Principes des Systèmes d'Exploitation

PLAN DU COURS

- **Fonctions d'un OS**
- Machine Logique / Langage
- Développement de Programmes
- Notion de Processus
- Processus Unix

Fonctions d’un OS

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

Gestion du Matériel (1)

- Gestion des composants « primaires »
  - Processeur
  - Support architectures multi-processeurs
  - Mémoire centrale
  - Support Mémoire Virtuelle (MMU, TLB, etc.)
  - Interruptions
  - Composants intégrés (Timers, Compteurs, etc.)
- Mise en œuvre dans module(s) spécifique(s) du noyau de l'OS
  - Répertoire **arch** du noyau Linux
  - Un sous-répertoire par type de CPU
    - **arch/i386** pour microprocesseur 32 bits Intel 80386

Gestion du Matériel (2)

- Gestion des périphériques
  - Identification des [types de] périphériques
  - Support du « Plug and Play »
- Mise en œuvre des Entrées/Sorties
  - Support bus d’entrées/sorties
  - Drivers par types de périphériques
- Administration
  - Désignation **non ambiguë** des périphériques
  - Collecte de statistiques

Gestion de l'Information

- Stocker programmes et données
  - Fichiers, Bases de Données
- Structuration de l'espace de stockage
  - Notions de partition, de répertoire
  - Agrégation dynamique d'espaces additionnels
- Mécanismes de désignation des fichiers
  - Liens physiques, liens symboliques
  - Points de montage de systèmes de fichiers
- Mécanismes de contrôle d'accès aux fichiers

Gestion des Communications

- Interface Homme-Machine (**IHM**)
  - Gestion des interactions avec les utilisateurs
- Échanges de données entre programmes
  - Locaux
    - Mémoire partagée
    - Tubes de communication (« **pipes** »)
  - Distants
    - Protocoles de Communication (**TCP/IP**)

Contrôle de l’Exécution

- Démarrage/Arrêt de programmes
- Enchaînement de l‘exécution des programmes
  - Séquentiel, parallèle
- Aide à la mise au point de programme
  - Détection et notification d'exceptions
  - Pose et traitement de points d'arrêt
- Mesures
  - Du temps
  - Statistiques (I/O, mémoire, etc.)

Sécurité et Protection

- Identification des Utilisateurs
- Mécanismes de Contrôle d'Accès
  - Mots de passe
  - Droits d'accès
- Support de Communications
  - Confidentialité
  - Authentification
- Surveillance (« Monitoring »)
  - Enregistrements d'événements

Classement des OS

- Dédiés / A Usage Général
- Interactifs (IHM) on non
- Enfouis
- Isolés / Communicants
- Frontières de plus en plus floues
  - Box Internet
    - Système dédié ?
    - Système enfoui ?
    - Système pas interactif ?

Propriétés des OS

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

PLAN DU COURS

- Fonctions d'un OS
- **Machine Logique / Langage**
- Développement de Programmes
- Notion de Processus
- Processus Unix

Noyau d'OS = Machine Logique

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

Niveaux d'Abstraction (1)

Langage de Commandes (**Shell**) + Utilitaires

Environnement de Développement
Langages + Bibliothèques

Operating **S**ystem **ABI**

Machine physique

Niveaux d'Abstraction (2)

- Langage de commandes (« shell » d'Unix)
  - Fichiers de programmes (binaires, shell scripts)
  - Manipulation fichiers de données
- Langages de programmation
  - Environnement défini par le langage
    - Compilé ou Interprété
  - Peut inclure extensions (objets/fonctions)
    - Mises en œuvre par bibliothèques ou par interpréteur
- Ensemble des appels système du noyau

Shell - Langage de Commandes (1)

- Commande de lancement de programme
  - <nom_du_programme> [paramètres]
  - Exemple : `cp file1 file2`
- Par défaut, attend fin programme courant
- Lancement de programme en arrière plan
  - <nom_programme_serveur> [paramètres] &
- Notion de « shell script »
  - Fichier contenant des commandes shell
  - Lancé comme un programme
    - `sh script.sh`

Shell - Langage de Commandes (2)

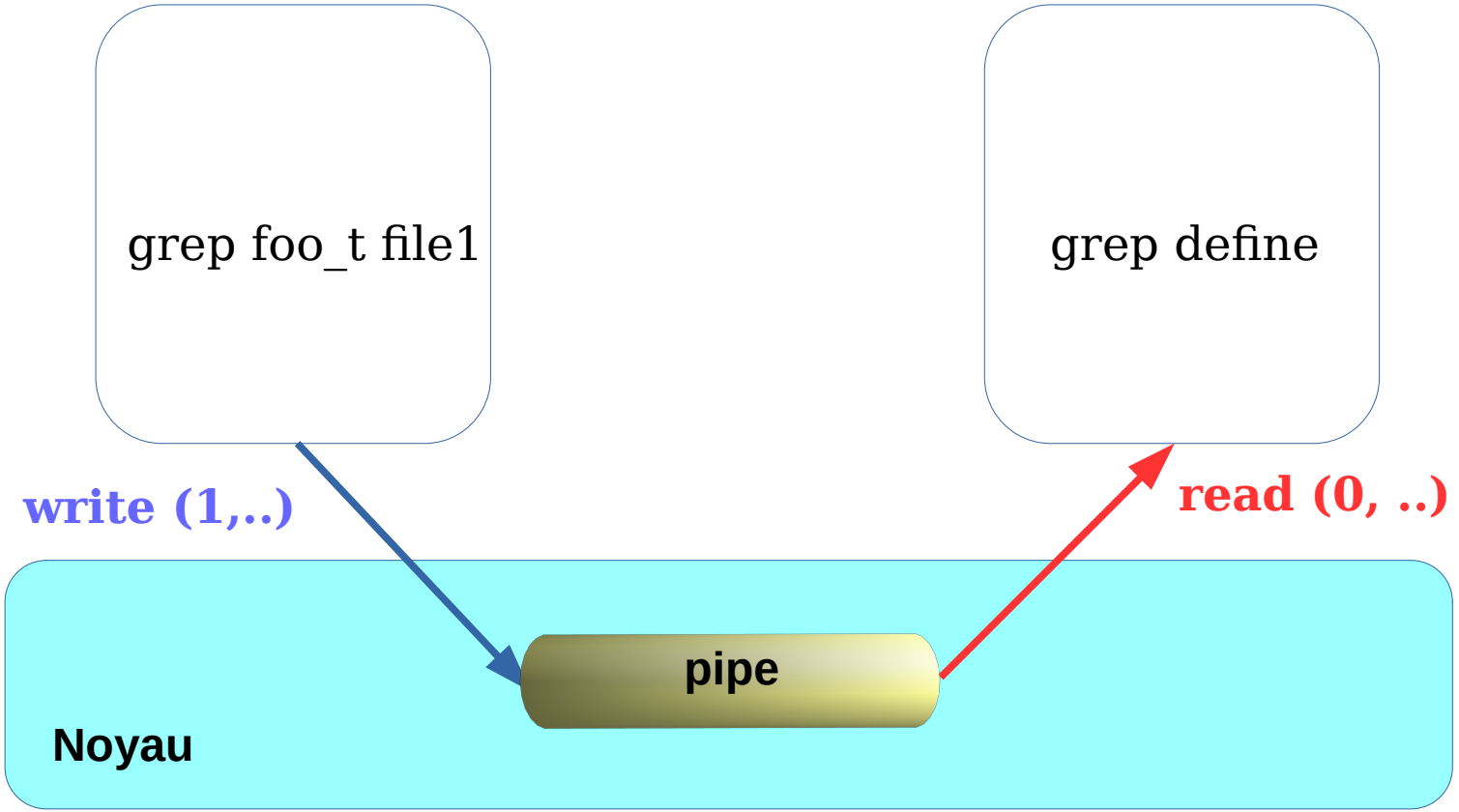
- Intègre directives de programmation
- Variables
  - <nom_de_variable>=valeur
  - TERM=xterm
  - PATH=/bin:/usr/bin/
- Directives de chemins d'exécution
  - if, else, elif, for, case, until, while ...
- Support de la notion de fonction

Shell - Langage de Commandes (3)

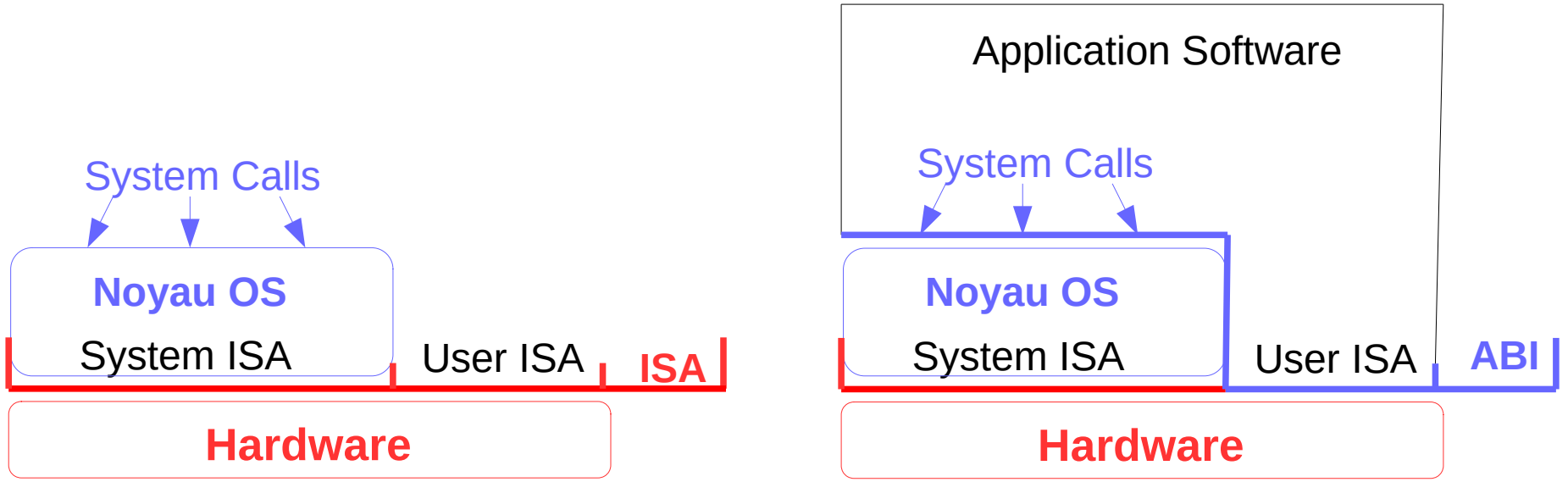
- Entrées/Sorties standard
- Entrée standard (0) et sortie standard (1)
  - Défaut : 0 = clavier, 1 = écran
  - Peuvent être redirigées sur un fichier
    - `grep "foo" file1 > foo_in_file1`
- Notion de tube (« pipe »)
  - Programmes se comportant comme des filtres
  - Modèle du producteur / consommateur
  - Producteur : sortie standard redirigée sur entrée du pipe
  - Consommateur : entrée standard redirigée sur sortie du pipe

Shell - Langage de Commandes (4)

- Principe des pipes
  - `grep "foo_t" file1 | grep "define"`



Interfaces Machines (1)



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

Interfaces Machines (2)

- **ISA**
  - Toutes les instructions du CPU
  - Par extension, inclut
    - Architecture mémoire
    - Mécanismes d'interruptions
    - Périphériques d'Entrées/Sorties
  - Indépendante des OS
- **ABI**
  - Instructions CPU « neutres » (non privilégiées)
  - Ensemble des appels système du noyau

PLAN DU COURS

- Fonctions d'un OS
- Machine Logique / Langage
- **Développement de Programmes**
- Notion de Processus
- Processus Unix

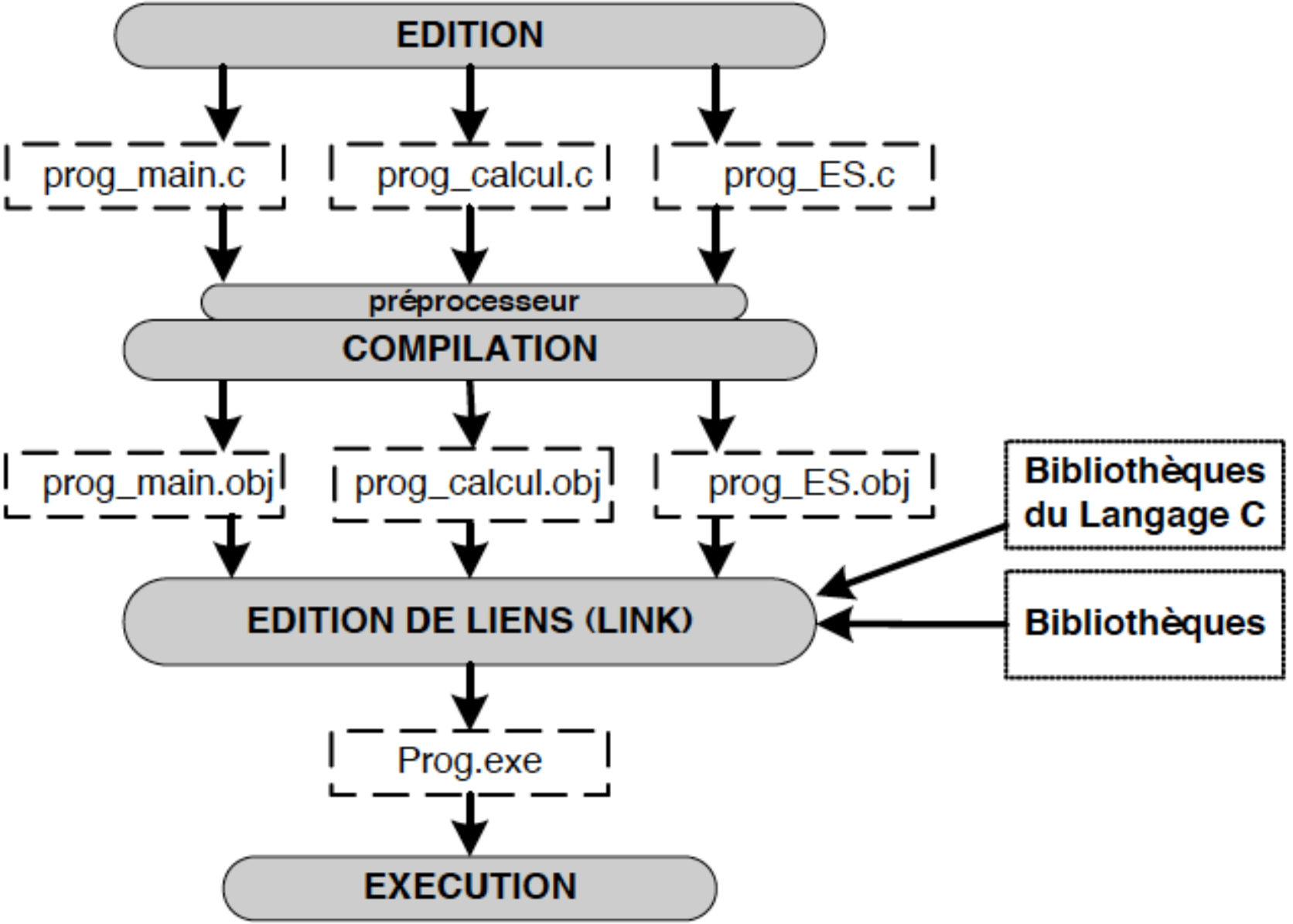
Conception de Programmes

- Conception Modulaire
  - Fichiers source séparés
  - Unités de compilation indépendantes
  - Structuration par programmation objet
  - Parties dépendantes OS / CPU
  - Fonctions communes
- Interfaces décrits dans fichiers dédiés
  - 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
- Édition de liens
  - Combinaison de fichiers objets + bibliothèques
  - Statique ou Dynamique

Étapes Développement de Programmes



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

- Compilation Croisée
  - Machine hôte : compilation des programmes
  - Machine cible : exécution des programmes
- Compilation croisée
  - Machine hôte ≠ Machine cible (CPU/OS)
  - Bibliothèques + include .h 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
  - Format {code + données} pouvant être combiné avec d'autres fichiers objet 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

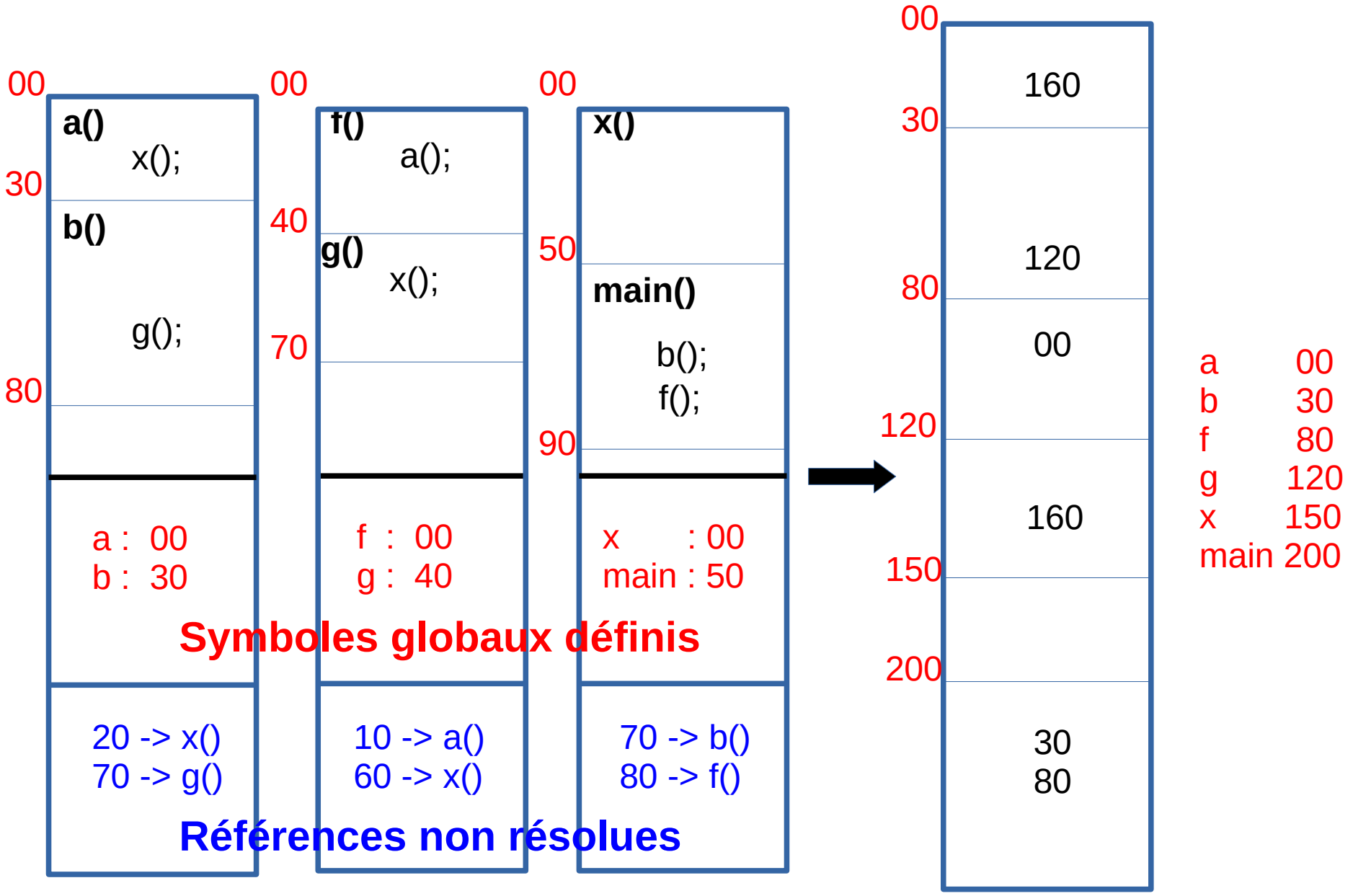
Bibliothèques

- 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
    - **RPC** (**R**emote **P**rocedure **C**all)

Édition de Liens

- Fichiers objets + Bibliothèques → Binaire Exécutable
- Effectuée par Éditeur de Liens
- Résout références non résolues
  - D'abord entre fichiers objets
  - Puis avec symboles exportés par bibliothèques
- Regroupe et reloge données et code des fichiers objet
  - Regroupe et reloge sections **text** et **data**
  - Accumule taille et reloge sections **bss**

Édition de Liens (2)



Édition de Liens Statique

- Code des fonctions de bibliothèque utilisées inclus dans chaque fichier binaire exécutable
- Dupliqué en mémoire centrale
- Binaire exécutable auto-suffisant
  - Toujours exécutable en cas de "disparition" des fichiers de bibliothèques
  - Protégé contre modifications ultérieures des fonctions de bibliothèque

Édition de Liens Dynamique (1)

- Bibliothèques Partagées
- Symboles des fonctions/données importées uniquement référencés dans chaque binaire exécutable
  - Code fonctions de bibliothèque chargé dynamiquement en mémoire centrale et partagé par tous les programmes
  - Copie privée des données pour chaque programme exécuté
- Code bibliothèques partagées peut être pré-chargé en mémoire à des adresses mémoires pré-établies
  - Revient à édition de liens statiques
  - Sans duplication code dans fichiers binaires

Édition de Liens Dynamique (2)

- Bibliothèques partagées chargées en mémoire dynamiquement
- Évite de dupliquer en mémoire code des fonctions de bibliothèques partagées
- Références externes des fonctions résolues
  - Au lancement d'un programme
    - Windows **.DLL** (**D**ynamic **L**ink **L**ibrary)
  - Durant l'exécution, lors du premier appel
    - Linux **.so** (**S**hared **O**bject)

Édition de Liens Dynamique (3)

- Fichiers **.so** sur Linux / GCC
- Compilés en **PIC** (**P**osition **I**ndependant **C**ode)
  - Fonctions de bibliothèques relogeables et exécutables à n'importe quelle adresse
- Accès indirect aux variables / fonctions globales par **GOT** (**G**lobal **O**ffset **T**able)
  - Contient adresses données et fonctions chargées dynamiquement
- Invocation fonctions partagées via table d'indirection : **PLT** (**P**rocedure **L**inkage **T**able)

Édition de Liens Dynamique (4)

- Éditeur de liens dynamique Linux : **ld.so**
  - Nom stocké dans section **.interp** des fichiers binaires des programmes
  - Chargé par le noyau au lancement du programme
  - https://man7.org/linux/man-pages/man8/ld.so.8.html

```
-> readelf -p .interp /bin/gcc

Vidange textuelle de la section « .interp » :
  [     0]  /lib64/ld-linux-x86-64.so.2

-> file /lib64/ld-linux-x86-64.so.2
/lib64/ld-linux-x86-64.so.2: symbolic link to /lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
```

PLAN DU COURS

- Fonctions d'un OS
- Machine Logique / Langage
- Développement de Programmes
- **Notion de Processus**
- Processus Unix

Notion de Processus (1)

- Historiquement, concept logiciel introduit pour la mise en œuvre de programmes concurrents
  - Partage (transparent) d'une machine entre utilisateurs
  - Représente chacun des programmes séquentiels exécutés en parallèle sur la machine
- Contexte d'exécution d'un programme
  - Contexte processeur (valeur des registres CPU)
  - Contexte mémoire (espace d'adressage virtuel)
  - Attributs : identificateur, priorité, droits, etc.
  - Ressources allouées par le noyau

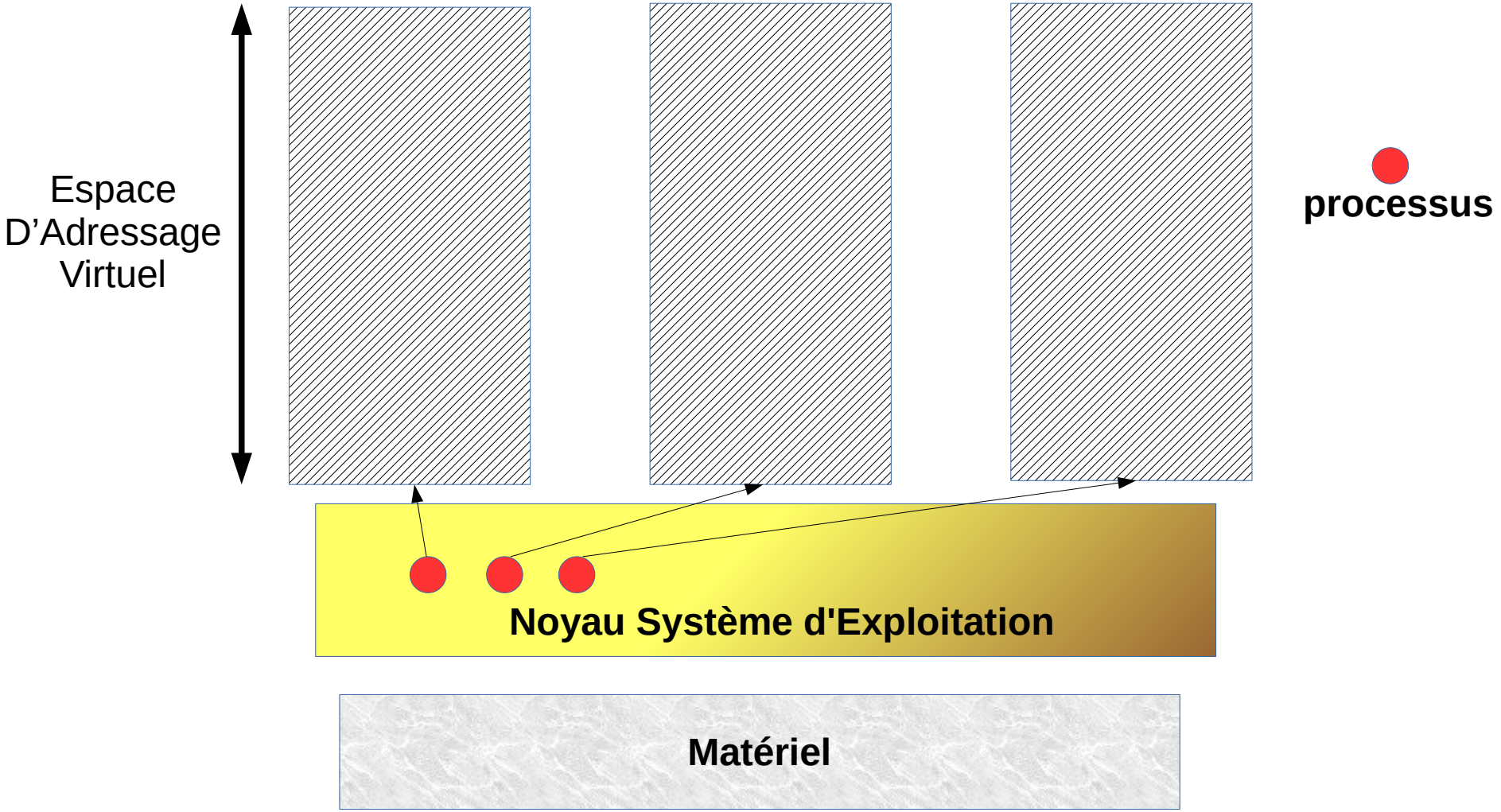
Notion de Processus (2)

- Contexte d'attribution des ressources
- CPU
  - Politiques d’ordonnancement (« scheduling »)
  - Points de préemption du système
- Mémoire centrale
- Espace de swap sur disque
- Objets temporaires du système
  - Ports de communication (TCP, UDP)
- Buffers d'Entrées/Sorties

Notion de Processus (3)

- Machine logique incarnant l'exécution d'un programme dans un environnement privé
- Représentée par structure de donnée incluant
  - État [d'une partie] de la machine physique sous-jacente
    - Sauvegarde valeur registres CPU
    - Allocation mémoire centrale
    - Support espace d’adressage virtuel
  - État exécutif : stoppé, prêt, etc.
  - Contexte Entrées/Sorties

Notion de Processus (4)



Espace d'Adressage Virtuel (1)

- Espace d'adressage indépendant des adresses physiques et de [la taille de] la mémoire centrale
- Mode d'adressage virtuel supporté par CPU
  - Utilise adresses virtuelles pour accéder aux instructions et aux données en mémoire centrale
- Correspondance entre adresse virtuelle et adresse physique
  - Établie dynamiquement par un mécanisme matériel de conversion intégré dans le CPU
  - En fonction d'informations gérées par le noyau du système

Espace d'Adressage Virtuel (2)

- Décomposé en régions de tailles variables
  - Gérées par le noyau du système
- Région de code du programme exécuté
- Région des données globales du programme
- Tas (« heap ») : région d'allocation dynamique de mémoire
  - `malloc()/free()`
  - `new()/delete()`
- Pile d'exécution
  - Pas d'extension dynamique
  - Taille pré-définie

PLAN DU COURS

- Fonctions d'un OS
- Machine Logique / Langage
- Développement de Programmes
- Notion de Processus
- **Processus Unix**

Processus Unix

- Création
  - Appel système `fork()`
  - Processus fils clone du père
  - Seules différences :
    - « pid » (**P**rocess **Id**entifier)
    - « ppid » (**P**arent **P**rocess **Id**entifier)
- Terminaison
  - Appel système `exit(status)`
  - Libère toutes les ressources, sauf structure de données « proc » associée au processus
  - Processus passe à l'état « zombie »
- Destruction
  - Appel système `wait(&status)`
  - Invoqué par processus père
  - Libère structure de données « proc » associée au processus fils

Contexte Système Processus Unix

- Contexte Entrées/Sorties
  - Répertoire racine
  - Répertoire courant
  - Table de descripteurs de fichiers (entier >= 0)
  - **0** : entrée standard (lecture)
  - **1** : sortie standard (écriture)
  - **2** : sortie standard messages d’erreurs du programme
- Contexte Mémoire
  - Composition Espace d'Adressage Virtuel
  - Mémoire physique allouée au processus

Création / Terminaison Processus

```
main() {
    pid_t 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);
        wait(&status);
    } else
        perror("fork() system call failed");
}
```