

Copyright (c)

Systèmes d'exploitation et supports architecturaux – 3A SLR F2B303A

Ronan KERYELL
Robert RANNOU

Département Informatique, TÉLÉCOM Bretagne

Novembre 2007
Version 1.18

- Copyright (c) 1986–2037 by Ronan.Keryell@enst-bretagne.fr.
This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).
- Si vous améliorez ces cours, merci de m'envoyer vos modifications ! ☺
- Transparents 100 % à base de logiciels libres (L^AT_EX,...)

Le cours I

- « Je suis contre les polys » (cf CdV) mais :
 - ▶ Cours « cliquable »
 - ▶ Dense ☺
 - ▶ Table des matières
- Pas de petites classes sur cette partie ~> posez des questions !!! ☺
- Beaucoup d'exemples basés sur Unix (accès aux sources) mais...
- Retenir idées et concepts plutôt que les exemples précis
- Partie programmation faite par Alain LEROY & Christophe LOHR
- Partie sur Windows faite par Daniel BOURGET
- Difficulté : comment contenter les candides et les cyborgs RÉSÉLiens ? ☺~> Challenge ! ☺



Problématique I

- Ubiquité de l'informatique
- Beaucoup d'applications
- Reposent sur des fonctionnalités basiques communes
 - ▶ Interagir avec l'extérieur
 - ▶ Assurer démarrage, vie et mort des programmes
 - ▶ Confort d'utilisation mais aussi de développement
 - ▶ Besoin de sécurité d'exécution
- De nombreux types d'ordinateurs existent
 - ▶ Assurer portabilité
 - ▶ Assurer pérenité des développements

Capitaliser l'expérience



Le plan I

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches d'OS
 - Entrées-sorties
 - Gestion du temps qui passe
- 4 Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
- 7 Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Introduction I

Fournit 2 types de services

- Machine virtuelle étendue plus agréable que la vraie machine brute
 - ▶ Plusieurs programmes fonctionnent en même temps
 - ▶ Plusieurs utilisateurs
 - ▶ Mémoire arbitrairement grande
 - ▶ Fichiers
 - ▶ Interfaces (graphiques) sympathiques
 - ▶ ...
- Détails cachés de manière *transparente*
- Gestion optimale des ressources
 - ▶ Processeurs
 - ▶ Mémoire
 - ▶ Périphériques d'entrée-sortie
 - ▶ Gestion de la qualité de service (surtout en mode multi-utilisateur...)



Introduction II

- Latence, temps de réponse : mode interactif
- Débit : centre de calcul
- Contraintes temps réel : gestion d'un processus industriel
- Tolérance aux pannes : centrales nucléaires, avions
- ...



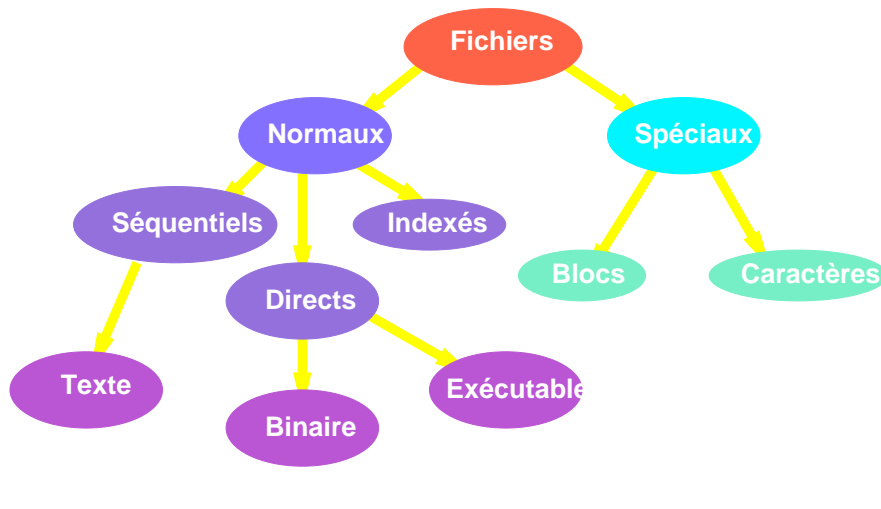
Machine Virtuelle Étendue I

- Processus : abstraction de processeur virtuel
 - ▶ Programme s'exécutant sur un processeur
 - ▶ Contexte d'exécution
 - ▶ Protection
- Atomicité : qui *paraît* insécable
 - ▶ Lecture, écriture
 - ▶ Transactions : début, fin, abandon possible sans casse
- Fichier abstrait
 - ▶ Conteneur de données, de programmes, répertoires
 - ▶ Périphérique (disquette)
 - ▶ Contrôle de n'importe quidans Unix...



Abstraction de fichier I

Dans un système imaginaire...



Exemples de fichiers et fichiers spéciaux I

- Contrôler des « périphériques » (/dev sous Unix)
 - ▶ Périphérique de stockage : /dev/fd0 (disquette),...
 - ▶ Terminal /dev/tty (clavier, écran, souris /dev/mouse, /dev/mouse1)
 - ▶ Lien de communication /dev/eth0, socket
 - ▶ cp truc /dev/lp ou copy truc prn: imprime le contenu de truc
 - ▶ cat /dev/zero > /dev/null met des 0 à la poubelle
 - ▶ cat /dev/random > a crée un fichier de caractères aléatoires
 - ▶ Mémoire (principale ou secondaire) : /dev/mem (fichier contenant une copie de la mémoire de l'ordinateur)
 - ▶ Informations sur des périphériques (/dev/sndstat) (mélange des genres historique...)
- Sockets : tuyaux
 - ▶ Entre machine (PF_INET,...)
 - ▶ xwd -root -display pigeon:0 | xwd : entre processus



Exemples de fichiers et fichiers spéciaux II

- ▶ Pipes nommés
- Processus dans /proc
 - ▶ Fichier mémoire processus
 - ▶ Fichiers utilisés par processus
- Contrôle et information système /sys



Transparence et opacité I

« C'est transparent à l'utilisateur » ≡ Il ne voit rien ~ ; c'est opaque ! ☹

- Hétérogénéité
 - ▶ Modèles et marques d'ordinateur
 - ▶ Taille des mots et rangement des octets dans les mots
 - ▶ Systèmes d'exploitation différents
 - ▶ Périphériques différents
- Localisation
 - ▶ Fichier local ou non
 - ▶ Ordinateur distant ou pas
- Migration & mobilité
 - ▶ Serveurs qui se déplacent
 - ▶ Objets migrants
 - ▶ Ressources distribuées
- Réplication (tolérance aux pannes et performances)



Transparence et opacité II

- ▶ Multiplication des serveurs
- ▶ Mécanisme de caches
- Concurrence et parallélisme
 - ▶ Sérialisabilité
- ↪ Définition de l'interface de programmation du système
 - Objets définis dans l'interface
 - Relations inter-objets ?
 - Comment communiquer ?

Transparence et opacité III

Exemple de concept : partage de données dans une variable globale (locale ou distante)

Le système va masquer la réalité :

- Réseau haut-débit
- Mémoire globale
- Mécanismes de cache à cohérence forte (atomique)

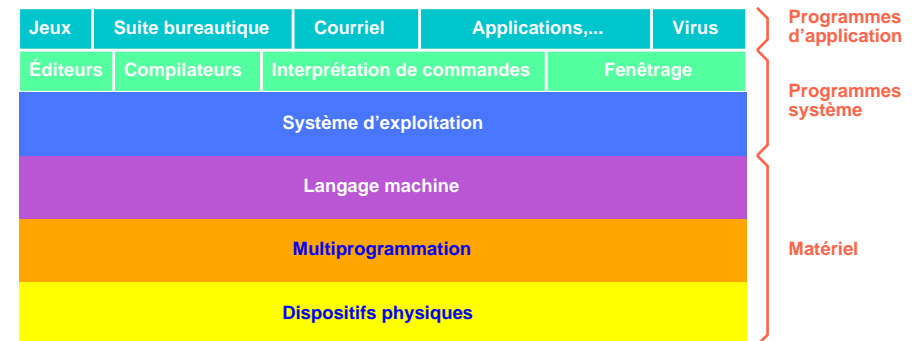
Nirvana des systèmes I

- Vrai système distribué
- Pas un système réseau
- Vision monosite
- Espace de nommage unique (infini...)
- ⚠ ¿ Comment concilier transparence et performance ?

↪ **Compromis cacher ≠ gérer...**

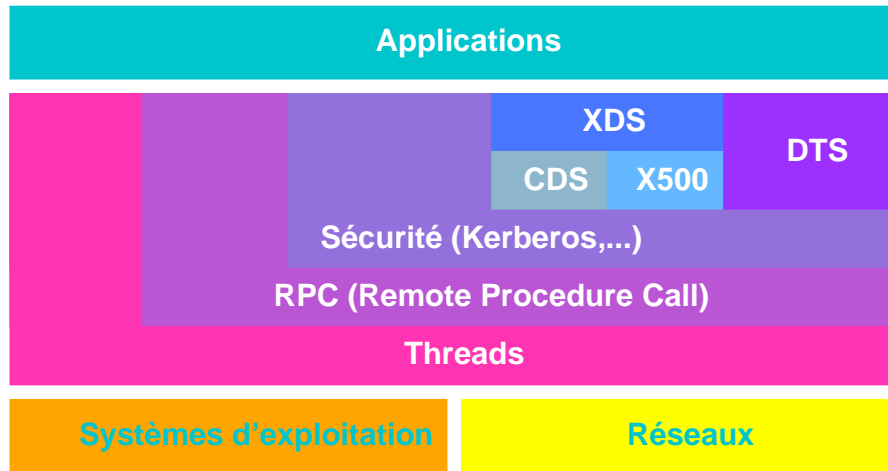
⚠ Bien comprendre comment cela marche si on ne veut pas tomber dans des pièges cachés... ↪ Ce cours !

Un ordinateur dans une perspective logicielle I

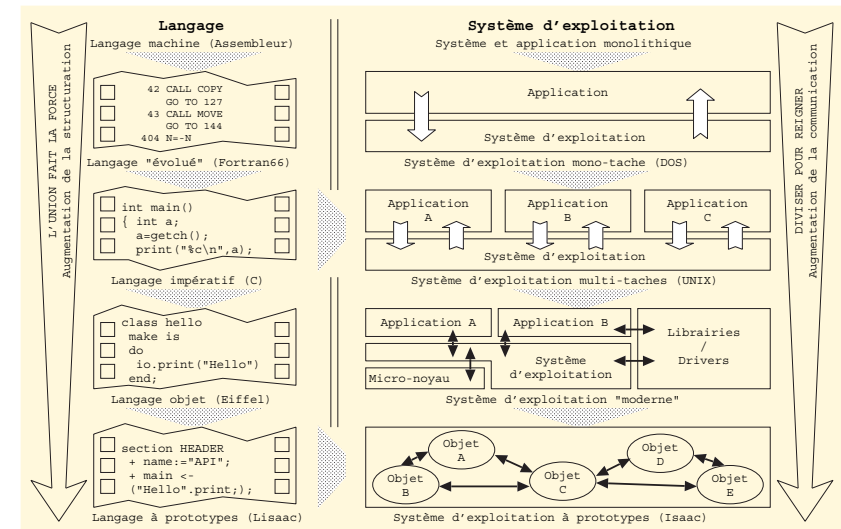


⚠ Le grand public ne voit que le haut et le fenêtrage...

DCE — Distributed Computing Environment I



Évolution logicielle I



Évolution logicielle II

Extrait de « Le projet Isaac : une alternative objet de haut niveau pour la programmation système »,
Benoît Sonntag, CFSE2005



Pourquoi ce cours en 1A/2A... puis en 3A ? I

- ; Parce que !
- Informatique et donc systèmes d'exploitation partout
- Vernis aux futurs ingénieurs dans la salle
- Comprendre la problématique
- 🚧 Sirènes graphiques : une interface graphique ne fait que cacher la complexité qui réapparaît en cas de problème...
- Nécessité de comprendre comment cela fonctionne !
- Plein d'astuces réutilisables dans la vie de tout les jours
 - ▶ Programmation
 - ▶ Optimisation
 - ▶ Gestion de production
 - ▶ Gestion de son potager
 - ▶ Gestion de son agenda
 - ▶ Gestion de son carnet de bal
 - ▶ ...



Bibliographie I

- « *Systèmes d'exploitation* », Andrew TANENBAUM, 2^{ème} édition, Pearson Education, 2003
- « *Virtual Machines — Versatile Platforms for Systems and Processes* », James E. SMITH & Ravi NAIR. Morgan Kaufmann/Elsevier, 2005 (contient aussi un cours d'architecture à la fin)
- Les bases (un peu d'histoire) : « *Systèmes d'exploitation des ordinateurs : principes de conception/CROCUS* », Paris : Dué, J. Briat, B. Canet, E. Cleemann, J.C. Derniame, J. Ferrié, C. Kaiser, S. Krakowiak, J. Mossière, J.-P. Verjus, 1978
<http://cnum.cnam.fr/fSYN/8CA2680.html>

- Cours du CNAM

▶ <http://deptinfo.cnam.fr/Enseignement/CycleProbatoire/SRI/Systèmes>



Bibliographie Linux I

- Que les sources soient avec vous ! ☺
 - ▶ Récupérer le noyau Linux <http://kernel.org>
 - ▶ Le code hypertextuel <http://lxr.linux.no> (et plus proche de nous <http://kernel.enstb.org>)
 - ▶ Penser à utiliser les *tags* dans son éditeur favori
 - `make TAGS` crée un fichier TAGS pour les Emacs
 - `make tags` crée un fichier tags pour les vi
- Livres
 - ▶ « *Linux Kernel Development* » Robert LOVE. Novell Press, 2^{ème} édition, 12 janvier 2005
 - ▶ « *Linux Device Drivers* », Jonathan CORBET, Alessandro RUBINI et Greg KROAH-HARTMAN. O'Reilly, 3^{ème} édition, février 2005.
<http://lwn.net/Kernel/LDD3>
- Gazette de discussion sur Linux <http://www.kernel-traffic.org>
- <http://lwn.net/Kernel> Linux kernel development



Bibliographie II

- ▶ <http://deptinfo.cnam.fr/Enseignement/CycleA/AMSI>
- ▶ <http://deptinfo.cnam.fr/Enseignement/CycleSpecialisation/ACCOV>
- ▶ <http://deptinfo.cnam.fr/Enseignement/CycleSpecialisation/SAR>
- <http://cui.unige.ch/~billard/systemeII/> cours en français de David Billard
- « *UNIX Internals : The New Frontiers* », Uresh Vahalia, October, 1995, Prentice Hall Engineering/Science/Mathematics
http://www.phptr.com/ptrbooks/esm_0131019082.html
- « *Operating systems : a modern perspective* », Gary J. Nutt, Addison Wesley, 1997
- « *The Magic garden explained : the internals of UNIX System V release 4 : an open systems design* », B. Goodheart ; J. Cox, Prentice-Hall, 1994



Bibliographie Linux II

- The linux-kernel mailing list FAQ <http://www.tux.org/lkml> (un peu vieux)
- Index of Documentation for People Interested in Writing and/or Understanding the Linux Kernel
<http://jungla.dit.upm.es/~jmseyas/linux/kernel/hackers-docs.html>
- <http://lwn.net/Articles/2.6-kernel-api> 2.6 API changes. A regularly-updated summary of changes to the internal kernel API
- <http://lwn.net/Articles/driver-porting> The Porting drivers to 2.6 series : over 30 articles describing, in detail, how the internal kernel API has changed in the 2.6 release
- « *Linux Internals* », Moshe Bar, 2000, The McGraw-Hill Companies, Inc.
- « *Linux Kernel 2.4 Internals* », Tigran Aivazian,
<http://www.moses.uklinux.net/patches/lki.html>



Bibliographie Linux III

- « The Linux Kernel 2.0.33 », David A Rusling,
<http://www.linuxhq.com/guides/TLK/tlk.html>

Historique I

<http://www.computerhistory.org>
<http://histoire.info.online.fr>

- 1945–1955
 - ▶ Pas de système d'exploitation
 - ▶ Tout faire à la main
- 1955–1965
 - ▶ Langages de plus haut niveau
 - ▶ Générations de machines
- 1965–1985
 - ▶ Multiprogrammation
 - ▶ Temps partagé
 - ▶ Gros systèmes
- 1985–...
 - ▶ Ordinateurs personnels
 - ▶ Stations de travail

Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion

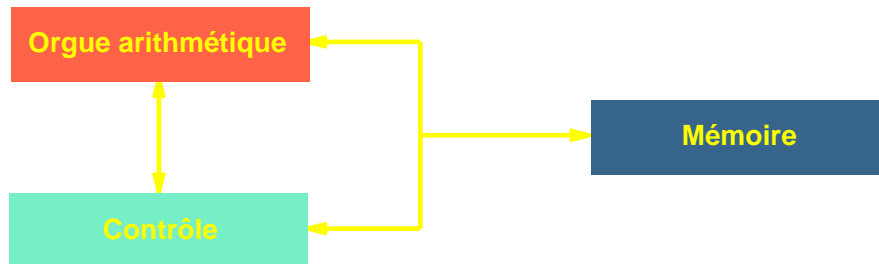
Historique II

- ▶ Réseaux (Internet)
- ▶ Systèmes distribués

<http://www.computer.org/50/history>

1945–1955 : ordinateur séquentiel I

Principes de ECKERT, MAUCHLY et VON NEUMANN, années 1940 :
une mémoire contient des données *ET* un programme



+ Entrées-sorties sur un des chemins de données

↪ concept d'ordinateur universel en 1946 : ENIAC. 18000 tubes à vide, 30 tonnes, 174 kW, 5000 +/s, 333 ×/s à 10 chiffres, modifié plus tard avec programme en mémoire

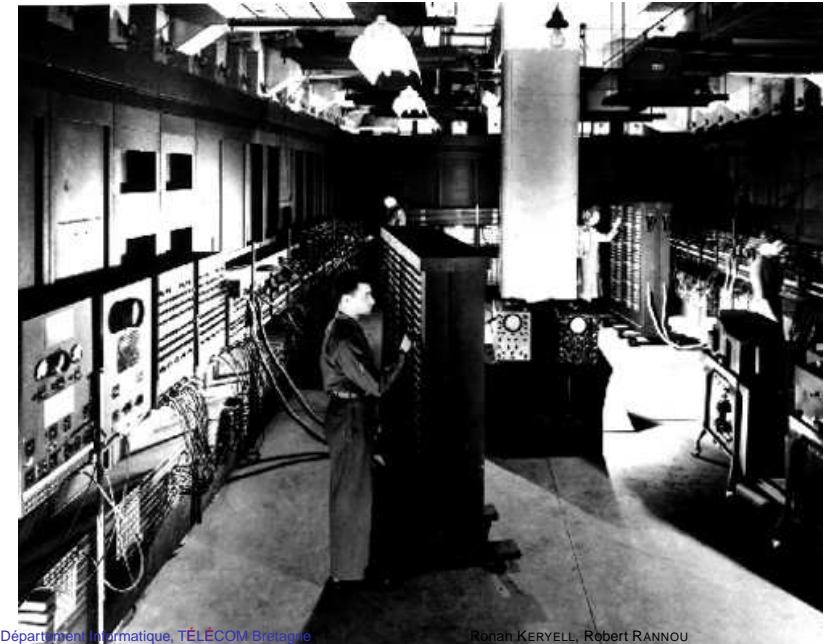


1955–1965 : Jusqu'à l'OS I

- Toujours qu'un seul utilisateur, réservation par tranche horaire
- Traitement par lots : faire la queue
- Mono-programmation
- 1957 : John BACKUS (IBM), langage & compilateur Fortran ↪ ↗
vitesse de programmation
- 1957 : Seymour CRAY, CDC1604, supercalculateur tout transistor
- 1957 : Disque dur IBM 305 RAMAC (≈ 2 réfrigérateurs)
- 1959 : Bull Gamma 60, instructions de parallélisme (parce que la mémoire était trop rapide par rapport au processeur !)
- Spécialisation des tâches pour optimiser globalement les coûts en utilisant plusieurs ordinateurs au lieu d'un seul plus gros ;



1945–1955 : ordinateur séquentiel II



1955–1965 : Jusqu'à l'OS II

- ▶ Petit IBM 1401 pour faire du transfert cartes perforées ↪ bandes magnétiques (données)
- ▶ Gros calcul sur IBM 7094
- ▶ Petit IBM 1401 pour faire du transfert bandes magnétiques ↪ imprimantes (données)



1965–1985 : Multiprogrammation, temps partagés I

- Gros systèmes
- Multiprogrammation : plusieurs programmes résident en mémoire
- Temps partagé : terminaux interactifs
- *spool* : *Simultaneous Peripheral Operation On Line* : on empile les requêtes d'impression et on fait autre chose
- Mémoire
 - ▶ Pagination : gestion plus fine simplifiée
 - ▶ Segmentation : différents espaces d'adressage
- Technologie
 - ▶ Transistors puis circuits intégrés, micro-processeur \equiv brique de base de l'informatique
 - ▶ Disques magnétiques ↗
 - ▶ Mécanisme d'interruption



Unix Story I

- En attendant la suite, Ken Thomson de BTL écrit un jeu *Space Travel* qu'il fait tourner sur un PDP-7 (machine pas trop chère)
- Problème : pas d'environnement de développement sur PDP-7 et nécessité de faire de l'assemblage croisé sur Honeywell 635 roulant GECOS
- Pour faciliter le développement du jeu, développement d'un système d'exploitation pour le PDP-7 : système de fichier simple (*s5fs*), système de gestion de processus, interpréteur de commande (*shell*)
- Le système devient auto-suffisant et est nommé *Unix* en 1969, jeu de mots en opposition à *Multics*
- Portage d'Unix sur PDP-11 et développement de l'éditeur de texte *ed* et du système de composition de texte *runoff*



1965–1985 : Multiprogrammation, temps partagés II

- ▶ Accès direct à la mémoire
- ▶ Écrans \pm graphiques
- ▶ Modems
- ▶ « Grosses » mémoires
- ▶ Loi de Moore
- Projet MULTICS : *MULTiplexed Information and Computing Service*
 - ▶ MIT, Bell Telephone Laboratories de AT&T, General Electric
 - ▶ Notion de « Computer Grid » (qui revient de nos jours...)
 - ▶ Offrir puissance de calcul pour toute la ville de Boston : le Minitel avant l'heure
 - ▶ Plus difficile que prévu \leadsto abandonné mais grande influence dans la communauté

<http://www.multicians.org/>



Unix Story II

- Développement de langage interprété *B* utilisé pour développer les outils
- Dennis Ritchie fait évoluer le langage en *C* dont le succès a largement dépassé le cadre d'Unix
- 1972 : 10 machines sous Unix...
- Unix réécrit en *C* en 1973 et la distribution version 4 contient elle-même *cc*
- L'université de Berkeley récupère une licence (gratuite à cause d'un procès antitrust de 1956 entre AT&T et Western Electric Company)
- Travaux à SRI de Doug ENGELBART sur interfaces graphiques dans les années 1960 repris ensuite chez Xerox PARC
- La version 7 de 1979 est la première version réellement portable



Unix Story III

- Beaucoup d'améliorations fournies par les utilisateurs eux-mêmes (de même que BSD & Linux maintenant) favorisé par le côté non commercial
- MicroSoft et Santa Cruz Operation collabore sur un portage pour i8086 : Xenix
- Portage sur machine 32 bits (Vax-11) en 1978 : UNIX/32V qui est récupérée par Berkeley (<http://www.lpl.arizona.edu/~vance/www/vaxbar.html> VaxBar)
- Rajout d'utilitaires (csh de Bill Joy) et d'un système de pagination
- La DARPA donne un contrat à Berkeley pour implémenter IP : BSD
 - ▶ Dernière version en 1993 : 4.4BSD. En tout : apport des *socket*, d'IP, d'un *fast file system* (FFS), des signaux robustes, la mémoire virtuelle



Unix Story IV

- ▶ Société BSDI créée pour vendre 4.4BSD *lite* en 1994, débarrassé de tout code d'origine AT&T
- 1982 : loi antitrust qui éclate AT&T en baby-Bell dont le AT&T Bell Laboratories qui peut alors commercialiser Unix
 - ▶ 1982 : System III
 - ▶ 1983 : System V
 - ▶ 1984 : System V release 2 (SVR2)
 - ▶ 1987 : System V release 3 (SVR3) introduit les IPC (InterProcess Communications : mémoire partagée, sémaphores), les *STREAMS*, le *Remote File Sharing*, les bibliothèques partagées,...
 - ▶ Base de nombreux Unix commerciaux
- 1982 : Bill Joy quitte Berkeley pour fonder Sun Microsystems. Adaptation de 4.2BSD en SunOS qui introduit le *Network File System*, interface de système de fichier générique, nouveau mécanisme de gestion mémoire



1985– : Micro-ordinateurs I

- Montée en puissance du microprocesseur
- Micro-ordinateurs \equiv ordinateur à base de microprocesseur : quasiment tout ordinateur
- Explosion du multi-fenêtrage \rightsquigarrow interfaces plus sympathiques
- Gros ordinateur (parallèle) \rightsquigarrow rassemblements de nombreux processeurs
- Apparition des systèmes distribués
- Stations de travail
- Généralisation des réseaux d'abord pour partager disques coûteux : NFS
- WWW devenu synonyme d'Internet 30 après
- 1981 : le premier PC



1985– : Micro-ordinateurs II

- ▶ Grand retour en arrière (...pour les spécialistes) : mono-programmation : MS-DOS 1.0, PC-DOS 1.0
- ▶ Système de fichier primitif (...mais robuste), pas de répertoire,...
- Milieu des années 1980 à Carnegie-Mellon University développe Mach, un micro-noyau avec des serveurs implémentant une sémantique 4BSD. OSF/1 & NextStep sont basés sur Mach
- Steve Jobs crée NeXT Computer après avoir été écarté d'Apple
- 1987 Andrew Tanenbaum publie « *MINIX : A UNIX Clone with Source Code for the IBM PC* »
<http://www.cs.vu.nl/~ast/minix.html>
- 1987 : AT&T achète 20% de Sun \rightsquigarrow prochaines version de SunOS basées sur System V : SunOS 5 (Solaris 2)



1985– : Micro-ordinateurs III

- NeXT Computer avec NeXTStep 0.8 (mélange de Mach 2.5 4.3BSD, interface graphique basée sur Display Postscript, programmation en Objective-C)
- 1989 : co-développement AT&T-Sun de SVR4 : inclut les fonctionnalités de SVR3, 4BSD, SunOS & Xenix. Création d'Unix Systems Laboratories pour développer et vendre Unix
- Tim Berners-Lee crée le premier brouteur WWW sur NeXT au CERN en 1990
- Novell achète une partie d'USL en 1991 pour développer UnixWare (Unix + Netware) et tout USL en 1993
- Linus Torwald récupère Minix sur PC i80386 et le développe en Linux en 1991
http://www.dina.dk/~abraham/Linus_vs_Tanenbaum.html
- NeXTStep 486 en 1992



1985– : Micro-ordinateurs IV

- 1994 : première version publique de Linux : 1.0. Développements pris en main par des programmeurs répartis sur Internet. Intégration des utilitaires GNU
- Arrivée de NT (*New Technology*) : mélange de MS-DOS, MacOS, VMS et Unix
- Novell cède la marque Unix au X/Open puis Sun rachète les droits de SVR4 à Novell en 1994
- NeXT et Sun définissent OpenStep. Repris par la suite dans SunOS, HP-UX et NT en partie. Continue dans <http://www.gnustep.org>
- Chorus, société française, développe un micro-noyau
- Rachat de Chorus par Sun. → JavaOS ?



1985– : Micro-ordinateurs V

- Un système Unix \equiv programmes utilisateurs + bibliothèques + utilitaires + système d'exploitation qui fournit le support d'exécution et les services
- Unix tourne sur toutes les plates-formes depuis les systèmes embarqués jusqu'aux supercalculateurs massivement parallèles
- Mac OS X server sort en 1999 revampant NeXTStep et OpenStep
- 2000 : Windows 2000 : plus stable, plus gros. Exposé passionnant sur l'ingénierie du projet dans « From NT OS/2 to Windows 2000 and Beyond - A Software-Engineering Odyssey »
<http://www.usenix.org/events/usenix-win2000/invitedtalks>
- 2001 : Windows XP : encore plus stable, encore plus gros
- MacOS X 10.3 (Panther) en octobre 2003
<http://www.kernelthread.com/mac/osx>



Le plan



Concepts de base I

- Multiprogrammation
- Temps partagé
- Pagination & segmentation

Multiprogrammation I

- \leadsto Mieux utiliser processeur
- Partage la mémoire entre plusieurs processus
- Recouvrement du temps d'entrées-sorties avec du calcul
- Protection des programmes entre eux



Code translatable I

- ⚠ Éviter des conflits d'adresses entre programme
 - Ne pas figer (à la compilation) les adresses des objets manipulés par un programme, les bouts d'exécutables (bibliothèques de fonctions, . . .)
 - Solutions possibles
 - ▶ Calculer les adresses au moment du chargement
 - ▶ Utiliser un mécanisme matériel : adresse relative par rapport à une base
 - ▶ Tout sous-traiter à une unité de traduction d'adresse
 - La solution logicielle est tout de même utilisée pour
 - ▶ Compilation séparée : fusion de .o par édition de liens
 - ▶ Bibliothèques dynamiques
 - ▶ Chargement de code à la volée
 - ▶ Compilation de code à la volée

Code translatable II

- \exists Langages interprétés : JVM pour Java, PostScript, Forth, PERL, Python, SmallTalk, BibTeX, . . .
- Accélération avec une phase de compilation préalable (JIT : *Just In Time* compilation pour Java/JVM)
- Cache de code précompilé
- ▶ ⚠ Virus par débordement de variables : arriver à écraser des données pour faire exécuter du code quelconque (Cf mon cours 3A IT S301)

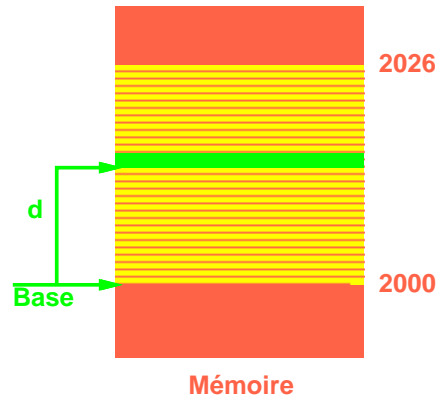
Base et déplacement I

- Remplacer $*p$ par $*(base + d)$
- Changer de place en mémoire : changer base
- Protection mémoire : vérifier toujours

$$d \in [0, d_{\max}]$$

Registres $\begin{cases} base = 2000 \\ d_{\max} = 26 \end{cases}$
 $\Rightarrow 2000 \leq base + d \leq 2026$

- Typiquement dans processeur à mode d'adressage compliqué : CISC



Ronan KERYELL, Robert RANNOU

49 / 309

Base et déplacement II

8b 95 74 e3 ff ff mov 0xffffe374(%ebp),%edx

sur x86, mais sans vérification des bornes



Ronan KERYELL, Robert RANNOU

50 / 309

Temps partagé – tourniquet I

- Donner illusion de disposer d'une machine à soi tout seul
- Lié à l'invention du terminal interactif
- Changer de programme à chaque quantum de temps
- Privilégier les requêtes peu gourmandes par rapport aux programmes de calcul : faire des heureux facilement avec *caisse moins de 10 Articles*

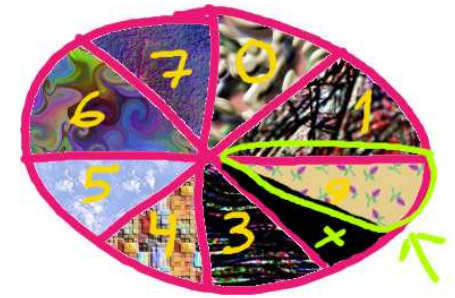


Ronan KERYELL, Robert RANNOU

51 / 309

Blocage dans le tourniquet I

- Blocage possible d'un processeur avant la fin de son quantum de temps
- Libère du temps pour les autres



Ronan KERYELL, Robert RANNOU

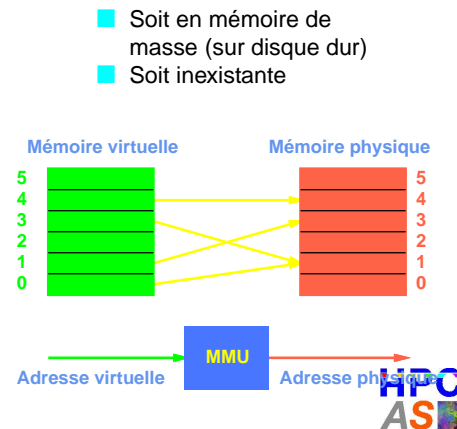
52 / 309

Pagination I

- Mémoire trop petite pour contenir tout un programme
- ~> Concept de mémoire virtuelle : fournir à l'utilisateur une mémoire géante

- Casser la mémoire physique en pages

- ▶ Rajouter composant de traduction entre mémoire virtuelle et mémoire physique : *Memory Management Unit*
- ▶ Programmes utilisent des adresses virtuelles
- ▶ Page de mémoire virtuelle
 - Soit mémoire physique



Pagination II

- Fonctionne bien avec le principe de multiprogrammation : si une page n'est pas là, exécute un autre programme en attendant son chargement

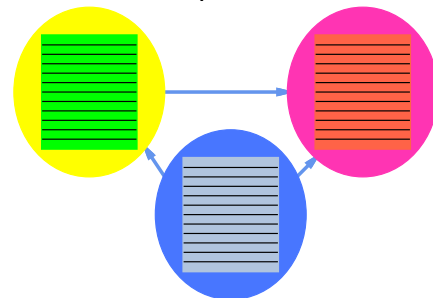
Segmentation I

- Programme, système, applications : non monolithiques

- ▶ Modules
- ▶ Couches
- ▶ Objets
- ▶ ...

~> Segments de mémoire indépendants

- Addressage propre à chaque segment : permet de croître arbitrairement sans conflit (une baguette n'a que 2 croûtons... ☺)
- ▶ Segment pour la pile
- ▶ Segment pour le code



Segmentation II

- ▶ Segment pour des données globales
- ▶ Segment pour le système d'exploitation
- ▶ Segment pour les new Carottes()
- ▶ Segment pour les new Lapin()
- ▶ ...
- Protection spécifique à chaque segment : lecture, écriture, exécution
- Moins fondamental avec grands espaces d'adressage sur 64 bits
- Reste des scories dans x86 qui ont repris le mode d'adressage du Mitra 15 (i8086) puis du Mitra 125 (386) de CII avec les mêmes noms !
DS, ES, CS, ...

Le plan

- 1 Historique
- 2 Concepts de base
- 3 **Concurrence & Parallélisme**
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Concurrence et parallélisme I

- Applications & algorithmes complexes avec des choses indépendantes
- Simplification de la programmation : détacher les tâches les unes des autres
- Gestion explicite des tâches
 - ▶ Co-routines
 - ▶ `setjmp()/longjmp()`
 - ▶ Gestion à la main... ☹
- Si plusieurs processeurs : possibilité d'exécuter plusieurs tâches en parallèle



Le plan

- 1 Historique
- 2 Concepts de base
- 3 **Concurrence & Parallélisme**
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Tâches I

- Besoin de simplification de la programmation (productivité)
- Abstraction de la notion de tâche
 - ▶ Programme s'exécutant (actif) sur un processeur virtuel ~→ processus (lourd) dans Unix
 - ▶ Contexte d'exécution : ressources virtuelles (espace mémoire, fichiers...)
 - ▶ Atomicité possible (transactions...)
- Gestion directe par le système d'exploitation



Le plan

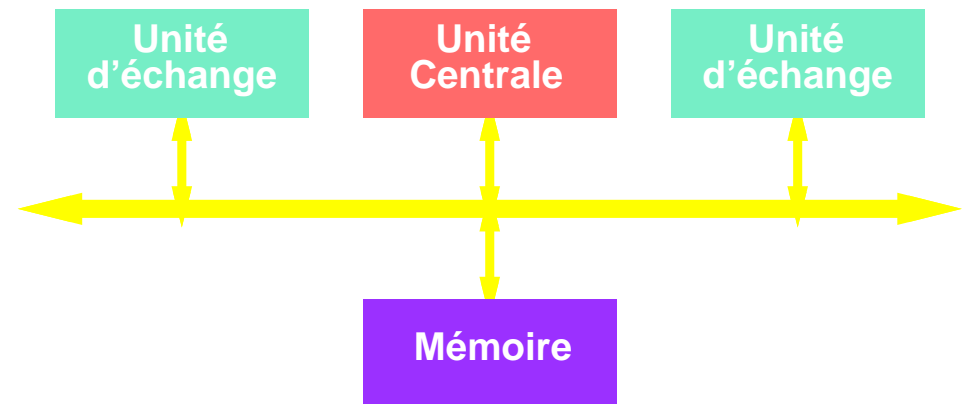
- | | |
|---|--|
| <ul style="list-style-type: none"> 1 Historique 2 Concepts de base 3 Concurrence & Parallélisme <ul style="list-style-type: none"> ● Introduction ● Hypothèses sur architecture matérielle ● Notion de noyau ● Processus lourds & légers, <i>threads</i> ● Plus de détail des tâches dans Linux ● Entrées-sorties ● Gestion du temps qui passe ● Ordonnancement | <ul style="list-style-type: none"> 4 Gestion mémoire 5 Virtualisation 6 Les entrées-sorties <ul style="list-style-type: none"> ● Disques ● Formattage ● RAID ● ZFS ● Pilote de périphérique 7 Systèmes de fichiers distants <ul style="list-style-type: none"> ● NFS 8 Conclusion |
|---|--|

Schéma d'un monoprocesseur II

- Unité centrale : interprète instructions des programmes applicatifs et du système d'exploitation
- Unités d'échange (contrôleur, canal) : assurent le suivi des entrées-sorties
- Mémoire : accessible via le bus par unité centrale et unités d'échange
Pour décharger le processeur : accès direct à la mémoire par unités d'échange

Direct Memory Access (DMA)

Schéma d'un monoprocesseur I

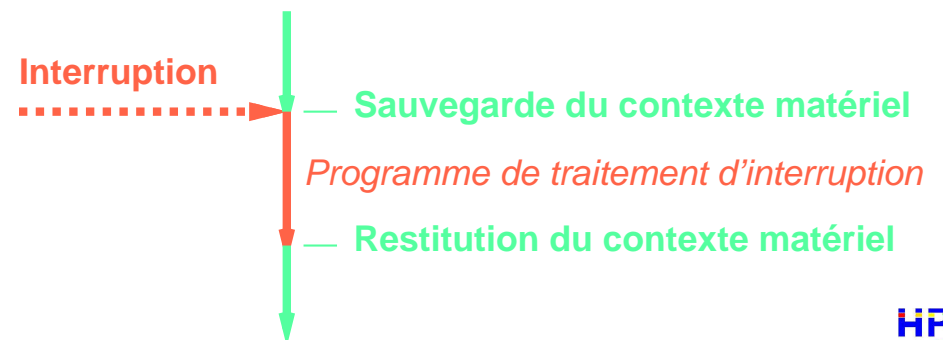


Mécanisme d'interruptions I

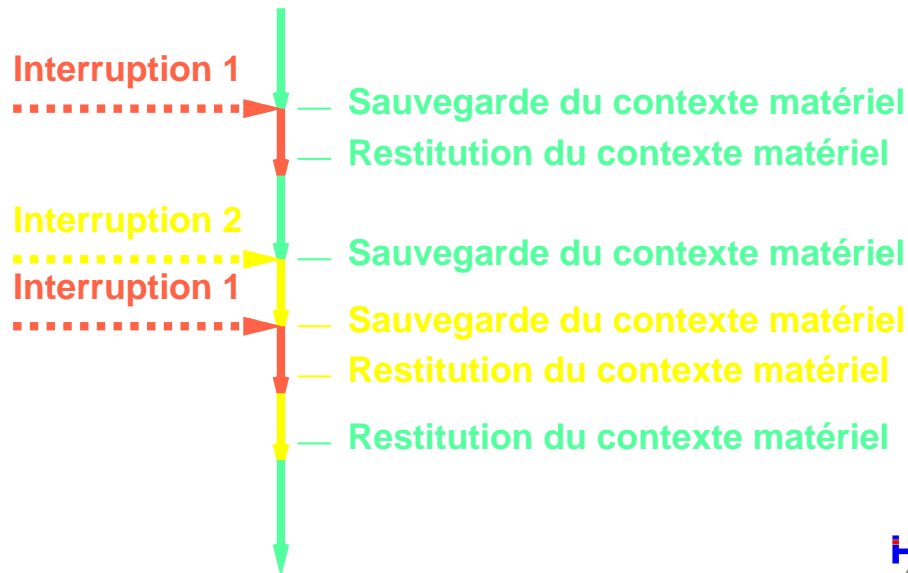
Besoin de surveiller et réagir

Comment éviter de passer son temps à surveiller les E/S ?

Interruption \equiv événement prioritaire qui interrompt déroulement normal d'un programme en cours d'exécution sur une unité centrale



Mécanisme d'interruptions II



Types d'interruptions :

• Interruptions matérielles

Systèmes d'exploitation et supports architecturaux – 3A SLR F2B303A

Concurrence & Parallélisme

Hypothèses sur architecture matérielle

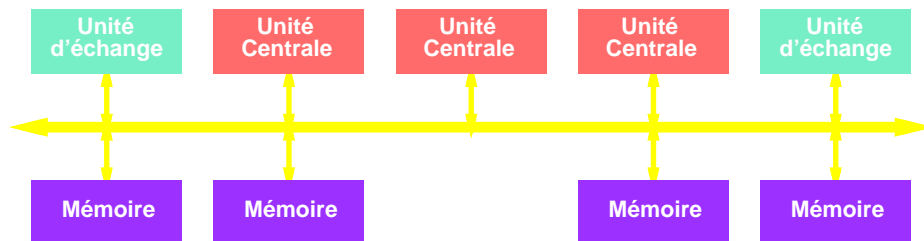
Ronan KERYELL, Robert RANNOU

65 / 309



Schéma d'un multiprocesseur I

Mémoire commune



- Plusieurs unités centrales pour augmenter la puissance : interprètent instructions des programmes applicatifs et du système d'exploitation
- Mémoire : plusieurs bancs pour augmenter taille et débit



Mécanisme d'interruptions — détail I

- Contexte matériel : registres à sauvegarder quand survient une interruption
 - ▶ Compteur ordinal
 - ▶ Sommet de pile de programme
 - ▶ Mot d'état du programme
 - ▶ État du pipeline interne
- Dépend du processeur
- Possibilité de masquer les interruptions
 - Éventuellement automasquage pour éviter récursion infinie
- Niveaux (priorités) d'interruptions
 - Parer au plus urgent
- Structure de pile pour accepter un nombre arbitraire d'interruptions en cours

Département Informatique, TÉLÉCOM Bretagne

Systèmes d'exploitation et supports architecturaux – 3A SLR F2B303A

Concurrence & Parallélisme

Hypothèses sur architecture matérielle

Ronan KERYELL, Robert RANNOU

66 / 309



Interruptions & multiprocesseur I

Hypothèses :

- Chaque processeur peut masquer localement les interruptions
- Interruption matérielle aiguillée vers processeur
 - ▶ Ne masquant pas interruptions
 - ▶ Si possible celui exécutant le processus de plus faible priorité



Classes d'architectures matérielles I

- Architectures centralisées
 - ▶ Monoprocresseurs
 - ▶ Multiprocresseurs à mémoire partagée
- Architectures réparties
 - ▶ Multiprocresseurs à mémoire distribuée : communication par messages
 - ▶ Réseaux



Notion de noyau I

- Couche de logiciel, voire de matériel
- Met en œuvre le concept de processus
- Tourne dans une machine physique ou virtuelle



Le plan

- | | |
|---|--|
| <ul style="list-style-type: none"> 1 Historique 2 Concepts de base 3 Concurrence & Parallélisme <ul style="list-style-type: none"> • Introduction • Hypothèses sur architecture matérielle • Notion de noyau • Processus lourds & légers, <i>threads</i> • Plus de détail des tâches dans Linux • Entrées-sorties • Gestion du temps qui passe • Ordonnancement | <ul style="list-style-type: none"> 4 Gestion mémoire 5 Virtualisation 6 Les entrées-sorties <ul style="list-style-type: none"> • Disques • Formattage • RAID • ZFS • Pilote de périphérique 7 Systèmes de fichiers distants <ul style="list-style-type: none"> • NFS 8 Conclusion |
|---|--|



Noyau Unix I

Plus qu'un noyau minimal : presque tout le système d'exploitation

- Gestion mémoire
- Systèmes de gestion de fichiers
- Entrées-sorties de bas niveau
- Entrées-sorties de haut niveau
- Sécurité
- ...

Mais évolution vers des Unix modulaires...



Notion de micro-noyau I

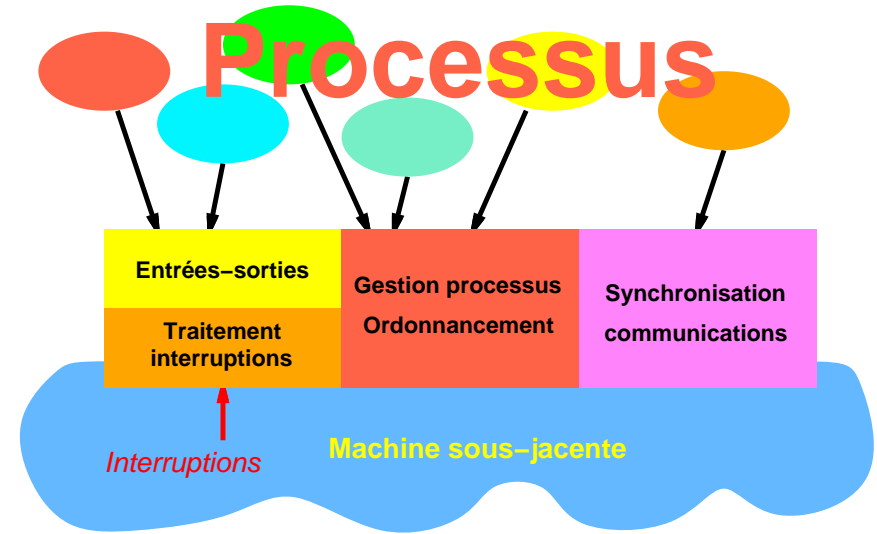
Mises en œuvres de la concurrence I

Faire tourner plusieurs choses

- Sur machine nue
 - ▶ Système d'exploitation lui-même
 - ▶ Application de contrôle de procédé, temps réel « dur »
- Sur machine virtuelle (qui permet la concurrence)
 - ▶ Temps réel : émulation d'un système d'exploitation sur un autre système d'exploitation
 - ▶ Temps simulé : simulation numérique, prévision météorologique,...

Dans la suite on se focalise sur concurrence et systèmes d'exploitation

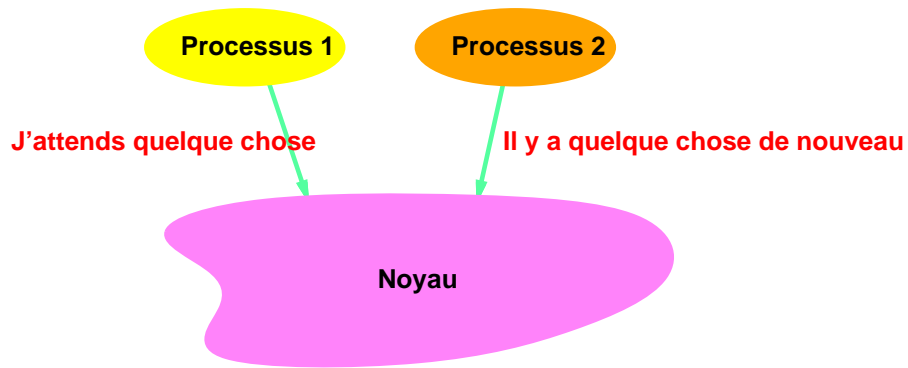
Notion de micro-noyau II



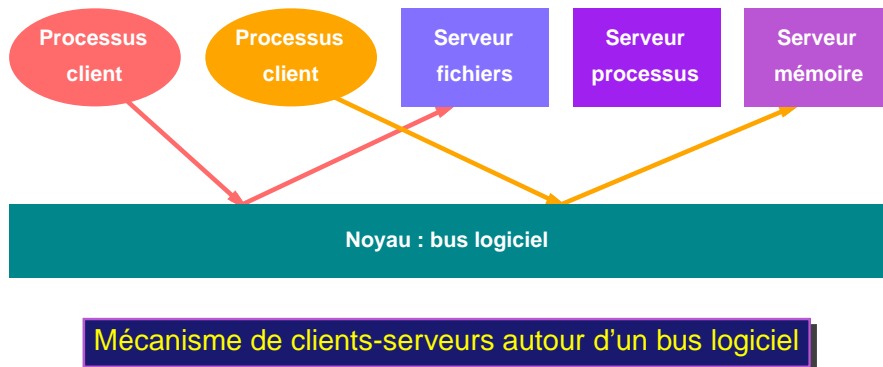
Fonctions d'un micro-noyau I

- Partager le ou les processeurs entre les processus
- Gérer naissance et mort des processus
- Cacher les interruptions (entrées-sorties et horloge)
- Fournir un service d'entrées-sorties de bas niveau
- Fournir outils d'exclusion mutuelle, de synchronisation et de communication inter-processus

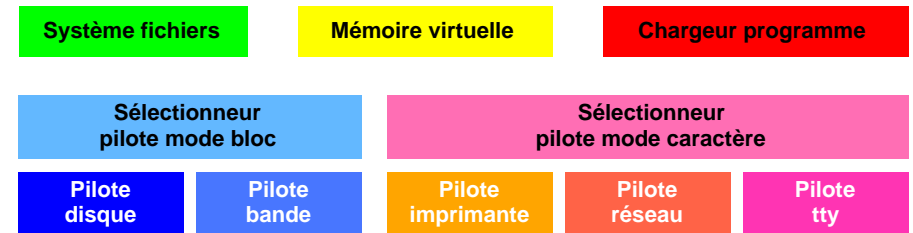
Noyau et interface de programmation I



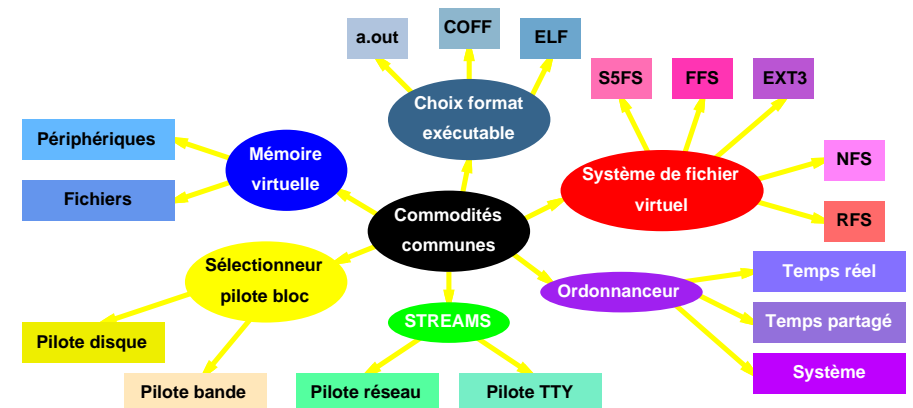
Séparer politique et mécanisme I



Structure Unix traditionnel (ab initio) I



Structure Unix moderne I

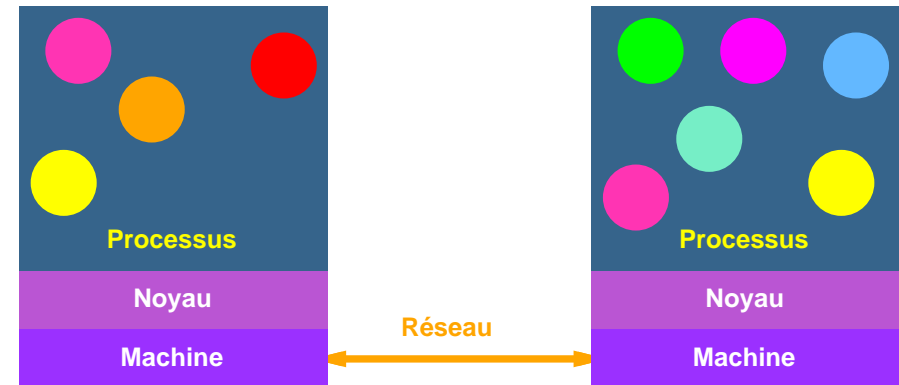


Centralisation contre distribution I

2 types de systèmes

- Systèmes centralisés : 1 seul noyau
 - ▶ Monoprocasseur
 - ▶ Multiprocasseur à mémoire partagée
- Systèmes distribués ou réseau : autant de noyau que de processeurs
 - ▶ Multiprocasseur sans mémoire commune
 - ▶ Réseaux d'ordinateurs, stations de travail

Systèmes distribués I



Autant de noyaux que de machines (qui peuvent être des multiprocesseurs à mémoire partagée...)



Quand donner contrôle au noyau ? I

- Quand un processus actif appelle une fonction système (du noyau)
 - ▶ Créer un processus
 - ▶ Attendre la fin d'un processus
 - ▶ Indiquer que l'on a terminé
 - ▶ Se bloquer en attente d'un événement
 - ▶ Demander une entrée-sortie de bas niveau
 - ▶ Attendre un certain temps
 - ▶ Envoyer un message
 - ▶ Recevoir un message
- Quand une interruption matérielle survient

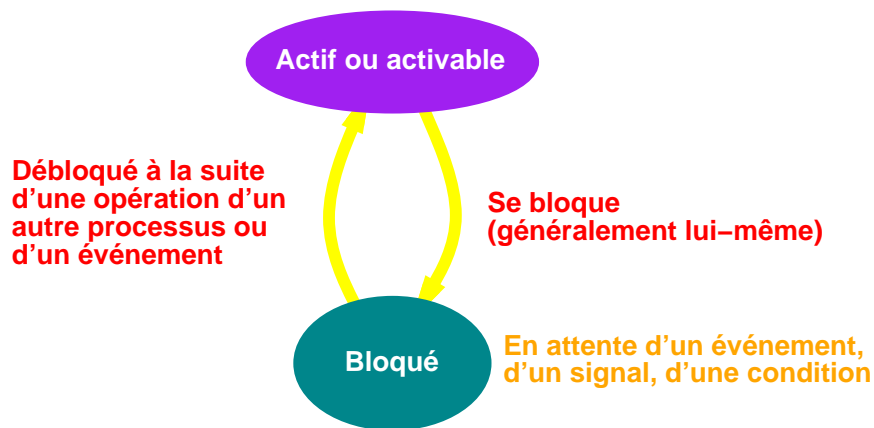


Informations gérées par le noyau I

- Données globales au noyau (temps, mémoire,...)
- Connaît tous les processus (table des processus)
- Descripteur de chaque processus
 - ▶ Contexte matériel
 - ▶ Une partie du contexte logiciel
- Gère transitions entre états des processus



Graphe des états d'un processus I



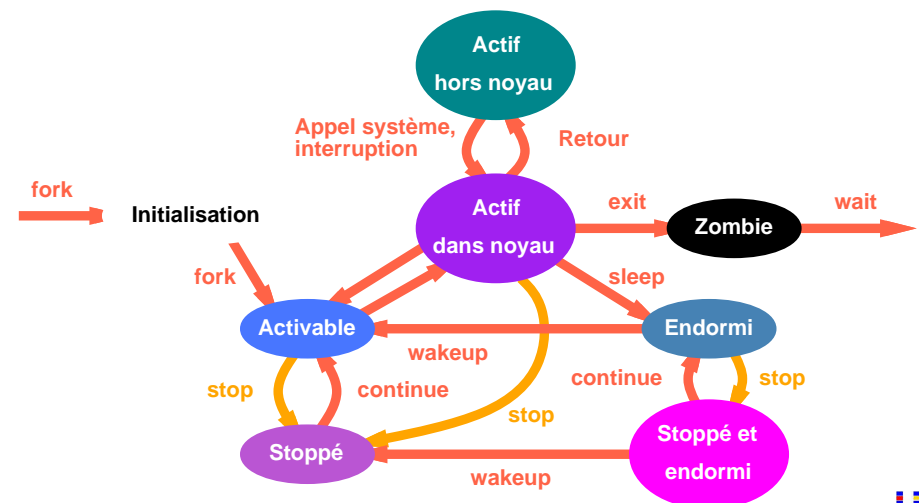
Atomicité des actions noyau I

- De nombreuses actions concurrentes dans le système : mise à jour liste processus, modification entrées-sorties à faire,...
- ~ Besoin d'assurer cohérence des informations gérées par le noyau
- Utilisation de l'exclusion mutuelle
 - ▶ Mono-processeur : masquage des interruptions
 - ▶ Multiprocesseur à mémoire commune : masquage des interruptions et construction de verrous avec des instructions spéciales insécables (*test and set*, *swap*, *XRM*,...)

⚠ Il y a aussi les dispositifs d'échange qui peuvent modifier la mémoire...



Graphe des états d'un processus sous Unix I



Contexte matériel I

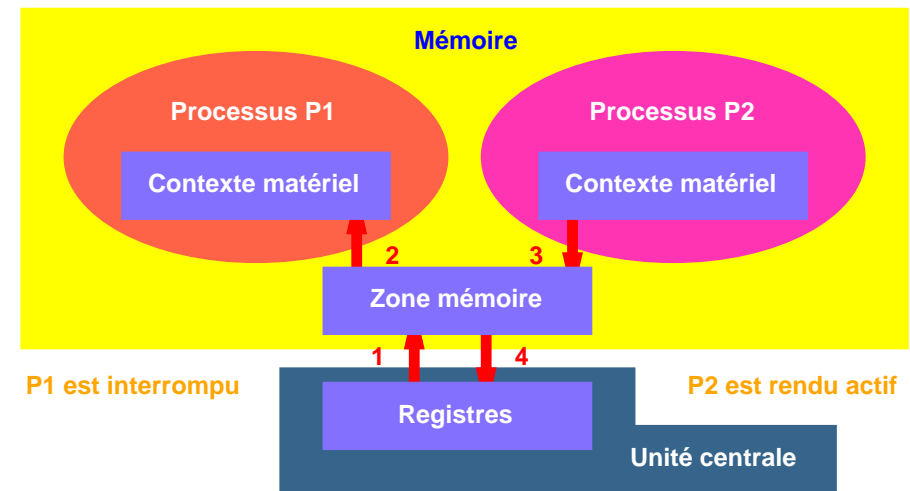
- Processus arrêtés lorsqu'une interruption matérielle ou logicielle survient
- Sauvegarde (minimale) des registres de la machine nécessaires au (re)fonctionnement d'un processus
 - ▶ Registres de travail
 - ▶ Compteur ordinal (pointe instruction courante)
 - ▶ Mot d'état du programme (bits de protection, résultats d'opération)
 - ▶ Pointeur de pile
 - ▶ Éventuellement état du pipeline sur processeur complexes,...



Sauvegarde et restitution de contexte I

Comment modifier l'exécution des processus ?

Sauvegarde et restitution de contexte II

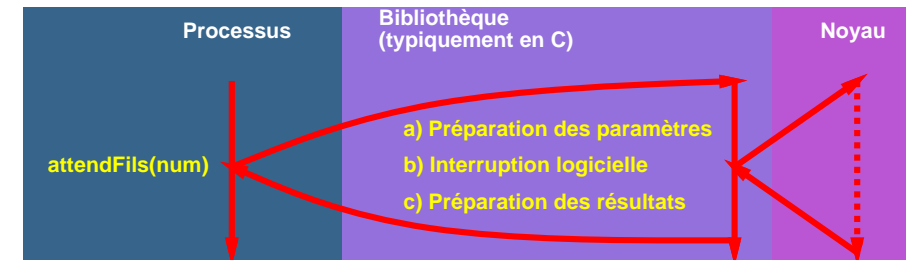


Sauvegarde et restitution de contexte III

- 1 P1 actif
- 2 Une interruption survient. Les registres de la machine sont sauvegardé dans une zone mémoire (selon le processeur, cela peut aussi être un jeu de registres supplémentaire). Passage en mode superviseur
- 3 Le noyau s'exécute
- 4 Le noyau fait une copie de cette zone dans le descripteur du processus P1 (contexte matériel)
- 5 Le noyau choisit de rendre le processus P2 actif. Il copie le contenu du contexte matériel du processus P2 dans la zone mémoire
- 6 L'instruction de retour sur interruption copie la zone mémoire dans les registres du processeur
- 7 P2 commence/continue sa vie



Appel système I



Appel moniteur/BIOS I

Certains systèmes ont une couche d'abstraction matérielle supplémentaire

- Portabilité accrue : *Basic Input/Output System* BIOS sur PC) pour accéder aux ressources d'un PC sans faire appel à des programmes sur disque → contenu en mémoire permanente (ROM, Flash)
- Possibilité de debug, variables d'environnement (Sun)
- Des cartes d'entrées-sorties peuvent rajouter des fonctionnalités

```
void point(int x,int y) {
    _CX = x ;
    _DX = y ;
    _AL = couleur ;
    _AH = 0x0C ;
    geninterrupt(0x10) ;           /* Fonction 0Ch de l'int. 10h */
}
```



Contexte logiciel d'un processus I

Regroupe

- Ce que le noyau a besoin de savoir sur le processus
 - ▶ Minimal si micro-noyau
- Ce que le reste du système d'exploitation a besoin de savoir
 - ▶ Si processus léger (*thread*) : peu de choses
 - ▶ Processus lourds beaucoup de choses :
 - Gestion des processeurs
 - Gestion de la mémoire
 - Gestion des fichiers



Sous Unix, système et noyau ne font qu'un...



Entrée et sortie du noyau I

- Entrée
 - ▶ Sauvegarde du contexte matériel
 - ▶ Passage en mode superviseur/noyau
 - ▶ Masquage local des interruptions
 - ▶ Si multiprocesseur, prélude de boucle d'attente active (avec *test & set*, XRM,...) sur certaines structures du noyau
- Sortie
 - ▶ Si multiprocesseur, postlude d'exclusion mutuelle
 - ▶ Démasquage local des interruptions
 - ▶ Restitution du contexte matériel et passage en mode utilisateur/esclave

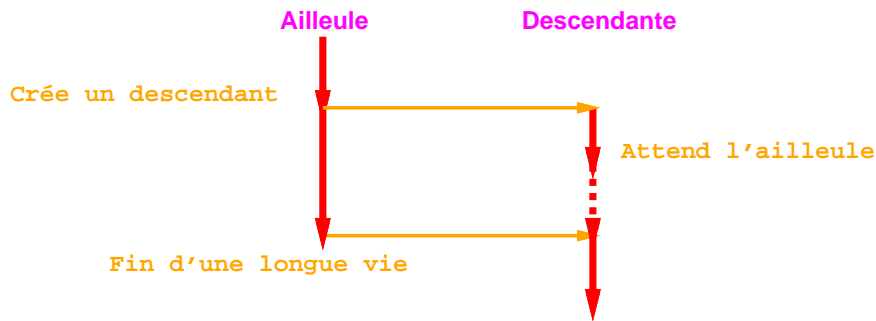


Le plan

- | | |
|---|--|
| <ol style="list-style-type: none"> 1 Historique 2 Concepts de base 3 Concurrence & Parallélisme <ul style="list-style-type: none"> ● Introduction ● Hypothèses sur architecture matérielle ● Notion de noyau ● Processus lourds & légers, <i>threads</i> <ul style="list-style-type: none"> ● Plus de détail des tâches dans Linux ● Entrées-sorties ● Gestion du temps qui passe ● Ordonnancement | <ol style="list-style-type: none"> 4 Gestion mémoire 5 Virtualisation 6 Les entrées-sorties <ul style="list-style-type: none"> ● Disques ● Formattage ● RAID ● ZFS ● Pilote de périphérique 7 Systèmes de fichiers distants <ul style="list-style-type: none"> ● NFS 8 Conclusion |
|---|--|



Vie et mort d'un processus I



Synchronisation & concurrence I

- Application utilisateur
 - ▶ Si application mono-thread pas de problème de partage de ressource
 - ▶ Si multithread/multiprocessus, possibilité de conflits d'accès à des ressources, interblocages,...
 - ▶ ~ Utilisation de primitives atomiques : verrous,...
 - ▶ Dans le pire des cas, arrêt des tâches possibles
- Tâche dans le noyau
 - ▶ Intrinsèquement multitâche
 - ▶ ~ Utilisation de primitives atomiques : verrous,...
 - ▶ Pas de système d'exploitation pour veiller...
 - ▶ ... Si conflits d'accès ou étreintes mortelles : plantage du système ☹



Préemption dans le noyau I

- Plusieurs tâches concurrentes dans le noyau, voire parallèles si plusieurs processeurs
- Interruptions asynchrones
- Préemption en standard dans noyau Linux 2.6 : tâches du noyau peuvent être aussi interrompues
- Meilleure réactivité... ☺
- ...source de bugs améliorée ☹
- Bien soigner partage de ressources pour éviter interblocages
- Ne pas tomber dans excès inverse d'avant 2.6 : gros cli/sti

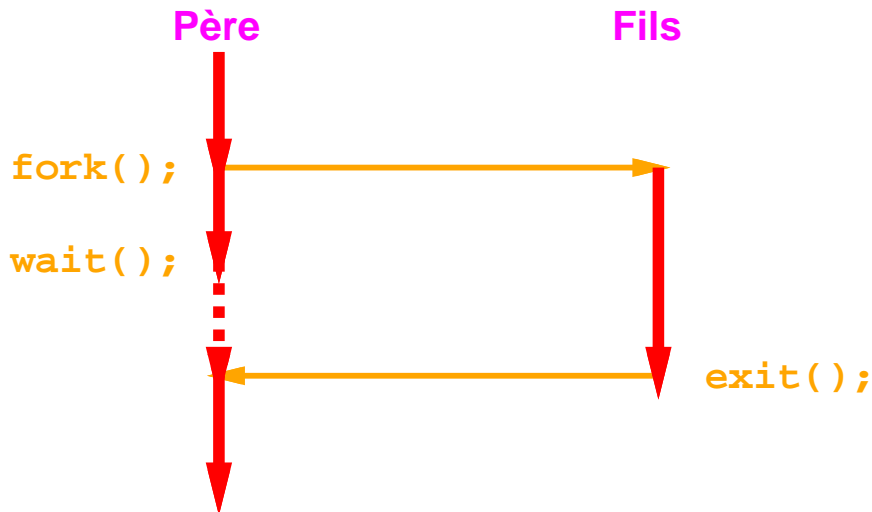


4 états d'exécution possible I

- Un processus lourd ou léger en cours d'exécution en mode utilisateur
- Un processus lourd ou léger en cours d'exécution en mode noyau
- Une tâche noyau en cours d'exécution
- Mode interruption dans noyau



Vie et mort d'un processus version Unix I



Vie et mort d'un processus version Unix II

- `fork()` crée un clone du père
 - ▶ Ne partagent que les valeurs initiales
- Souvent suivi d'un `exec()` pour exécuter un autre programme dans le processus
 - ▶ En fait optimisation de `fork()` : partage des pages de mémoire et copie seulement lors de la première écriture (cow *Copy On Write*)
 - ▶ Par le passé pré-COW, introduction du `vfork()` BSD optimisé pour un `exec()` : le fils *utilise* l'espace mémoire du père qui est rendu après l'`exec()`... mais appel système toujours présent



Distinguer le père du fils en Unix ? I

Clones parfaits pas toujours utiles ~ brisure de symétrie

```
/* Code du père */
resultat = fork();
/* Code du père ET du fils */
if (resultat < 0) {
    perror("Le fork() s'est viandé grave :-( !");
    exit(2002);
}
if (resultat == 0) {
    /* Je suis le fils */
    if (execve("/un/autre/programme") < 0) {
        perror("L'exec() dans le fils n'a pas marché !");
        exit(2003);
    }
}
```



Distinguer le père du fils en Unix ? II

```
/* On n'exécutera jamais ici. */
}
else {
    /* Je suis le père et resultat contient le numéro du fils */
    ...
}
```



Contexte logiciel processus lourd UNIX I

- État du processus (stoppé, activable,...)
- Date de lancement du processus
- Temps unité centrale utilisé
- Identificateur du processus (PID *process identification*)
- Identificateur du père du processus
- Pointeur sur le segment de code (les instructions)
- Pointeur sur le segment des variables globales (initialisées à 0)
- Pointeur sur le segment BSS (variables pré-initialisées par le programmeur)
- Table (de bits) des signaux en attente
- Identité du propriétaire-utilisateur UID (*user identification*) réel, UID effectif (droits temporaires empruntés à un utilisateur)



Contexte logiciel UNIX en 2 parties I

Pour des raisons d'optimisation mémoire centrale, division du contexte d'un processus :

- *U area* informations utiles lorsque le processeur est actif
 ↪ peuvent être stockées sur disque lorsque le processus ne tourne pas
- *Proc area* informations indispensables au noyau en permanence

⚠ Commandes d'information sur les processus (*ps*, *top*,...) peuvent provoquer du *swap* car besoin d'accéder à l'*U area*



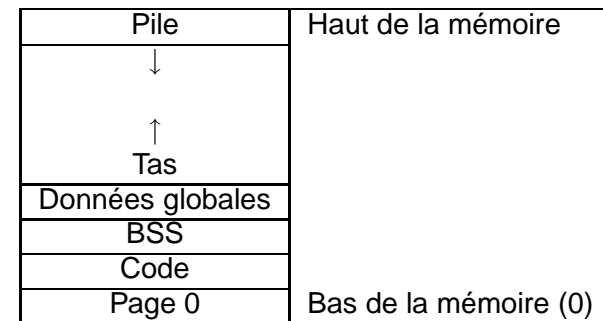
Contexte logiciel processus lourd UNIX II

- Identité du groupe GID (*group identification*) réel, GID effectif (droits temporaires empruntés à un groupe)
- Droits d'accès par défaut sur les fichiers créés (masque UMASK)
- Répertoire de travail courant
- Répertoire racine (vision du monde changeable pour enfermer des processus : sécurité, test,...)
- Descripteurs (objet) des fichiers manipulés par le processus



Espace virtuel des processus UNIX I

Chaque processus lourd UNIX possède un espace d'adressage (virtuel) propre



- Besoin d'une convention (collective) de bon usage et programmation



Espace virtuel des processus UNIX II

- Pile contient au départ arguments de

```
int main(int argc, char *argv[], char *environ[])
```

- ▶ Les `argc` paramètres `argv` de la ligne de commande
- ▶ Variables d'environnement dans `environ`
- ▶ BSS données initialisées
- ▶ Données globales initialisées à 0
- ▶ Tas utilisé pour les variables allouées par `malloc()` ou `new`
- ▶ Code d'un programme (lecture seule) partageable par plusieurs processus
- ▶ Chasse au bugs : la page 0 ne contient pas de mémoire : permet de déclencher les erreurs d'accès * (NULL)



Communications inter-processus sous UNIX II

- ▶ IPC (*InterProcess Communication*) : sémaphores, mémoire partagée,...



Communications inter-processus sous UNIX I

- `fork()` crée un clone du père
 - ▶ Ne partagent que les valeurs initiales
 - ▶ En fait optimisation : partage des pages de mémoire et copie seulement lors de la première écriture
- Par construction espaces d'adressages propres et étanche
 - ▶ Pas de données partagées par défaut → un peu plus complexe pour communiquer
 - ▶ Communications par fichiers (à protéger avec des verrous). Ex. courrier entre facteur et lecteur de courrier

```
int fcntl(int fd, int cmd, struct flock *lock);
```

- ▶ Communication par messages (*pipe*, *socket*,...). Ex. envoi de courrier et entre facteurs

```
int socket(int domain, int type, int protocol);
```

```
int pipe(int filedes[2]);
```

```
int socketpair(int domain, int type, int protocol, int sv[2]);
```



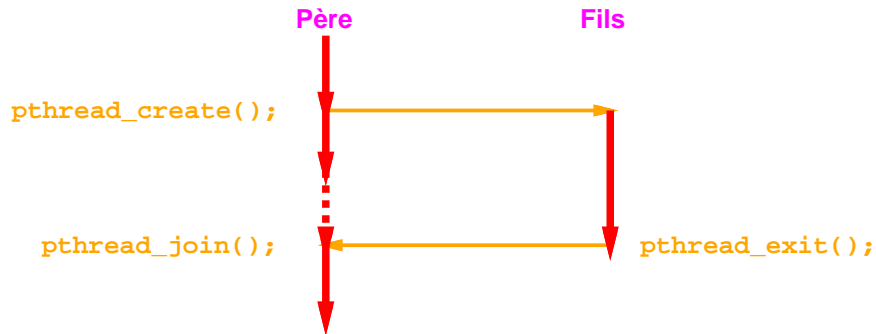
Processus légers I

- Besoin de concurrence plus fine dans un programme
- Processus lourd peut contenir plusieurs fibres d'exécutions (*threads of execution*, processus légers)
- Partagent au sein d'un processus lourd Unix
 - ▶ Mémoire (données globales)
 - ▶ Fichiers
 - ▶ Signaux
 - ▶ Données internes au noyau (état processeur,...)
- Mais les threads disposent en propre
 - ▶ Numéro (*thread-id*)
 - ▶ Contexte matériel : image des registres de la machine (compteur ordinal, pointeur de pile,...)
 - ▶ Pile
 - ▶ Masque de signaux UNIX
 - ▶ Priorité
 - ▶ Mémoire locale via le tas global et la pile

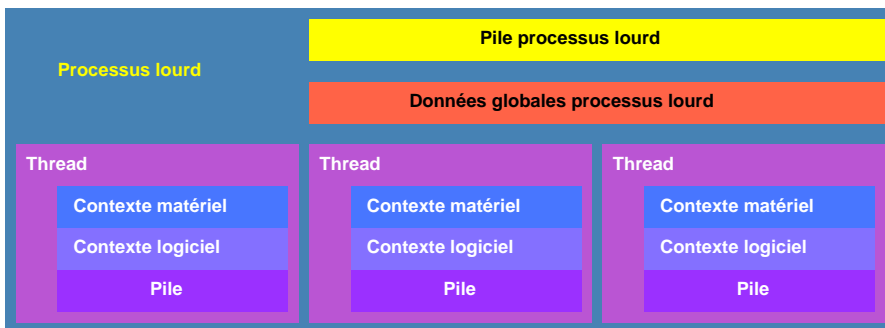


Vie et mort d'un processus version thread I

Thread POSIX



Threads utilisateurs dans un processus I



Threads POSIX I

- Créer un *thread*

```
int pthread_create(pthread_t *thread,
                  const pthread_attr_t *attr = null,
                  void *(*entry) (void *),
                  void *arg)
```

entry indique la fonction à exécuter avec les paramètres *arg*

- Pour terminer

```
void pthread_exit(void *status)
```

Transmet dans *status* une cause de terminaison

- Attendre la fin d'un fils *thread*

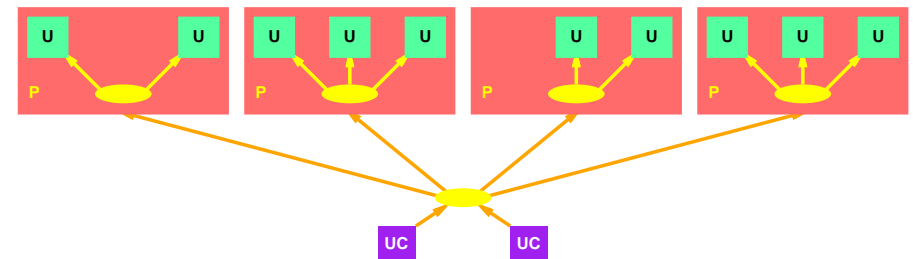
```
int pthread_join(pthread_t *thread ,
                 void **status) ;
```

Récupère dans *status* une information sur la cause de la terminaison



2 niveaux d'ordonnanceur I

Où choisir de multiplexer exécution concurrente ?



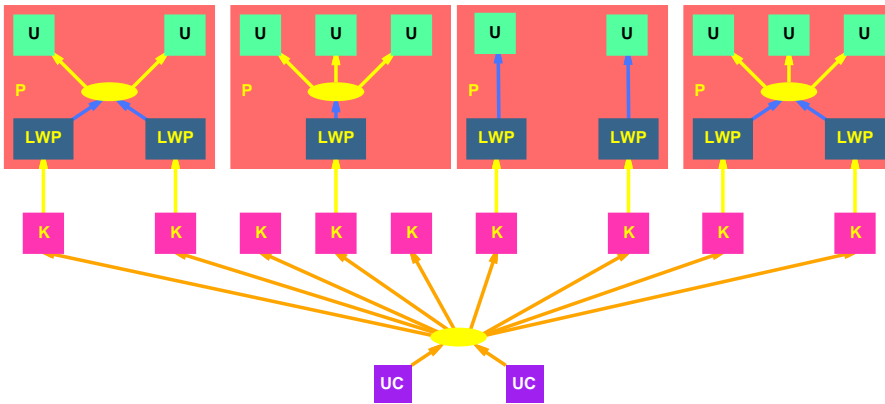
- Ordonnanceur du noyau : processus plutôt lourds

- Ordonnanceur utilisateur dans chaque processus lourds : *threads* plutôt légères

⚠ blocages sur E/S... ☹



Avec ordonnanceur de threads noyau I



Quid entre processus lourd et léger ? I

```
int clone(int (*fn)(void *), void *child_stack, int flags, void *arg);
```

Linux propose appel système `clone()` permettant de choisir ce qui est partagé

- Espace mémoire
- Descripteurs de fichiers (fichiers ouverts)
- Gestion des signaux
- Espace de nommage du système de fichiers



Les processus légers dans Linux I

- Pas de gestion spécifique dans Linux (contrairement à Solaris (LWP) ou Windows)
- Processus léger \equiv processus lourd comme un autre... où on précise qu'on partage certaines choses (mémoire,...)
- Suppose que processus gérés de manière naturellement « légère » (création, changement de contexte,...)
- Si pas suffisant, utiliser une bibliothèque niveau utilisateur



Processus et threads : combien ça coûte ? I

Dans une vieille version de Solaris (2.4) sur un vieux Sun

	Temps de création	Synchro sémaphore
User thread	52 μ s	66 μ s
LWP	350 μ s	? 390 μ s
Processus	1700 μ s	200 μ s

Compromis d'utilisation



Des processus sans système d'exploitation ? I

- Processus et concurrence \equiv concept de programmation important
- Difficile de porter tous les SE sur toutes les plateformes ☹
- Pour des systèmes embarqués, pas toujours les ressources pour : microcontrôleur minuscule,...
- Idée si on ne veut pas écrire des co-routines à la main : traduire (compiler) les appels systèmes thread POSIX du programme en programme qui gère les tâches à la main dans espace utilisateur
- Génère un programme C séquentiel compilable par un compilateur C pour n'importe quoi
 - ▶ *Atomic execution block* (AEB)
 - ▶ séparés par du code qui gère le choix des AEB à exécuter



Le plan

- | | |
|--|--|
| 1 Historique | 4 Gestion mémoire |
| 2 Concepts de base | 5 Virtualisation |
| 3 Concurrence & Parallélisme | 6 Les entrées-sorties |
| <ul style="list-style-type: none"> ● Introduction ● Hypothèses sur architecture matérielle ● Notion de noyau ● Processus lourds & légers, <i>threads</i> ● Plus de détail des tâches dans Linux | <ul style="list-style-type: none"> ● Disques ● Formattage ● RAID ● ZFS ● Pilote de périphérique |
| <ul style="list-style-type: none"> ● Entrées-sorties ● Gestion du temps qui passe ● Ordonnancement | <ul style="list-style-type: none"> 7 Systèmes de fichiers distants ● NFS 8 Conclusion |



Des processus sans système d'exploitation ? II

- Alexander G. DEAN. *Compiling for concurrency : Planning and performing software thread integration*. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, Austin, TX, Dec 2003.
- *Lightweight Multitasking Support for Embedded Systems using the Phantom Serializing Compiler*, André C. NÁCUL and Tony GIVARGIS, DATE2005



Différents usages de clone() I

- `sys_fork()` [[arch/i386/kernel/process.c](#)]

```
1  asmlinkage int sys_fork(struct upt_regs *regs)
2  {
3  __return__do_fork(SIGCHLD, __regs.esp, &__regs, 0, NULL, NULL);
4  }
```

- `sys_vfork()` [[arch/i386/kernel/process.c](#)]

```
1  asmlinkage int sys_vfork(struct upt_regs *regs)
2  {
3  __return__do_fork(CLONE_VFORK | CLONE_VM | SIGCHLD, __regs.esp, &__regs, 0, NULL, NULL);
4  }
```

- `sys_clone()` [[arch/i386/kernel/process.c](#)]



Différents usages de clone() II

```

1  asmlinkage int sys_clone(struct upt_regs *regs)
2  {
3      unsigned long clone_flags;
4      unsigned long newsp;
5      int __user *parent_tidptr, *child_tidptr;
6
7      clone_flags = regs.ebx;
8      newsp = regs.ecx;
9      parent_tidptr = (int __user *)regs.edx;
10     child_tidptr = (int __user *)regs.edi;
11     if (!newsp)
12         newsp = regs.esp;
13     return do_fork(clone_flags, newsp, 0, parent_tidptr, chil
14 }
```

Le vrai boulot : do_fork() [kernel/fork.c]



Différents usages de clone() III

```

1  long do_fork(unsigned long clone_flags,
2              unsigned long stack_start,
3              struct upt_regs *regs,
4              unsigned long stack_size,
5              int __user *parent_tidptr,
6              int __user *child_tidptr)
7  {
8      struct task_struct *p;
9      long pid = alloc_pidmap();
10     ...
11     p = copy_process(clone_flags, stack_start, regs, stack_size, paren
12     ...
13     if (!(clone_flags & CLONE_STOPPED))
14         wake_up_new_task(p, clone_flags);
15     else
16         p->state = TASK_STOPPED;
17     if (clone_flags & CLONE_VFORK)
18         wait_for_completion(&vfork);
19     return pid;

```



Différents usages de clone() IV

20 }

- Le fils est réveillé : essaye de le faire tourner avant le père pour optimiser le COW en cas d'exec() rapide
- copy_process() fait un dup_task_struct(current) et initialise la structure de la tâche



Tâches noyau I

- Même le noyau peut avoir besoin de processus pour faire des choses de manière concurrente
 - ~ tâches noyau (convention de nommage : tâche comme raccourci de processus noyau)
 - Création ? Comme les processus utilisateurs
 - ▶ Si processus utilisateur, création de processus utilisateur avec clone() ou (v)fork
 - ▶ Si tâche noyau, clone() crée une tâche noyau
- Exemple du chargement dynamique de module (si rajout de périphériques,...)

Remarques

- ▶ ⚠ Un processus utilisateur ne peut créer que des processus utilisateurs... Si besoin tâche noyau : modifier le noyau ou mettre dans un module
- ▶ ⚠ Une tâche noyau n'a aucun mode de protection, accède à toutes les ressources,... Sans filet!



Exemple de liste de processus I

- Outils ps, top,...

```
keryell@an-dro:~$ ps auxww
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	1584	80	?	S	04:46	0:00	init [?
root	2	0.0	0.0	0	0	?	SN	04:46	0:00	[ksofti
root	3	0.0	0.0	0	0	?	S<	04:46	0:00	[events
root	2141	0.0	0.0	2984	176	?	Ss	04:46	0:00	/sbin/l
bind	2171	0.0	0.1	29740	736	?	Ssl	04:46	0:00	/usr/sl
...										
keryell	9620	0.0	0.1	4612	904	pts/4	R+	08:52	0:00	ps aux

- Des entrées dans /proc pour chaque processus dont *self*



Représentation des tâches en interne II

- ▶ Priorité
- ▶ Gestion des machines parallèles NUMA
- ▶ ...

- Chaque processus a une (petite) pile dans le noyau
- Chaque processus utilisateur a aussi une pile dans espace mémoire utilisateur
- Reliés par une double liste chaînée
- Tout n'est pas nécessaire pour faire des changement de tâches
 - ▶ Hiérarchisation
 - ▶ Le minimum vital est en haut de la pile noyau du processus
 - ▶ Rapide (cache) et facile (déplacement pointeur de pile) à accéder

thread_info [[include/linux/thread_info.h](#)]



Représentation des tâches en interne I

- Chaque processus est représenté par une task_struct [[include/linux/sched.h](#)] de 1,7 Ko sur ordinateur 32 bits
- Contient tout ce que le noyau a besoin de connaître sur un processus pour le faire fonctionner
 - ▶ État (bloqué ou pas)
 - ▶ Espace mémoire
 - ▶ Exécutable associé
 - ▶ Parenté
 - ▶ Droits, capacités
 - ▶ Fichiers ouverts
 - ▶ Espace de nommage (montages particuliers, autre racine,...)
 - ▶ Domaines d'exécutions (simulation d'autres systèmes,...)
 - ▶ Audit
 - ▶ Files d'attente d'entrées/sorties
 - ▶ Statistiques d'usage



Représentation des tâches en interne III

```

1  struct thread_info {
2      struct task_struct *task;
3      struct exec_domain *exec_domain;
4      unsigned long flags;
5      unsigned long status;
6      __u32 cpu;
7      __s32 preempt_count;
8      mm_segment_t addr_limit;
9      struct restart_block restart_block;
10     unsigned long previous_esp;
11     ...
12     __u8 supervisor_stack[0];
};

```



Accès aux tâches I

- `current_thread_info()` pointe vers le descripteur courant

```
1 static inline struct thread_info *current_thread_info(void)
2 {
3     struct thread_info *ti;
4     __asm__ ("andl_%%esp,%0;_": "=r" (ti) : _"^(THREAD_SIZE-1)");
5     return ti;
6 }
```

- `current()` pointe vers le descripteur courant de la `task_info` complète
- Adresse pas pratique pour un utilisateur (si processus migre,...)
 - ↪ notion de *pid* sur 16 ou 32 bits dans `task_info`
 - ▶ Limite dans `/proc/sys/kernel/pid_max` changeable pour gros serveurs 64 bits...
 - ▶ Allocation de *pid* par un bitmap



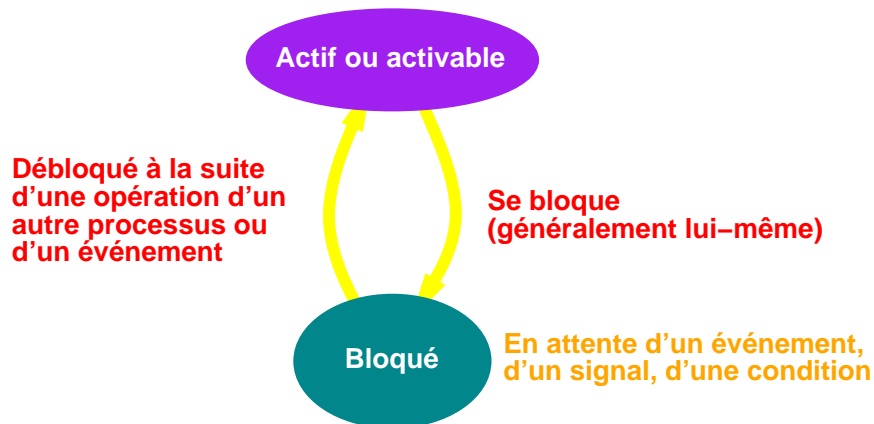
Accès aux tâches II

- ▶ Fonction de hachage pour retrouver rapidement une tâche à partir de son *pid*

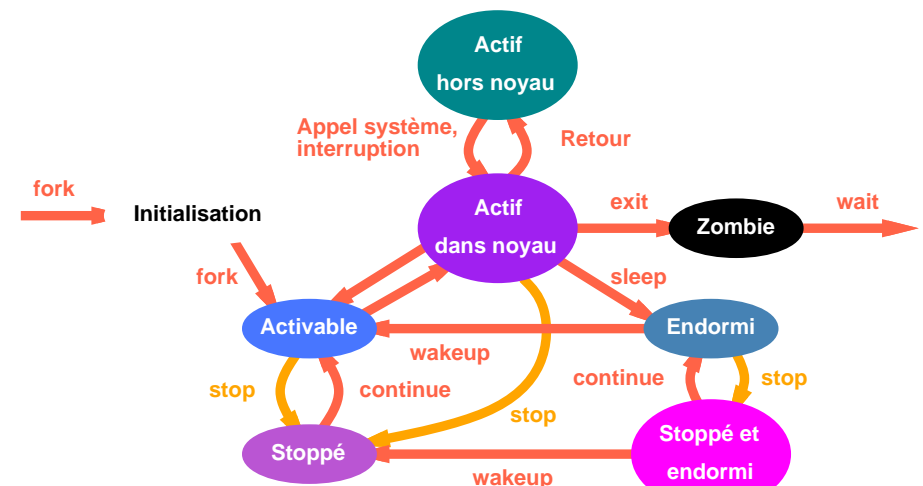
- `[kernel/pid.c]`



Graphe des états d'un processus I



Un processus Unix dans tous ses états I



États internes dans Linux I

Dans descripteur processus `task_struct` [`include/linux/sched.h`] :
champ `task`

- `#define TASK_RUNNING 0` : le processus est en train de fonctionner ou est sur une file d'attente pour. Si processus en mode utilisateur forcément en train de... s'exécuter ☺
R dans `ps/top`
- Le processus est bloqué en attente (endormi) d'un événement ou entrée-sortie pour repasser dans l'état `TASK_RUNNING`
 - ▶ `#define TASK_INTERRUPTIBLE 1` : le processus peut être réveillé aussi par un signal
S dans `ps/top`



États internes dans Linux III

- `#define TASK_TRACED 8` : un processus est en train de tracer tout ce qu'il fait (`strace` pour lister appels systèmes, `gdb` pour déboguer,...)
- `#define EXIT_ZOMBIE 16` : le processus n'existe plus. État juste pour retourner `task_struct.exit_code` au processus parent quand il fera un `wait()`
 - ▶ Si pas de parent, `init` joue le rôle de parrain et fait un `wait()` de complaisance
 - ▶ Si parent mal écrit et ne fait jamais de `wait()`, développement des zombies... ☹
- `#define EXIT_DEAD 32` : paix à son âme



États internes dans Linux II

- ▶ `#define TASK_UNINTERRUPTIBLE 2` : idem mais ne peut pas être réveillé par un signal. Utile si processus veut dormir sans interruption, si événement attendu rapidement et réserve de ressources importantes,...
D dans `ps/top`
⚠ Un signal `SIGKILL` ne peut même pas en venir à bout... Mais peut-être voulu si des ressources ont été bloquées et ne seraient pas libérées. Problème si bug néanmoins ☹
~ Éviter si possible
- `#define TASK_STOPPED 4` : est stoppé suite à `SIGTSTOP` (~Z du shell,...) `SIGSTOP` (ne peut pas être contré) ou suite à une entrée-sortie alors qu'il est en tâche de fond (`SIGTTIN` & `SIGTTOU`). Continue si `SIGCONT`
T dans `ps/top`

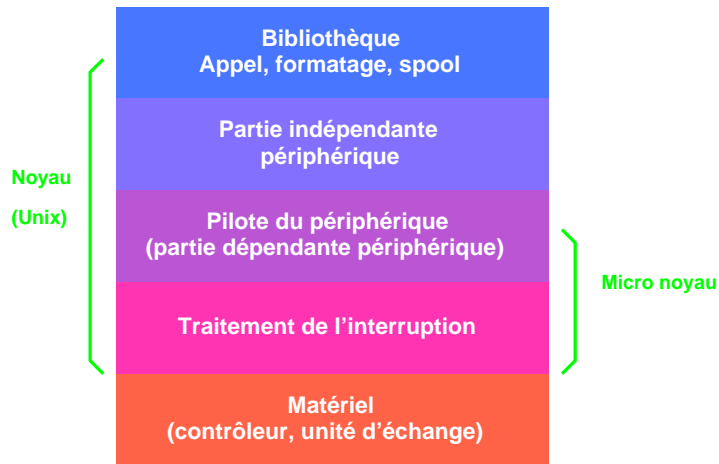


Le plan

- | | |
|--|--|
| 1 Historique | 4 Gestion mémoire |
| 2 Concepts de base | 5 Virtualisation |
| 3 Concurrence & Parallélisme | 6 Les entrées-sorties |
| <ul style="list-style-type: none"> ● Introduction ● Hypothèses sur architecture matérielle ● Notion de noyau ● Processus lourds & légers, <i>threads</i> ● Plus de détail des tâches dans Linux | <ul style="list-style-type: none"> ● Disques ● Formattage ● RAID ● ZFS ● Pilote de périphérique |
| <ul style="list-style-type: none"> ● Entrées-sorties ● Gestion du temps qui passe ● Ordonnancement | <ul style="list-style-type: none"> 7 Systèmes de fichiers distants ● NFS |
| | 8 Conclusion |



Entrées-sorties physiques (bas niveau) I



Le plan

- | | |
|---|--|
| <ul style="list-style-type: none"> 1 Historique 2 Concepts de base 3 Concurrence & Parallélisme <ul style="list-style-type: none"> ● Introduction ● Hypothèses sur architecture matérielle ● Notion de noyau ● Processus lourds & légers, <i>threads</i> ● Plus de détail des tâches dans Linux ● Entrées-sorties ● Gestion du temps qui passe ● Ordonnancement | <ul style="list-style-type: none"> 4 Gestion mémoire 5 Virtualisation 6 Les entrées-sorties <ul style="list-style-type: none"> ● Disques ● Formattage ● RAID ● ZFS ● Pilote de périphérique 7 Systèmes de fichiers distants <ul style="list-style-type: none"> ● NFS 8 Conclusion |
|---|--|



Déroulement d'une entrée-sortie physique I

- 1 Processus 1 s'exécute (actif)
- 2 Exécution d'une fonction de demande d'entrée-sortie
- 3 Appel au noyau via interruption logicielle
 - 1 Préparation de l'entrée-sortie
 - 2 Lancement de l'entrée-sortie
 - 3 Ordonnanceur
- 4 ~ Processus 1 bloqué (dans et hors noyau)
- 5 Interruption matérielle de fin d'entrée-sortie
 - 1 Traitement de fin d'entrée-sortie
 - 2 Ordonnanceur
- 6 Processus 1 activable



Temps I

- Attribution des ressources à tour de rôle
- ~ Temps : crucial dans un système d'exploitation
- Périphérique particulier qui donne le temps : l'horloge
- Besoin
 - ▶ Connaître l'heure
 - ▶ Faire des choses à intervalles régulier
- Part d'une fréquence régulière de base qu'on divise d'un bon facteur
- Base selon richesse et précision :
 - ▶ Oscillateur RC
 - ▶ Filtre céramique
 - ▶ Courant secteur 50 Hz
 - ▶ Résonateur à quartz (éventuellement thermostaté)
 - ▶ Stations radios (GPS, France Inter ou BBC en GO,...)



Temps II

- Horloge atomique : utilise des fréquences de transition entre 2 états atomiques

[http://www.obs-besancon.fr/www/tf/equipes/vernotte/echelles/echelle:](http://www.obs-besancon.fr/www/tf/equipes/vernotte/echelles/echelle)
<http://www.chez.com/tempsatomique/nouvellepage3.htm>

- Nouvelle définition du temps : « *La seconde est la durée de 9 192 631 770 périodes de la radiation correspondant à la transition entre les deux niveaux hyperfins de l'état fondamental de l'atome de Césium 133* »
- Idée : coupler un oscillateur sur la résonance d'atomes d'un jet de césium 133 (le plus lent possible. ...)
- MASER à hydrogène : meilleur en stabilité à court terme
- Horloges à cellule de rubidium, au mercure,...

Merci à Jean François DUTREY du Laboratoire de métrologie Temps-Fréquence pour ses précisions!



L'heure et sa distribution I

- Des horloges partout...
- ...Mais rarement à l'heure, ni synchronisées ☹
- Problématique si systèmes de fichiers distribués et Makefile par exemple ou corrélation de phénomènes physiques
- Besoin de synchronisation globale : diffusion d'une référence par un moyen quelconque et recalage
 - Radiodiffusion
 - Radio Frankfurt 10 MHz
 - Modulation porteuse France Inter GO
 - RDS de la bande FM
 - Satellite GPS
 - Protocoles réseau
 - NTP : Distribution du temps & Mesure statistique du temps de transmission depuis un serveur de référence



Génération d'événements I

- Mécanisme matériel avec 1 registre T et un compteur C
- À chaque coup d'horloge faire


```
C--
if (C == 0) {
    C = T
    Générer signal ou interruption
}
```
- Exemple d'usage à chaque interruption
 - Gérer un temps macroscopique en incrémentant un autre compteur (⬆ si débordement du compteur...)
 - Rendre activables ou stoppés certains processus
 - Redonner la main à l'ordonnanceur qui choisit quel processus faire tourner



L'heure et sa distribution II

```
chailly99-keryell > ntpq -p
remote          refid          st t when poll reach  delay  offset  disp
=====
*orgenoy        canon.inria.fr   2 u  51  64  377   0.66   0.255   0.14
```



Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Priorité I

Associer une priorité à chaque processus en fonction de

- Objectifs globaux du système (système à temps partagé, traitement par lots, contrôle de procédés industriels,...)
- Caractéristiques de chaque processus (échéance temporelle, périodicité, temps processeur récemment consommé, temps récemment passé en sommeil,...)
- Caractéristiques de chaque périphérique : ne pas ralentir des périphériques qui sont déjà lents,...



Ordonnancement I

- Système d'exploitation multitâche
 - ▶ Plein de processus veulent tourner
 - ▶ Un ou plusieurs processeurs
 - ▶ De nombreuses possibilités
 - ▶ Lesquels faire tourner en premier ?
- 2 objectifs difficiles à atteindre
 - ▶ S'assurer d'un bon taux d'utilisation des processeurs
 - ▶ S'assurer que chaque processus a le service qu'il souhaite
- Satisfaction des processus interactif au détriment des travaux par lots
- Coût des changements de contexte, des transferts de mémoire principale-mémoire secondaire,...
- Processus souvent connus qu'à l'exécution ☹️ ~> ordonnancement dynamique



Types de multitâche I

- Multitâche coopératif
 - ▶ Chaque processus a prévu de passer la main aux autres
 - ▶ Nécessite une architecture logicielle précise
 - ▶ Comportement assez prédictible
 - ▶ Difficile de faire des choses complexes
- Multitâche préemptif
 - ▶ Même si une tâche n'a pas prévu de s'arrêter le système peut en faire tourner une autre à la place après un quantum de temps
 - ▶ Globalement plus simple à mettre en place
 - ▶ Pas de garantie facile à assurer sur les contraintes

Systèmes d'exploitations modernes généralistes : font les deux (Linux 2.6, Solaris,...)



Quelques politiques d'ordonnancement I

- Premier Arrivé Premier Servi (PAPS) ou *First In First Out* (FIFO)
- Tourniquet (*round robin*) ou partage du temps
- Priorité statique (temps réel) ou dynamique
- *Shortest job first...* Mais nécessite de connaître la durée de la tâche (future)
- *Earliest deadline first...* Mais nécessite de connaître la date de fin (future)
- *Smaller period first (Rate Monotonic scheduling)*... Nécessite de préciser les intervalles de lancement
- ...



Temps partagé – tourniquet I

- Donner illusion de disposer d'une machine à soi tout seul
- Lié à l'invention du terminal interactif
- Changer de programme à chaque quantum de temps
- Privilégier les requêtes peu gourmandes par rapport aux programmes de calcul : faire des heureux facilement avec *caisse moins de 10 Articles*



Processus dirigé par le calcul ou les E/S I

Programme \equiv calculs + E/S

... mais rarement équilibré. Souvent 2 aspects se dégagent :

- Processus orienté calcul
 - ▶ Gros calculs
 - ▶ Prémption fréquente gâcherait du temps (système, cache, swap,...)
 - ▶ Réactivité moins évidente à l'utilisateur
- Processus orienté entrées-sorties
 - ▶ Calculs souvent bloqués par des E/S
 - ▶ Prémption souvent évitée par blocage sur E/S
 - ▶ Réactivité plus évidente à l'utilisateur (si c'est lui l'E/S !)

D'un point de vue rentabilité processeur, intéressant d'avoir les 2 en même temps



Blocage dans le tourniquet I

- Blocage possible d'un processeur avant la fin de son quantum de temps
- Libère du temps pour les autres



Politique par priorité I

Associer une priorité à chaque processus et faire fonctionner processus voulant tourner qui est le plus prioritaire

Priorité choisie en fonction de

- Objectifs globaux du système (système à temps partagé, traitement par lots, contrôle de procédés industriels,...)
- Caractéristiques de chaque processus (échéance temporelle, périodicité, temps processeur récemment consommé, temps récemment passé en sommeil,...) si on veut une politique a priorité plus dynamique
- Caractéristiques de chaque périphérique : ne pas ralentir des périphériques qui sont déjà lents,...



Politique à priorités dans Unix II

- Le super utilisateur peut augmenter la priorité
- Ordonnancement :
 - ▶ Faire tourner les processus de plus forte priorité entre eux en tourniquet
 - ▶ Un processus utilisateur qui a dépassé son quantum de temps est remis en queue de sa file d'attente (tourniquet)
 - ▶ Toutes les secondes, on recalcule les priorités sur le thème

$$p_{\text{base}} + \text{temps}_{\text{utilisation processeur}}$$

Avec p_{base} donné par la commande `nice` (0 par défaut)

- Rajout aussi d'ordonnanceur « temps réel » en plus dans les Unix modernes : permet d'avoir aussi des processus qui tournent avec des contraintes fortes



Politique à priorités dans Unix I

Priorité (p_{base})	Exemple	Type
+19 (+ faible)	Utilisateur d'écran	Plutôt utilisateur
⋮	⋮	
+1		
0 (standard)	Jeux vidéo	Plutôt système
-1	Numérisation d'un cours	
⋮	⋮	
-20 (+ forte)		

- Processus « noyau » ont une priorité (négative en Unix...) forte
- Processus utilisateurs ont une priorité (positive en Unix...) faible
- Un utilisateur peut seulement baisser la priorité d'un processus utilisateur ☺ (modifie sa base) à l'aide de la commande `nice`



Toutes les priorités dans Linux I

[`include/linux/sched.h`]

```

1  /*
2  * Priority of a process goes from 0..MAX_RT_PRIO-1, valid RT
3  * priority is 0..MAX_RT_PRIO-1, and SCHED_NORMAL tasks are
4  * in the range MAX_RT_PRIO..MAX_RT_PRIO-1. Priority values
5  * are inverted: lower p->prio value means higher priority.
6  *
7  * The MAX_USER_RT_PRIO value allows the actual maximum
8  * RT priority to be separate from the value exported to
9  * user-space. This allows kernel threads to set their
10 * priority to a value higher than any user task. Note:
11 * MAX_RT_PRIO must not be smaller than MAX_USER_RT_PRIO.
12 */
13 #define MAX_USER_RT_PRIO 100
14 #define MAX_RT_PRIO MAX_USER_RT_PRIO
15
16 #define MAX_PRIO (MAX_RT_PRIO + 40)
```



Toutes les priorités dans Linux II

[kernel/sched.c]

```

1  /*
2   * Convert user-nice values [-20...0...19]
3   * to static priority [MAX_RT_PRIO..MAX_PRIO-1],
4   * and back.
5   */
6  #define NICE_TO_PRIO(nice) ((MAX_RT_PRIO + (nice) + 20)
7  #define PRIO_TO_NICE(prio) ((prio) - MAX_RT_PRIO - 20)
8  #define TASK_NICE(p) (PRIO_TO_NICE((p)->static_prio))

10 /*
11  * 'User priority' is the nice value converted to something we
12  * can work with better when scaling various scheduler parameters,
13  * it's a [-0...39] range.
14  */
15 #define USER_PRIO(p) ((p) - MAX_RT_PRIO)
16 #define TASK_USER_PRIO(p) (USER_PRIO((p)->static_prio))
17 #define MAX_USER_PRIO (USER_PRIO(MAX_PRIO))

```



Exemple de processus Unix avec top I

```

6 root      9   0   0   0   0 S   0.0   0.0   0:00.00 bdflush
7 root      9   0   0   0   0 S   0.0   0.0   1:29.13 kupdated
381 root    5 -10 90400 1480 1408 S   0.0   0.3  33:38.10 XFree86

```

Autres commandes : `ps -ef` (Système V), `ps auxww` (BSD),...



Exemple de processus Unix avec top I

top permet de visualiser les processus de manière interactive

Tasks: 134 total, 5 running, 129 sleeping, 0 stopped, 0 zombie
 Cpu(s): 97.4% user, 2.6% system, 0.0% nice, 0.0% idle
 Mem: 514572k total, 505464k used, 9108k free, 41872k buffers
 Swap: 2048248k total, 57620k used, 1990628k free, 150544k cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
26487	even	16	0	185m	185m	784	R	95.7	36.8	158:20.05	prose
15440	keryell	9	0	2800	1944	1704	R	3.3	0.4	8:23.54	sshd
26540	keryell	11	0	1000	1000	768	R	0.7	0.2	0:00.60	top
17441	keryell	9	0	31244	28m	27m	S	0.3	5.6	12:18.96	mozilla-bin
1	root	9	0	436	412	388	S	0.0	0.1	1:26.93	init
2	root	9	0	0	0	0	S	0.0	0.0	0:00.14	keventd
3	root	9	0	0	0	0	S	0.0	0.0	0:36.84	kapmd
4	root	19	19	0	0	0	S	0.0	0.0	1:37.74	ksoftirqd_CPU
5	root	9	0	0	0	0	S	0.0	0.0	59:20.41	kswapd



Choix du quantum de temps dans Linux I

- Un quantum (*time slice*) identique pour tout le monde n'est pas la solution optimale
 - Trop long : peu interactif pour les autres
 - Trop court : trop de temps perdu en changements de contextes
- Clair qu'idéalement un petit quantum est préférable \leadsto souvent autour de 50 ms dans les Unix
- Mais tâche plus prioritaire tournera plus souvent même si petit quantum
- Idée : allouer un quantum variable en fonction de la priorité ou interactivité et faire un tourniquet sur tous ces processus jusqu'à épuisement des quanta
 - Quantum par défaut : 100 ms
 - Processus plus interactif ou plus prioritaire : tend vers 800 ms
 - Processus moins interactif ou prioritaire : tend vers 5 ms



Choix du quantum de temps dans Linux II

- ▶ Un quantum peut être consommé en plusieurs changement de contexte dans un tour de tourniquet (50 passage de 1 ms pour un processus très interactif par exemple)
- ▶ Recalcul des quanta après chaque tour
- Prémption lorsque
 - ▶ Un processus passe dans l'état `RUNNING` et que sa priorité est supérieure à celle du processus en cours d'exécution : changement de processus
 - ▶ Un processus a terminé son quantum de temps



File d'exécution (*runqueue*) I

- Tous les processus actifs/activables sont rangés dans une file d'exécution (*runqueue*)/processeur
- Un extrait de *runqueue* [[kernel/sched.c](#)] :

```

1  struct runqueue {
    spinlock_t lock;
3   unsigned long nr_running;
    #ifdef CONFIG_SMP
5   unsigned long cpu_load;
    #endif
7   unsigned long long nr_switches;
    unsigned long nr_uninterruptible;
9
    unsigned long expired_timestamp; /* Date de l'échange d'arrays */
11  unsigned long long timestamp_last_tick;
    task_t *curr, *idle;
13  struct mm_struct *prev_mm; /* Espace mémoire de la tâche précédente
    prio_array_t *active, *expired, arrays[2];

```



Ordonnanceur Linux 2.6 I

- Bonne interactivité même si forte charge
- Essaye de respecter une certaine équité : pas de misère
- Optimise le cas courant de quelques processus actifs mais tient la charge
- Algorithme d'ordonnancement indépendant du nombre de processeurs : $\mathcal{O}(1)$
- Chaque processeur a sa liste de processus : bonne montée en charge parallèle (extensibilité) car pas de conflit
- Exploite la localité des processus sur processeurs (affinité) : meilleure utilisation des caches,...
- Tâche de migration de processus vers des processeurs moins chargés
- Gère le rajout et la disparition des processeurs



File d'exécution (*runqueue*) II

```

15  int best_expired_prio;
    atomic_t nr_iowait;
17  ...
    /* Des choses pour le multiprocesseur et des statistiques */
19  }

```



Tableaux de priorité I

- Sur chaque file d'exécution (*runqueue*) avec n processus il faut
 - Trouver tâche plus prioritaire
 - Exécuter tâche plus prioritaire
 - La mettre après exécution si quantum expiré dans une liste des tâches ayant consommé leur quantum de temps
- ~ Besoin d'une bonne structure de données pour éviter d'aller à la pêche en $\mathcal{O}(n)$ comme dans Linux 2.4...
- Idee : s'inspirer du *bucket sort* des facteurs de la poste
 - Mettre autant de files qu'il y a de niveaux de priorité
 - Mettre chaque tâche dans la file correspondante
 - Exécuter tâche de la file non vide de plus forte priorité
- ~ Recherche en $\mathcal{O}(\#prio)$
- Certains processeurs ont des instructions pour trouver position premier bit à 1 dans une chaîne (ffs,...) en $\mathcal{O}(\log \text{sizeof}(\text{int}))$, en général 1 cycle



Recalculer les quanta de temps I

- Idee de base
 - Itérer sur toutes les tâches
 - Si une tâche a usé son quantum de temps, le recalculer (en fonction de plein de paramètres : priorité, interactivité, histoire,...)
 - ⚠ Grosse boucle $\mathcal{O}(n)$ avec des mécanismes d'exclusion mutuelle partout assez incompatible avec du temps réel... ☹

Version Linux 2.6

```

1 struct runqueue {
2     ...
3     prio_array_t *active, *expired, arrays[2];
4     ...
5 }

```

- On alloue un nouveau tableau de priorité *expired* pour les tâches ayant usé leur temps



Tableaux de priorité II

- Construire 1 tableau de bits associé aux files avec bit à 1 si file non vide
- Trouver en $\mathcal{O}(\frac{\#prio}{\text{sizeof}(\text{int})} \log \text{sizeof}(\text{int}))$ la position de la première file non vide
- Si 140 niveaux de priorité et mots de 32 bits, au plus 5 ffs pour parcourir le champ de 160 bits dans *sched_find_first_bit()*
- Comme cela ne dépend plus de n , on parle d'ordonnanceur à temps constant en $\mathcal{O}(1)$

```

1 #define BITMAP_SIZE (((MAX_PRIO+1+7)/8)+sizeof(long)-1)/sizeof(long)
2 struct prio_array {
3     unsigned int nr_active;
4     unsigned long bitmap[BITMAP_SIZE];
5     struct list_head queue[MAX_PRIO];
6 };

```



Recalculer les quanta de temps II

- Dès qu'une tâche de *active* a usé son quantum, on recalculer son quantum et on la met au bon endroit dans le tableau *expired*
- Lorsque le tableau de priorité *active* est vide on échange les 2 tableaux et on recommence
- Fait dans *schedule()* [[kernel/sched.c](#)]

```

1 array = rq->active;
2 if (unlikely(!array->nr_active)) {
3     /*
4      * Switch the active and expired arrays.
5      */
6     schedstat_inc(rq, sched_switch);
7     rq->active = rq->expired;
8     rq->expired = array;
9     array = rq->active;
10    rq->expired_timestamp = 0;
11    rq->best_expired_prio = MAX_PRIO;
12 } else
13    schedstat_inc(rq, sched_noswitch);

```



Recalculer les quanta de temps III

- ▶ Devient donc un petit $\mathcal{O}(n)$ moins violent que dans le 2.4



Calcul des priorités dynamiques et quanta I

- Un processus utilisateur a par défaut une priorité de sympathie envers les autres (*nice* $\in [-20, 19]$) : `static_prio`
- `effective_prio()` [[kernel/sched.c](#)] calcule la priorité dynamique `prio`

```
1 /*
2  * effective_prio - return the priority that is based on the static
3  * priority but is modified by bonuses/penalties.
4  */
5 /* We scale the actual sleep average [0...MAX_SLEEP_AVG]
6  * into the -5...0...+5 bonus/penalty range.
7  */
8 /* We use 25% of the full 0...39 priority range so that:
9  */
10 /* 1) nice + 19 interactive tasks do not preempt nice 0 CPU hogs.
11  * 2) nice - 20 CPU hogs do not get preempted by nice 0 tasks.
12  */
13 /* Both properties are important to certain workloads.
```



La fonction d'ordonnancement `schedule()` I

- Appelée
 - ▶ Explicitement par une tâche noyau altruiste
 - ▶ À la fin d'un quantum de temps d'un processus
 - ▶ Après chaque changement de priorité ou d'état

- `schedule()` [[kernel/sched.c](#)]

```
1 idx = sched_find_first_bit(array->bitmap);
2 queue = array->queue[idx];
3 next = list_entry(queue->next, task_t, run_list);
```



Calcul des priorités dynamiques et quanta II

```
15 /*
16  * #define CURRENT_BONUS(p) \
17  *     (NS_TO_JIFFIES((p)->sleep_avg) * MAX_BONUS / \
18  *     MAX_SLEEP_AVG)
19  * static int effective_prio(task_t *p)
20  * {
21  *     int bonus, prio;
22  *     if (rt_task(p))
23  *         return p->prio;
24  *     bonus = CURRENT_BONUS(p) - MAX_BONUS / 2;
25  *     prio = p->static_prio - bonus;
26  *     if (prio < MAX_RT_PRIO)
27  *         prio = MAX_RT_PRIO;
28  *     if (prio > MAX_PRIO - 1)
29  *         prio = MAX_PRIO - 1;
30  *     return prio;
31  */
```



Calcul des priorités dynamiques et quanta III

33 }

- En gros si une tâche dort beaucoup elle est interactive et donc elle gagne un bonus de priorité
- Un nouveau processus part avec un grand `sleep_avg` pour bien démarrer dans la vie

```

1  /*
   * task_timeslice() scales user-nice values [-20...0...19]
3  * to time slice values: [800ms...100ms...5ms]
   *
5  * The higher a thread's priority, the bigger timeslices
   * it gets during one round of execution. But even the lowest
7  * priority thread gets MIN_TIMESLICE worth of execution time.
   */
9  #define MIN_TIMESLICE → max(5 * HZ / 1000, 1)
   #define DEF_TIMESLICE → (100 * HZ / 1000)
11 #define SCALE_PRIO(x, prio) \

```



Sommeil & réveil I

- Les tâches ont aussi besoin de se reposer
- Évite les attentes actives inutiles
- Permet de faire du travail utile pendant ce temps
- Gestion des états `TASK_INTERRUPTIBLE` et `TASK_UNINTERRUPTIBLE`
- Introduction de files d'attentes (*wait queue*) pour collectionner processus à réveiller sur un événement particulier
- Le réveil des tâches en attente est fait par `__wake_up()` [[kernel/sched.c](#)]

Exemple de `do_clock_nanosleep()` [[kernel/posix-timers.c](#)]



Calcul des priorités dynamiques et quanta IV

```

→ max(x * (MAX_PRIO - prio) / (MAX_USER_PRIO / 2), MIN_TIMESLICE)
13
static unsigned int task_timeslice(task_t *p)
15 {
    → if (p->static_prio < NICE_TO_PRIO(0))
17 → return SCALE_PRIO(DEF_TIMESLICE * 4, p->static_prio)
    → else
19 → return SCALE_PRIO(DEF_TIMESLICE, p->static_prio)
}

```



Sommeil & réveil II

```

1 long
2 do_clock_nanosleep(clockid_t which_clock, int flags, struct timespec
   {
4   ...
   /* Crée un élément de liste abs_wqueue avec la tâche courante comme
6   valeur, style abs_wqueue = CONS(current, NIL) */
   DECLARE_WAITQUEUE(abs_wqueue, current);
8   ...
   init_timer(&new_timer);
10  new_timer.expires = 0;
   new_timer.data = (unsigned long) current;
12  new_timer.function = nanosleep_wake_up;
   abs = flags & TIMER_ABSTIME;
14  ...
   if (abs && (posix_clocks[which_clock].clock_get !=
16  posix_clocks[CLOCK_MONOTONIC].clock_get))
   /* En gros fait
18  nanosleep_abs_wqueue = CONS(current, nanosleep_abs_wqueue)
   add_wait_queue(&nanosleep_abs_wqueue, &abs_wqueue)

```



Sommeil & réveil III

```

20
21  __new_timer.expires = __jiffies + __left;
22  __set_current_state(TASK_INTERRUPTIBLE);
23  add_timer(&new_timer);
24
25  schedule();
26  ...
27  return 0;
28  }

```



Équilibrage de charge I

- Multiprocesseur capable d'exécuter plusieurs processus en parallèle
- Pour des raisons d'efficacité, tâches associées à un processus
- Il se peut que des processeurs soient beaucoup plus chargés que d'autres... ☹
- `schedule()` [[kernel/sched.c](#)] appelle régulièrement `load_balance()` [[kernel/sched.c](#)]
 - ▶ Si pas de déséquilibre de plus de 25 % ne fait rien
 - ▶ Sinon prend une tâche de haute priorité (à équilibre principalement) qui n'a pas tourné depuis longtemps (cache...) et essaye de la bouger



Préemption I

- Effectué par `schedule()` [[kernel/sched.c](#)] qui appelle `context_switch()` [[kernel/sched.c](#)]
- Peut arriver
 - ▶ Processus utilisateur : au moment de revenir dans l'espace utilisateur depuis un appel système ou une interruption
 - ▶ Tâche noyau : après un retour d'interruption, sur blocage, appel explicite à `schedule()` ou lorsque le noyau redevient préemptif
- Une tâche dans le noyau est préemptible si elle ne possède aucun verrou de posé (comptabilisés par le champ `preempt_count` de la `thread_info`)
- La préemption est globalement contrôlée par le drapeau `need_resched` qui est positionné si quelqu'un a besoin d'avoir la main



Affinité tâche-processeur I

- Une tâche tourne sur *un* processeur
- Est-ce indépendant de la localisation ?
 - ▶ Fonctionnellement oui...
 - ▶ ... ⚠ performances!
- Architecture sous-jacente complexe
 - ▶ Mémoires caches dans les processeurs
 - ▶ Architectures NUMA (*Non Uniform Memory Access*)
 - ▶ Architectures hétérogènes : cartes réseaux et processeurs
 - Faire tourner un processus qui traite des paquets d'une interface réseau sur un processeur proche de celle-ci

~ **Besoin de contrôler finement la localisation**

Cf. `man` de `taskset(1)`, `sched_setaffinity(2)`,
`sched_getaffinity(2)`



Unix : 2 ordonnancement I

Interaction avec mémoire virtuelle et mémoire secondaire car 1 processeur ne peut exécuter que des instructions en mémoire principale

- Au niveau bas : ordonnanceur choisit de rendre actif 1 processus activable *présent en mémoire principale*
- Au niveau haut : ordonnanceur gère le va-et-vient (*swap*) entre la mémoire principale et la mémoire secondaire. Modifie les priorités pour favoriser processus qui viennent de rentrer en mémoire principale, etc.

Choix averti de l'ordonnanceur → noyau a constamment en mémoire principale une table de l'ensemble des processus avec données nécessaire au fonctionnement de l'ordonnanceur



Politiques temps réel I

En plus de la politique non temps réel SCHED_NORMAL il y a 2 politiques plus temps réel dans Linux 2.6

- SCHED_FIFO
 - ▶ Une telle tâche passera toujours avant une SCHED_NORMAL
 - ▶ Prémption
 - ▶ Tourne tant qu'elle n'est pas bloquée ou fait un `sched_yield()` ou une tâche temps réel plus prioritaire veut tourner
- SCHED_RR
 - ▶ Comme SCHED_FIFO mais avec des quanta de temps
 - ▶ Tourniquet entre processus de même priorité

Pas de garantie dure mais permet de faire plus de chose que SCHED_NORMAL classique de base
Cf. `man chrt(1)`, `sched_setscheduler(2)`



Ordonnancement et va et vient I

- Certaines informations liées à un processus restent en mémoire principale
 - ▶ Paramètres d'ordonnancement
 - ▶ Adresses sur disques des segments/pages du processus
 - ▶ Informations sur les signaux UNIX acceptés
 - ▶ ...
- D'autres informations suivent le processus quand il est mis (*swap out*) en mémoire secondaire
 - ▶ Contexte matériel
 - ▶ Table de descripteurs de fichiers
 - ▶ Pile du noyau (stockage des variables locales du noyau lorsqu'il traite le processus) et la pile « utilisateur » (stockage des variables locales lors des calculs)



Petite conclusion sur ordonnancement Linux I

- De gros progrès dans Linux 2.6 !
- Noyau préemptif
- Rajout de priorités fixes temps réel
- Ordonnanceur efficace et rapide : regarder <http://developer.osdl.org/craiger/hackbench/index.html>
- Nouvelles politiques d'ordonnancement temps réel en plus de Unix dynamique classique
- Parti du monde du PC Linux aborde sans complexe toute l'étendue informatique : des systèmes embarqués aux super-calculateurs parallèles NUMA
- Monde du logiciel libre
 - ▶ Pas captif d'un produit fermé
 - ▶ On a les sources pour regarder dedans et adapter !
 - ▶ Plein de documentation et d'exemples disponibles
 - ▶ Grande communauté (support gratuit ou payant)



Des ennuis : inversion de priorité I

Supposons

- 3 processus P_1 , P_2 et P_3
- Un système à priorités fixes dures
- Priorités $p(P_1) > p(P_2) > p(P_3)$



Solutions possibles I

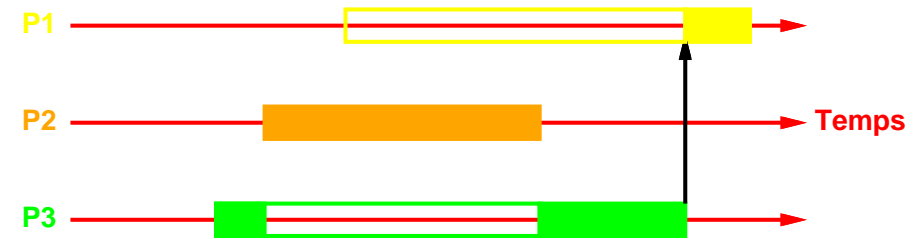
Solutions possibles :

- Héritage de priorité : faire hériter (provisoirement!) P_3 de la priorité de P_1
- *Priority Ceiling Algorithms* : associer à une ressource une priorité qui sera prêtée aux tâches qui utilisent ou attendent cette ressource



Des ennuis : inversion de priorité II

Cas pathologique :



- P_1 attend un message de P_3
- P_2 fonctionne à la place de P_3 car plus prioritaire
- Paradoxe : P_2 fonctionne à la place de P_1 et est donc plus prioritaire !



Mars Pathfinder Mission on July 4th, 1997

http://www.research.microsoft.com/mbj/Mars_Pathfinder

The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface. Successes included its unconventional "landing"-bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web.



Tâche martienne de priorité forte I

A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus (1553 bus). Access to the bus was synchronized with mutual exclusion locks (mutexes).

Tâche martienne de priorité moyenne I

The spacecraft also contained a bus communications task that ran with medium priority.



Tâche martienne de priorité faible I

The meteorological data gathering task (ASI/MET) ran as an infrequent, low priority thread, and used the information bus to publish its data. When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. If an interrupt caused the information bus thread to be scheduled while this mutex was held, and if the information bus thread then attempted to acquire this same mutex in order to retrieve published data, this would cause it to block on the mutex, waiting until the meteorological thread released the mutex before it could continue.

Bug martien I

Most of the time this combination worked fine. However, very infrequently, it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.



Debug martien I

- Vitesse de la lumière Terre-Mars ≈ 14 mn ☹
- Récupérer une version du système temps réel (vxWorks) spécialement instrumentée pour le debug avec collecte de trace d'exécutions
- Faire tourner avec les mêmes tâches que sur Mars une maquette identique
- Bug apparu au bout de 18 heures
- Analyse : sémaphore utilisé sans option d'héritage de priorité (non mis par défaut pour des raisons d'optimisation)
- Besoin de modifier le code sur... Mars !
- Conception d'une rustine logicielle (*patch*)
- Mise en place avec une procédure spéciale... qui avait été prévue !



197 / 309

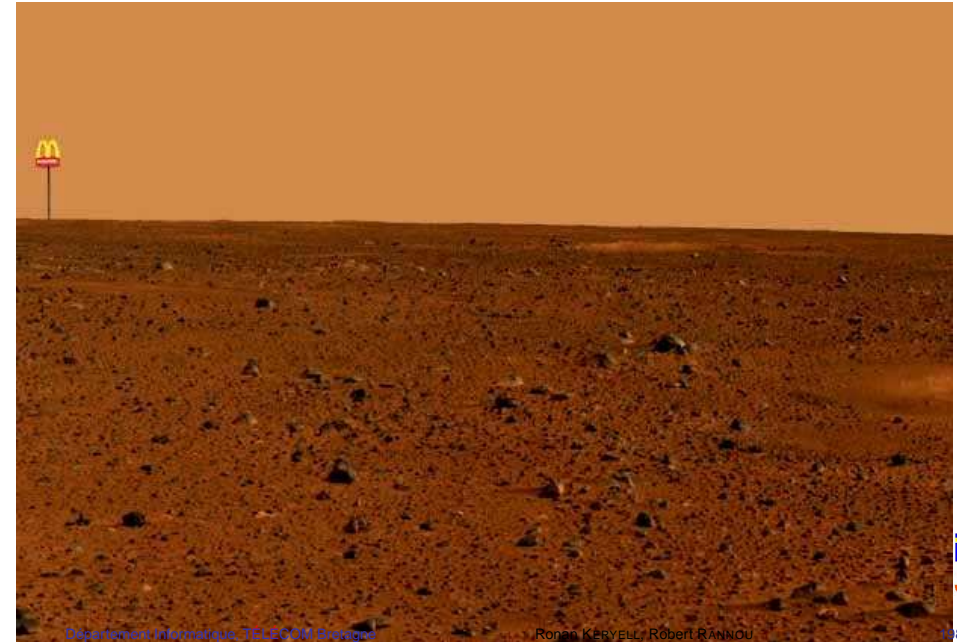
Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



199 / 309

Debug martien II



198 / 309

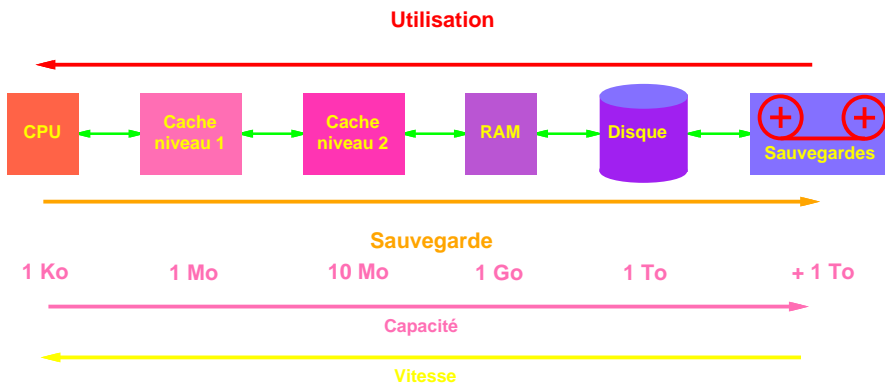
Gestion des mémoires I

- Plusieurs types de mémoire
 - ▶ Grosses mémoires : pas chères
 - Magnétiques : disques durs, bandes, (disquettes),...
 - Optiques : DVD, (CDROM),...
 - Problème : lentes! ☹
 - ▶ Mémoires rapides :
 - FLASH-ROM : moyennement rapide
 - DRAM : rapide
 - SRAM : très rapide
 - Problème : faible capacité, très chères ☹
- À moins d'être très riche (cf vieux supercalculateur Cray,...) besoin d'un compromis !
- Idée : garder le plus près possible de l'utilisation les données dans la mémoire la plus rapide

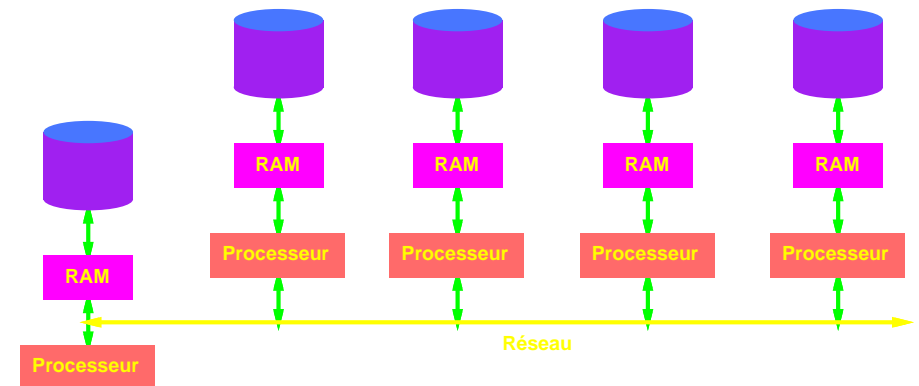


200 / 309

Hiérarchie mémoire I



Hiérarchie mémoire version réseau I



Dépasser des limitations d'adressage I

- Réduction du coût du matériel dans applications grand public (lave linge,...) : microcontrôleur 1-4-8 bits
- Mémoire adressable < mémoire physique
- Comment accéder à des grosses mémoires lorsqu'on ne peut manipuler que des données sur 8 voire 16 bits (65536 valeurs d'adresses) ?
- Utilisation d'une fonction de translation $a_p = f(a_v)$ donnant l'adresse physique à partir d'une adresse virtuelle



Dépasser des limitations d'adressage II

- Cas des vieux PC avec i8088 : utilisation d'un registre de segments pour accéder à 1 Mo avec des registres d'adresse sur 16 bits :

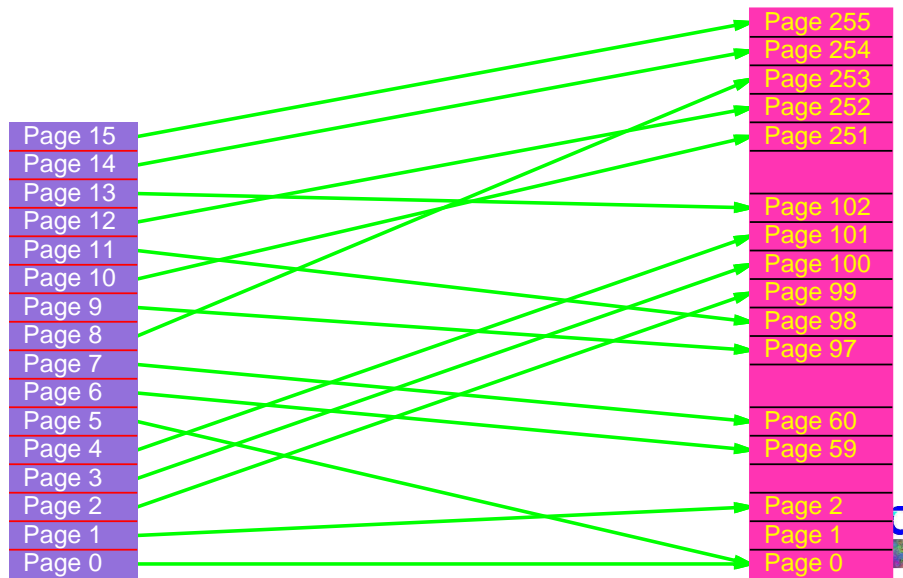
$$a_p = a_v + 16 \times s$$

Permet simplement d'avoir plusieurs programme simultanément chargés en mémoire (multiprogrammation) : pour changer la zone du programme exécuté changer s

- Cas du Goupil 3 avec 6809 : pagination avec accès possible à 16 pages de 4 Ko parmi 256 pages (1 Mo) sélectionnable avec l'aide du système d'exploitation



Dépasser des limitations d'adressage III



Intérêts de la mémoire virtuelle I

- Adresse physique des objets pas forcément connue à la compilation ni l'édition de lien
- Permet de changer adresse d'un objet au chargement ou en cours d'exécution
- Permet de faire apparaître un objet à *plusieurs* adresses (duplication)
- Permet simplement d'avoir plusieurs programme simultanément chargés en mémoire (multiprogrammation)
- Autorise une meilleure gestion de la mémoire : permet de donner une vision plus propre de la mémoire
Par exemple peut donner un gros bloc de mémoire à une application même si physiquement il n'y a pas assez de mémoire contiguë, sans avoir à la compacter
- Partage de zones mémoire par plusieurs processus

Dépasser la mémoire physique I

- Cas avec processeurs courants 32 ou 64 bits : mémoire physique (1 Go) < mémoire adressable (16 Eo)
- Simuler la mémoire manquante avec de la mémoire *moins chère* : mémoire secondaire (sur disque dur)
- Utilise grand espace disponible pour se simplifier la vie : on peut espacer les objets sans (trop) compter
Par exemple si besoins de piles de taille inconnue pour des processus légers p sur une machine à 64 bits, on peut les placer aux adresses $p.2^{32}$ et elles peuvent grossir chacune jusqu'à 4 Go

Intérêts de la mémoire virtuelle II

- ▶ Code si plusieurs instances du même programme
- ▶ Variables globales si processus légers
- ▶ Segments de mémoire partagée demandés explicitement par différents processus via les IPC (Inter Process Communication)
- Permet de virtualiser le concept de mémoire de manière arbitraire : cela ressemble à de la mémoire mais c'est un disque, un fichier, un écran,....

Mémoire virtuelle (traduction d'adresse) généralement effectuée par une MMU (*Memory Management Unit*)

Grande mémoire virtuelle I

- En général taille mémoire virtuelle \gg mémoire physique
- Comment réaliser la MMU qui calcule une adresse physique sur 40 b à partir d'une adresse virtuelle sur 64 b ?

$$f : a_v \longrightarrow a_p$$

$$N/2^{64} \mathbb{N} \longmapsto (N/2^{40} \mathbb{N})^{2^{64}}$$

Nécessite une table de 2^{64} entrées de 40 bits ! Soit bien plus que la mémoire de tout ordinateur ☹



Grande mémoire virtuelle II

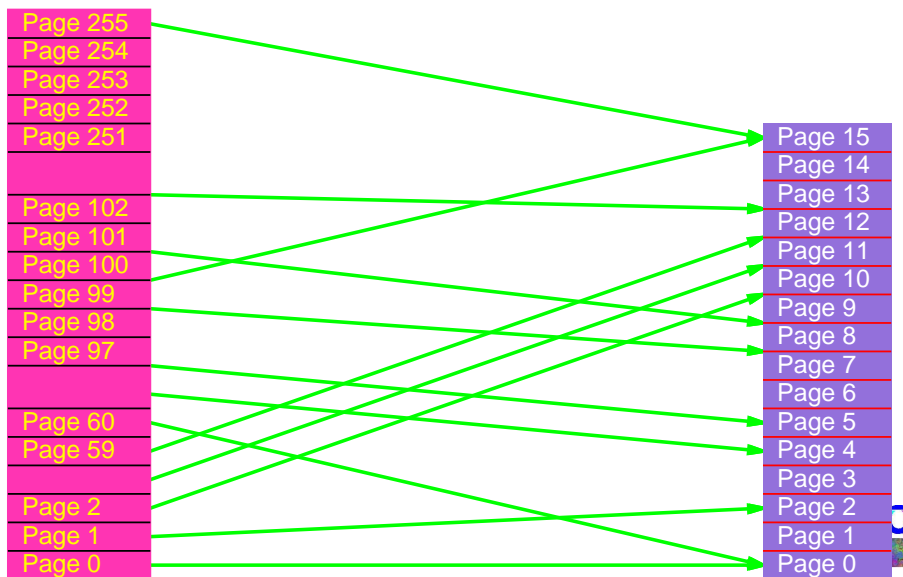
- ☕ Idée 1 : raisonner au grain d'une page de taille t et non plus d'une case mémoire :

$$a_p = f\left(\frac{a_v}{t}\right) + (a_v \% t)$$

→ Divise la table de la MMU par t ! Si pages de 4 Ko, plus « que » 2^{52} pages



Grande mémoire virtuelle III



Traduction des adresses des pages I

Récupère l'adresse de la page physique à la ligne correspondant à l'adresse de la page virtuelle

	<i>Existence</i>	<i>Page physique</i>
0	0	
1	1	0x23fe
2	1	0x5fde
3	1	0x2345
4	0	
5	1	0x0
6	0	
	⋮	⋮

Traduction de l'adresse virtuelle 0x37890 (3 | 0x7890) en adresse physique 0x23457890 (0x2345 | 0x7890)



Tailles de pages I

Compromis

- Petites pages
 - ▶ Grande liberté de traduction
 - ▶ Nécessite de nombreuses entrées de traduction
- Grandes pages
 - ▶ Utile pour allouer rapidement de grosses zones de mémoire
 - ▶ Petite table des pages

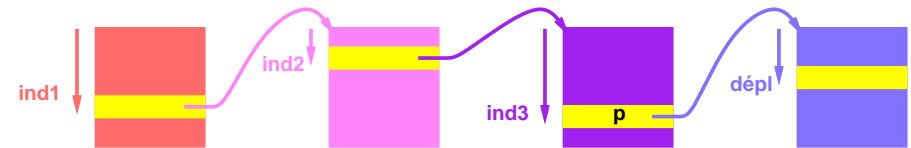
~ Certains microprocesseurs ont plusieurs tailles de pages possibles



Pagination à 3 niveaux sur SPARC I

- Besoin de compacter la représentation de la table (creuse) de traduction
- Choix par exemple d'une pagination à 3 niveaux de table dans les processeurs

ind1 | ind2 | ind3 | dépl traduit en p | dépl



Défaut de page I

- Si une page n'existe pas (invalide) la MMU génère une interruption
- Le système d'exploitation
 - ▶ Regarde dans la table des pages s'il existe une traduction correcte (fait matériellement par certains processeurs)
 - ▶ Si la page existe mais pas en mémoire physique par exemple, on la charge depuis la mémoire secondaire (mécanisme d'échange ou *swap*)
 - ▶ Éventuellement on évacue des pages depuis la mémoire physique (si pages modifiées) pour faire de la place
- Cette interruption peut remonter au niveau de l'application et être utilisée (signal *segmentation violation* sous Unix)



Vision objet de la mémoire virtuelle I

- Mécanisme de défaut de page : déclenche l'exécution d'une fonction quelconque par interruption
- Permet de surcharger les « méthodes » écriture(*type *addr*, *type v*) et lecture(*type *addr*) d'une zone mémoire
- Permet de simuler n'importe quel comportement
 - ▶ Mémoire artificielle
 - ▶ Émulation d'un autre ordinateur
 - ▶ Mémoire vidéo
 - ▶ Mémoire virtuellement partagée (SVM) transmise ailleurs par réseau
 - ▶ ...



Mémoire virtuelle et localité I

- Localité spatiale : si un objet est référencé, des objets proches en mémoire seront référencés bientôt (tableaux, structures, variables locales, variables d'instances)
- Localité temporelle : si un objet est référencé, il sera à nouveau référencé (boucles récursivité, ...)

→ Utiliser ces observations pour choisir les pages à échanger



Accélérer la traduction d'adresse I

- Utiliser une mémoire associative pour stocker les traductions les plus courante : cache spécialisé (*Translation Look-aside Buffer*)
- Marche bien car localité spatiale se retrouve au sein d'une page
- Table des pages consultée (par le matériel ou le système d'exploitation) que lorsque la traduction n'est pas dans le TLB
- Le contenu du TLB constitue l'espace de travail (*working set*). En cas de changement de processus on intérêt à sauvegarder et restaurer ces traductions pour garder la localité



Limiter les défauts de pages I

Comment éviter que le système passe son temps à traiter des défaut de pages ?

→ Bien choisir la page à vider

- Réserver un espace à chaque processus dans la mémoire physique
Le processus qui génère un défaut de page cherche une page à vider parmi ses propres pages
- Ou bien, faire le choix des pages à vider parmi toutes les pages en mémoire



Contenu d'un TLB I

- Numéro de page virtuelle
- Numéro de page physique
- Bit de validité
- Bit indiquant si la page a été modifiée par une écriture du processeur
- Bits de protection (autorisation) d'écriture, de lecture, d'exécution
- Éventuellement bit indiquant que la page a été lue ou accédée (exercice : comment le simuler ?)



Choix des pages à enlever du TLB I

Nombreux algorithmes possibles

- FIFO (première rentrée, première sortie) : un peu violent
- NRU (*Not Recently Used*) : régulièrement met à 0 le bit indiquant l'accès. Si à la fin du quantum de temps le bit est toujours à 0 : candidat à l'éjection
- LRU (*Least Recently Used*) : garder la date de dernière utilisation de chaque page et virer la plus ancienne

Donner de l'espace aux processus I

- Un processus doit avoir suffisamment de pages pour avancer sans trop de défauts de pages
- Si pas possible, virer de la mémoire centrale un autre processus en concurrence
- Ne pas libérer des pages partagées par d'autres processus avec le processus qui nous intéresse
- Ne pas libérer des pages bloquées pour des entrées-sorties
- Garder du mou en mémoire centrale pour l'allocation de nouveaux espace : existence en tâche de fond d'un démon qui vide des pages régulièrement

Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion

Architectures virtuelles I

≡ Différences entre ce que voit l'utilisateur et la réalité

- Communication par message \nrightarrow réseau
 - ▶ Paradigme par messages peut utiliser de la mémoire partagée
- Réseau \nrightarrow communication par message
 - ▶ Paradigme variables partagées au dessus d'un réseau avec des messages

Transparence

...  ; mais attention parfois aux performances !

Machines virtuelles matérielles I

- IBM OS/360 : traitement par lots des années 1960
- Aller plus loin : volonté de partage de temps
- Système à temps partagé :
 - ▶ Multiprogrammation
 - ▶ Machine étendue avec interface plus sympathique que la vraie
- ☕ Idée : dissocier les 2
- Permet de faire tourner *n'importe quoi* dans la machine virtuelle (VM *virtual machine*)
- Programme utilisateur dans chaque VM/370, voire un autre système d'exploitation dans VM/370 : CMS spécialisé pour l'interactif, Unix,...



Intérêts de la virtualisation I

- Sécurité
 - ▶ Attaques cernées dans la VM
 - ▶ 1 VMM prouvé qui délimite des VM avec des OS non prouvés
- Factorisation ressources
 - ▶ Machines (pas la peine d'avoir 1 Mac + 1 PC sous Windows + 1 Sun sous Solaris + 1 truc sous Linux +...)
 - ▶ Processeurs
 - ▶ Disques
 - ▶ Interfaces réseau
- Administration simplifiée
 - ▶ 1 administration + clonage
- Tolérance aux pannes
 - ▶ Migration de machines virtuelles complètes en cas de panne ou de charge trop élevée
 - ▶ Snapshot d'état



Intérêts de la virtualisation II

- SOA (Service-Oriented Architecture)
 - ▶ Infrastructures à la demande
 - ▶ Hébergement moins cher
 - ▶ Mutualisation
- Permet de faire tourner vieilles applications sur matériel qui n'existe plus (autocom Alcatel, musées de l'informatique...)
- Mise au point de pilotes ou OS sous débogueur dans VM
- Recherche en architecture : simulation au cycle près sans vrai matériel
 - ▶ Projet de processeur CryptoPage à TÉLÉCOM Bretagne
- Co-conception (*codesign*) OS-matériel

... Au prix d'une \pm légère perte d'efficacité



Machines virtuelles logicielles I

- Principe encore vivant :
 - ▶ JVM : Java \rightsquigarrow JVM *compile once, run everywhere*
 - ▶ Langages PERL (Parrot), Python, C# (CDL),...
 - ▶ Emacs
 - Langage de programmation Emacs-LISP, interpréteur et machine virtuelle (exécute du *bytecode*)
 - Système d'exploitation portable : processus, intercommunication,...
 - Système de multi-fenêtrage
 - Peut aussi servir d'éditeur de texte... ☺
 - ▶ Faire tourner des PCs virtuels dans des PCs : VMware, QEMU, vieux *plex86* <http://www.plex86.org/>,...
 - ▶ Faire tourner des PCs sur n'importe quoi (PC, Mac, Sun,...) : BOCHS <http://www.bochs.com/>, QEMU, PTLsim (précis au cycle près)...



Machines virtuelles logicielles II

- ▶ <http://www.deanliou.com/WinRG/WinRG.htm> « *Microsoft's Windows RG (Really Good Edition)* » de James CLIFFE : des applications Windows dans Windows RG en Flash qui tourne dans une machine virtuelle Flash qui tourne dans un navigateur WWW qui tourne dans un processus qui tourne sur un processeur physique (qui tourne dans...)



Machines virtuelles - fonctionnement I

- Moniteur de machine virtuelle juste au dessus du matériel
- Prend en charge la multiprogrammation
- Copie conforme de matériel si bas niveau : simule
 - ▶ Modes noyau, utilisateur
 - ▶ Mémoire virtuelle
 - ▶ Interruptions
 - ▶ Dispositifs virtuels d'entrée sortie : lèvent une interruption et la machine physique fait l'opération
 - ▶ Lecture de secteurs disques sur un disque virtuel en commandant le contrôleur disque virtuel (*minidisk* sur IBM VM 370)
 - ▶ Écriture dans la mémoire écran : déclenche une exception au moment de l'écriture dans la mémoire → transformé en écriture dans une fenêtre de l'écran physique
- L'aspect multiprogrammation reste simple : commuter des machines virtuelles



Machines virtuelles - fonctionnement II

- ⚠ Difficile à optimiser
 - ▶ Fait tourner 2 processus identiques sur 2 OS identiques en même temps sans partage de pages...
 - ▶ Écriture dans l'écran virtuel même si fenêtre non visible...
- VMware & QEMU : simule plusieurs machines x86 sur une machine x86
 - ▶ Utilise le x86 sous-jacent pour exécuter la majorité du code → rapide
 - ▶ Mémoire simulée par la mémoire virtuelle via la MMU
 - ▶ Périphériques détournés par la MMU et simulés
- BOCHS <http://www.bochs.com/> : simulateur complet de machine virtuelle à processeur A sur processeur B
 - ▶ Instructions de A interprétées par B ou traduites (compilées) et exécutées par B
 - ▶ Mémoire et périphériques gérés par l'interpréteur



Machines virtuelles - fonctionnement III

- Nouveaux processeurs rajoutent des instructions pour virtualiser de manière efficace instructions superviseurs
 - ▶ AMD Pacifica, Intel VT
 - ▶ Il faut aussi virtualiser matériel car si requêtes DMA et ES partent sur bus **physiques**... ☹
 - Apparition de matériel qui gère virtualisation
 - Interfaces Ethernet physique gérant plusieurs Ethernet virtuels
 - Canaux DMA gérant virtualisation et MMU virtuelle

Cf. biblio



Paravirtualisation I

- Faire tourner un OS dans une machine virtuelle est difficile
 - Un OS ne suppose pas qu'il doit économiser le processeur
 - Dispositifs d'E/S gèrent rarement virtualisation

☕ Idée ☕

Modifier OS pour prendre en compte machine virtuelle hôte ~
paravirtualisation

Exemple : Xen qui tourne dans un OS hôte pour bénéficier infrastructure (gestion mémoire, périphériques...)

- Modification ordonnanceur pour passer la main aux autres OS
- Modification pilotes de périphériques pour passer la main à périphériques (gros du boulot, source de bugs et baisse de performance)



Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Paravirtualisation II

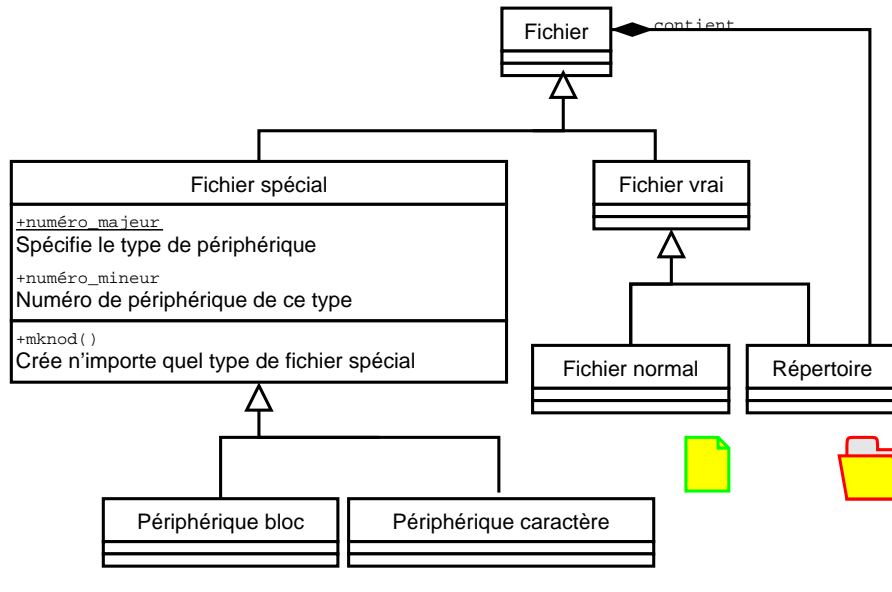
- OS hôte tourne dans « domaine » 0
Nécessite d'avoir sources de l'OS...



Hiérarchie de classes de fichiers Unix simplifiée I



Hiérarchie de classes de fichiers Unix simplifiée II



Descripteur de fichier I

- Fichier désigné par un chemin (nom) absolu (/bin/sh) ou relatif au répertoire courant du processus (essai.c)
 - Chaînes de caractères sont traitées inefficacement par les processeurs : pénible
 - Fichier ≡ objet dans système d'exploitation. Sous Unix :
 - Contient une référence au système de fichiers contenant ce fichier
 - Contient une référence au v-nœud (v-node) du fichier dans son système de fichier. v-nœud implémenté par exemple par un i-nœud
 - Comment manipuler depuis n'importe quel langage (orienté objet ou non) ?
- Besoin d'état par accès et non plus par fichier :
 - Stocke un pointeur de lecture/d'écriture courante
 - Des droits d'accès

Descripteur de fichier II

- Idee en Unix : accéder à chaque fichier à partir d'un entier (positif), le *descripteur de fichier*, représentant l'objet d'accès au fichier et non le fichier lui-même
- Association d'un chemin d'accès à un descripteur de fichier = *ouvrir* un fichier


```
int fd = open(char *path, int flags, int perms)
```

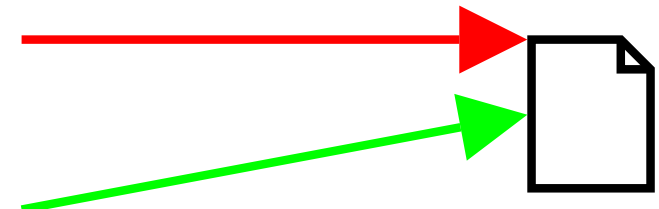
 - flags permet de choisir de pouvoir faire des lectures ou écritures par la suite, de créer un nouveau fichier,...
 - perms permet de changer les droits par défaut (fichier exécutable ? Mon voisin a le droit de le lire ?...)
- Désassociation d'un descripteur de fichier lorsqu'on n'en a plus besoin = *fermer* un fichier


```
int close(int fd)
```

Sémantique des fichiers Unix I

- Un fichier sous Unix peut être utilisé par plusieurs processus en même temps
- Il y a autant de descripteur qu'il y a d'utilisations différentes du fichier

P1 : fd1



P2 : fd2

- Les écritures ou lectures sont atomiques

Sémantique des fichiers Unix II

- On a aussi des mécanismes de verrous (indicatifs, obligatoires, bloquants ou non) via l'appel `fcntl()`



Quelques méthodes associées à des fichiers II

- `int lstat(const char *file_name, struct stat *buf)` : idem mais dans le cas d'un lien analyse le lien au lieu de la cible
- `int symlink(const char *oldpath, const char *newpath)` : crée un lien symbolique
- `int link(const char *oldpath, const char *newpath)` crée un lien *hard* (une nouvelle entrée) dans un répertoire pour un fichier déjà existant
- `int unlink(const char *pathname)` supprime une entrée d'un répertoire, voire efface le fichier
- `int pipe(int filedes[2])` crée une paire de descripteurs de fichier : on peut lire dans `filedes[0]` ce qu'on écrit dans `filedes[1]`
- `int socket(int domain, int type, int protocol)` crée un descripteur de fichier de communication



Quelques méthodes associées à des fichiers I

Cf man `[-s] 2`, outre `open()` et `close()` déjà vus à utiliser avec `#include <unistd.h>`

- `ssize_t read(int fd, void *buf, size_t count)` : lit à partir de la position courante au plus `count` caractères
- `ssize_t write(int fd, const void *buf, size_t count)` : essaye d'écrire `count` caractères
- `off_t lseek(int fildes, off_t offset, int whence)` : déplace le point courant en absolu, relatif ou depuis la fin selon `whence`
- `int stat(const char *file_name, struct stat *buf)` : récupère toutes les caractéristiques d'un fichier
- `int fstat(int fd, struct stat *buf)` : idem sur un fichier ouvert




Quelques méthodes associées à des fichiers III

⚠ Quelques blagues avec les gros fichiers (>2Go) sur un ordinateur ne manipulant pas des données 64 bits (32 bits...) : nécessité de proposer une version 32 et 64 bits de certains appels systèmes... ☹



Passer un descripteur à ton voisin I

Et si on veut partager un fichier ET l'endroit courant ?

-  Partage du *même* descripteur de fichier
 - Duplication d'un descripteur via `dup()` et `dup2()`
 - Naturellement dupliqués via le clonage (`fork()`)
 - Hérité via `exec()` : base du shell qui peut manipuler les fichiers de ses enfants pour gérer |, < ou >...
- Conventions de numérotation des descripteurs de fichier pour un processus Unix :
- ▶ 0 : entrée standard (`stdin`)
 - ▶ 1 : sortie standard (`stdout`)
 - ▶ 2 : sortie pour les messages d'erreur (`stderr`, en général non tamponnée dans les bibliothèques)

Un processus peut ne connaître que ces descripteurs alors que c'est le shell qui leur a associé des fichiers ou autres



Les fonctions d'E/S du C(++) I

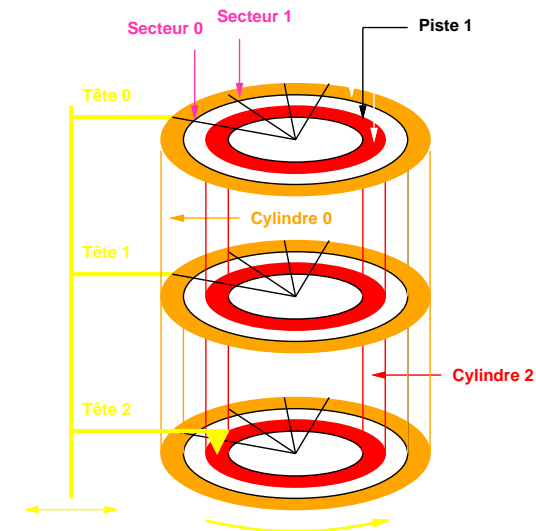
Le C (stdio.h) ou les stream de C++ en rajoute une couche

Remarque : cela fait partie de la bibliothèque, pas vraiment du système d'exploitation...


- read, write,... : assez bas niveau
- Ces appels systèmes provoquent un passage dans le noyau à chaque fois : lourd
- Besoin de fonctions légères et de plus haut niveau : tamponne les entrées-sorties et envoie le tout par bloc ~ moins de passage dans le noyau
- [f]printf()
- [f]scanf()
- fopen()
- fclose()



(Disques magnétiques) II



Unix FFS optimisé pour les disques I

- Partition (ou tranche) : ensemble de cylindres consécutifs  ↗ localité
- Allocation dans des cylindres consécutifs
- Allocation dans des secteurs consécutifs avec un saut (temps de rotation)
- Laisse des cylindres vides régulièrement pour allouer plus rapidement de nouveaux secteurs

Problèmes des caches dans les contrôleurs disque qui éloignent de la réalité...




Partitionnement des disques II

- Chaque système d'exploitation a sa convention de partitionnement (n'est pas déterminé au niveau du disque lui-même)
- Solaris découpe en 8 ou 10 partitions avec comme convention l'usage courant
 - 0 : contient /
 - 1 : du swap
 - 2 : tout le disque (déborde sur les autres...)
 - 3 : /export sur un serveur
 - 4 : /export/swap sur un serveur
 - 5 : /opt
 - 6 : /usr
 - 7 : /home ou /export/home
 - 8 : sur PC contient le système de boot et pointe au début du disque





Partitionnement des disques I

- Découpe des disques pour des usages différents
- Augmente la localité (et donc performances) des accès au sein de chaque partition
- Limites infranchissables (contre certains utilisateurs expansifs)
- Fournit des zones brutes pouvant avoir chacune leur système de fichier (indépendant et même de type différent) voir sans (swap, base de donnée)
- Peuvent avoir des politiques d'exportation différentes sous NFS
-  Éviter d'avoir 2 partitions qui se recouvrent sans raison...
- Partitionnement fait automatiquement et graphiquement par la procédure d'installation
- Mais en cas de problème sur un disque, de remplacement, de changement du partitionnement : connaissance utile



Partitionnement des disques III

9 : sur PC contient les blocs alternatifs utilisés à la place d'autres en panne et pointe après la partition 8

- Sur PC nécessité d'une « convention collective des OS » pour faire du multi-OS. Convention de partitionner un disque jusqu'en 4 partitions via `fdisk`. Solaris prend une de ces partitions et la repartitionne avec son propre système
-  La loi de Murphy veut que le partitionnement choisi n'est jamais le bon... En général, / et le swap sont trop petits
-  Loi de Murphy numéro 2 : difficile de changer le partitionnement dynamiquement...



Utilitaire format I

Utilisation :

- Installation d'un nouveau disque :
 - ▶ Formattage
 - ▶ Partitionnement
- Affiche les disques reconnus sur le système
- Affiche des informations et leur partitionnement
- Test d'un disque
- Réparation d'un disque
- Destruction du contenu (sensible...) avant renvoi



format à l'œuvre II

Adresses aussi en *cylindre/tête/bloc*

Format propose un partitionnement par défaut

- `current` décrit le disque courant


```
format> cu
Current Disk = c0t4d0: bassine
<SEAGATE-ST39102LW-0004 cyl 6922 alt 2 hd 12 sec 214>
/sbus@1f,0/espdma@e,8400000/esp@e,8800000/sd@4,0
```
- `format` reformate le disque
- `backup` récupère un label (la table des matières du disque) de secours en cas de perte du label principal
- `analyse` permet de tester le disque avec un effet plus ou moins destructeur



format à l'œuvre I

Les commandes sont abrégées

- `partition` gère et affiche le partitionnement (`prtvtoc` donne aussi l'information)

```
partition> p
Current partition table (original):
Total disk cylinders available: 253 + 2 (reserved cylinders)
```

Part	Tag	Flag	Cylinders	Size	Blocks
0	root	wm	3 - 28	203.95MB	(26/0/0) 417690
1	swap	wu	29 - 170	1.09GB	(142/0/0) 2281230
2	backup	wm	0 - 252	1.94GB	(253/0/0) 4064445
3	unassigned	wm	0	0	(0/0/0) 0
4	unassigned	wm	0	0	(0/0/0) 0
5	unassigned	wm	0	0	(0/0/0) 0
6	usr	wm	171 - 252	643.23MB	(82/0/0) 1317330
7	unassigned	wm	0	0	(0/0/0) 0
8	boot	wu	0 - 0	7.84MB	(1/0/0) 16065
9	alternates	wu	1 - 2	15.69MB	(2/0/0) 32180



format à l'œuvre III

- `repair` répare 1 bloc du disque en le rajoutant dans la liste des défectueux et en remet un autre à la place. En cas de problème matériel
- `defect` permet de gérer la liste des défauts (1 gros disque est rarement parfait...)
- `volname` donne un nom au disque. Au CRI on donne des noms de conteneurs pour s'y retrouver. `goutte` aura moins d'octets que `bassine`. Même nom qu'on retrouve monté
- `label` entérine les modifications

`/etc/format.dat` contient les paramètres de formatage (géométrie, etc) des disques connus

- Disque récent (SCSI-2) en bon état : informe directement `format`
- Sinon, lire la documentation ou récupérer un `format.dat` récent (ou le contraire !)



Étape fdisk sur PC I

- Partage du disque entre plusieurs OS

```
Total disk size is 788 cylinders
Cylinder size is 16065 (512 byte) blocks
```

Partition	Status	Type	Cylinders		Length	%
			Start	End		
=====	=====	=====	=====	=====	=====	=====
1	Active	IFS: NTFS	0	50	51	6
2		DOS-BIG	51	101	51	6
3		Solaris	102	356	255	32
4		UNIX System	357	592	236	30

SELECT ONE OF THE FOLLOWING:

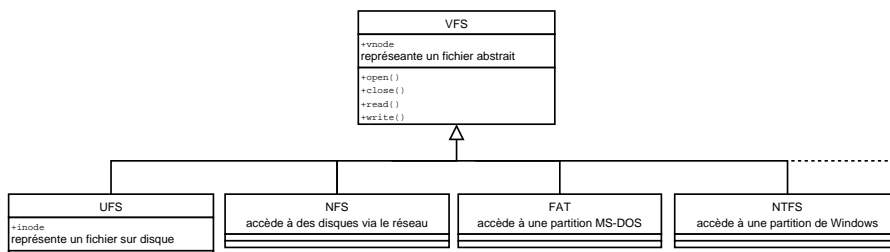
1. Create a partition
2. Specify the active partition
3. Delete a partition
4. Exit (update disk configuration and exit)
5. Cancel (exit without updating disk configuration)

Enter Selection:



Interface de système de fichier VFS I

Le Virtual File System permet un héritage au sens objet



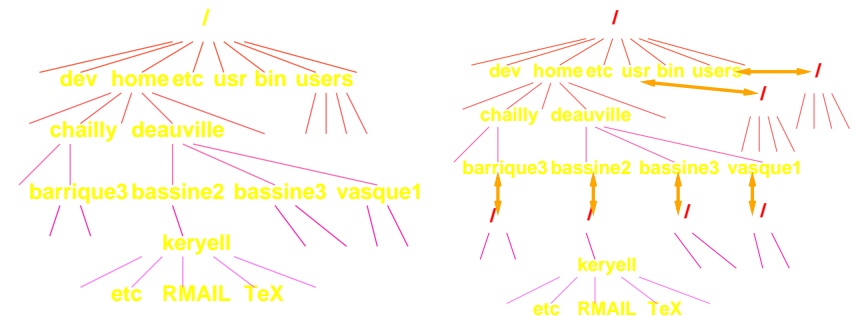
Étape fdisk sur PC II

- 1 seule partition Solaris par disque
- Partition Solaris alignée sur 1 cylindre
- Épargner le *Master Boot Record* sur le cylindre 0
- Subtilité
 - 1 On formate le disque avec `format`
 - 2 On partitionne globalement avec `fdisk` appellable directement depuis `format`
 - 3 On partitionne la partition Solaris générée avec `format` à nouveau...
- Mode non interactif pour extraire des configurations et configurer de manière précise un disque brute style `/dev/rdisk/c0t0d0p0`.
Intérêt pour faire des installations automatiques multi-système d'exploitation



Montage/démontage d'un système de fichiers I

- Pour accéder à un système de fichier : montage pour attacher le système à un répertoire de la hiérarchie préexistante



- / est toujours monté (lancement du noyau) et indémontable

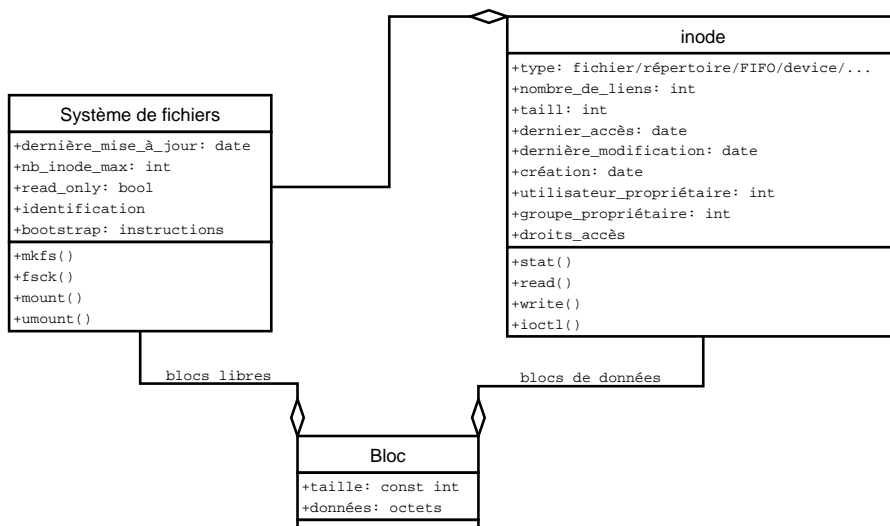


Montage/démontage d'un système de fichiers II

- Montage cache les fichiers préexistants dans le répertoire
- Démontage du système de fichier fait réapparaître d'éventuels fichiers préexistants
- Démontage possible seulement si plus aucun process n'utilise le système de fichier
- Démontage utile pour faire une sauvegarde d'une partition en étant sûr que personne ne la modifie
- Arrêt du système utilise une procédure de démontage



Structure de système de fichier Unix II



Structure de système de fichier Unix I



Structure de répertoire virtuel Unix I

<i>inode</i>	<i>Nom</i>
12345	.
67890	..
2004	toto
4	Schtroumfette_nue.divX



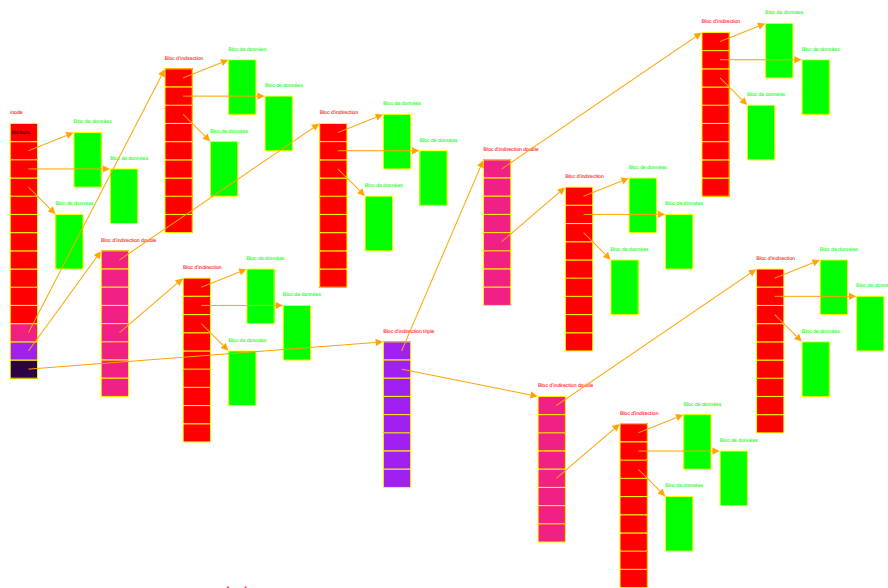
Transformer de la mémoire disque en fichiers I

- Langages de programmation de haut niveau : manipulation de structure de données, d'objets,... Assez loin de la vraie vie de l'occupation mémoire (sauf pour celui qui écrit le compilateur ☺)
- Dans un système de fichier : aplatir une structure de donnée sous forme de flux d'octets ou de blocs de données
- Compromis à trouver
 - ▶ Minimum de surcoût mémoire
 - ▶ Rapidité d'accès en lecture ou écriture
 - ▶ Possibilité de rajouter ou d'enlever des fichiers sans (trop) fragmenter la mémoire
 - ▶ Tolérance aux pannes (redondance)
 - ▶ Optimisation de cas courants ou pas (gros fichiers séquentiels,...)
 - ▶ ...



Transformer de la mémoire disque en fichiers Unix

II



Transformer de la mémoire disque en fichiers Unix I

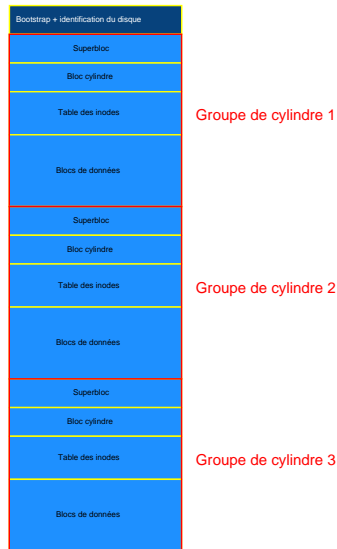
- Unité de base : le bloc
- Peut contenir des données ou des pointeurs vers d'autres blocs



Structure globale du FFS I



Structure globale du FFS II



Systèmes de fichiers sous Solaris II

S5FS : lecture et écriture sur des disques au format System V sur PC

- Système de fichiers de type accès distant

NFS : Network File System pour accéder à des fichiers distants comme s'ils étaient locaux (modulo des différences de performance)

- Système de fichiers virtuels

CacheFS : Cache File System pour stocker localement une copie rapide. CD-ROM, Intranet distant,...

TMPFS : Temporary File System pour stocker en mémoire pour aller très vite. Configuration par défaut de /tmp (accélération des compilations...) qui est doublement volatil

Systemes de fichiers sous Solaris I

- Utilise le Virtual File System : définit une interface permettant de rajouter assez simplement un nouveau type de système de fichiers
- Masque les détails : possibilité de lire, écrire, consulter, etc. quel que soit le type de système de fichiers (local, distant,...)
- Systèmes de fichiers de type disque local

UFS : Unix File System, basé sur le FFS 4.3BSD,
Système de fichier par défaut

HSFS : High Sierra et ISO-9660 (version officielle de la précédente) pour CD-ROM. Lecture seule.
Extension Rock Ridge fournissant la sémantique UFS (sauf les liens durs et... l'écriture !)

PCFS : lecture et écriture sur des disques au format MS-DOS (typiquement disquettes)

Systèmes de fichiers sous Solaris III

LOFS : Loopback System pour faire apparaître à un autre endroit une partie de la hiérarchie (y compris montages NFS)

PROCFS : Process System montre la liste des processus en train de tourner sous forme de répertoires. Utilisé par des outils de debug et d'analyse

5 autres systèmes de fichiers à usage interne sans administration particulière

- LOFI *Loopback file driver* permet de générer un pilote brut à partir d'une image fichier

► Montage de l'image d'un CD-ROM

```
lofiadm -a $CD/sol-8-u5-sparc-v1.iso
mount -F hsfs -o ro /dev/lofi/1 /mnt
```

- ▶ Montage de l'image d'une disquette

- Tâche de l'administrateur ?

Systèmes de fichiers sous Solaris IV

- ▶ Créer de nouveaux systèmes de fichiers
- ▶ Rendre les ressources locales et distantes accessibles aux utilisateurs
- ▶ Connexion et ajout de nouveaux disques
- ▶ Mise en place d'une *excellente* politique de sauvegarde
- ▶ Vérification et correction des fichiers endommagés
Pour les hackers : `fsdb` un débogueur de système de fichiers pour récupérer un accident...
- Commandes générique : `mount`, `umount`, `mkfs`, `fsck`,... acceptent l'option `-F fs-type` et appellent en fait `mount`, `umount_fs-type`, `mkfs_fs-type`, `fsck_fs-type`,... Voir les documentations de ces dernières commandes pour les détails intrinsèques



UFS journalisé II

Extended Fundamental Type (EFT) pour avoir des numéros d'utilisateurs, de groupes et de devices sur 32 bits

Large file systems système de fichiers de 1 To en tout. Pratique si stripping/RAIDs à la DiskSuite

Large files pour fichiers dépassant les 2 Go. Par défaut

Comme l'information des superblocs est critique, elle est répliquée dans tous les groupes de cylindres et décalée de telle manière qu'elle soit répartie en plus sur tous les plateaux



UFS journalisé I

- Unix File System est celui utilisé par défaut sous Solaris. Extension du FFS 4.3BSD. La partition est divisée en groupes de cylindres

Boot Block 8 Ko permettant le démarrage. Existe même si pas partition de boot

Superblock contient les informations sur le système de fichier : taille, statut, label, taille des blocs, date de dernière modification, nom du dernier répertoire de montage, etc.

Contient des drapeaux précisant le fonctionnement

État *clean*, *stable*, *active*, *logging* et *unknown*. Permet de savoir où en est le disque lors d'un accident. *clean*, *stable* ou *logging* ne nécessitent pas de `fsck`



UFS journalisé III

Inodes contiennent toutes les informations sur un fichier sauf son nom : type (normal, répertoire, device,...), mode, propriétaire et groupe, taille, dates,... et tableau de 15 adresses de blocs de données. L'adresse 13 pointe vers un bloc d'adresses, l'adresse 14 pointe vers un bloc d'adresses de blocs d'adresses et l'adresse 15 encore un niveau de plus pour les très gros fichiers

Blocs de données stockent le contenu des fichiers et des répertoires (fichiers de noms et d'adresses d'inodes). Blocs de taille 8 Ko ou 1 Ko (fragments) par défaut

Blocs libres blocs non utilisés (ni inodes, ni données, ni blocs d'adresse) par groupe de cylindre. Garde trace de la fragmentation pour limiter sa propagation



UFS journalisé IV

Pour des raisons de performance, on arrête le remplissage du disque à 90 % de la capacité pour ne pas perdre trop de temps à chercher de la place

- Journalisation
 - ▶ Penser les modifications aux fichiers sous forme de transactions
 - ▶ Stocker les transactions dans un journal
 - ▶ Appliquer (plus tard) les transactions au système de fichier
 - ▶ Après accident, lors du redémarrage les transactions incomplètes sont éliminées mais les transactions complètes sont prises en compte → cohérence maintenue
 - ▶ Plus besoin de faire tourner de longs `fsck` au démarrage
 - ▶ Démarré par option `-o logging` au montage
 - ▶ Le journal est alloué dans la liste de blocs vides
- `mkfs -F ufs` permet de créer un système de fichier en spécifiant tous les paramètres



Vérification d'un système de fichiers I

- Beaucoup de choses sont faites de manière asynchrone pour accélérer un système de fichiers. `fsflush` effectue les écritures en tâche de fond
- `sync` re-synchronise les disques avec ce que pense l'utilisateur (utile si obligé d'arrêter salement une machine)
- Suite à un reboot intempestif ou une panne matérielle, structures de données incohérentes dans le système de fichier : fichiers à moitié effacés, superbloc endommagé,...
- Lancement d'un `fsck` au démarrage si un système de fichier n'est pas marqué *clean* (démonté proprement à l'arrêt), *stable* (non démonté proprement à l'arrêt mais non modifié après le dernier `sync` ou `fsflush` avant l'arrêt) ou *log* (système de fichier journalisé). Parcours de toute la structure du disque : long ! Mais analyse de plusieurs disques en parallèle






UFS journalisé V

- `newfs` crée un système de fichier standard en appelant `mkfs -F ufs` avec des paramètres par défaut

```
deauville-root > newfs -v /dev/rdisk/c0t4d0s5
newfs: construct a new file system /dev/rdisk/c0t4d0s5: (y/n)? y
mkfs -F ufs /dev/rdisk/c0t4d0s5 5926944 214 12 8192 1024 64 2 167 6144 t 0 -1 8 128
/dev/rdisk/c0t4d0s5: 5926944 sectors in 2308 cylinders of 12 tracks, 214 sectors
2894.0MB in 61 cyl groups (38 c/g, 47.65MB/g, 7936 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
32, 97840, 195648, 293456, 391264, 489072, 586880, 684688, 782496, 880304,
978112, 1075920, 1173728, 1271536, 1369344, 1467152, 1561376, 1659184,
1756992, 1854800, 1952608, 2050416, 2148224, 2246032, 2343840, 2441648,
2539456, 2637264, 2735072, 2832880, 2930688, 3028496, 3126304, 3224112,
3321920, 3419728, 3517536, 3615344, 3713152, 3810960, 3908768, 4006576,
4104384, 4202192, 4300000, 4397808, 4495616, 4593424, 4691232, 4789040, 4886848,
4984656, 5082464, 5180272, 5278080, 5375888, 5473696, 5571504, 5669312,
5767120, 5864928, 5962736, 6060544, 6158352, 6256160, 6353968, 6451776, 6549584, 6647392, 6745200, 6843008, 6940816, 7038624, 7136432, 7234240, 7332048, 7429856, 7527664, 7625472, 7723280, 7821088, 7918896, 8016704, 8114512, 8212320, 8310128, 8407936, 8505744, 8603552, 8701360, 8799168, 8896976, 8994784, 9092592, 9190400, 9288208, 9386016, 9483824, 9581632, 9679440, 9777248, 9875056, 9972864, 10070672, 10168480, 10266288, 10364096, 10461904, 10559712, 10657520, 10755328, 10853136, 10950944, 11048752, 11146560, 11244368, 11342176, 11439984, 11537792, 11635600, 11733408, 11831216, 11929024, 12026832, 12124640, 12222448, 12320256, 12418064, 12515872, 12613680, 12711488, 12809296, 12907104, 13004912, 13102720, 13200528, 13298336, 13396144, 13493952, 13591760, 13689568, 13787376, 13885184, 13982992, 14080800, 14178608, 14276416, 14374224, 14472032, 14569840, 14667648, 14765456, 14863264, 14961072, 15058880, 15156688, 15254496, 15352304, 15450112, 15547920, 15645728, 15743536, 15841344, 15939152, 16036960, 16134768, 16232576, 16330384, 16428192, 16526000, 16623808, 16721616, 16819424, 16917232, 17015040, 17112848, 17210656, 17308464, 17406272, 17504080, 17601888, 17699696, 17797504, 17895312, 17993120, 18090928, 18188736, 18286544, 18384352, 18482160, 18579968, 18677776, 18775584, 18873392, 18971200, 19069008, 19166816, 19264624, 19362432, 19460240, 19558048, 19655856, 19753664, 19851472, 19949280, 20047088, 20144896, 20242704, 20340512, 20438320, 20536128, 20633936, 20731744, 20829552, 20927360, 21025168, 21122976, 21220784, 21318592, 21416400, 21514208, 21612016, 21709824, 21807632, 21905440, 22003248, 22101056, 22198864, 22296672, 22394480, 22492288, 22590096, 22687904, 22785712, 22883520, 22981328, 23079136, 23176944, 23274752, 23372560, 23470368, 23568176, 23665984, 23763792, 23861600, 23959408, 24057216, 24155024, 24252832, 24350640, 24448448, 24546256, 24644064, 24741872, 24839680, 24937488, 25035296, 25133104, 25230912, 25328720, 25426528, 25524336, 25622144, 25719952, 25817760, 25915568, 26013376, 26111184, 26208992, 26306800, 26404608, 26502416, 26600224, 26698032, 26795840, 26893648, 26991456, 27089264, 27187072, 27284880, 27382688, 27480496, 27578304, 27676112, 27773920, 27871728, 27969536, 28067344, 28165152, 28262960, 28360768, 28458576, 28556384, 28654192, 28752000, 28849808, 28947616, 29045424, 29143232, 29241040, 29338848, 29436656, 29534464, 29632272, 29730080, 29827888, 29925696, 30023504, 30121312, 30219120, 30316928, 30414736, 30512544, 30610352, 30708160, 30805968, 30903776, 31001584, 31099392, 31197200, 31295008, 31392816, 31490624, 31588432, 31686240, 31784048, 31881856, 31979664, 32077472, 32175280, 32273088, 32370896, 32468704, 32566512, 32664320, 32762128, 32859936, 32957744, 33055552, 33153360, 33251168, 33348976, 33446784, 33544592, 33642400, 33740208, 33838016, 33935824, 34033632, 34131440, 34229248, 34327056, 34424864, 34522672, 34620480, 34718288, 34816096, 34913904, 35011712, 35109520, 35207328, 35305136, 35402944, 35500752, 35598560, 35696368, 35794176, 35891984, 35989792, 36087600, 36185408, 36283216, 36381024, 36478832, 36576640, 36674448, 36772256, 36870064, 36967872, 37065680, 37163488, 37261296, 37359104, 37456912, 37554720, 37652528, 37750336, 37848144, 37945952, 38043760, 38141568, 38239376, 38337184, 38434992, 38532800, 38630608, 38728416, 38826224, 38924032, 39021840, 39119648, 39217456, 39315264, 39413072, 39510880, 39608688, 39706496, 39804304, 39902112, 40000000, 40097808, 40195616, 40293424, 40391232, 40489040, 40586848, 40684656, 40782464, 40880272, 40978080, 41075888, 41173696, 41271504, 41369312, 41467120, 41564928, 41662736, 41760544, 41858352, 41956160, 42053968, 42151776, 42249584, 42347392, 42445200, 42543008, 42640816, 42738624, 42836432, 42934240, 43032048, 43129856, 43227664, 43325472, 43423280, 43521088, 43618896, 43716704, 43814512, 43912320, 44010128, 44107936, 44205744, 44303552, 44401360, 44499168, 44596976, 44694784, 44792592, 44890400, 44988208, 45086016, 45183824, 45281632, 45379440, 45477248, 45575056, 45672864, 45770672, 45868480, 45966288, 46064096, 46161904, 46259712, 46357520, 46455328, 46553136, 46650944, 46748752, 46846560, 46944368, 47042176, 47139984, 47237792, 47335600, 47433408, 47531216, 47629024, 47726832, 47824640, 47922448, 48020256, 48118064, 48215872, 48313680, 48411488, 48509296, 48607104, 48704912, 48802720, 48900528, 49000000, 49097808, 49195616, 49293424, 49391232, 49489040, 49586848, 49684656, 49782464, 49880272, 49978080, 50075888, 50173696, 50271504, 50369312, 50467120, 50564928, 50662736, 50760544, 50858352, 50956160, 51053968, 51151776, 51249584, 51347392, 51445200, 51543008, 51640816, 51738624, 51836432, 51934240, 52032048, 52129856, 52227664, 52325472, 52423280, 52521088, 52618896, 52716704, 52814512, 52912320, 53010128, 53107936, 53205744, 53303552, 53401360, 53499168, 53596976, 53694784, 53792592, 53890400, 53988208, 54086016, 54183824, 54281632, 54379440, 54477248, 54575056, 54672864, 54770672, 54868480, 54966288, 55064096, 55161904, 55259712, 55357520, 55455328, 55553136, 55650944, 55748752, 55846560, 55944368, 56042176, 56139984, 56237792, 56335600, 56433408, 56531216, 56629024, 56726832, 56824640, 56922448, 57020256, 57118064, 57215872, 57313680, 57411488, 57509296, 57607104, 57704912, 57802720, 57900528, 58000000, 58097808, 58195616, 58293424, 58391232, 58489040, 58586848, 58684656, 58782464, 58880272, 58978080, 59075888, 59173696, 59271504, 59369312, 59467120, 59564928, 59662736, 59760544, 59858352, 59956160, 60053968, 60151776, 60249584, 60347392, 60445200, 60543008, 60640816, 60738624, 60836432, 60934240, 61032048, 61129856, 61227664, 61325472, 61423280, 61521088, 61618896, 61716704, 61814512, 61912320, 62010128, 62107936, 62205744, 62303552, 62401360, 62499168, 62596976, 62694784, 62792592, 62890400, 62988208, 63086016, 63183824, 63281632, 63379440, 63477248, 63575056, 63672864, 63770672, 63868480, 63966288, 64064096, 64161904, 64259712, 64357520, 64455328, 64553136, 64650944, 64748752, 64846560, 64944368, 65042176, 65139984, 65237792, 65335600, 65433408, 65531216, 65629024, 65726832, 65824640, 65922448, 66020256, 66118064, 66215872, 66313680, 66411488, 66509296, 66607104, 66704912, 66802720, 66900528, 67000000, 67097808, 67195616, 67293424, 67391232, 67489040, 67586848, 67684656, 67782464, 67880272, 67978080, 68075888, 68173696, 68271504, 68369312, 68467120, 68564928, 68662736, 68760544, 68858352, 68956160, 69053968, 69151776, 69249584, 69347392, 69445200, 69543008, 69640816, 69738624, 69836432, 69934240, 70032048, 70129856, 70227664, 70325472, 70423280, 70521088, 70618896, 70716704, 70814512, 70912320, 71010128, 71107936, 71205744, 71303552, 71401360, 71499168, 71596976, 71694784, 71792592, 71890400, 71988208, 72086016, 72183824, 72281632, 72379440, 72477248, 72575056, 72672864, 72770672, 72868480, 72966288, 73064096, 73161904, 73259712, 73357520, 73455328, 73553136, 73650944, 73748752, 73846560, 73944368, 74042176, 74139984, 74237792, 74335600, 74433408, 74531216, 74629024, 74726832, 74824640, 74922448, 75020256, 75118064, 75215872, 75313680, 75411488, 75509296, 75607104, 75704912, 75802720, 75900528, 76000000, 76097808, 76195616, 76293424, 76391232, 76489040, 76586848, 76684656, 76782464, 76880272, 76978080, 77075888, 77173696, 77271504, 77369312, 77467120, 77564928, 77662736, 77760544, 77858352, 77956160, 78053968, 78151776, 78249584, 78347392, 78445200, 78543008, 78640816, 78738624, 78836432, 78934240, 79032048, 79129856, 79227664, 79325472, 79423280, 79521088, 79618896, 79716704, 79814512, 79912320, 80010128, 80107936, 80205744, 80303552, 80401360, 80499168, 80596976, 80694784, 80792592, 80890400, 80988208, 81086016, 81183824, 81281632, 81379440, 81477248, 81575056, 81672864, 81770672, 81868480, 81966288, 82064096, 82161904, 82259712, 82357520, 82455328, 82553136, 82650944, 82748752, 82846560, 82944368, 83042176, 83139984, 83237792, 83335600, 83433408, 83531216, 83629024, 83726832, 83824640, 83922448, 84020256, 84118064, 84215872, 84313680, 84411488, 84509296, 84607104, 84704912, 84802720, 84900528, 85000000, 85097808, 85195616, 85293424, 85391232, 85489040, 85586848, 85684656, 85782464, 85880272, 85978080, 86075888, 86173696, 86271504, 86369312, 86467120, 86564928, 86662736, 86760544, 86858352, 86956160, 87053968, 87151776, 87249584, 87347392, 87445200, 87543008, 87640816, 87738624, 87836432, 87934240, 88032048, 88129856, 88227664, 88325472, 88423280, 88521088, 88618896, 88716704, 88814512, 88912320, 89010128, 89107936, 89205744, 89303552, 89401360, 89499168, 89596976, 89694784, 89792592, 89890400, 89988208, 90086016, 90183824, 90281632, 90379440, 90477248, 90575056, 90672864, 90770672, 90868480, 90966288, 91064096, 91161904, 91259712, 91357520, 91455328, 91553136, 91650944, 91748752, 91846560, 91944368, 92042176, 92139984, 92237792, 92335600, 92433408, 92531216, 92629024, 92726832, 92824640, 92922448, 93020256, 93118064, 93215872, 93313680, 93411488, 93509296, 93607104, 93704912, 93802720, 93900528, 94000000, 94097808, 94195616, 94293424, 94391232, 94489040, 94586848, 94684656, 94782464, 94880272, 94978080, 95075888, 95173696, 95271504, 95369312, 95467120, 95564928, 95662736, 95760544, 95858352, 95956160, 96053968, 96151776, 96249584, 96347392, 96445200, 96543008, 96640816, 96738624, 96836432, 96934240, 97032048, 97129856, 97227664, 97325472, 97423280, 97521088, 97618896, 97716704, 97814512, 97912320, 98010128, 98107936, 98205744, 98303552, 98401360, 98499168, 98596976, 98694784, 98792592, 98890400, 98988208, 99086016, 99183824, 99281632, 99379440, 99477248, 99575056, 99672864, 99770672, 99868480, 99966288, 100064096, 100161904, 100259712, 100357520, 100455328, 100553136, 100650944, 100748752, 100846560, 100944368, 101042176, 101139984, 101237792, 101335600, 101433408, 101531216, 101629024, 101726832, 101824640, 101922448, 102020256, 102118064, 102215872, 102313680, 102411488, 102509296, 102607104, 102704912, 102802720, 102900528, 103000000, 103097808, 103195616, 103293424, 103391232, 103489040, 103586848, 103684656, 103782464, 103880272, 103978080, 104075888, 104173696, 104271504, 104369312, 104467120, 104564928, 104662736, 104760544, 104858352, 104956160, 105053968, 105151776, 105249584, 105347392, 105445200, 105543008, 
```

Vérification d'un système de fichiers III

-  `fsck` n'a aucun moyen de réparer le *contenu* des fichiers...
-  Ne pas monter a priori de disque local via `/etc/vfstab` sans préciser que le `fsck` doit être fait au démarrage. Un - dans `/etc/vfstab` indique pas de `fsck`, 1 pour `fsck` séquentiel dans l'ordre du `/etc/vfstab` et plus que 1 pour dire que les `fsck` sont ensuite faits en parallèle sur les disques
-  `fsck` ne remplace pas les RAID et encore moins les sauvegardes ! Évite juste les restaurations en cas de problèmes mineurs
- Pour hackers et pompiers le débogueur de système de fichiers : `fsdb`, `fsdb_ufs`,...

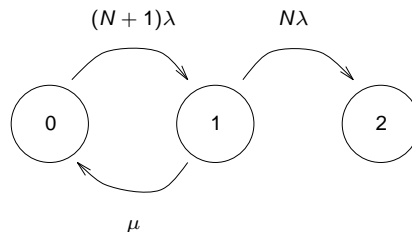


Redundant Array of Inexpensive Disks I

Mettre plus de disques pour compenser les pannes

Chaîne de MARKOV modélisant un RAID où 0 ou 1 disque peut être en panne sans perte de données :

- λ constante de panne d'un disque ($1/MTBF_1$)
- μ constante de réparation.
Supposition : $\lambda \ll \mu$



Durée avant perte de données :

$$MTTDL \approx \frac{\mu}{N \times (N+1)\lambda^2}$$



RAID I

Augmenter débit et capacité mais diminuer coût \leadsto paralléliser les disques !

Problème : *Mean Time Between Failure* de plusieurs disques.

Endurance de N disques pendant un temps t :

$$R_N(t) = (R_1(t))^N$$

Est-ce bien utile ?

$$\lim_{N \rightarrow \infty} R_N(t) = 0$$

Si $MTBF_1 = 30000$ heures, alors $MTBF_{1000} = 30$ heures...



Types de RAIDs I

RAID-0 : pas de redondance ! *stripping* sur plusieurs disques

RAID-1 : tout est doublé

- cher : moitié du disque utile
- rapide

RAID-2 : rajouter C disques par D disques pour code de détection et correction d'erreur, $C \geq \log_2(D + C + 1)$

RAID-3,4,5 : si un contrôleur sait quand le disque est en panne (CRC sur disque, etc.) \leadsto seule la parité suffit.

$$p_i = a_i \oplus b_i \oplus c_i \quad (1)$$

$$b_i = a_i \oplus c_i \oplus p_i \quad (2)$$



Types de RAIDs II

Problème : tout accès en écriture nécessite un accès à la parité \Rightarrow goulet d'étranglement !

RAID-5 : répartition de la parité cycliquement sur les disques pour paralléliser les accès :


	\bar{D}_1	\bar{D}_2	\bar{D}_3	\bar{D}_4
B_1	a_1	b_1	c_1	p_1
B_2	p_2	a_2	b_2	c_2
B_3	c_3	p_3	a_3	b_3
B_4	b_4	c_4	p_4	a_4
B_5	a_5	b_5	c_5	p_5

∃ autres combinaisons de RAID

Concepts de ZFS I

- Copie sur écriture pour avoir toujours vieilles données valides et journalisation
- Permet de rajouter facilement modèle transactionnel
- Codes de vérification pour détecter corruption
- Réplication avec RAID-Z, évite corruption RAID-5 (si panne de courant entre écriture donnée et parité) car copie sur écriture. Adaptation taille des bandes en fonction débit de chaque disque
- Inspection des fichiers et réparation en tâche de fond
- Optimise parallélisme et ordonnancement des ressources, E/S dans le désordre en respectant graphe de dépendance, tableau noir (*scoreboard*)
- Instantanés (*snapshots*) (lecture seule) et clones (lecture-écriture) en temps constant. Pratique pour sauvegardes

ZFS de OpenSolaris Sun I

- Essayer de dépasser limitations systèmes de fichiers classiques
 - ▶ Intégrité des données
 - ▶ Extensibilité
 - ▶ Sémantique transactionnelle
 - ▶ Administration simple
 - ▶ Disparition de la limite de disques ou partitions
 - ▶ Gère ordre des octets
-  Zetabyte File System

<http://opensolaris.org/os/community/zfs/docs>

Concepts de ZFS II

- Possibilité de faire des différences d'instantanés (sauvegardes incrémentales, réplication à distance)
- Compression des données à la volée permet de réduire E/S d'un facteur 2–3 et peut être gagnant outre gain en capacité
- Permet d'exporter des pseudo-blocs de disque : swap, bases de données, systèmes de fichiers... pour bénéficier avantages ZFS
- Identifiants sur 128 bits
 - ▶ 2^{64} caractères par fichier
 - ▶ 2^{48} fichiers par répertoires accédés par table de hachage
 - ▶ 2^{64} instantanés
 - ▶ 2^{64} disques

Déploiement de ZFS I

- (Open)Solaris
- MacOS X
- BSD
- Linux : problème de licence GPL pour ajouter quelque chose dans le noyau dont ZFS en CDDL. Possible de tourner en mode utilisateur avec FUSE mais ↘ ↘ performances ☹



Contrôleur ou pilote de périphérique I

- Besoin de contrôler les périphériques de bas niveau par le système d'exploitation, voire par l'utilisateur
- Pendant logiciel au périphérique matériel : *device driver* ou contrôleur/pilote de périphérique
- Essaye de réutiliser un maximum de code
- Exemple de Solaris (SVR4) : *Device Driver Interface Driver-Kernel Interface* (DDI/DKI)
 - ▶ Définit les méthodes que doit implémenter un contrôleur de périphérique (`man -s 9e intro`)
 - ▶ Méthodes du noyau utilisables par le contrôleur de périphérique : gestion des DMA, des interruptions, de la mémoire, des messages d'erreur... (`man -s 9f intro`)
- Augmente la portabilité du système d'exploitation

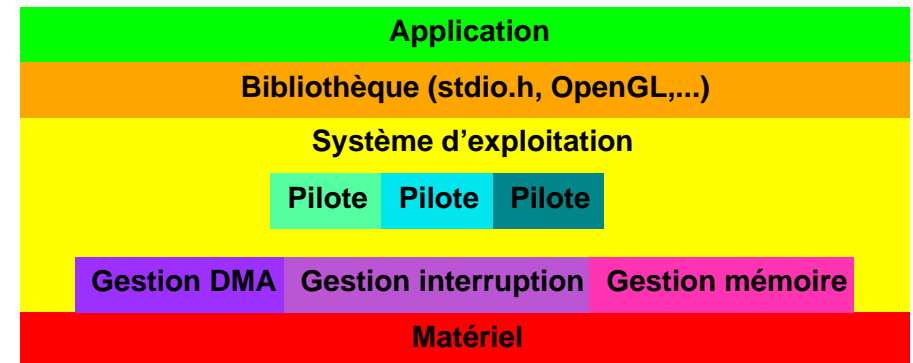


Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Contrôleur ou pilote de périphérique II



Types de pilotes de périphériques I

- Certains périphériques ne peuvent envoyer ou recevoir des données que par une taille fixe de données sinon erreur
 - ▶ Clavier, imprimante : *caractère* par *caractère* (cela se passe en général bien pour en lire plusieurs)
 - ▶ Réseau : des paquets Ethernet entier par exemple en mode *brut*
 - ▶ Disque dur, disquette : par bloc ou secteur (style 512 octets) en mode brut

Ce sont des périphériques de type *caractère*. Nom malheureux car regroupe aussi bien des pilotes gérant effectivement des caractère ou des paquets de données (mode brut ou *raw*)

- Besoin parfois d'avoir un périphérique utilisable avec n'importe quel nombre d'octets
 - ▶ Faire des `read`, `write`, `printf` sans soucis
 - ▶ Le système offre la possibilité de rajouter une couche de tampon pour cacher cette taille fixe



Quelques méthodes d'un pilote DDI/DKI I

Minimum à écrire pour avoir un pilote : `open`, `read` ou `write` et `close`

`chpoll` : poll entry point for a non-STREAMS character driver
`close` : relinquish access to a device
`ioctl` : control a character device (sert à tout et n'importe quoi)
`mmap` : check virtual mapping for memory mapped device
`open` : gain access to a device
`print` : display a driver message on system console
 `put` : receive messages from the preceding queue (STREAMS)
`read` : read data from a device
 `srv` : service queued messages



Types de pilotes de périphériques II

- ▶ Exemple : si écriture de 10 octets sur disque, le système lit 2 blocs du disque concernés par ces octets, les modifie et les réécrit

Ce sont des périphériques de type bloc (nom malheureux encore...)

- Certains pilotes sont mieux gérés avec une architecture en couche :
 - ▶ Protocoles réseau (couches)
 - ▶ Terminaux (caractères de contrôles, gestion des modems ou liaison série,...)
 - ▶ FIFO

Pilote de type STREAM



Quelques méthodes d'un pilote DDI/DKI II

`strategy` : perform block I/O. Appelé par le noyau pour faire des accès par bloc et remplir vider ses tampons pleins pour le pilote en mode bloc. Aussi utilisé par `read` et `write` si périphérique orienté bloc mais utilisé en mode brut (*raw*) donc caractère

`write` : write data to a device

Il y a aussi des choses purement Solaris (gestion tolérance aux pannes, hibernation de la machine,...)



Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 **Systèmes de fichiers distants**
 - **NFS**
- 8 Conclusion

Pouvoir délocaliser les fichiers I

- Dès début années 1970 développement des réseaux et protocoles de transfert de fichiers : UUCP, FTP,...
- Pas très élégant car pas de vision globale du système
- Émergence de systèmes de fichiers permettant des accès transparents aux fichiers distants dans les années 1980
 - ▶ *Network File System* (NFS) de Sun Microsystems
 - ▶ *Remote File Sharing* (RFS) d'AT&T
 - ▶ *Andrew File System* (AFS) de Carnegie-Mellon University qui a évolué en *Distributed File System* d'OSF/DCE

Propriétés importantes I

Un système de fichiers distribué peut avoir :

- Transparence de l'accès distant
- Transparence de la localisation
- Nom indépendant de la localisation
- Mobilité possible de l'utilisateur
- Tolérance aux pannes
- Extensibilité
- Mobilité des fichiers

Considérations de conception I

- Espace de nommage : uniforme ou pas
 - ▶ Espace de nommage uniforme : la localisation n'apparaît pas
 - ▶ Un client peut greffer (« monter ») une arborescence distante dans sa propre hiérarchie avec un nom, voire un usage (FTP,...), dépendant de la machine distante
- Fonctionnement avec ou sans état du serveur
Certaines requêtes possèdent un état qu'il faut stocker quelque part : *open*, *lseek*,...
 - ▶ Serveur sans état persistant : chaque client doit envoyer des requêtes auto-suffisantes (position dans le fichier où on doit écrire,...). Serveur plus simple
 - ▶ Serveur avec état persistant : conserve des informations sur les clients, moins de trafic réseau mais plus complexe (récupération de l'état en cas de plantage du serveur,...)

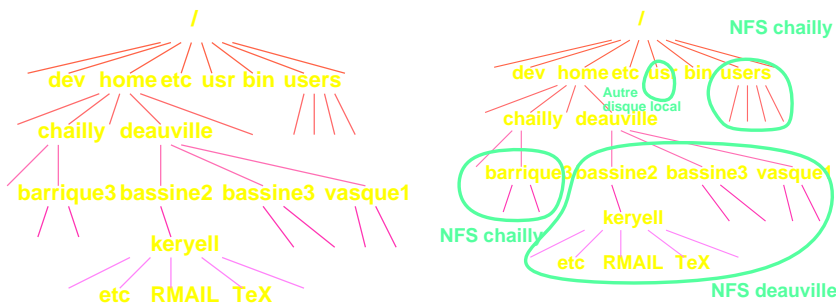
Considérations de conception II

- Sémantique du partage
 - ▶ Sémantique Unix : toute modification par un client doit être visible immédiatement par un autre client. Contrainte forte ~ \ performances
 - ▶ Sémantique de session : modifications propagées aux autres clients qu'au niveau du `close` ou à intervalle régulier,...
- Méthodes d'accès à distance : pas toujours limitées à un simple modèle client-serveur dans le cas d'un serveur à état à cause du mécanisme de récupération de panne du serveur



Network File System (NFS) I

- Introduit en 1985 avec la version 2 par Sun Microsystems
- Accès transparent à des systèmes de fichiers distants



- Standard *de facto*
- Modèle client-serveur



Le plan

- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire
- 5 Virtualisation
- 6 Les entrées-sorties
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Network File System (NFS) II

- ▶ Serveur de fichiers exporte un ensemble de fichiers
- ▶ Clients de NFS accèdent à ces fichiers
- ▶ Possible avoir machines à la fois clients et serveurs
- ▶ Communications par des *Remote Procedure Call*



Buts de conceptions I

- Non restreint à Unix
- Protocole indépendant du matériel
- Mécanisme simple de récupération après plantage du client ou serveur
- Accès transparents pour les applications : pas de noms, bibliothèque ou compilation spécifiques
- Sémantique Unix maintenue dans le cas de clients Unix
- Performances NFS comparables à celles des disques locaux
- Réalisation indépendante de la couche transport



NFS : sans état I

- Pas d'état concernant le client dans le serveur
- Chaque requête auto-suffisante et indépendante des autres
- Pas de mécanisme d'open ou close
- Les READ et WRITE doivent contenir leur propre *offset*
- Plantage du client : besoin de remonter le système de fichiers
- Plantage du serveur : client répète ses requêtes jusqu'à une réponse. Pas de différence entre serveur lent ou serveur qui redémarre...
- Sans état
 - ▶ Protocole séparé pour le verrouillage (NLM)
 - ▶ Toute modification des fichiers doivent être écrits sur disques du serveur avant de répondre au client car en cas de plantage client ne serait jamais au courant de la perte d'information... ~ Lent, accélérateurs matériels, NFS v.3



Composants de NFS I

- Protocole RPC : interaction entre client et serveur, invocation de fonctions distantes avec passage d'arguments locaux et récupération des résultats
- *External Data Representation* (XDR) : encodage des informations indépendant de l'architecture matérielle
- Programme du serveur NFS : gère requêtes des clients
- Programme du client NFS : transforme appels systèmes aux fichiers distants en appels RPC aux serveurs NFS
- Protocole de montage : gère montage et démontage des systèmes de fichiers NFS
- Plusieurs processus « démons » : `nfsd` gère les requêtes NFS et `mountd` gère montage sur serveur, `biod` gère sur le client les entrées-sorties asynchrones à des blocs de fichiers
- Rajout mécanisme de verrou sur fichiers via NFS avec *Network File System* et *Network Status Monitor* (`lockd` et `statd`)



Le plan



Conclusion I

- Informatique et donc systèmes d'exploitation : partout !
- Ingénieur → comprendre comment cela marche
- ∃ quelque chose derrière les interfaces graphiques
- Savoir bien utiliser le système
- Donne plein d'exemples de bonnes idées pour la gestion de ressources ☺
- Continuer l'apprentissage du C



Entrée et sortie du noyau	94	Le plan	140
Contexte logiciel d'un processus	95	Entrées-sorties physiques (bas niveau)	141
• Processus lourds & légers, threads		Déroutement d'une entrée-sortie physique	142
Le plan	96	• Gestion du temps qui passe	
Vie et mort d'un processus	97	Le plan	143
Prémption dans le noyau	98	Temps	144
Synchronisation & concurrence	99	Génération d'événements	146
4 états d'exécution possible	100	L'heure et sa distribution	147
Vie et mort d'un processus version Unix	101	• Ordonnancement	
Distinguer le père du fils en Unix ?	103	Le plan	149
Contexte logiciel processus lourd UNIX	105	Ordonnancement	150
Contexte logiciel UNIX en 2 parties	107	Priorité	151
Espace virtuel des processus UNIX	108	Types de multitâche	152
Communications inter-processus sous UNIX	110	Quelques politiques d'ordonnancement	153
Processus légers	112	Processus dirigé par le calcul ou les E/S	154
Vie et mort d'un processus version thread	113	Temps partagé – tourniquet	155
Threads POSIX	114	Blocage dans le tourniquet	156
Threads utilisateurs dans un processus	115	Politique par priorité	157
2 niveaux d'ordonnancement	116	Politique à priorités dans Unix	158
Avec ordonnanceur de threads noyau	117	Toutes les priorités dans Linux	160
Les processus légers dans Linux	118	Exemple de processus Unix avec top	162
Quid entre processus lourd et léger ?	119	Choix du quantum de temps dans Linux	164
Processus et threads : combien ça coûte ?	120	Ordonnanceur Linux 2.6	166
Des processus sans système d'exploitation ?	121	File d'exécution (runqueues)	167
• Plus de détail des tâches dans Linux		Politique à priorité	169
Le plan	123	Recalculer les quanta de temps	171
Différents usages de clone()	124	La fonction d'ordonnancement schedule()	174
Tâches noyau	128	Calcul des priorités dynamiques et quanta	175
Exemple de liste de processus	129	Sommeil & réveil	179
Représentation des tâches en interne	130	Prémption	182
Accès aux tâches	133	Équilibrage de charge	183
Graphe des états d'un processus	135	Affinité tâche-processus	184
Un processus Unix dans tous ses états	136	Unix : 2 ordonnancement	185
États internes dans Linux	137	Ordonnancement et va et vient	186
• Entrées-sorties		Politiques temps réel	187
		Petite conclusion sur ordonnancement Linux	188

Copyright (c)	2	Le plan	57
Le cours	3	• Introduction	
Problématique	4	Le plan	58
Le plan	5	Concurrence et parallélisme	59
Introduction	6	Tâches	60
Machine Virtuelle Étendue	8	• Hypothèses sur architecture matérielle	
Abstraction de fichier	9	Le plan	61
Exemples de fichiers et fichiers spéciaux	10	Schéma d'un monoprocesseur	62
Transparence et opacité	12	Mécanisme d'interruptions	64
Nirvana des systèmes	15	Mécanisme d'interruptions — détail	66
Un ordinateur dans une perspective logicielle	16	Schéma d'un multiprocesseur	67
DCE — <i>Distributed Computing Environment</i>	17	Interruptions & multiprocesseur	68
Évolution logicielle	18	Classes d'architectures matérielles	69
Pourquoi ce cours en 1A/2A... puis en 3A ?	20	• Notion de noyau	
Bibliographie	21	Le plan	70
Bibliographie Linux	23	Notion de noyau	71
1 Historique		Noyau Unix	72
Le plan	26	Notion de micro-noyau	73
Historique	27	Mises en œuvres de la concurrence	75
1945–1955 : ordinateur séquentiel	29	Fonctions d'un micro-noyau	76
1955–1965 : Jusqu'à l'OS	31	Noyau et interface de programmation	77
1965–1985 : Multiprogrammation, temps partagés	33	Structure Unix traditionnel (ab initio)	78
Unix Story	35	Séparer politique et mécanisme	79
1985– : Micro-ordinateurs	39	Structure Unix moderne	80
2 Concepts de base		Centralisation contre distribution	81
Le plan	44	Systèmes distribués	82
Concepts de base	45	Quand donner contrôle au noyau ?	83
Multiprogrammation	46	Informations gérées par le noyau	84
Code translatable	47	Graphe des états d'un processus	85
Base et déplacement	49	Graphe des états d'un processus sous Unix	86
Temps partagé – tourniquet	51	Atomicité des actions noyau	87
Blocage dans le tourniquet	52	Contexte matériel	88
Pagination	53	Sauvegarde et restitution de contexte	89
Segmentation	55	Appel système	92
3 Concurrence & Parallélisme		Appel moniteur/BIOS	93
Département Informatique, TÉLÉCOM Bretagne			
Systèmes d'exploitation et supports architecturaux – 3A SLR F2B303A			
Table des matières			
Des ennuis : inversion de priorité	189	6 Les entrées-sorties	
Solutions possibles	191	Le plan	235
Mars Pathfinder Mission on July 4th, 1997	192	Hiérarchie de classes de fichiers Unix simplifiée	236
Tâche martienne de priorité forte	193	Descripteur de fichier	238
Tâche martienne de priorité faible	194	Sémantique des fichiers Unix	240
Tâche martienne de priorité moyenne	195	Quelques méthodes associées à des fichiers	242
Bug martien	196	Passer un descripteur à ton voisin	245
Debug martien	197	Les fonctions d'E/S du C(++)	246
4 Gestion mémoire		• Disques	
Le plan	199	(Disques magnétiques)	247
Gestion des mémoires	200	Unix FFS optimisé pour les disques	249
Hiérarchie mémoire	201	Partitionnement des disques	250
Hiérarchie mémoire version réseau	202	• Formatage	
Dépasser des limitations d'adressage	203	Utilitaire format	253
Dépasser la mémoire physique	206	format à l'œuvre	254
Intérêts de la mémoire virtuelle	207	Étape fdisk sur PC	257
Grande mémoire virtuelle	209	Interface de système de fichier VFS	259
Traduction des adresses des pages	212	Montage/démontage d'un système de fichiers	260
Tailles de pages	213	Structure de système de fichier Unix	262
Pagination à 3 niveaux sur SPARC	214	Structure de répertoire virtuel Unix	264
Défaut de page	215	Transformer de la mémoire disque en fichiers	265
Vision objet de la mémoire virtuelle	216	Transformer de la mémoire disque en fichiers Unix	266
Mémoire virtuelle et localité	217	Structure globale du FFS	268
Limiter les défauts de pages	218	Systèmes de fichiers sous Solaris	270
Accélérer la traduction d'adresse	219	UFS journalisé	274
Contenu d'un TLB	220	Vérification d'un système de fichiers	279
Choix des pages à enlever du TLB	221	• RAID	
Donner de l'espace aux processus	222	RAID	282
5 Virtualisation		Redundant Array of Inexpensive Disks	283
Le plan	223	Types de RAID	284
Architectures virtuelles	224	• ZFS	
Machines virtuelles matérielles	225	ZFS de OpenSolaris Sun	286
Intérêts de la virtualisation	226	Concepts de ZFS	287
Machines virtuelles logicielles	228	Déploiement de ZFS	289
Machines virtuelles - fonctionnement	230	• Pilote de périphérique	
Paravirtualisation	233		

Le plan	290	● NFS	
Contrôleur ou pilote de périphérique	291	Le plan	302
Types de pilotes de périphériques	293	Network File System (NFS)	303
Quelques méthodes d'un pilote DDI/DKI	295	Buts de conceptions	305
		Composants de NFS	306
		NFS : sans état	307
7 Systèmes de fichiers distants			
Le plan	297		
Pouvoir délocaliser les fichiers	298	8 Conclusion	
Propriétés importantes	299	Le plan	308
Considérations de conception	300	Conclusion	309

