Initiation à l'Informatique
Systèmes d'exploitation

Catalin Dima

Systèmes d'exploitation

- Qu'est-ce qu'un système d'exploitation ?
- Système d'exploitation et architecture.
- Concepts de base dans les systèmes d'exploitation :
  - (système de) fichiers, processus, multiprogrammation, interface avec les utilisateurs, ordonnancement, sécurité, etc.

Architecture d'un ordinateur

- Très simplifiée !
- Vue "physique" sur un système informatique.

  - Processeur
  - Mémoire

  - Écran, imprimante, ...
  - Disques durs, CD-ROMs, sticks mémoire...
  - Claviers, Écrans tactiles...
  - Connexions réseau

Système d'exploitation : chargé de faire le lien entre la partie haute et celle en bas de cette figure.

Structure “logique” d’un système informatique

- Niveau matériel :
  - Composants physiques.
  - Microarchitectures, microprocesseurs.
  - Langage machine, assembleur, microprogrammation des pilotes (drivers).
- Niveau système d’exploitation :
  - Noyau SE = routines standardisées de gestion de ressources (appels système).
  - Gestion des entrées/sorties.
  - Gestion de requêtes d’accès aux ressources.
- Niveau intermédiaire (parfois SE, parfois applications) :
  - Compilateurs, éditeurs, multifenêtrage, interprètes de commandes.
- Niveau applications :
  - Système de réservation de billets en ligne.
  - Système bancaire.

SE : contours

SE =

Un ensemble de programmes qui s'exécutent à tout moment sur l'ordinateur

+

Un ensemble de fonctionnalités pour rendre l'utilisation de l'ordinateur plus facile.

- Masquer le matériel.
- Fournir des méthodes standard d'accès aux ressources.
- Remplacer les opérateurs.
- Permettre un usage efficace des ressources, indépendamment des programmes à executer.
- Permettre un usage honnête des ressources à tous les utilisateurs honnêts.

Concepts de base dans un SE

- *Fichier* = unité de stockage des informations.
- *Processus* = programme en exécution, avec son *environnement*.
- *Noyau du système d'exploitation* = ensemble de fonctionnalités internes au système, utilisées lors de toute interaction processus/architecture.
- *Appel système* = moyen standardisé (procedure, fonction) pour exécuter une tâche élémentaire dans le système.
  - Interface entre processus et architecture, fournie par le système d'exploitation.
- *Environnement* (ou espace de travail) pour chaque processus, utilisateur, groupe = ensemble de paramètres utilisés par le processus/utilisateur/groupe dans ses interactions avec le système.
  - Répertoires de travail, commandes associées, quotas...

Pourquoi des appels système ?

- Processeur exécute programmes en langage machine
  - Éventuellement écrits en langage assembleur.
- Processeur doit interactionner avec périphériques.
- Périphériques mettent à disposition au système des fonctionnalités très basiques
  - Écrire un nbre de *blocs* sur disque, envoyer untel signal à l'écran...
  - Pilote (driver) du périphérique, dépendant du vendeur !.
- On peut tout écrire dans ces langages mais c'est pas si facile...
  - Copier un fichier entre un disque dur Maxtor et un stick USB ?...
- Appels système : standardisation des fonctionnalités
  - Accès standardisé aux fichiers, création standardisée des processus, communication standardisée entre processus etc.

Fichiers – raisons d'être

- Programme : suite d'*instructions* données au processeur pour exécuter une certaine tâche.
- Quand un programme se termine, d'où va le processeur chercher un autre programme qu'on veut rouler ?
- Les programmes doivent être stockés sur des supports externes.
  - Mémoire = volatile !
- Parfois les programmes peuvent s'échanger des données entre eux
  - Mais sans que l'émetteur et le récepteur soient exécutés en même temps !

– Solutions : *fichiers*.

Fichiers et systèmes de fichiers

- **Fichier** :
  - Ensemble d'informations en relation entre elles : programmes, données.
  - Unité "logique" de stockage de l'information.
- **Le système de fichiers** :
  - L'ensemble des fonctionnalités mises en oeuvre pour la gestion des fichiers dans un SE – partie *essentielle* du système d'exploitation qui gère les fichiers.
- Fonctionnalités d'un système de fichiers :
  - Correspondance entre fichiers et dispositifs physiques = placement sur disques/bandes magnétiques/mémoires flash...
  - Organisation interne et externe des fichiers.
  - Gestion des requêtes pour l'accès aux fichiers.
  - Protection des fichiers.

Caractéristiques des fichiers

- Nommage de fichiers : convention des extensions (`c, o, bak, htm(l), jpg, gif, mp3, pdf, txt, exe` etc.)
- Chemin d'accès.
- Structure interne : séquence d'octets/enregistrements, structure arborescente.
- Type : exécutable, archive à distance, partagé, etc.
- Accès aux fichiers : séquentiel/aléatoire.
- Protection des fichiers : accéssigne en lecture/écriture/exécution seulement, qui (utilisateur) peut accéder au fichier, protection par mot de passe/chiffrement.
- Opérations sur les fichiers : lecture, modification, lancement en exécution, etc.
- Divers caractéristiques de gestion : archive, taille, date de création, propriétaire, etc.

Certaines caractéristiques gardés en tant qu'**attributs**.

Répertoires

- Conteneurs de fichiers = classeurs/pochettes.
- Dans la plupart des SE, organisés en arborescence.
- Possibilité de *partager* des fichiers (W98, W2000, XP, Unix/Linux) :
  - Pour rendre l'accès plus facile, il est permis à plusieurs répertoires de "contenir" le même fichier/répertoire.
  - En fait, on a plusieurs *liens* vers le même fichier/répertoire.
  - *Problème* : possibilités d'avoir des cycles dans le graphe de dépendance.
- Protection des répertoires : nom du proprietaire, droits d'accès.

Chemins d'accès

- Répertoires et fichiers organisés dans une arborescence.
  - Et si deux fichiers avec le même nom ?
- Identification d'un fichier selon son **chemin d'accès absolu**
  - Son nom...
  - ... précédé par la séquence de répertoires qui le contiennent.
  - Exemple (W98) : `C :\users\dima\toto.txt`.
  - Exemple (Unix/Linux) : `/home/users/dima/toto.txt`.
- Si plusieurs supports externes existants
  - W98+ = *volume*, affectation d'une lettre.

    `A : B : C : D : E :` etc.
  - Unix/Linux : toute l'arborescence des fichiers doit hériter du *répertoire racine*, `/`.

    E.g. `/media/usbdisk`, ou `/media/cdrom`.

Exemples de systèmes de fichiers

- FAT (File Allocation Table) : le contenu d'un répertoire est gardé dans une *liste chaînée.*
- EXT2 (Extended fiesystem – pour Linux) : le contenu d'un répertoire est organisé en une liste indexée à deux niveaux.
- ISO9660 (système de fichiers des CDs) : allocation contigüe.
- NTFS (New Technology File System) : propriété Microsoft, détails de construction non publics...
  - Compression et cryptage transparent, contrôle d'accès, journalisation, quotas, etc.

Partitions

- On possède un espace externe de stockage (disque dur), mais on veut le diviser en plusieurs *partitions*.
  - Parce qu'on veut installer deux (ou plusieurs) systèmes d'exploitation : double boot (ou triple, ou...).
  - Pour des questions de sauvegarde, fiabilité, securité, pour diminuer l'étendue des dégêts en cas de partition corrompue, etc.
- Sur les disques durs des PCs, le Master Boot Record = premier secteur du disque, contient une Table de partitions : tableau des adresses des partitions sur le disque.
  - Que 4 entrées – *partitions primaires*.
  - Mais sur chaque partition, on peut créer des sous-partitions = *partitions logiques*.
- Une partition devient un *disque logique*
  - On le voit comme si notre ordinateur aurait plusieurs disques durs.
- Outils pour manipuler les partitions : `fdisk` (ligne de commande), PartitionMagic (payant), etc.
- Nécessaires si on a un ordinateur avec un seul système d'exploitation (Windows) et on veut en installer un autre (Linux).

Multiprogrammation – raisons d'être

- Programme = durée de vie, activité variée.
- Et si on ne pouvait lancer qu'un seul programme à la fois ?
  - Séquentialisation rendant la vie impossible des utilisateurs !
- Programme utilisant des accès aux supports de mémoire externe ou aux périphériques
  - On serait obligé d'attendre la terminaison de l'affichage à l'imprimante pour pouvoir continuer à travailler.
- Solution = **multiprogrammation** ! = méthodologie permettant d'avoir plusieurs programmes qui sont apparaiment en exécution à chaque moment.
  - Pour les PC, on parle plutôt de multitasking.

Défis de la multiprogrammation

- Séparation de l'espace d'adressage de chaque programme – garde-fous contre du code malveillant.
  - Séparation du code...
  - ... mais si on a plusieurs répliques du même programme, ne serait-il moins coûteux s'ils partageraient leur code ?
  - Séparation des données...
  - ... mais si les programmes doivent coopérer pour bien fonctionner, comment devraient-ils s'échanger des informations ?
- Décisions sur le moment où un programme doit laisser la main au suivant
  - Quel est le suivant ?
  - Quel est le bon moment ? Peut-on l'approximer, au moins ?
  - Que faire avec le programme qu'on vire ? avec ses données se trouvant en mémoire vive ? Quand pourra-t-il reprendre son exécution ?

Processus

- **Processus** = programme entre le début et la fin de son exécution.
- Plusieurs processus peuvent coexister au même moment dans un système.
- Le nombre de processus *actifs* est égal au nombre de processeurs de l'architecture matérielle.
- États d'un processus :
  - Créé, mais pas encore actif.
  - Actif, le processeur exécute une partie de son code.
  - Inactif, a laissé la main à un autre processus, sans se terminer.
  - Terminé, mais possédant encore qqs infos utiles au SE.
- L'état inactif peut être raffiné davantage.
  - Bloqué, car étant en train de faire des E/S sur périphériques, donc en attente des résultats rapportés par ces périphériques.
  - Suspendu, car ayant reçu un *signal* spécial de la part du système (pour laisser la main aux autres pour divers raisons).

Programmes et processus

- Un *programme* est un ensemble d'instructions à exécuter par l'ordinateur.
- ... et comporte des déclarations de *variables*, utilisées au cours de son exécution.
- Quand on lance en exécution un programme, on doit
  - Réserver de l'espace mémoire pour le code et les données du programme.
  - ... et aussi pour les éventuelles données qui seront créées en cours d'exécution (données dynamiques).
  - Initialiser les paramètres dont le programme a besoin pour s'exécuter : données de lancement, bibliothèques de fonctions auxiliaires, fichiers, etc.
  - Aller chercher la premiere instruction du programme, la mettre dans le compteur du programme et lancer, enfin, l'exécution de ce qui se trouve dans ce compteur.
- À certains instants, le programme intéragit avec le système d'exploitation :
  - Quand il attend qu'on tape = appel système (car opération E/S).
  - Quand il veut (c.à.d. il exécute du code pour) afficher qqchose, ou lire/écrire sur disque.
  - Quand il veut communiquer avec d'autres programmes.
  - Quand le système l'interrompt, pour qu'il laisse la main à un autre programme.

Caractéristiques d'un processus

- Espace d'adresses : espace en mémoire ou le processus réside et peut lire/écrire
  - code, données, pile d'appels.
- Valeur de registres : compteur du programme, pointeur de pile, autres registres spécifiques au processus et au système d'exploitation.
- Valeurs de variables : variables internes.
- Informations utiles pour le système d'exploitation :
  - pour l'ordonnancement : priorité, files d'attente ;
  - pour la gestion de la mémoire : si le code du programme est decoupé en pages/segments, alors les adresses de ceux-ci.
  - état des entrées-sorties : table de fichiers ouverts, périphériques alloués au processus
  - comptabilisation : quantité de temps processeur ou temps réel utilisé, limites de temps, no. de compte, identification du processus :
    - **PID** = Process Identification Descriptor.
    - **PPID** = Parent Process Identification Descriptor.

Hiérarchie des processus

- Tout processus est lancé par un autre processus : son *père*.
- Chaque processus père est responsable de vérifier si tous ses fils se sont bien terminés.
- Les fils peuvent hériter (i.e. copier) du *code* et des *données* du père.
- Au démarrage : processus *chargeur*, l'ancêtre de tout processus.
- À chaque instant, l'ensemble de processus forme *un arbre*.
- Lancer un processus par double click dans une fenêtre de l'Explorer (W98) = le processus Explorer crée un *fils* dont le code sera celui du programme clické.
  - Le processus Explorer ne se termine pas, et ne se bloque pas non plus !

Ordonnancement

- L'UC est une ressource et son emploi doit satisfaire des contraintes d'*exclusion mutuelle*, d'*efficacité*, d'*équité*.
  - Exclusion mutuelle : ne pas entrelacer deux processus sans que leur contexte soit **sauvegardé**.
  - Efficacité : avoir des processus en attente (passive) le moins de temps possible.
  - Équité : ne pas "enfamer" les processus qui attendent la libération de l'UC.
- **Changement de contexte** entre processus interrompu et processus destiné à reprendre le controle du processeur :
  - Sauvegarde de l'état du processus interrompu.
  - Chargement de l'état du processus choisi pour être exécuté.
- **Ordonnancement** = ensemble de procédures implémentant les contraintes ci-dessus.

Types d'ordonnancement

- Préemptif : un processus peut être interrompu pour laisser la main à un autre (ce qui se passe en W98, Unix/Linux, etc.).
- Round-Robin : chaque processus reçoit une tranche de temps (prédéfinie), pendant laquelle il a la possesion du processeur.
  - *Liste circulaire* des processus.
- À priorités : certains processus peuvent s'avérer être plus importants que d'autres, alors on choisit toujours ceux qui sont les plus importants.
  - Vieillisement : si le processus est actif, sa priorité se dégrade avec le passage du temps.
- Temps réel :
  - Cas des systèmes sensibles : aviation, centrales nucléaires...
  - On connait, au moment de l'exécution, le necessaire de temps pour chaque processus.
  - Si les processus ne se terminent pas dans certains délais – catastrophe.
  - On prend des décisions d'ordonnancement selon ces données.

Interfaces avec les utilisateurs

Systèmes de lancement de programmes :

- Multifenêtrage : W98, XP, Vista, XWindows...
- Invités de commandes (DOS), Shells (Unix/Linux).
- Ensemble de commandes disponibles pour l'utilisateur = **commandes**.
  - La plupart sont pour la gestion de fichiers.
  - Mais aussi de gestion d'utilisateurs, configuration de système, journalisation, etc.

Activités au démarrage d'un ordinateur

- Tout processeur n'exécute que du code qui se trouve dans la mémoire primaire (RAM).
- Mais tout système d'exploitation se trouve dans des fichiers, en mémoire externe (disques, sticks USB, CDs/DVDs...).
- On a besoin alors d'un petit *lanceur de programmes* (boot loader), qui ne sait que chercher sur disque un premier programme et le charger en mémoire.
  - Ce qui fait déjà un morceau de code non-négligeable...
  - Est, dans la plupart des PCs, placé dans le ROM – le BIOS.
  - En général, lit le MBR du disque bootable – qui contient, donc, du code d'amorçage.
- Souvent (e.g. double boot), celui-ci lance un autre lanceur (2e niveau), qui va lancer le système.
- Tâches à exécuter au démarrage :
  - Vérifier de l'état des composants du système – avant le boot loader.
  - Charger les pilotes des périphériques – le premier processus lancé par le boot loader.
  - Lancer des processus de configuration du système (e.g. réseau) et d'intéraction avec l'utilisateur (e.g. multifenêtrage).

Défis de la conception des systèmes ouverts et multi-utilisateurs

- *Intrusion* dans des systèmes informatiques, en se faisant passer pour un autre (angl. *spoofing*), ex. par vol, casse ou récupération non-authorisée de mots de passe, mais aussi par usurpation d'identité (adresse IP) ou par "ingénierie sociale".
- *Accès* non-autorisé des utilisateurs d'un système informatique aux informations ou ressources auxquelles, normalement, ils n'ont pas l'autorisation.
- *Fuites d'information* en cours de transport à travers des canaux de communication.
- *Prise de contrôle* pure et simple d'une machine.
- *Deni de service* (angl. DoS *Denial of service*) : attaque contre une ressource (d'habitude des serveurs) par envoi d'une quantité excessive de demandes (de connexion) qui vont saturer la ressource et vont la rendre inutilisable par les utilisateurs légitimes.

Types de services de sécurité

- *Sécretisation*, *confidentialité*, *intimité* : l'assurance que ses actions/données/communications ne soient pas examinées par des personnes/parties non-authorisées.
- *Authenticité* : l'assurance que l'accès/la conversation se passe avec une ou plusieurs personnes/parties légitimes.
- *Integrité* : l'assurance que ses propres données ne soient pas modifiées ou corrompues sans son accord, ou que ses communications ne soient pas pas modifiés après avoir été envoyées.
- *Contrôle d'accès* : la prévention de l'accès non-authorisé à une ressource, ou de l'utilisation d'une ressource dans un mode non-authorisé.
- *Non-repudiation* : l'impossibilité de nier ou désavouer une action/transaction/message dont on est la source.

Concepts de base dans la sécurité

- *Système de contrôle d'accès.*
- *Système de detection d'intrusion.*
- *Pare-feu* : système de contrôle du traffic entre l'exterieur et l'intérieur du système (ça peut inclure le contrôle dans les deux sens).
- *Cryptographie* : codage (chiffrement) des données selon lequel, sans connaître la clé de codage, le contenu du message est *pratiquement impossible* à retrouver.
  Cryptanalyse : étude d'un système de chiffrement à partir des messages cryptés, en vue du decryptage sans connaître apriori la clé.
- *Protocôles de sécurité* : schémas de communication sécurisée, bâtis sur des systèmes cryptographiques et assurant plusieurs services de sécurité.

La sécurité dans les systèmes informatiques n'est pas que de la cryptographie !