Systèmes d'exploitation centralisés

1SN

7 mars 2019

Calendrier

Semaine	*Cours*	*TDs*	*TPs*	*Séances en autonomie*	*Echéances*
10 (4/3)	Introduction			TD Shell	
12 (18/3)	Mécanismes de base			TP Shell (commandes de base)	• Quiz Shell
13 (25/3)	Processus			TP Shell (langage de script)	• Rendu TPs Shell
14 (1/4)		• Processus • Signaux	Processus		• Quiz 1 *Début minishell*
15 (8/4)	Fichiers		Signaux		
16 (15/4)		Fichiers (E/S)	Fichiers		• Quiz 2 • Rendu intermédiaire minishell
19 (6/5)	Mémoire	Fichiers (tubes)	Fichiers (tubes)		*Retours / rendus intermédiaires*
20 (13/5)	Virtualisation	Mémoire	Fichiers (select)		• Rendu final minishell *Début minichat*
21 (20/5)		Mémoire virtuelle	Mémoire virtuelle		• Quiz 3
22 (27/5)					• Examen • Rendu minichat

- 30h encadrées (6 CM, 6 TD, 6 TP) ;
- 20 à 40 h de travail personnel.
- 2 miniprojets

Présentation du cours

Objectifs

Culture essentielle

- supervision et gestion des activités en cours (*processus*)
- mise en œuvre des systèmes (contrôle des ressources et des processus) : heuristiques et mécanismes de base
- conception de logiciels complexes

Compétences essentielles

programmation (utilisant le) système d'exploitation (Unix)

Page de l'enseignement : http ://moodle-n7.inp-toulouse.fr (UE Archi-Syst.)
Contact : mauran@enseeiht.fr

Sources et références

- *Précis de systèmes d'exploitation*, G. Padiou, distribué en cours
- Cours d'introduction (L3) de S. Krakowiak, disponible sur internet
- R. et A. Arpaci-Dusseau,
  Operating Systems : three easy pieces, disponible sur internet
- Jean-Marie Rifflet et Jean-Baptiste Younès,
  Programmation et communication sous UNIX. Éditions
- (un peu) plus sur la page Moodle...

Première partie

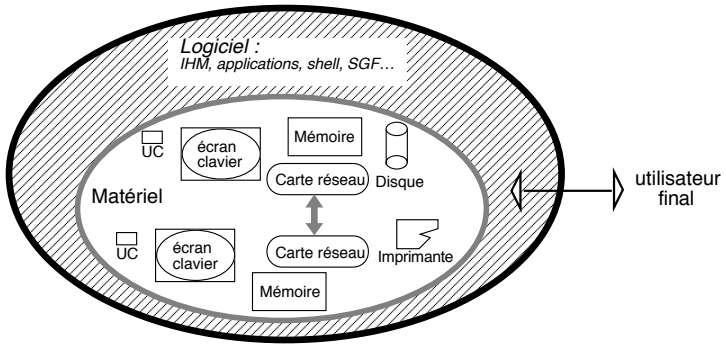
Quelques jalons

Contenu de cette partie

- Comment aborder un problème complexe ?
  → notion de module
- Quel est le rôle d'un système d'exploitation (SX) ?
- Brève histoire des systèmes d'exploitation
  - recherche constante d'efficacité ou de meilleures performances
  - schémas de solutions aux problèmes rencontrés

Plan

Anatomie d'un système informatique (1)



- constituants nombreux, variés, concurrents
- utilisateurs nombreux, variés, concurrents

Les systèmes informatiques sont complexes

Diversité des composants
- UC : alimentation, processeur, chipset, carte mère, cache, bus, RAM, carte graphique, réseau...
- périphériques : écran, scanner, imprimante, DVD, disque dur, souris, enceintes, clavier...

Chaque composant a ses particularités
- fonctionnement différent
- technologies spécifiques

→ gestion complexe : technique, ad hoc, peu réutilisable, opaque

Parallélisme des composants
→ besoin de protocoles pour
- gérer les interactions entre composants
- coordonner l'action des composants pour réaliser un objectif commun

Pour le concepteur du système : imprévisibilité (en général)
- des usages effectifs du système
- des évolutions de l'environnement d'utilisation

→ besoin de flexibilité, et d'adaptabilité

Plan

Comment traiter la complexité des systèmes informatiques ?

→ Adapter le système aux capacités humaines

- limites cognitives
- efficacité faible

Deux stratégies classiques

- diviser pour régner (analyser)
- concentrer, élaguer (synthétiser, modéliser, abstraire)

Version informatique de ces stratégies : notion de module

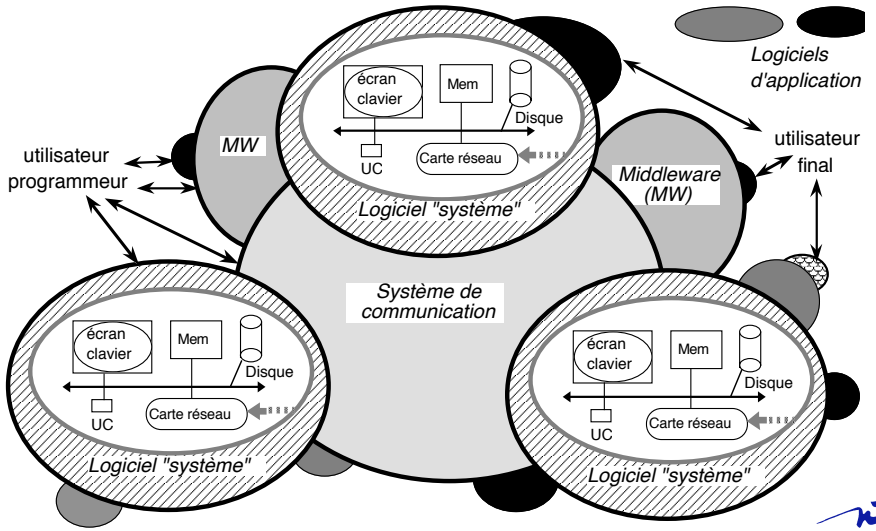
Principe

- Décomposer un système informatique en un ensemble de modules (ou « services »)
  Chaque module joue un rôle, réalise un service
  - précis (bien identifié)
  - spécifique (pas de recoupement/doublon entre modules)
- Chaque module est caractérisé par une interface, qui propose une abstraction du service réalisé
  La forme de l'interface peut varier, selon l'utilisateur visé :
  formel (textuel) → graphique (métaphore)

Exemples (dans le domaine informatique)

mail, ftp, éditeur graphique, bureau...

Exemple : anatomie d'un système informatique (2)



Abstraction des services par des modules

- Un service est caractérisé par une interface
  - interface = ensemble des fonctions fournies aux «clients» du service
  - chaque fonction est définie par
    - son format (la description de son mode d'utilisation) : sa syntaxe
    - sa spécification (la description de son effet) : sa sémantique
  - ces descriptions doivent être
    - complètes (y compris les cas d'erreur)
    - non ambiguës
- L'interaction entre utilisateur et service suit un schéma requête/réponse (cf appel procédural)
- L'interface est séparée, distincte de l'implémentation du service

- portabilité
- maintenance
- standardisation

} → l'interface reste stable, les réalisations peuvent changer selon le contexte

- protection : l'interface est un passage obligé pour l'accès au service (*encapsulation*)

Combiner (composer) les modules (1/2)

Conception descendante (pelures d'oignon, poupées russes)

- Le système informatique est conçu comme une hiérarchie (une succession) de services
- Chaque service (machine virtuelle)
  Simplifie (abstrait) la machine précédente et/ou ajoute une fonction à la machine précédente

Exemple : langages de programmation

Générateur d'applications
Langage de haut niveau
Assembleur
Langage machine

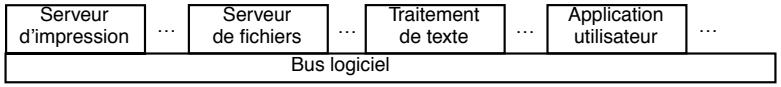
Exemple : Système THE (Dijkstra, 1968)

Applications
Gestion des E/S
Gestion des terminaux
Gestion de la mémoire
Gestion du processeur
Matériel

Combiner (composer) les modules (2/2)

Construction de systèmes ouverts : fédération de services
Le système informatique est conçu comme

- un ensemble de services (de même niveau),
- reliés par un bus logiciel, qui gère les interactions entre services



Exemples

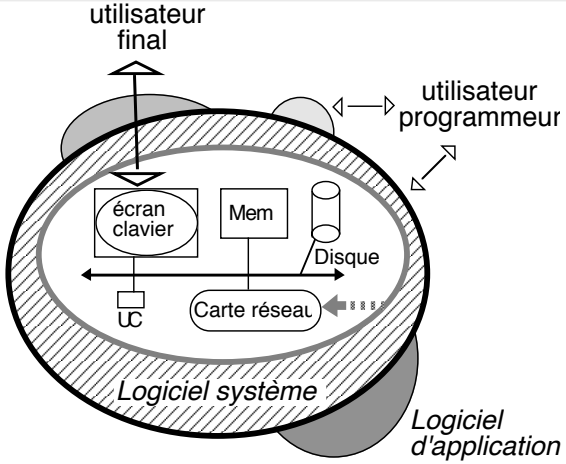
- micronoyaux (Mach, Hurd, L4),
- intergiciels (CORBA, Web Services)

Plan

Rôle d'un SX

Interface des ressources matérielles pour les applications

(→ virtualisation)



Comment simplifier l'accès aux ressources matérielles ?

Virtualisation

- Proposer une interface simplifiée d'accès aux ressources
  - masquant les détails de mise en œuvre
  - éliminant les contraintes physiques des ressources réelles (ressources partagées, en quantité limitée, non fiables...)
    → ressources virtuelles
  - offrant des opérations plus évoluées, abstraites
    exemple (souris) : régulation du déplacement du curseur, gestes (double clic, glisser-déposer...)
- Implanter cette interface : gérer de manière autonome les différentes ressources (mémoire, processeurs, périphériques, programmes...)
  → allouer, partager, protéger piloter les ressources.

Remarque : principe d'interposition

Pour être efficace, l'interface doit être un **passage obligé** :
l'accès et la manipulation directs des ressources doivent être réservés au SX.
(Sinon, le SX n'a pas une vision exacte/cohérente de la ressource qu'il gère)

Outillage

Fournir un environnement d'assistance à l'utilisation des ressources

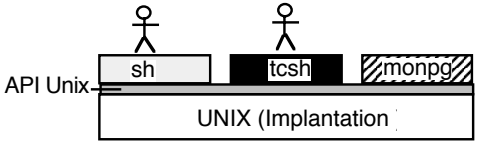
Services système : un panorama

Service	*Ressource physique*	*Ressource virtuelle*
gérer des activités (traitements utilisateurs ou système)		
exécution	processeur	processus
coordination	mécanisme d'interruptions	signaux
gérer des ressources		
accès aux données	mémoire (RAM)	mémoire virtuelle
stockage	disque	fichier
périphériques	imprimante, réseau	*notion unique :*
	écran, clavier	flot d'E/S
interface utilisateur	RAM video, souris	fenêtre, pointeur

environnement de base

- interpréteurs de commandes : *shell, interface graphique (bureau)*
- éditeurs de texte
- communication : *ftp, mail, news, chat*
- administration de données : *copie, archivage, compression...*
- outils de développement : *compilateurs, débogueurs, versions, archives, dépendances*

Interfaces du système système d'exploitation (SX) (1/2)



Un SX présente en général deux (types d') interfaces

Interface utilisateur, ou interface de commande

- destinée à un usager humain (IHM)
- composée
  - d'un ensemble de commandes (programmes utilitaires)
  - d'un interpréteur de commandes (shell), qui permet de saisir et transmettre au SX les requêtes de l'utilisateur
- le langage de commande peut être
  - *textuel* (ex Unix : `rm *.o`)
  - *graphique* (ex : déplacer l'icone d'un fichier vers la corbeille)

Algorithme de principe de l'interpréteur

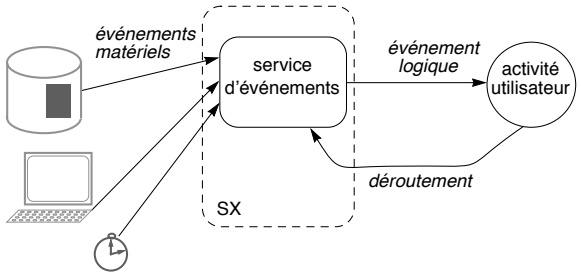
```
Interpréter(){
  while (true) {
      écran.Afficher(">") ;
      Commande c = Ligne.Lire() ;
      if (c.valide()) c.Exécuter() ;
      else écran.Afficher(c.Erreur);
  }
}
```

Interfaces du système système d'exploitation (SX) (2/2)

***Interface programmatique** (ou API : Application Programming Interface)*

Fournit un environnement d'exécution abstrait (simplifié) aux programmes dont l'exécution est gérée par le système

- service de gestion et d'accès aux ressources et activités
  → bibliothèque de procédures (appels systèmes)
- filtrage (abstraction) des événements matériels
  → service d'événements logiques (exemple : signaux Unix)



Exemple d'usage des interfaces (Unix)

But : recopier un fichier dans un autre

Interface programmatique (en C) :

le fragment de code ci-contre réalise la copie en utilisant les procédures read() et write() (de l'API système).

```
...
while (bytesread = read(from_fd,buf,BLKSIZE)){
 if ((bytesread == -1)&&(errno != EINTR))break;
 else if (bytesread > 0){
  bp =buf;
  while(byteswritten=write(to_fd,bp,bytesread)){
   if ((byteswritten == -1)&&(errno != EINTR))
     break;
   else if (byteswritten == bytesread)break;
   else if (byteswritten > 0){
     bp += byteswritten;
     bytesread -= byteswritten;
   }
  }
  if (byteswritten == -1) break;
 }
}
...
```

Interface de commande :

le programme cp, lancé à partir de l'interpréteur de commande réalise directement la copie :

```
cp fich1 fich2
```

Documentation en ligne : commande `man`

- `man -s 1 <nom de la commande>` : commandes (option par défaut)
- `man -s 2 <nom de la commande>` : appels système

Plan

Rôle d'un SX : permettre de « mieux » utiliser le matériel

Fonctions / Modèle } simples → { Fonctions évoluées { efficacité / confort d'utilisation ; Modèle... aussi simple que possible

Modèle initial ≈ ordinateur individuel

- Modèle simple
  - unité de temps : exécution du traitement « en temps réel », sans interruption
  - unité d'action : exécution exclusive d'un unique traitement à la fois, sans partage
  - unité de lieu : exécution localisée sur une machine
  - identité entre constructeur, programmeur et utilisateur (pas de problème de communication entre les différents rôles)
- Fonctions du SX = bibliothèque d'accès aux ressources

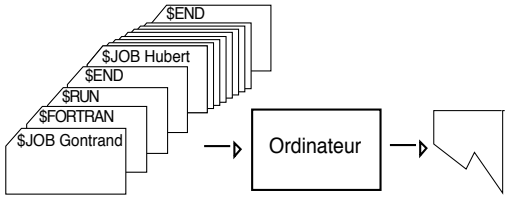
Accroître la part de temps processeur consacrée aux calculs (1/3)

2ème génération (avant 1965) : Univac 1103, IBM 7030

Traitements par lots *(temps différé, batch)*

Idée : déléguer la saisie des programmes et des commandes

→ *cartes de commande* pour { séparer les travaux (lots) ; décrire les traitements à lancer



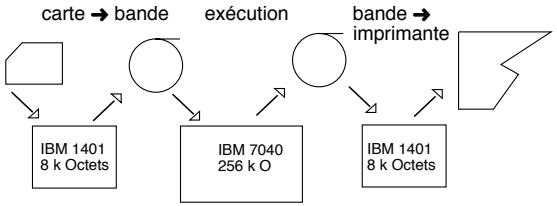
- Abandon de l'interactivité (le travail doit être préparé)
- Séparation constructeur/opérateur/programmeur

Accroître la part de temps processeur consacrée aux calculs (2/3)

Découplage entre calcul et Entrées/Sorties (E/S)

Idée : réduire le temps consacré par l'UC à la gestion des échanges avec les périphériques (E/S)

E/S synchrones *: l'UC effectue les calculs* ***et*** *gère les E/S*

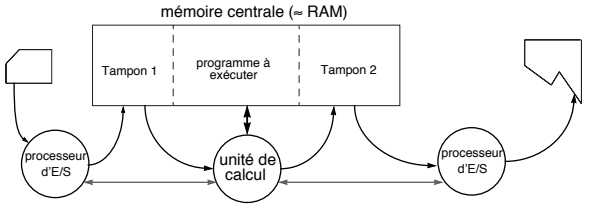


- Utiliser des périphériques rapides → temps d'E/S réduit
- Préparer les E/S : transférer les données des périphériques lents vers les périphériques rapides de manière indépendante de l'UC
  - → Hiérarchie de mémoires
  - → E/S « virtuelles » (format «logique» d'E/S)

Accroître la part de temps processeur consacrée aux calculs (3/3)

E/S asynchrones *: les périphériques ont leur propre processeur*

- l'UC effectue les calculs et lance les E/S (mais ne les gère pas)
- UC et processeurs périphériques partagent un tampon mémoire



Remarques

- communication entre UC et périphériques indépendants :
  - surveillance périodique (scrutation),
  - ou signal du périphérique (interruptions) (1955)
- le tampon permet d'amortir les variations de vitesse E/calculs/S
- idée analogue (IBM, 1960) : utiliser le disque comme tampon pour les travaux d'impression (spool (Simultaneous Peripheral Operation OnLine))

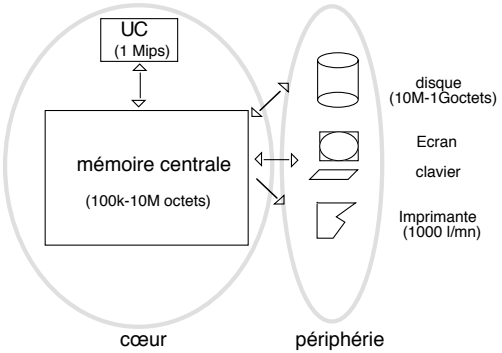
Conclusion sur la deuxième génération

- séparation programmation / administration de la machine
  - → directives accompagnant les programmes
    - → langages de commande
- découplage (asynchronisme) entre activités de calcul et d'E/S
  - échange des données via des tampons mémoire partagés
  - mécanisme d'interruption
    - → communication par événements
- mise en place de hiérarchies de mémoires,
  - mémoire centrale (rapide)/ mémoire secondaire (volumineuse)
  - stratégie : réserver la mémoire centrale aux données utiles au calcul en cours
- matériel : apparition des transistors

Équilibrer l'utilisation des ressources (1/3)

3ème génération (avant 1980) : OS 360, VM/370, VMS, CTSS, Multics

Situation



Disparité des capacités (stockage, vitesse) entre cœur et périphérie

→ un traitement ne peut utiliser *toutes* les ressources *en permanence*
→ mise en place d'activités concurrentes (système multiprogrammé)

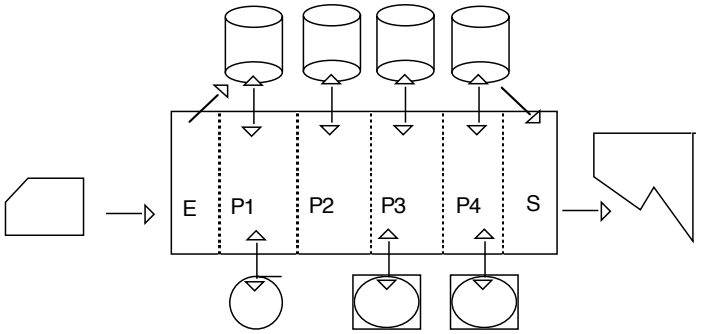
Pari : diversité des besoins des applications (sinon, effet de convoi)

Équilibrer l'utilisation des ressources (2/3)

Mener plusieurs traitements de front ⇒ partager les ressources

- temporellement (processeur, imprimante)
- physiquement (mémoire...)

Exemple : *partage physique de la mémoire centrale*

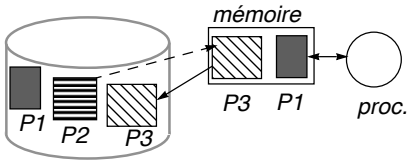


Équilibrer l'utilisation des ressources (2/3)

Exemple : *partage temporel de la mémoire centrale par va-et-vient (swapping)*

Idée

- utiliser le disque pour stocker les images mémoire de processus
- multiplexage temporel de la mémoire centrale entre images mémoire



• *pendant l'exécution de P1 :*
sauvegarde de P3, puis chargement de P2

Raffinement de l'idée
pagination (va-et-vient sur des *fragments* d'image mémoire (pages))

Conclusion sur la troisième génération

- exécution concurrente d'applications
- compétition pour les ressources (mémoire, processeur, périphériques...)
  → nécessité de protéger les ressources et contrôler leur allocation
- notion de machine virtuelle
  - chaque traitement en cours (processus) dispose de son environnement d'exécution : ensemble de ressources nécessaires
    → le SX gère l'état d'allocation des ressources aux processus :
    - par processus : ressources attendues et ressources obtenues
    - par ressource : processus élus et processus en attente
  - encapsulation des ressources par un arbitre (noyau/superviseur)
    - ayant seul directement accès aux ressources
    - disposant d'une vue globale de l'état du système
    - accès aux ressources → appels au noyau
- matériel : circuit intégrés

Améliorer le service rendu : interfaces utilisateur

- Multiprogrammation « interactive » : temps partagé
  (MULTICS ; TSO — Time Sharing Option)
  - possibilité d'exécuter un interpréteur de commandes parallèlement aux applications
    → retour au contrôle interactif de l'utilisateur sur l'avancement de ses programmes
  - glissement du calculateur au processeur de données
- Amélioration des modèles et interfaces utilisateur
  - interfaces graphiques : Alto, MacOS...
  - langages de scripts, filtres : Unix (processus, fichiers, filtres)

Améliorer le service rendu : interconnexion et répartition des systèmes

Quatrième génération (après 1980) : Unix

- apparition réseaux locaux et des micro-ordinateurs
  → usages nouveaux
  - partage de ressources (serveurs) : impression, stockage...
    → architectures client/serveur
  - communication entre utilisateurs : outils de travail coopératif
- intégration du parallélisme dans le modèle utilisateur
  parallélisme «utilitaire» (pour le SX) → parallélisme comme service
  → support au développement
  - de programmes parallèles
  - d'applications réparties sur des sites géographiquement distants

Améliorer le service rendu : systèmes répartis à large échelle

(à partir de 1990) : Amoeba, Andrew, Spring

Processeurs et réseaux
- de plus en plus performants
- de moins en moins coûteux

→ croissance très rapide
- du nombre de machines et de leurs possibilités d'interconnexion
- de la population de développeurs et d'utilisateurs
  → systèmes dynamiques, à large échelle
  - architectures ouvertes (micronoyaux, bus logiciels)
  - logiciels ouverts (libres, open source) : GNU, POSIX, Linux
  - accent mis sur la sécurité et la tolérance aux pannes
  - importance de la prise en compte du facteur d'échelle
    → développement des architectures pair à pair

Informatique ubiquitaire, enfouie, dans le nuage (à partir de 2005)

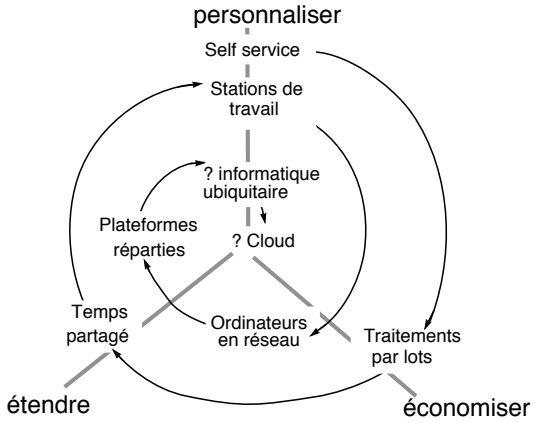
- banalisation des architectures multiprocesseurs
- croissance exponentielle des capacités de calcul et de stockage
- réseaux sans fil et très haut débit

→ mobilité et variabilité des environnements d'exécution

→ reprise et extension du modèle des systèmes classiques :
- dispositif d'interface proche de l'utilisateur, ressources et calculs virtualisés (et distants, mais la distance devient transparente)
- virtualisation des ressources poussée, pour permettre de la flexibilité dans le développement et l'exécution des applications

→ informatique comme service public : fermes de calcul et de stockage accessibles à distance (Cloud, Big data) : traitements de masses de données, calcul intensif, comptabilisés au volume.

Conclusion : éléments de prospective



- Au fur et à mesure que les «petites» machines se sophistiquent, leurs SX se compliquent (héritent de SX de « grosses » machines)

  Ex: Multics → UNIX → Linux

  ou : MS-DOS → Windows → Linux

- apparition « continue » de petites machines (PDA, systèmes enfouis), disparition progressive des très grosses machines (*mainframes*)

Bilan

Des programmes complexes qui ont un impact

- sur le génie logiciel
  - Modularité
  - Architectures en couches
  - Machine virtuelle
  - Parallélisme
- sur les architectures matérielles
  - Notion de mode d'exécution
  - Mécanismes de protection mémoire
  - Notion d'interruption, de déroutement.

Notions importantes (et/ou difficiles)

- modèle et espace d'exécution
- interface et abstraction
- ressources
- activités (processus)
- répartition