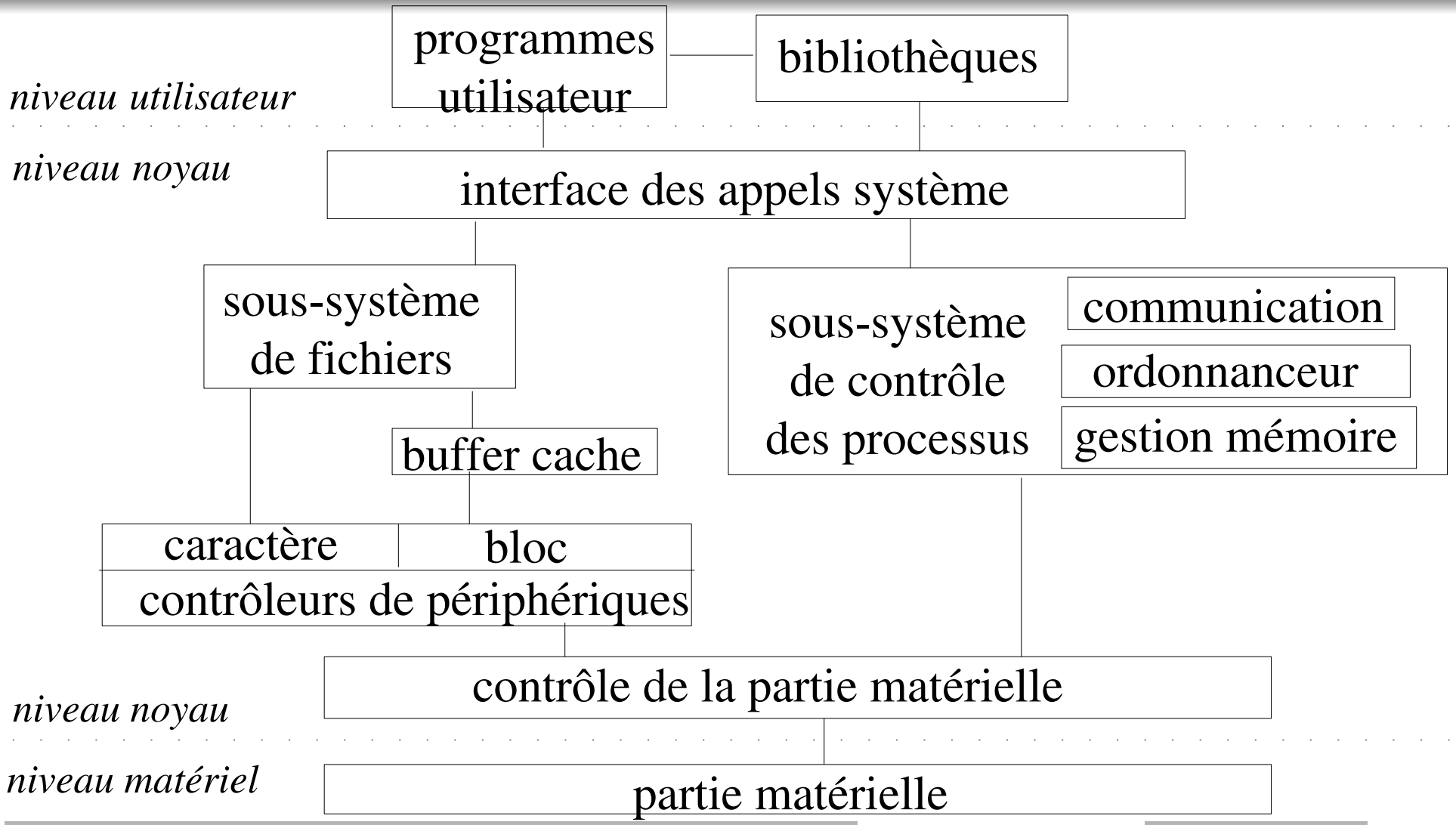
Les entrées / sorties sous Unix

Structure du noyau (1)



Structure du noyau (2)

- noyau = un ensemble d'appels système + algorithmes correspondants
- répond au nom d'un processus utilisateur ⇒ il est une partie de chaque processus utilisateur
- ⇒ réside en permanence en mémoire

Organisation générale des E/S sous Unix

- 2 types :
  - E/S en mode bloc (ou structuré) : correspond aux entrées sorties sur disque
  - E/S en mode caractère (non structuré) : correspond aux entrées/sorties sur terminaux
- opérations réalisées au travers du buffer cache

le buffer cache (1)

- but : minimiser la fréquence d'accès au disque par le noyau en gérant une réserve de tampons en données internes appelée buffer cache, qui contient les données des blocs disque les plus récemment utilisés.
- buffer cache = structure logicielle

≠ caches matériels pour accélérer les références à la mémoire

- lecture d'une donnée ⇒
  - lecture dans le cache
  - donnée présente ⇒ pas d'accès disque
  - sinon, accès disque et rangement dans le cache

le buffer cache (2)

- écriture d'une donnée ⇒
  - écriture dans le cache ⇒ donnée présente si relue plus tard
  - tente de minimiser les opérations d'écriture sur disque

- lecture séquentielle d'un fichier ⇒ anticipation sur lecture de bloc : 1 requête synchrone suivie d'une requête asynchrone

Avantages buffer cache (1)

- permet accès uniformes au disque (partie d'un fichier, d'un i-noeud, ...)

  ⇒ code plus modulaire (interface tous usages pour les E/S)

  ⇒ concepts du système plus simples

- réduction du trafic disque

  ⇒ ↗ capacités de traitement du système

  ⇒ ↘ temps réponse

- aide à la sécurité du système en assurant un accès sérialisé de plusieurs processus à un même bloc

Avantages buffer cache (2)

- remarque :
  - nombre tampons ↗ ⇒ trafic ↘
  - MAIS nombre tampons ↗⇒ mémoire disponible ↘⇒↗ swapping
- transmission d'une quantité faible de données ⇒ ↗ performances (mémorisation des données jusqu'à ce qu'il soit "économique" de les transmettre depuis ou vers le disque)

Inconvénients du buffer cache (1)

- panne fatale (crash) + écriture différée ⇒ l'utilisateur n'est jamais sûr qu'un appel système write a finalement conduit à une écriture sur disque
- tampon intermédiaire ⇒ copie de donnée intermédiaire
- transmission d'une quantité de données importante ⇒ ↘ performances

les appels système d'E/S (Rappels)

- ***ouverture d'un fichier :*** int open (char *ref, int mode)
- ***création d'un fichier :*** int creat(char *ref, int mode)
- ***lecture dans un fichier :*** int read(int desc, char *buf, int n)
- ***écriture dans un fichier :*** int write(int desc, char *buf, int n)
- ***fermeture de fichier :*** int close(int desc)

Autres appels système

- mknod : création d'un i-noeud
- stat, fstat : lecture d'un i-noeud
- lseek : déplacement du pointeur de fichier
- access : teste possibilité d'accéder à un fichier relativement à un mode donné
- link : création d'un lien (ie : une nouvelle référence pour un fichier)
- unlink : suppression d'un lien
- chdir : changement de catalogue de travail pour un processus
- etc ...

La Bibliothèque C standard (1)

- /lib/libc.a : le fichier d'archives correspondant
- pas d'instruction d'E/S en C $\Rightarrow$ appels système aux fonctions de la bibliothèque standard
- contient les fonctions permettant
  - les E/S bufferisées
  - manipulation des chaînes de caractères
  - conversion et classification de caractères
  - manipulation de temps
  - allocation mémoire
  - accès aux variables d'environnement

La Bibliothèque C standard (2)

- fournit une interface standard et portable entre Unix et d'autres systèmes d'exploitation qui supportent ces bibliothèques
- Les E/S standard : le coeur de la bibliothèque C standard
  - optimise la taille des données transmises entre les périphériques (disques et terminaux) et les programmes
  - manipulation des informations de la taille d'un bloc logique du disque ($\Rightarrow$ plus efficace)
  - $\Rightarrow$ en général plus efficace que les appels système
  - gestion par une structure de type FILE

La Bibliothèque C standard (3)

- structure FILE :
  - un descripteur de fichier
  - les opérations autorisées sur le fichier
  - un pointeur sur un buffer
- toute opération est effectuée par l'intermédiaire d'un pointeur sur une structure FILE
- accès aux E/S standard $\Rightarrow$ inclure le fichier *stdio.h*

  $\rightarrow$ accès aux macros, constantes et définitions nécessaires

le fichier stdio.h (1)

- constantes
  - BUFSIZ taille tampon en zone utilisateur
  - _NFILE nombre maximum de fichiers pouvant être ouverts
  - NULL pointeur Null
  - EOF caractère end-of-file
  - stdin entrée standard (descripteur 0)
  - stdout sortie standard (descripteur 1)
  - stderr sortie erreur standard (descripteur 2)

le fichier stdio.h (2)

- structures

```
struct iobuf
{  char *_ptr; /*pointeur courant dans le tampon*/
   int _ent;  /*nb car ds tampon après pos courante*/
  char *_base; /* pointeur sur tampon */
  char _flag;  /* précise le mode d'ouverture  */
  char _file;  /* num de desc  fich correspondant */
}
#define FILE struct iobuf
```

- structures de données : un tableau _iob de _NFILE éléments de type FILE

le fichier stdio.h (3)

- macro-définition de fonctions
  - ne peuvent pas être utilisées comme paramètres de fonctions (pas de point d'entrée)
  - ex : getch(p)

    getchar()

    putc(x, p)

    putchar(x)

    feof(p)

    ferror(p)

    fileno(p)

le fichier stdio.h (4)

- prédéfinition de types de fonctions
  - fonctions d'ouverture de fichiers
  - fonctions de lecture/écriture dans un fichier,
  - .....
  - fonctions de liaisons entre le tampon mémoire et le cache système
    - ✔ - fillbuf : remplit le tampon si besoin (_ent < 0)
    - ✔ - flushbuf : vide le tampon si besoin (_ent > BUFSIZ)

Ouverture d'un fichier

- ouverture du fichier au niveau système pour obtenir un descripteur et un cache
- réservation d'un emplacement dans le tableau _iob
- initialisation des différents champs dans cet emplacement
  - réservation dynamique d'un tampon
  - initialisation du tampon
  - etc...

Comparaison E/S standard et appels système (1)

```
#define N ....
main(){
        char buf[N];
        int f1, f2, n;
        f1 = open("fich", 0);
        f2 = creat("copie", 0666);
        while ((n=read(f1, buf), N)>0)
            write(f2, buf, n);
}
```

```
#include <stdio.h>


FILE *f1, *f2;
f1 = fopen("fich", "r");
f2 = fopen("copie", "w");
while ((fread(buf,1,N, f1)) != 0)
    fwrite(buf, 1, N, f2);
```

Comparaison E/S standard et appels système (2)

- Si N = 1024 : 1er programme légèrement plus efficace (appels système de plus bas niveau)
- Si N <<1024 : le 2ème est beaucoup plus efficace
- ex : N = 1 $\Rightarrow$ si fich contient p caractères
  - 1er programme :
    - ✔ p appels système read
    - ✔ p appels système write
  - 2ème programme :
    - ✔ (p/1024)+1 appels système read
    - ✔ (p/1024)+1 appels système write

Avantages / Inconvénients de la bibliothèque d'E/S (1)

- avantages :
  - ↘nombre d'appels au moniteur
  - ↗autonomie du processus et ↘ nombre d'interruptions
  - niveau plus élevé des fonctions (ex : E/S formatées)
  - portabilité
- inconvénients :
  - interruption des processus

Avantages / Inconvénients de la bibliothèque d'E/S (2)

```
main()  {
        int n;
        n = creat("toto", 0666);
        write(n "x", 1);
        for (;;);
}
```

```
#include <stdio.h>
    main() {
        FILE *f;
        f = fopen("toto, "w");
        putc('x', f);
        for (;;);
    }
```

- interruption des processus ⇒
  - - 1er programme : toto contient x
  - - 2ème programme : toto ne contient pas x (transfert entre tampon mémoire et cache non effectué)

Avantages / Inconvénients de la bibliothèque d'E/S (3)

- solution : capter les interruptions en réalisant des déroutements sur une fonction assurant le vidage des tampons avant de mettre fin au processus

☞ solution partielle :

- fflush(FILE *f) $\Rightarrow$ force le vidage du tampon associé
- rem : putc(c, f); fflush(f); $\Leftrightarrow$ write(fileno(f), &c, 1)