Principes des Systèmes d'Exploitation

Plan

- Fonctions d'un OS
- Machine Virtuelle / Langage
- Développement de Programmes
- Notion de Processus

Fonctions d'un OS

- Gestion de l'information
- Gestion des communications
- Contrôle d'exécution
- Gestion des composants matériels
- Partage des ressources
- Sécurité et protection

Gestion de l'Information

- Stocker programmes et données
  - Fichiers, bases de données
- Structuration de l'espace de stockage
  - Notions de répertoire, de partition
- Mécanismes de désignation des objets
  - Noms rémanents
  - Points de montage
- Mécanismes de contrôle d'accès aux objets

Gestion des Communications

- Interface Homme-Machine (IHM)
  - Gestion des interactions avec les utilisateurs
- Echanges de données entre programmes
  - Locaux
    - Mémoire partagée
    - Tubes de communication
  - Distants
    - Protocoles de communication

Contrôle d'exécution

- Démarrage/arrêt de programmes
- Enchaînement de programmes
  - Séquentiel, parallèle
- Aide à la mise au point de programme
  - Pose et traitement de points d'arrêt
  - Détection et notification d'exceptions
- Mesures
  - Du temps
  - Statistiques

Gestion du Matériel

- Gestion des périphériques
  - Identification des [types] de périphériques
  - Support du « plug-and-play »
- Mise en oeuvre des entrées/sorties
  - Drivers par types de périphériques
- Administration
  - Désignation des périphériques
  - Statistiques

Sécurité et Protection

- Identification des utilisateurs
- Mécanismes de contrôle d'accès
  - Mots de passe
  - Droits d'accès
- Support de communications
  - Confidentialité
  - Authentification
- Enregistrements d'événements

Classement des OS

- Dédiés / A usage général
- Interactifs (IHM) on non
- Enfouis
- Isolés / Communicants
- Observables / Invisibles

Propriétés des OS

- Portables
- Adaptables
- Ouverts
- Temps-réels
- Déterministes
- Tolérants aux pannes
- Distribués

Plan

- Fonctions d'un OS
- Machine Virtuelle / Langage
- Développement de Programmes
- Notion de Processus

OS = Machine Virtuelle

- Etendre interface machine physique
  - Fournir services de [plus] haut niveau
  - Indépendants du matériel sous-jacent
- Fournir contexte d'exécution de programmes
  - Mutuellement isolés les uns des autres
- Partager les ressources
  - Entre utilisateurs
  - Par extension, entre programmes concurrents

Niveaux Abstraction

- Langage de commandes (« shell » de Unix)
  - Fichiers de données
  - Fichiers de programmes (binaires, shell scripts)
- Langage de programmation
  - Environnement défini par le langage
  - Peut inclure extensions de haut-niveau (objets/fonctions répartis)
  - Mis en oeuvre par interpréteur ou par bibliothèques
- Interface système d'exploitation
  - Ensemble des appels système + objets associés

Niveaux Abstraction

Langage de Commandes (Shell) + utilitaires

Environnement de Développement
Langages + Bibliothèques

ABI

Machine physique

Shell et langage de commande

- Programme interactif de lancement de programmes
  - <nom_du_programme> [paramètres]

```
cp file1 file2
```

- Par défaut, attend fin commande courante
- Commande lancée en "background"

```
server_prog arg1 arg2 &
```

- Notion de "shell script"
  - Fichier contenant des commandes shell
  - Lancé comme une commande

```
sh script.sh
```

Shell et langage de commande

- Intègre langage de programmation
- Variables
  - <nom_de_variable>=valeur

    `TERM=xterm`
  - `PATH=/bin:/usr/bin/`

    Liste de répertoires contenant programmes
    Parcourue dans ordre d'apparition
- Directives de programmation
  - `if, else, elif, for`
- Notion de fonction

Shell – entrées/sorties standard

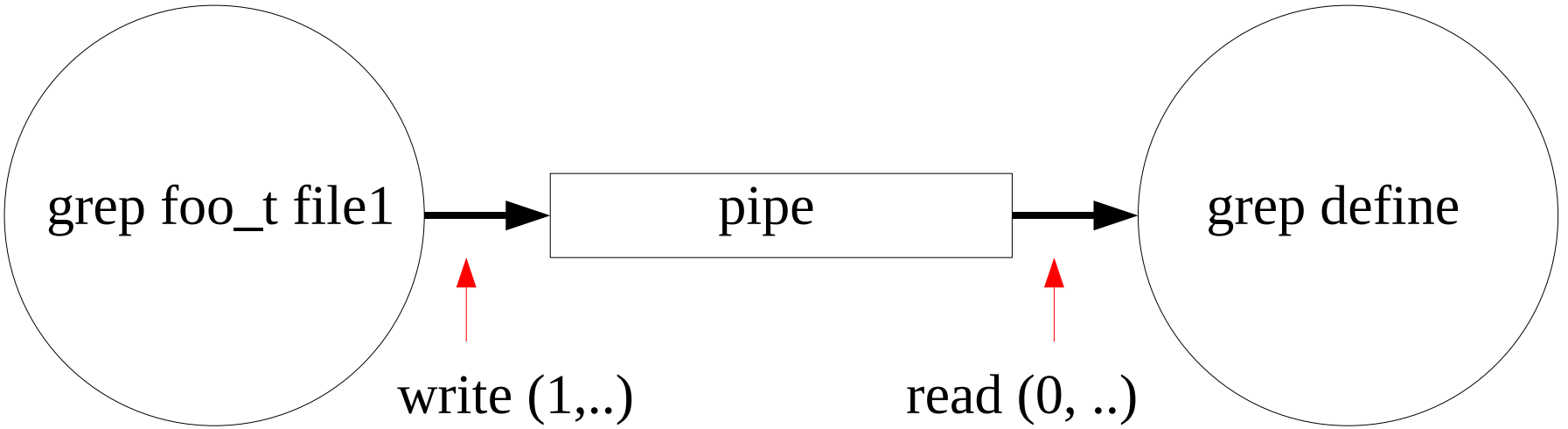
- Notion d'entrée (0) et de sortie (1) standard
  - Défaut : 0 = clavier, 1 = écran
  - Peuvent être redirigées sur un fichier

```
grep "foo" file1 > foo_in_file1
```

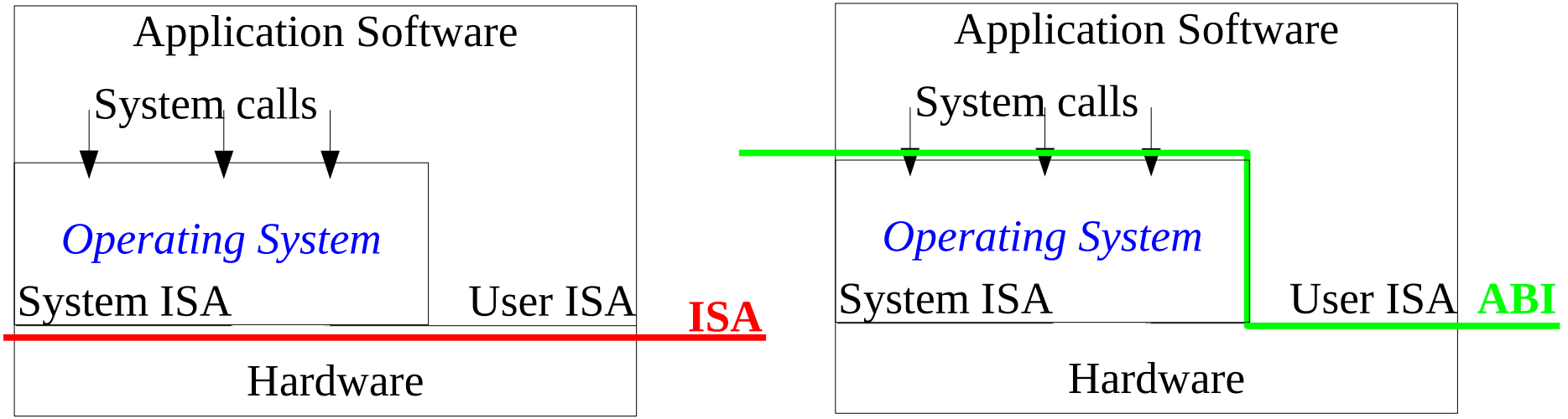
- Notion de *tube* (*pipe*) pour lancer des suites de commandes se comportant commes des filtres
  - Modèle du producteur / consommateur
  - Producteur : sortie standard redirigée sur entrée pipe
  - Consommateur : entrée standard redirigée sur sortie pipe

Exemple de pipe-line

```
grep "foo_t" file1 | grep "define"
```



Interfaces Machine



- ISA (**I**nstruction **S**et **A**rchitecture)
  - Interface de machine physique
- ABI (**A**pplication **B**inary **I**nterface)
  - Interface de machine virtuelle système

Interfaces Machine

- ISA
  - Instructions CPU : neutres + privilégiées
  - Par extension, inclut périphériques
  - Indépendante des OS
- ABI = Interface d'un OS sur un CPU
  - Instructions CPU neutres seulement
  - Ensemble des appels système

Plan

- Fonctions d'un OS
- Machine Virtuelle / Langage
- Développement de Programmes
- Notion de Processus

Développment de Programmes

- Conception modulaire - fichiers source séparés
  - Unités de compilation indépendantes
  - Structuration par programmation objet
  - Parties dépendantes OS / CPU
  - Fonctions communes
- Interfaces décrits dans « include files »
  - Définitions types de base, constantes, etc
  - Structure et interface des objets
  - Fonctions exportées par les modules

Environnement de Développement

- Compilation : fichier source -> fichier objet
- Bibliothèques
  - Fonctions mises en commun
  - Statiques ou partagées à l'exécution
- Edition de liens
  - Combinaison de fichiers objets + bibliothèques
  - Statique ou Dynamique
- Gestion des dépendances – makefiles

Compilation de Programmes

- Compilateur/Assembleur
  - Fichier source -> fichier objet
  - Instructions et données au format CPU
- Calcule et produit Méta-données
  - Informations de relogement en mémoire
  - Références données/fonctions importées (dites non résolues)
  - Références données/fonctions exportées
  - Informations de mise au point (optionnelles)

Compilation Croisée

- Machine hôte : compilation des programmes
- Machine cible : exécution des programmes
- Compilation croisée
  - Machine hôte ≠ Machine cible (CPU/OS)
  - Bibliothèques + include files de l'OS cible
- Compilation Croisée Canadienne
  - Compilation croisée appliquée au compilateur

Fichier Objet

- Binary File Formats (BFF)
  - exe (Windows) / a.out – coff – elf (Unix, Linux)
- Header: *magic* + localisation tables, sections
- Table de relogement
- Table des symboles
- Sections
  - Section *text* : instructions
  - Section *data* : données initialisées
  - Section *bss* : données non initialisées

Formats de Fichiers Objets

- Fichier objet relogeable
  - Format {Code + données} pouvant être combiné avec d'autres relogeables pour créer un exécutable
- Fichier objet exécutable
  - Format {Code et données} pouvant être chargé en mémoire et exécuté
- Fichier objet partageable
  - Format {Code et données} pouvant être chargé en mémoire et partagé entre exécutables

Bibliothèques

- Regroupement de fonctions mises en commun
- Services génériques
  - manipulations de chaînes de caractères
- Fonctions spécifiques
  - mathématiques, graphiques
- Invocation de services externes
  - Appels système
  - Invocation de serveurs distants

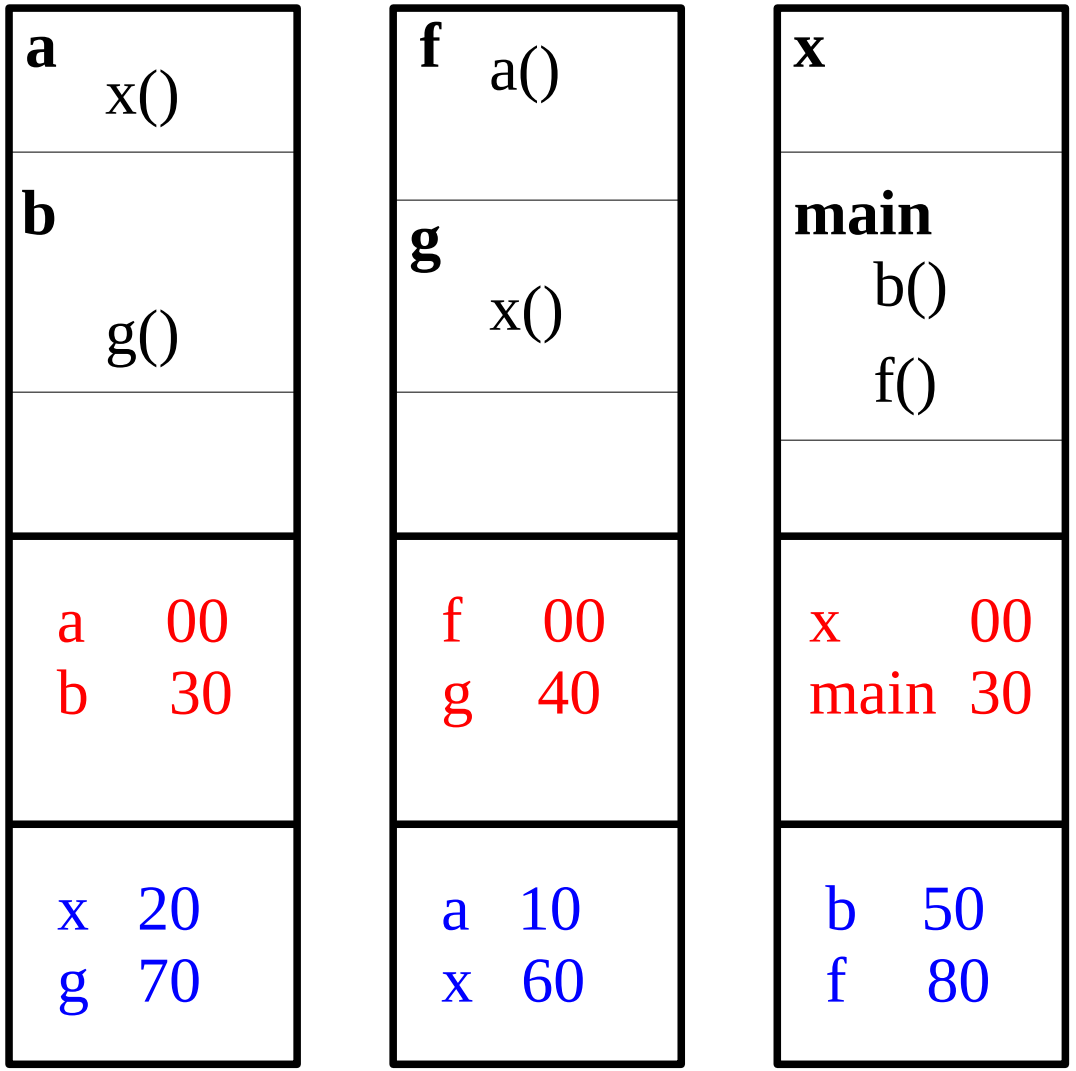
Bibliothèques - suite

- Bibliothèque statique
  - Dupliquée dans chaque programme exécutable qui l'utilise
- Bibliothèque partagée
  - code partagé par tous les programmes
  - copie privée des données par programme exécuté
- Pré-chargée avec adresses mémoires fixes
- Chargée dynamiquement

Edition de Liens Statique

- Ensemble fichiers relogeables + bibliothèques -> binaire exécutable
- Résoud références non résolues
  - D'abord entre fichiers relogeables
  - Puis avec ensemble de bibliothèques
- Regroupe et reloge données / fonctions
  - Regroupe et reloge sections text, data
  - Accumule et reloge section bss
  - Text & Data cadrées sur frontières page

Edition de Liens Statique



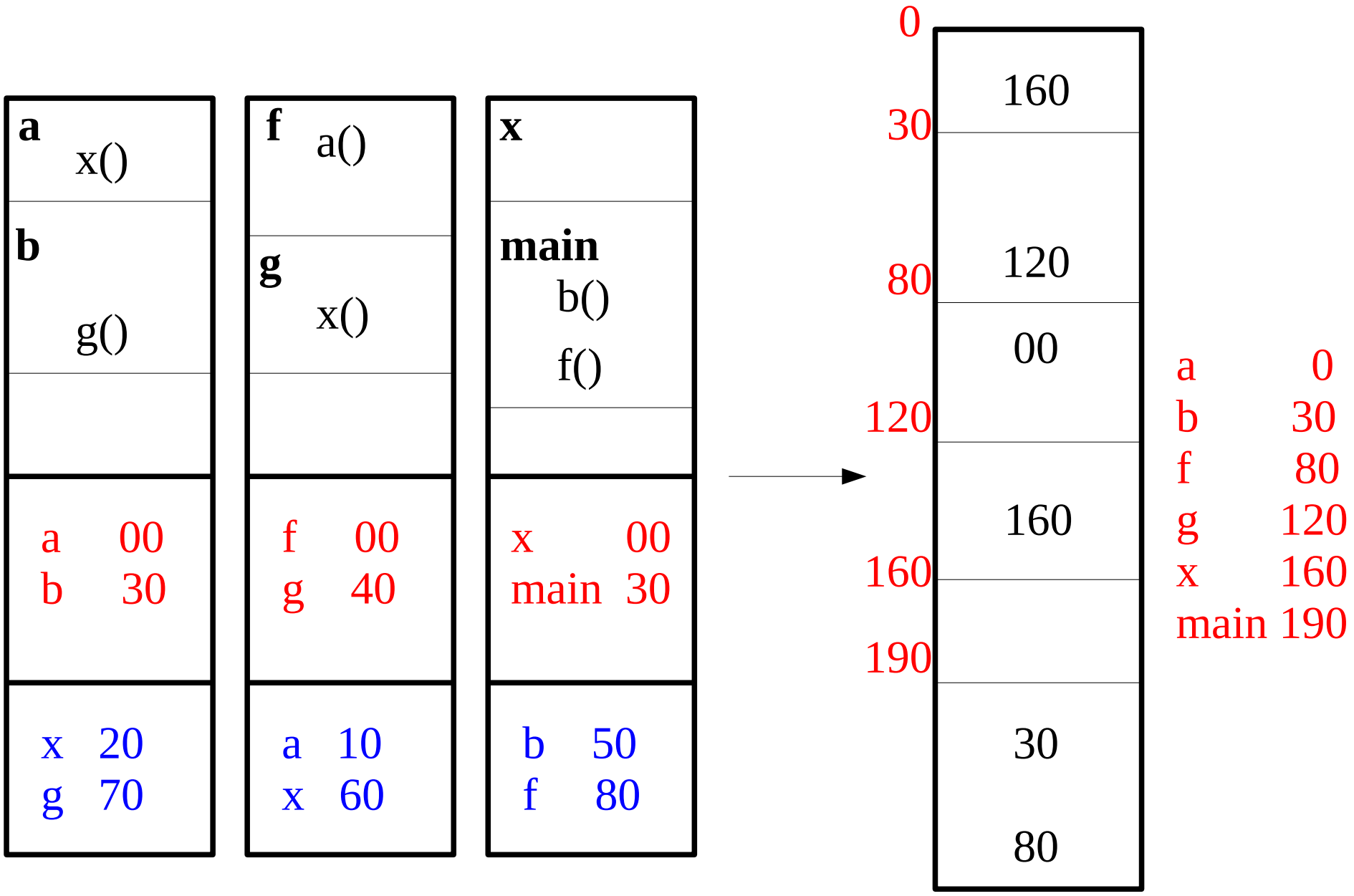
a 00
b 30

Symboles globaux définis

b 50
f 80

Références non résolues

Edition de Liens Statique



Edition de Liens Dynamique

- Bibliothèques partagées
  - Par liées avec un exécutable à l'édition de liens
  - Chargées en mémoire dynamiquement
  - Evite de dupliquer en mémoire code fonctions des bibliothèques partagées
- Références externes des fonctions résolues
  - Au chargement d'un programme
    - Windows .DLL (**D**ynamic **L**ink **L**ibrary)
  - Durant l'exécution, lors du premier appel
    - Linux .so (**S**hared **O**bject)

Edition de Liens Dynamique

- Fichiers .so sur Linux / GCC
- Compilés en PIC (**P**osition **I**ndependant **C**ode) exécutables à n'importe quelle adresse
- Accès indirect aux variables globales par GOT (**G**lobal **O**ffset **T**able) initialisée au chargement
- Lie dynamiquement fonctions partagées par table d'indirection : **P**rocedure **L**inkage **T**able (PLT)

Edition de Liens Dynamique (2)

- Editeur de liens dynamique Linux (*ld.so*)
  - Exécuté au chargement du programme
  - Partagé => résoud d'abord ses propres adresses
- OK sur Intel, ARM : MMU, adressage indirect
- Limitations sur autres processeurs
  - PowerPC : adressage relatif - offset sur 24 bits
  - TI DSPc6x : pas de MMU

Utilitaire « make » et makefile

- Définir règles de dépendances entre fichiers
  - Sources / include files
  - Objets / sources
  - Binaires / objets
- Associer commandes à exécuter
- Après modification d'un fichier **F**
  - Déterminer ensemble des fichiers dépendants de **F**
  - Effectuer opérations associées aux règles
- Comparaison date de modification des fichiers

Plan

- Fonctions d'un OS
- Machine Virtuelle / Langage
- Développement de Programmes
- Notion de Processus

Notion de Processus

- Historiquement, concept introduit pour la mise en oeuvre de programmes concurrents
  - partage (transparent) d'une machine entre utilisateurs
  - programmes séquentiels exécutés en parallèle
- Contexte d'exécution d'un programme
  - Contexte processeur (valeur des registres CPU)
  - Contexte mémoire (espace d'adressage virtuel)
  - Attributs (identificateur, priorité, droits)
  - Ressources allouées par OS

Processus - suite

- Un processus par utilisateur
  - Nombre maximum connu
  - Gestion ressources simple
  - IBM/VM, VAX/VMS
- Systèmes famille Unix (BSD, Solaris, Linux)
  - Plusieurs processus par utilisateur
  - Apparentés selon structure arborescente
  - Création/terminaison par appels systèmes

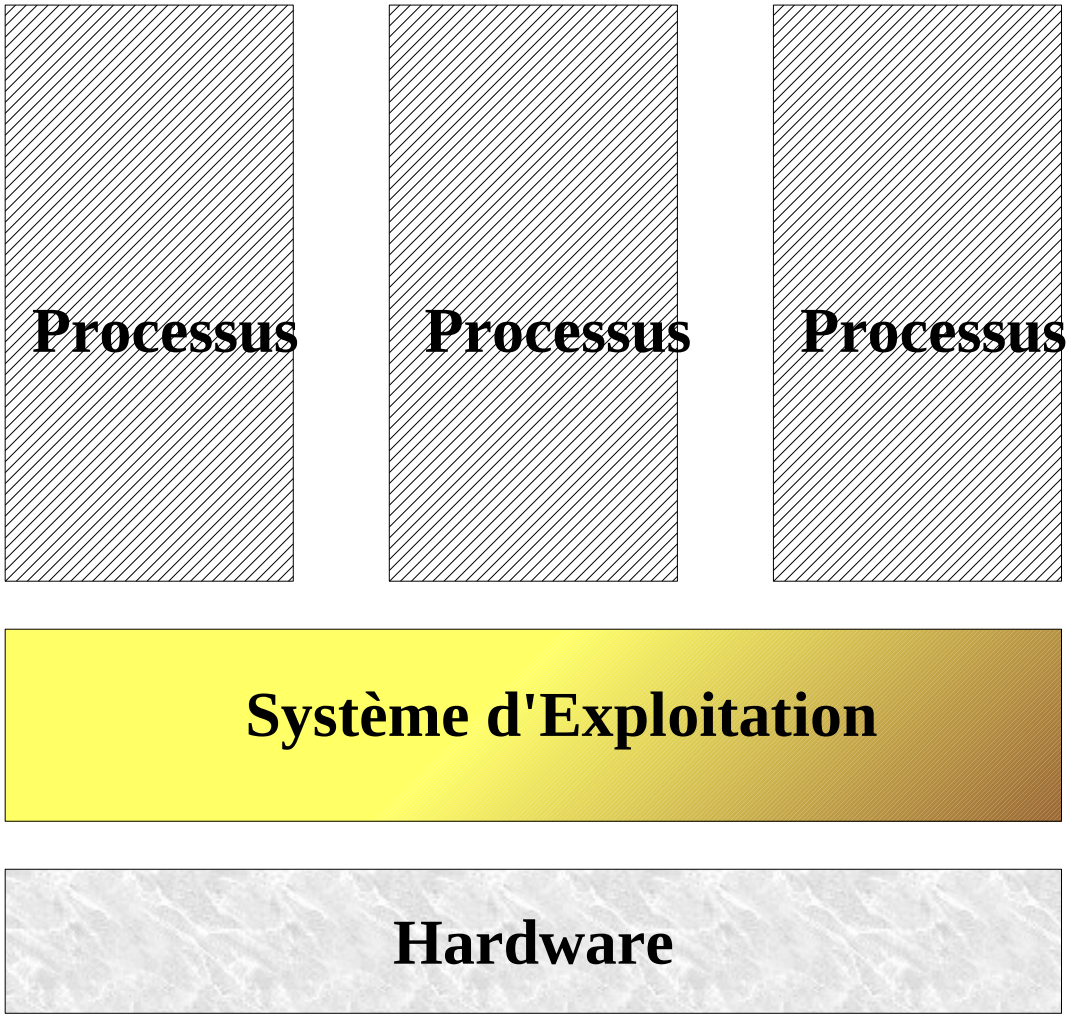
Processus - suite

- Entité d'allocation/répartition d'un CPU
  - Politiques de scheduling
  - Points de préemption du système
- Contexte d'attribution des ressources
  - Espace mémoire
  - Espace disque
  - Objets temporaires du système
    - Buffers d'I/O

Processus - suite

- Entité dynamique
  - entité active opérant sur son environnement en exécutant un programme
- Etat courant de la machine virtuelle exécutant le programme associé au processus
  - Inclut état [d'une partie] de la machine physique
  - Etat mémoire (adresses, tailles)
  - Etat vis à vis du système (stoppé, etc.)

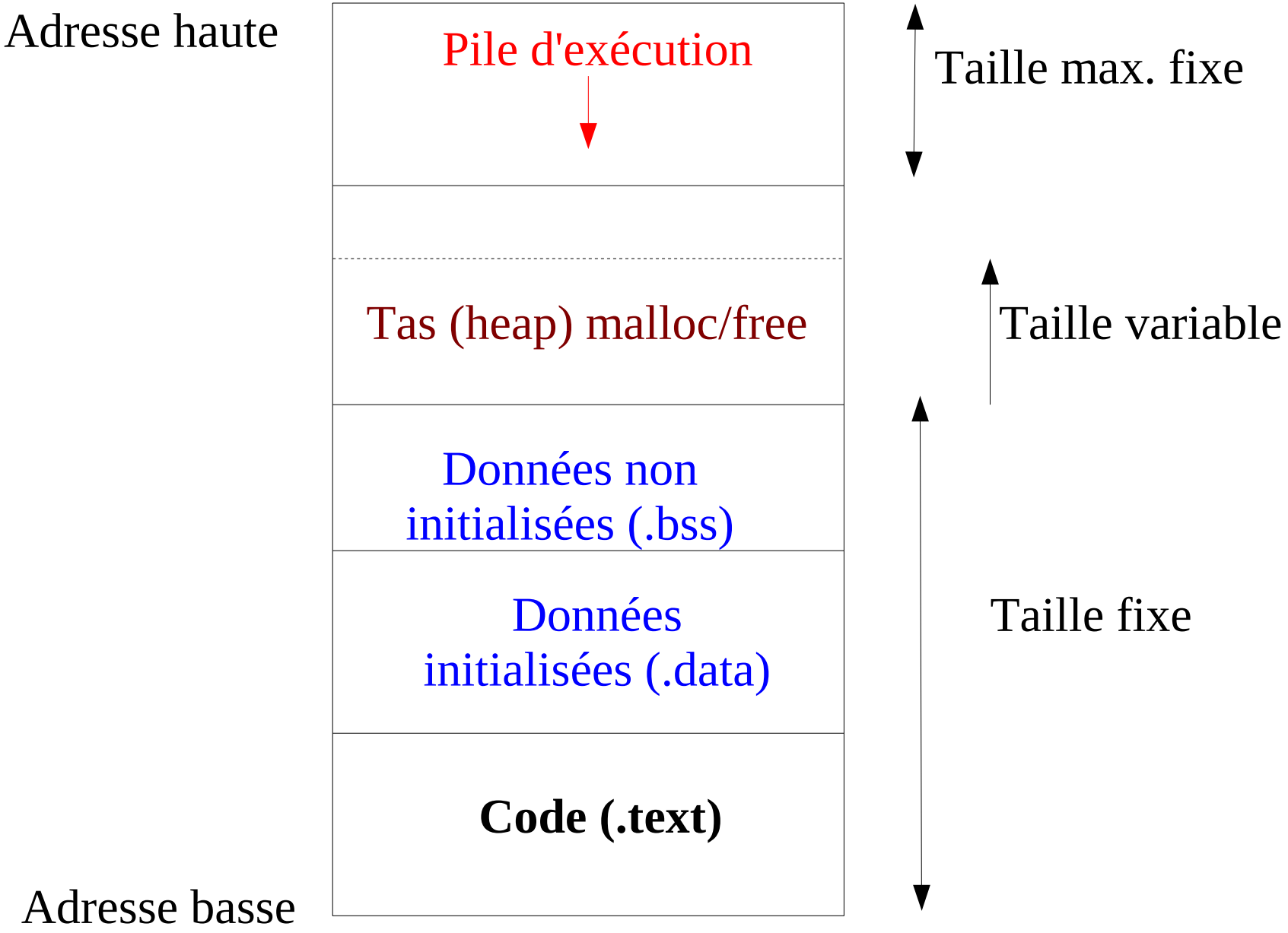
Processus - suite



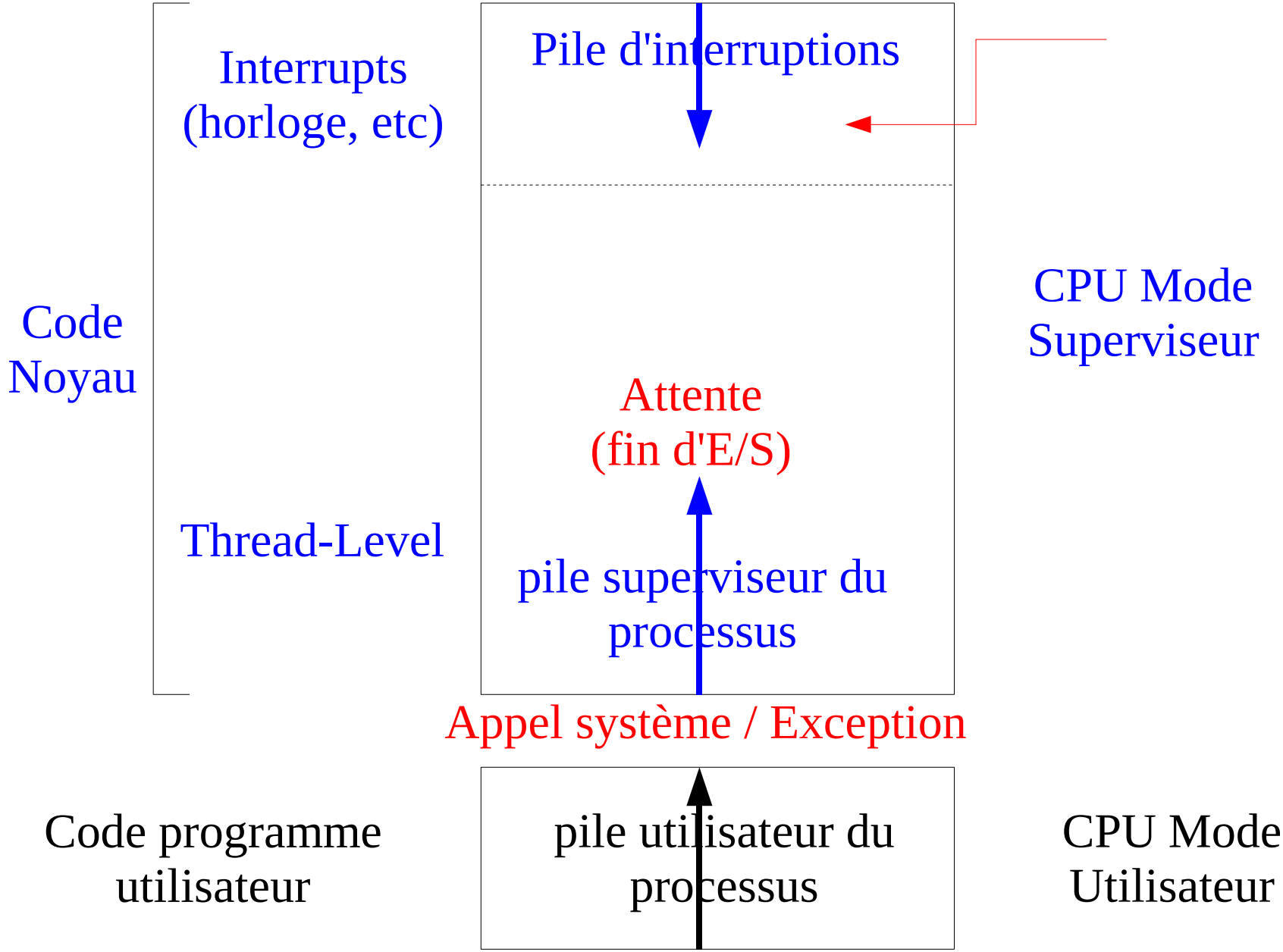
Espace d'adressage processus

- Code du programme exécuté
- Données globales du programme
- Mémoire allouée dynamiquement (heap)
  - `malloc/free` – `new/delete`
- Pile d'exécution
  - Pas d'extension dynamique
  - Taille pré-définie

Espace Mémoire d'un Processus



Modes d'exécution



Contexte système Processus Unix

- Contexte Entrées/Sorties
  - Répertoire racine
  - Répertoire courant
  - Table de descripteurs de fichiers (index >= 0)
    - 0 = entrée standard (lecture)
    - 1 = sortie standard (écriture)
- Contexte mémoire
  - Régions virtuelles (code, données, pile)
  - Pages mémoire physique

Processus Unix

- Creation : `pid = fork()`
  - processus fils clone du père
  - Seules différences :
    - `pid` (process identifier)
    - `ppid` (parent process identifier)
- Terminaison : `exit(status)`
  - Libère toutes les ressources, sauf structure `proc`
  - Processus passe à l'état « zombie »
- Destruction : `wait(&status)` par père
  - Libère structure `proc` du processus fils

Création / Terminaison processus

```
main() {
  pid = fork();
  if (pid == 0) { /* child processus */
    printf("child process, pid=%d\n", getpid());
    exit(0); /* never return */
  }
  if (pid > 0) {
    printf("parent process, child = %d\n", pid);
  } else
    perror("fork() system call failed");
}
```

Modèle de programme

```
int main(int argc, char* argv[], char* envp[])
/*
  argc: nombre éléménts dans tableau argv

  argv: tableau de chaines de caractères contenant
        le nom et les paramètres de lancement du
        programme

  envp: tableau de chaines de caractères contenant
        les variables d'environnement sous la forme
        nom=valeur
        (derniere entree du tableau à zero)
*/
{
    /* corps du programme */
}
```

Lancement d'un programme

Soit la commande suivante lancée depuis le *shell* pour copier le contenu du fichier `file1` dans le fichier `file2` :

```
/bin/cp file1 file2
```

La fonction `main` du programme `cp` est appellée avec les arguments suivants :

```
argc    : 3 (la commande contient 3 chaînes de caractères)
argv[0]: /bin/cp
argv[1]: file1
argv[2]: file2
```

Chargement d'un programme

- Appel système de la famille `exec()`
  - `exec("/bin/cp", "file1", "file2")`
- Remplace programme du processus courant par nouveau programme
  - Charge espace d'adressage avec code/données du nouveau programme à exécuter
  - Conserve contexte système (ressources, etc.)
  - Ré-initialise la pile d'exécution
  - Pas de retour possible ancien programme si erreur durant chargement nouveau programme

Chargement d'un programme

```
main() {
  pid = fork();
  if (pid == 0) { /* child process */
    execl("/bin/cp", "cp", "file1", "file2", 0);
    perror("execl() system call failed");
    exit(1); /* never return */
  }
  if (pid > 0) {
    wait(&exit_status); /* wait child dead */
  } else perror("fork() system call failed");
}
```

Exemple utilisation fork/exec