour transférer des données et signaux de contrôle*
- Utilisation d'**instructions E/S** de transfert de données  
*Transfert d'octets ou mots vers/depuis une adresse d'un port E/S*
- **Registres E/S mappés** dans l'espace d'adresses du processeur  
*Instructions standards de lecture/écriture sur adresse mappée*

# Adresse des ports

- Contrôleur graphique utilise les deux techniques

*Ports E/S pour opération de base et mémoire pour contenu écran*

- Utilisation de mémoire protégée pour la sécurité

*Empêcher des écritures erronées dans une mémoire mappée*

| Intervalle d'adresses | Périphérique                       |
|-----------------------|------------------------------------|
| 000–00F               | Contrôleur DMA                     |
| 020–021               | Contrôleur d'interruption          |
| 040–043               | Timer                              |
| 200–20F               | Contrôleur de jeu                  |
| 2F8–2FF               | Port série (secondaire)            |
| 320–32F               | Contrôleur disque                  |
| 378–37F               | Port parallèle                     |
| 3D0–3DF               | Contrôleur graphique               |
| 3F0–3F7               | Contrôleur du lecteur de disquette |
| 3F8–3FF               | Port série ( primaire)             |

# Port d'E/S

- Port d'E/S typiquement composé de quatre registres
  - data-in lu par l'hôte pour obtenir une entrée
  - data-out écrit par l'hôte pour envoyer une sortie
  - status état du contrôleur (commande finie, données dispo...)
  - control lance une commande ou change le mode
- Options de configuration du périphérique avec control

*Port série full/half duplex, check parité, mots de 7 ou 8 bits...*
- Possibilité d'avoir une petite puce FIFO dans le contrôleur

*Pour ne pas être limité à un seul registre pour data-in/out*

# Polling (1)

- **Protocole de communication** entre l'hôte et le contrôleur

*Protocole complet compliqué, mais handshaking basique*

- **Coordination** de la relation producteur-consommateur

- Exemple avec deux bits pour faire la coordination
- Contrôleur indique son état avec bit `busy` du registre `status`
- Envie de l'hôte avec bit `command-ready` du registre `command`

- L'hôte se met dans un mode d'**attente active** (*polling*)

*Attente de la disponibilité du contrôleur*

# Polling (2)

- Différentes étapes du **handshaking**
  - 1 L'hôte lit de manière répétée le bit **busy** en attendant 0
  - 2 L'hôte met à 1 le bit **write** du registre **command** et écrit un octet dans le registre **data-out**
  - 3 L'hôte met à 1 le bit **command-ready**
  - 4 Le contrôleur met à 1 le bit **busy** lorsqu'il le remarque
  - 5 Le contrôleur lit le registre **command**, remarque le bit **write**, et écrit le contenu de **data-out** sur le périphérique
  - 6 Le contrôleur met à 0 le bit **command-ready**, le bit **error** du registre **status** et le bit **busy**

# Attente active

- L'attente active peut occuper l'hôte

*Pendant ce temps, il ne peut rien faire d'autre comme opérations*

- Utilisable si le contrôleur et le périphérique sont rapides

*Ainsi l'attente ne sera pas trop longues*

- Attention aux pertes de données si pas lues

- Très petit buffer du contrôleur d'un port série
- Par exemple flux de données d'un clavier ou souris

# Interruption (1)

- Préférable que le contrôleur **interrompe** le CPU

*Pour signaler que la requête a été traitée et données disponibles*

- **Ligne requête/interruption** au niveau du CPU

*Le CPU vérifie cette ligne après exécution de chaque instruction*

- Sauvegarde de l'état actuel pour **traitement de l'interruption**

*Exécution de la routine de gestion de l'interruption*

- Traitement d'un **très grand nombre** d'interruptions

*Centaine/s sur une machine, centaine de milliers/s sur serveur*

# Interruption (2)

- Exécution du **gestionnaire d'interruptions**
  - 1 Détermination de la cause de l'interruption
  - 2 Exécution du code nécessaire à gérer l'interruption
  - 3 Instruction « `return from interruption` » qui restaure le CPU
- **Rôle** des différents acteurs impliqués
  - Contrôleur génère une interruption (*raise*)
  - Le CPU capture l'interruption (*catch*) et l'envoie au gestionnaire (*dispatch*)
  - Le gestionnaire libère l'interruption (*clear*)

# Gestion avancée d'interruption

- Fonctionnalités avancées nécessaires pour système réel
  - Retarder la gestion d'une interruption lors d'exécution critique
  - Trouver directement qui a généré l'interruption  
*Sans devoir poller tous les périphériques*
  - Préciser des priorités entre les différents périphériques
- Gérée par le CPU et par un **contrôleur d'interruption hardware**
- **Deux lignes** de requêtes d'interruption
  - Ligne non masquable pour erreur mémoire grave...
  - Ligne masquable qui est cachée lors d'exécutions critiques

# Vecteur d'interruption (1)

- Vecteur avec les **adresses** des gestionnaires d'interruption

*Taille du vecteur limitée à un certain nombre de places*

- **Adresse** communiquée lors d'une interruption

*Correspond à l'indice de l'entrée du vecteur d'interruption*

- Vecteur d'interruption à **deux niveaux**

*Vecteur d'interruption pointe sur une liste de gestionnaires*

# Vecteur d'interruption (2)

## ■ Vecteur d'interruption sur les Intel Pentium

*Masquable de 0 à 31 et le reste non masquable*

| Indice du vecteur | Description                 |
|-------------------|-----------------------------|
| 0                 | Erreur de division          |
| ...               | ...                         |
| 6                 | Opcode invalide             |
| 7                 | Périphérique pas disponible |
| ...               | ...                         |
| 11                | Segment pas présent         |
| 12                | Faute de pile               |
| 13                | Protection générale         |
| 14                | Défaut de page              |
| ...               | ...                         |
| 19–31             | Réservé pour Intel          |
| 32–255            | Interruptions masquables    |

# Niveau de priorité

- Association d'un **niveau de priorité** aux interruptions

*Le CPU peut retarder l'exécution des basses priorités*

- Possibilité de **préemption** du CPU

*Une interruption de haute priorité peut dépasser une autre*

# Interaction avec les interruptions

- Scan des bus hardware au démarrage du système

*Installation des gestionnaires dans le vecteur d'interruption*

- Plusieurs sources d'interruption

- Périphérique signale qu'une demande a été traitée
- Gestion d'exceptions : division par 0, accès mémoire protégée...

- Demande d'exécution de routines du kernel

*Défaut de page, appels système...*

# Accès direct à la mémoire

- Interaction avec disques par **E/S programmée**

*Poller le statut, transférer les octets de données...*
- Utilisation d'un **processeur spécialisé** pour accès mémoire

*Contrôleur DMA (direct-memory-access)*
- **Configuration** du DMA par le CPU
  - 1 Pointeurs vers source et destination du transfert
  - 2 Nombre d'octets à transférer

# Vol de cycle

- CPU et DMA connectés à la mémoire par le **même bus**

*Vol de cycle pendant que le DMA accès à la mémoire*

- **Gain de temps** global du système est positif

*Malgré les vols de cycles*

- Le CPU peut continuer de travailler avec **ses caches**

*D'où l'importance d'y précharger des choses utiles*

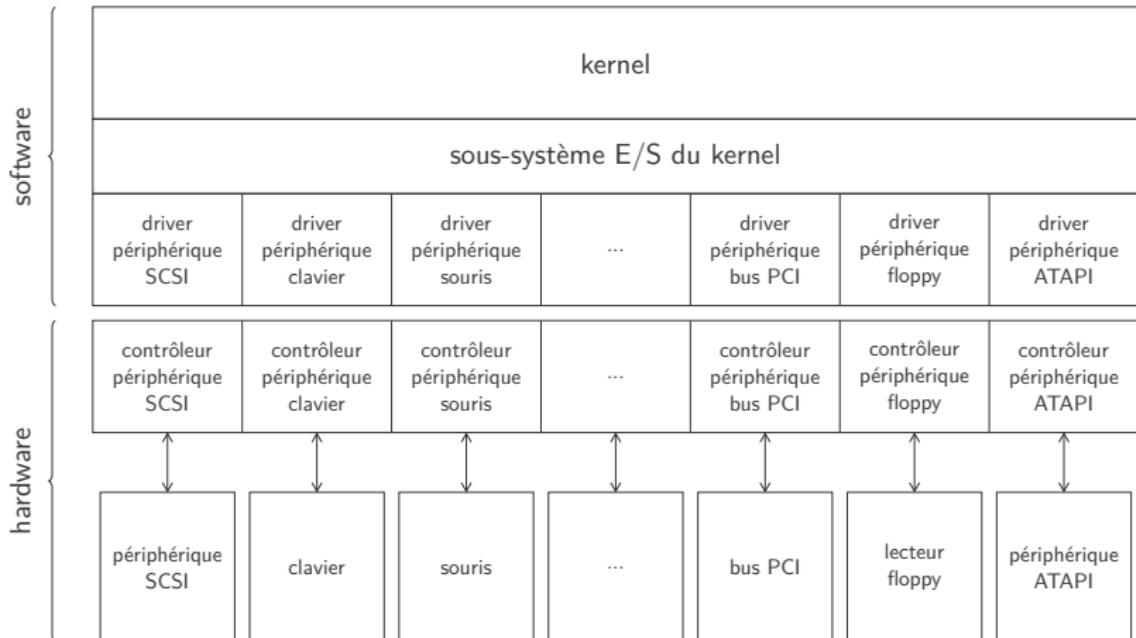
# E/S software



# Interface d'applications E/S

- Traitement **uniforme et standard** des périphériques E/S
  - Par exemple ouverture transparente d'un fichier*
- Identification des points communs d'une **classe de périphérique**
  - Établissement d'un ensemble standard de fonctions, une interface*
- Encapsulation des différences dans des **modules kernel**
  - Spécifique en interne, mais exporte l'interface*

# Structure E/S en kernel



# Caractéristiques des périphériques E/S

## ■ Nombreuses caractéristiques différentes des périphériques E/S

| Aspect            | Options  | Exemple                                  |
|-------------------|--|--|
| Mode de transfert | Caractère<br>Bloc  | Terminal<br>Disque                       |
| Méthode d'accès   | Séquentiel<br>Aléatoire  | Modem<br>CD-ROM                          |
| Transfert         | Synchrone<br>Asynchrone  | Banque magnétique<br>Clavier             |
| Partage           | Dédié<br>Partageable   | Bande magnétique<br>Clavier              |
| Vitesse           | Latence<br>Temps de recherche<br>Taux de transfert<br>Délai entre opérations |  |
| Direction E/S     | Lecture seule<br>Écriture seule<br>Lecture/écriture                          | CD-ROM<br>Contrôleur graphique<br>Disque |

# Back door

- Catégories de périphériques assez standard entre OS  
*E/S en blocs, en flux de caractères, mémoire mappée et en socket*
- Présence d'un **back door** dans la plupart des OS  
*Communication directe avec un périphérique*
- Sous Linux, existence de l'**appel système ioctl()**  
*Accède à une fonctionnalité spécifique à un driver*

# Périphérique par blocs/caractères

- Interface périphérique-bloc permet d'accéder aux disques

*Commandes de base comme read, write et seek*

- Application accède à travers un système de fichiers

*Isolation de l'application des détails de bas niveau*

- Accès mémoire comme une séquence linéaire de blocs

*Ce que fait l'OS et des applications spéciales comme les DBMS*

- Interface flux de caractères génère un flux de données

*Pratique pour clavier, souris, modem, imprimante, carte audio...*

# Périphérique réseau

- Interface E/S réseau différente des disques locaux

*La plus répandue est l'interface socket*

- Plusieurs opérations offertes par l'OS
  - Création d'un socket disponible localement
  - Attacher une adresse distante au socket local
  - Écouter les connexions entrantes sur un socket local
  - Envoyer et recevoir des données (paquets) sur la connexion

# Horloge et timer

- Horloge et timer hardware qui offrent trois fonctions de base
  - Obtenir le temps actuel
  - Obtenir un temps écoulé
  - Configurer un timer pour exécuter opération  $X$  au temps  $T$
- Pas de standardisation des appels systèmes
- Génération d'interruptions par les horloges et timers

*Plusieurs utilisations par le kernel et ses sous-systèmes*
- L'OS peut proposer des horloges virtuelles

*Gère une liste de demandes dispatchées sur les horloges hardware*

# E/S non bloquante et asynchrone

- Une **opération d'E/S bloque** l'application qui l'a initiée

*Le processus passe dans l'état WAITING, avant de revenir en READY*
- Possibilité de faire des opérations E/S **non bloquantes**

*Réception d'input souris/clavier tout en affichant infos à l'écran*
- **Deux techniques** utilisables pour du non bloquant
  - Plusieurs threads pour isoler opérations bloquantes
  - Appels systèmes asynchrones

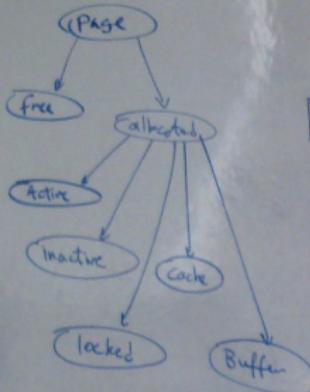
# E/S vectorisée

- Plusieurs opérations E/S en un seul appel système

*Par exemple, appel ready sous Unix*

- Plusieurs avantages aux E/S vectorisées

- Éviter surcharge changements de contexte et appels systèmes
- Possibilité de faire toutes les opérations en atomique



Application

Virtual Memory Subsystem

Physical memory (RAM)

2GB

Disk

2GB

Page out  
Page in

Swap out  
Swap in

inode table

# Sous-système E/S

# Ordonnancement E/S

- Déterminer l'**ordre dans lequel exécuter** des requêtes d'E/S

*Suivre l'ordre des appels systèmes est une mauvaise idée*

- Une **file d'attente** pour chaque périphérique
  - L'ordonnancement se fait par file
  - L'OS réarrange la file à chaque nouvelle requête qui rentre
- **Grosse mémoire** nécessaire pour les appels asynchrones
  - Retenir toutes les requêtes E/S asynchrones en cours
  - Stockage d'une table des états des périphériques

# Buffering

- Zone mémoire transitoire pour les données appelé **buffer**

*Pallie la différence de vitesse entre producteur et consommateur*

- Par exemple, stocker fichier qui vient d'un **modem sur disque**

- Buffer pour accumuler les données avant écriture disque
  - Double buffering pour permettre au modem de continuer

- Par exemple, en réseau pour gérer les **paquets fragmentés**

*Données n'arrivent pas forcément dans l'ordre d'envoi*

- Par exemple, pour faire une « **copy semantics** »

*Écriture d'un buffer vers le disque, mais si le contenu change ?*

# Caching

- Mémoire rapide pour stocker une **copie des données**

*Accès à la mémoire cache plus rapide que la mémoire originale*

- **Déférence** entre cache et buffer

*Buffer peut contenir unique copie de données, cache non*

- Possibilité d'avoir un **buffer cache**

*Données disque en buffer pour copy semantics, et comme cache*

# Spooling

- Un **spool** est un buffer de sorties pour un périphérique
  - Périphérique qui ne peut accepter flux de données entrelacés
  - Autoriser impression concurrente sur imprimante, par exemple
- L'OS **intercepte les demandes** et stocke dans un fichier disque
  - Spool de l'output de chaque application dans fichier séparé
  - Une fois le périphérique libre, envoi du fichier suivant

# Crédits

- <https://www.flickr.com/photos/shyb/2748309749>
- <https://www.flickr.com/photos/132889348@N07/27034807133>
- <https://www.flickr.com/photos/createdigitalmedia/2211729651>
- <https://www.flickr.com/photos/farrokhi/5820695338>