

Copyright (c)

Systèmes d'exploitation et supports architecturaux – 3A SLR F2B205A

Ronan KERYELL

HPC Project
TÉLÉCOM Bretagne, Département Informatique, HPCAS

29 octobre 2008
Version 1.20

- 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,...)



2 / 298

• Introduction

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 ! ☺



• Introduction

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 pérennité des développements

Capitaliser l'expérience



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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
 - Systèmes de fichiers distants
 - NFS
- 7 Conclusion
- 8 Conclusion

(I)

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...)
 - Latence, temps de réponse : mode interactif



Introduction

(II)

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



Machine Virtuelle Étendue

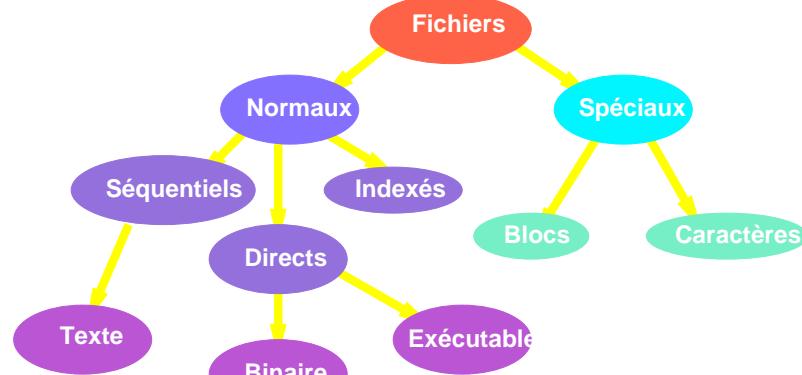
(I)

- 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 quoidans Unix...



Abstraction de fichier

Dans un système imaginaire...



Exemples de fichiers et fichiers spéciaux

- 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 | xwud : entre processus
 - ▶ Pipes nommés



Exemples de fichiers et fichiers spéciaux

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



Transparence et opacité

« 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 migrateurs
 - ▶ Ressources distribuées
- Réplication (tolérance aux pannes et performances)
 - ▶ Multiplication des serveurs



Transparence et opacité

(II)

- ▶ 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 ?



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 !



Transparence et opacité

(III)

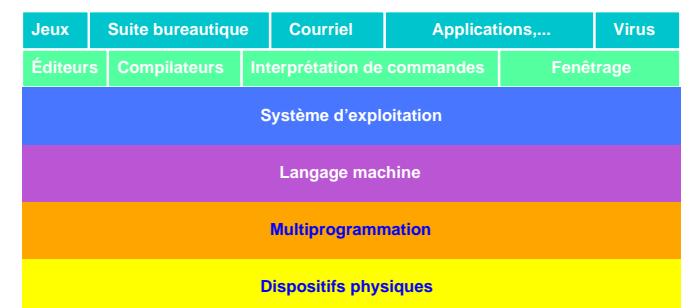
Exemple de concept : partage de données dans une variable globale (locale ou distante)
Le système va masquer la réalité :

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



Un ordinateur dans une perspective logicielle

(I)

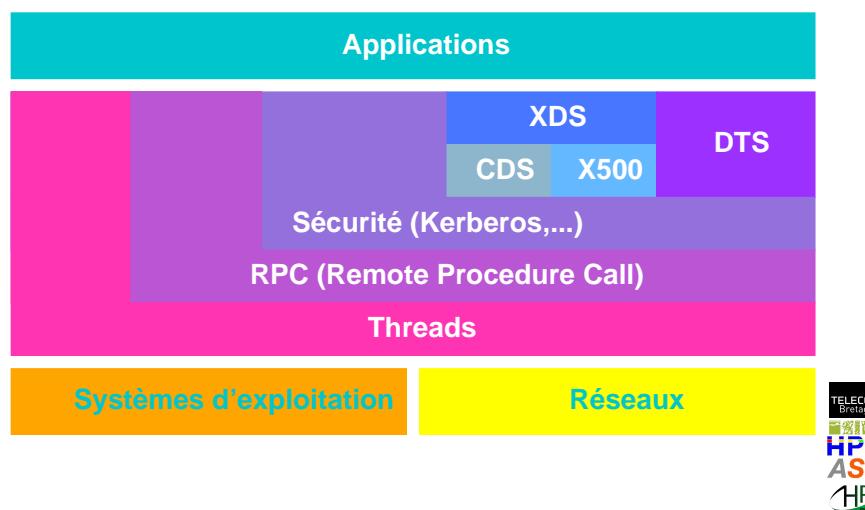


⚠ Le grand public ne voit que le haut et le fenêtrage...



DCE — Distributed Computing Environment

(I)



Évolution logicielle

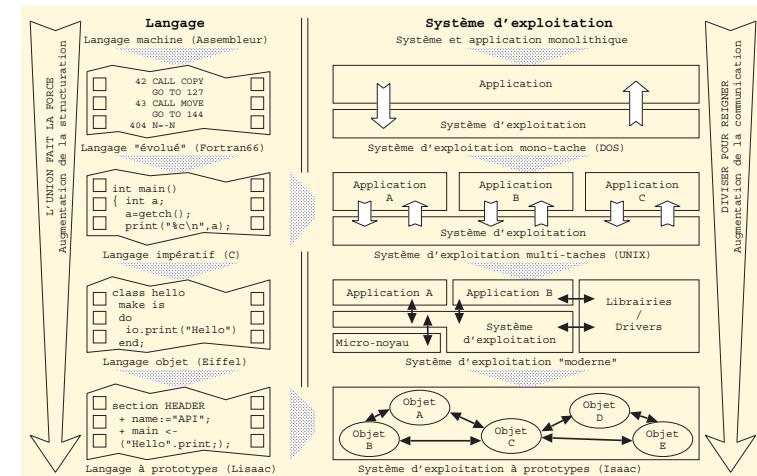
(II)

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



Évolution logicielle

(I)



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/SystemesAS/HPC>
 - <http://deptinfo.cnam.fr/Enseignement/CycleA/AMSI>



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



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 dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire

5 Virtualisation

- 6 Les entrées-sorties
 - Systèmes de fichiers
 - Disques
 - Formatage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



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
 - ▶ Réseaux (Internet)



Historique

(II)

- ▶ Systèmes distribués

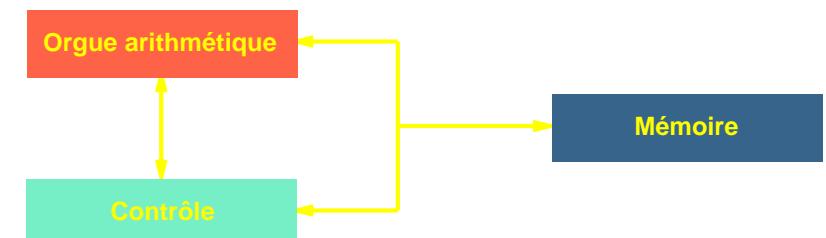
<http://www.computer.org/50/history>



1945–1955 : ordinateur séquentiel

(I)

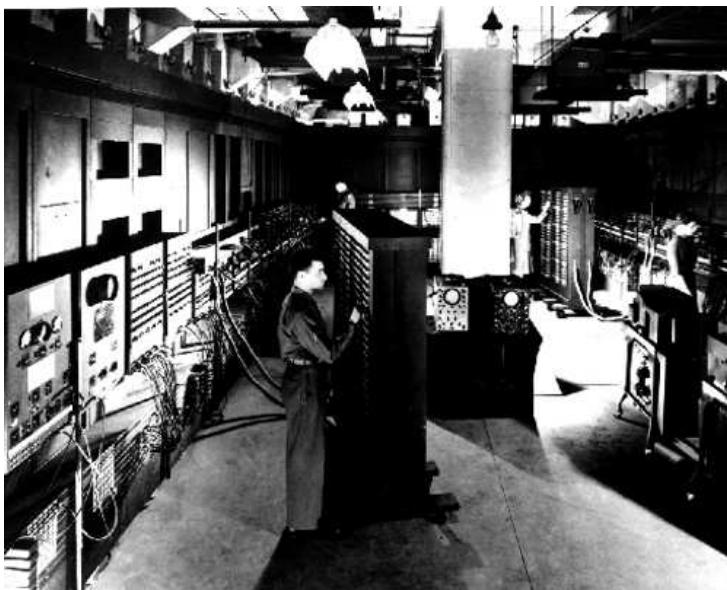
Principes de ECKERT, MAUCHLY et von NEUMANN, années 1940 : une mémoire contient des données *ET* un programme



- + 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



1945–1955 : ordinateur séquentiel



■ Systèmes d'exploitation et supports architecturaux – 3A SLR F2B205A

- Tout manuel
- Ordinateur à relais, voire à lampe
- Langage binaire au mieux, câblage avec des fils standard

Ronan KERYELL

(II)

(I)

1955–1965 : Jusqu'à l'OS

29 / 298



31 / 298



30 / 298

1955–1965 : Jusqu'à l'OS

- 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 (\approx 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 ;
 - ▶ Petit IBM 1401 pour faire du transfert cartes perforées ↗ bandes magnétiques (données)
 - ▶ Gros calcul sur IBM 7094

■ Systèmes d'exploitation et supports architecturaux – 3A SLR F2B205A

HPC Project + IT/TB/DI/HPCAS

Ronan KERYELL

30 / 298

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 ≡ brique de base de l'informatique
 - ▶ Disques magnétiques ↗
 - ▶ Mécanisme d'interruption

■ Systèmes d'exploitation et supports architecturaux – 3A SLR F2B205A

HPC Project + IT/TB/DI/HPCAS

Ronan KERYELL

32 / 298

■ Systèmes d'exploitation et supports architecturaux – 3A SLR F2B205A

HPC Project + IT/TB/DI/HPCAS

Ronan KERYELL

1965–1985 : Multiprogrammation, temps partagés (II)

- ▶ Accès direct à la mémoire
- ▶ Écrans ± 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 ↗ abandonné mais grande influence dans la communauté

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



33 / 298

Unix Story

(II)

- 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 véritablement portable
- Beaucoup d'améliorations fournies par les utilisateurs eux-mêmes (de même que BSD & Linux maintenant) favorisé par le côté non commercial



35 / 298

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*
- Développement de langage interprété *B* utilisé pour développer les outils



34 / 298

Unix Story

(III)

- 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
 - ▶ 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



36 / 298

Unix Story

(IV)

1985– : Micro-ordinateurs

(I)

- ▶ 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

(II)

1985– : Micro-ordinateurs

(III)

- ▶ 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 ↠ prochaines version de SunOS basées sur System V : SunOS 5 (Solaris 2)
- NeXT Computer avec NeXTStep 0.8 (mélange de Mach 2.5 4.3BSD, interface graphique basée sur Display Postscript, programmation en Objective-C)



- Montée en puissance du microprocesseur
- Micro-ordinateurs ≡ ordinateur à base de microprocesseur : quasiment tout ordinateur
- Explosion du multi-fenêtrage ↠ interfaces plus sympathiques
- Gros ordinateur (parallèle) ↠ 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
 - ▶ Grand retour en arrière (...pour les spécialistes) : mono-programmation : MS-DOS 1.0, PC-DOS 1.0



1985– : Micro-ordinateurs

(IV)

- 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. Reprise 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 ?
- Un système Unix ≡ 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



Le plan



1985– : Micro-ordinateurs

(V)

- 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>



Concepts de base

(I)

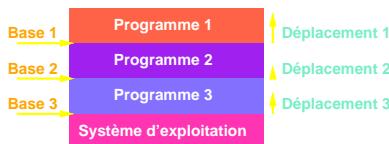
- Multiprogrammation
- Temps partagé
- Pagination & segmentation



Multiprogrammation

- 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

⚠ 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
 - Langages interprétés : JVM pour Java, PostScript, Forth, PERL, Python, SmallTalk, BibTeX, ...



Code translatable

- 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)

(II)



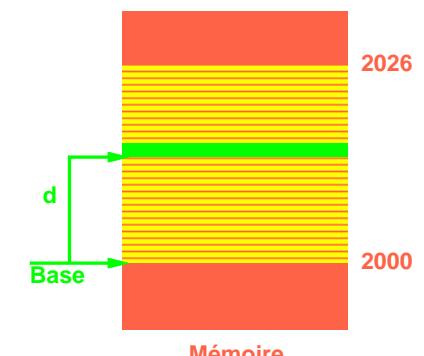
Base et déplacement

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

$$\begin{aligned} d &\in [0, d_{\max}] \\ \text{Registres } &\left\{ \begin{array}{l} \text{base} = 2000 \\ d_{\max} = 26 \end{array} \right. \\ \implies &2000 \leq \text{base} + d \leq 2026 \end{aligned}$$

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

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



Base et déplacement

sur x86, mais sans vérification des bornes

(II)

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*



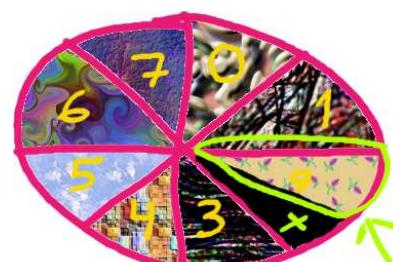
Blocage dans le tourniquet

(I)

Pagination

(I)

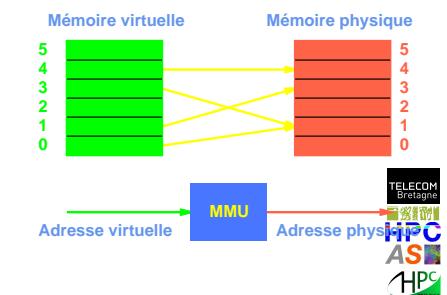
- Blocage possible d'un processeur avant la fin de son quantum de temps
- Libère du temps pour les autres



- 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

- Soit en mémoire de masse (sur disque dur)
- Soit inexisteante



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

(II)

- 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,...

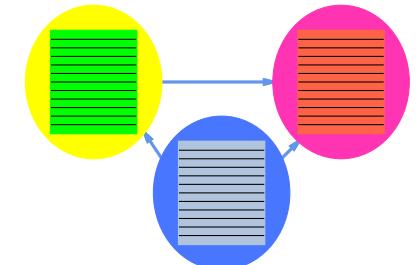


Segmentation

(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
 - Segment pour des données globales



Le plan

- Historique
- Concepts de base
- 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
- Gestion mémoire

- Virtualisation
- Les entrées-sorties
- Systèmes de fichiers
 - Disques
 - Formatage
 - RAID
 - ZFS
 - Pilote de périphérique
- Systèmes de fichiers distants
 - NFS
- Conclusion



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
 - Systèmes de fichiers
 - 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



Tâches

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



Le plan

(I)

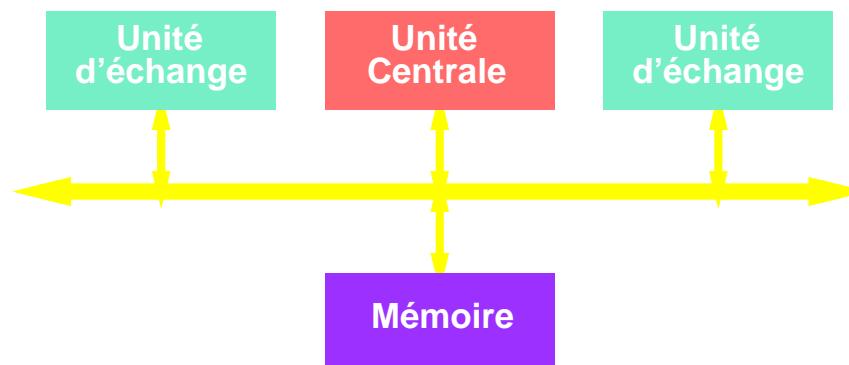
- 1 Historique
- 2 Concepts de base
- 3 Concurrence & Parallélisme
 - Introduction
 - Hypothèses sur architecture matérielle
 - Notion de noyau
 - Processus lourds & légers, *threads*
 - Plus de détail des tâches dans Linux
 - Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement
- 4 Gestion mémoire

- 5 Virtualisation
- 6 Les entrées-sorties
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Schéma d'un monoprocesseur

(I)



- Unité centrale : interprète instructions des programmes applicatifs et du système d'exploitation

Schéma d'un monoprocesseur

(II)

- 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 directe à la mémoire par unités d'échange

Direct Memory Access (DMA)

Mécanisme d'interruptions

(I)

Besoin de surveiller et réagir

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

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



Mécanisme d'interruptions

(II)



Mécanisme d'interruptions

(III)

Types d'interruptions :

- Externes : matérielles
Déclenchées par du matériel externe : horloge, clavier, réseau, entrée-sortie, fin de ligne vidéo, panne de courant,...
- Internes : interruptions logicielles, déroutement
Instruction de passage dans le noyau, division par 0, instruction inexistante, instruction non autorisée, accès mémoire invalide,...

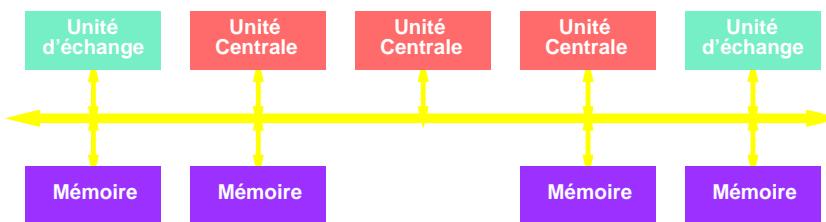
Le concept équivalent d'interruption existe au niveau logiciel : signaux Unix \equiv fonction (*handler*) exécutée sur certaines conditions (erreur, communication possible,...),...



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



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
 - ▶ Monoprocesseurs
 - ▶ Multiprocesseurs à mémoire partagée
- Architectures réparties
 - ▶ Multiprocesseurs à mémoire distribuée : communication par messages
 - ▶ Réseaux



Notion de noyau

► 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

- 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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Noyau Unix

► Notion de noyau

(I)

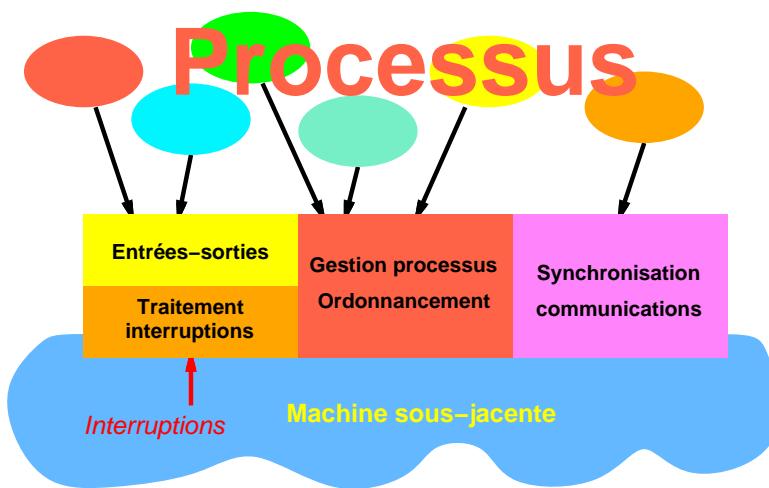
Plus qu'un noyau minimal : presque tout le système d'exploitation

- 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

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,...

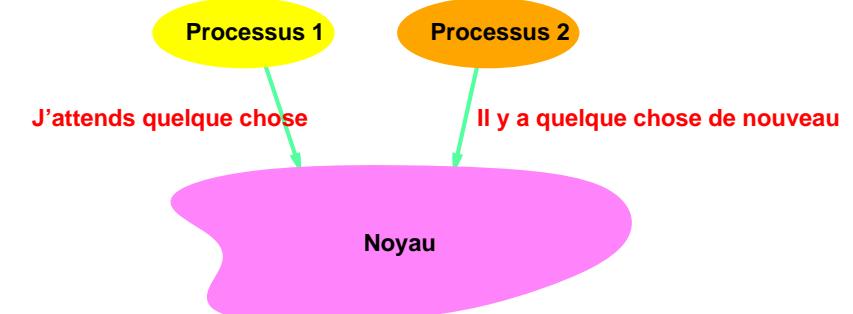
Dans la suite on se focalise sur concurrence et systèmes d'exploitation

Fonctions d'un micro-noyau

- 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

(I)

Noyau et interface de programmation

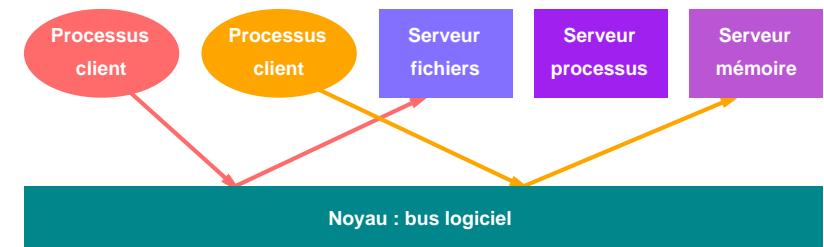


Structure Unix traditionnel (ab initio)



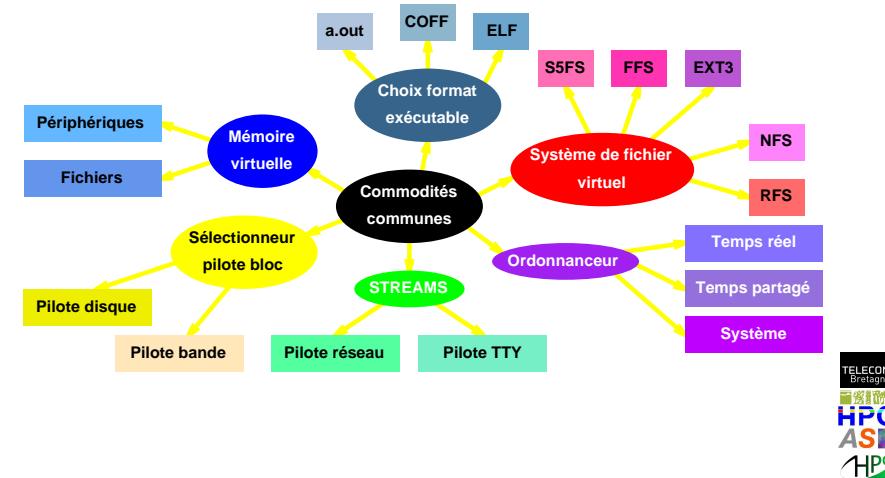
(I)

Séparer politique et mécanisme



Mécanisme de clients-serveurs autour d'un bus logiciel

Structure Unix moderne



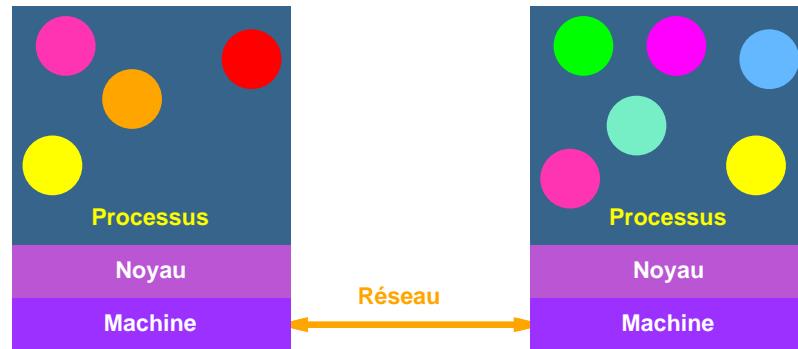
(I)

Centralisation contre distribution

2 types de systèmes

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

Systèmes distribués



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

(I)

Quand donner contrôle au noyau ?

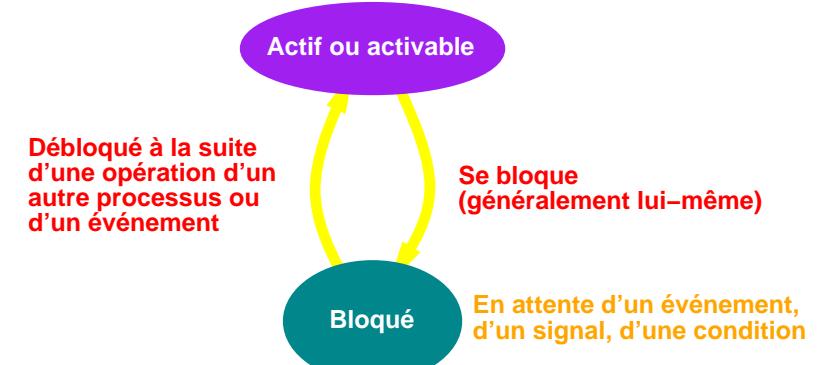
- Quand un processus actif appelle une fonction système (du noyau)
 - ▶ Créer un processus
 - ▶ Attendre la fin d'un processus
 - ▶ Indiquer que l'on a terminé
 - ▶ Se bloquer en attente d'un événement
 - ▶ Demander une entrée-sortie de bas niveau
 - ▶ Attendre un certain temps
 - ▶ Envoyer un message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

Informations gérées par le noyau

- 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

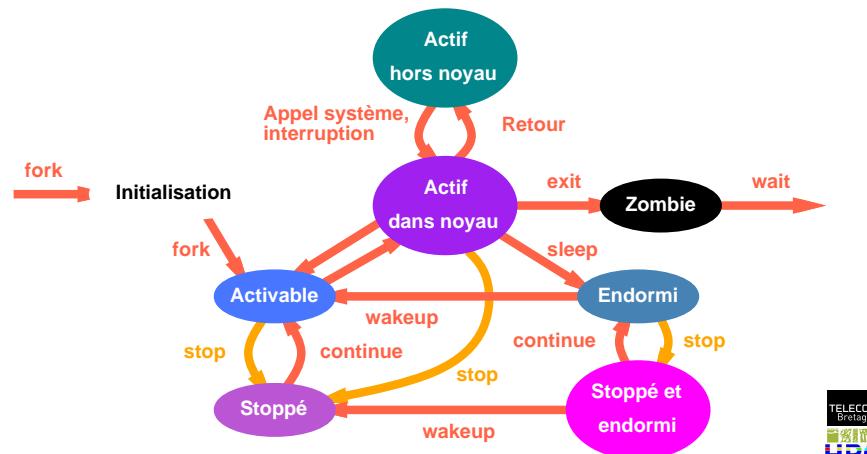
(I)

Graphe des états d'un processus



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)fondctionnement 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,...

Atomicité des actions noyau

(I)

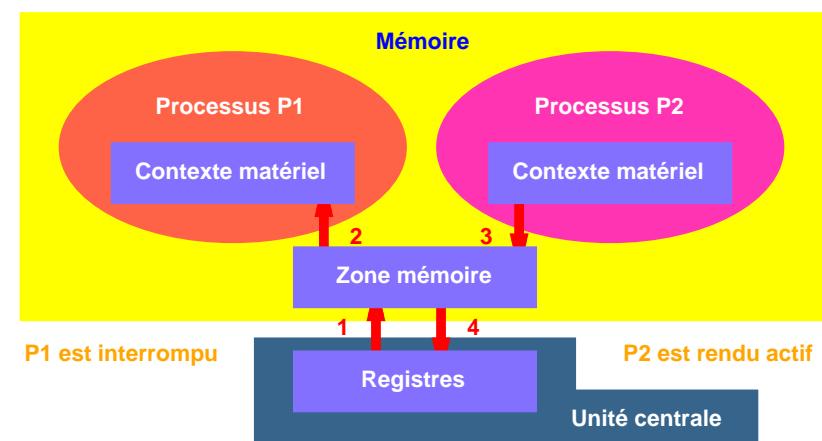
- De nombreuses actions concurrentes dans le système : mise à jour liste processus, modification entrées-sorties à faire,...
- ~ Besoin d'assurer cohérence des informations gérées par le noyau
- Utilisation de l'exclusion mutuelle
 - Mono-processeur : masquage des interruptions
 - Multiprocesseur à mémoire 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...

Sauvegarde et restitution de contexte

(I)

Comment modifier l'exécution des processus ?



Sauvegarde et restitution de contexte

(II)

- ➊ P1 actif
- ➋ 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
- ➌ Le noyau s'exécute
- ➍ Le noyau fait une copie de cette zone dans le descripteur du processus P1 (contexte matériel)
- ➎ Le noyau choisit de rendre le processus P2 actif. Il copie le contenu du contexte matériel du processus P2 dans la zone mémoire
- ➏ L'instruction de retour sur interruption copie la zone mémoire dans les registres du processeur
- ➐ P2 commence/continue sa vie



Appel moniteur/BIOS

► Notion de noyau

(I)

Certains systèmes ont une couche d'abstraction matérielle supplémentaire

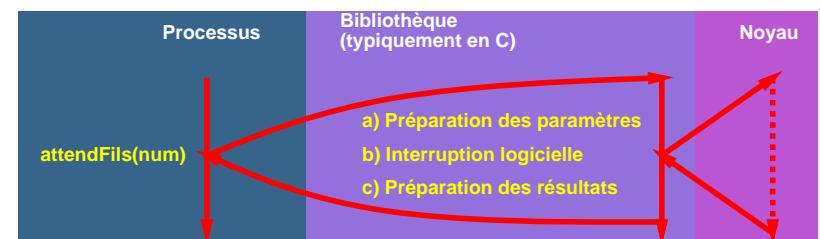
- Portabilité accrue : *Basic Input/Output System* BIOS sur PC) pour accéder aux ressources d'un PC sans faire appel à des programmes sur disque ↳ contenu en mémoire permanente (ROM, Flash)
- Possibilité de debug, variables d'environnement (Sun)
- Des cartes d'entrées-sorties peuvent rajouter des fonctionnalités

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



Appel système

(I)



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



Contexte logiciel d'un processus

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...

(I)

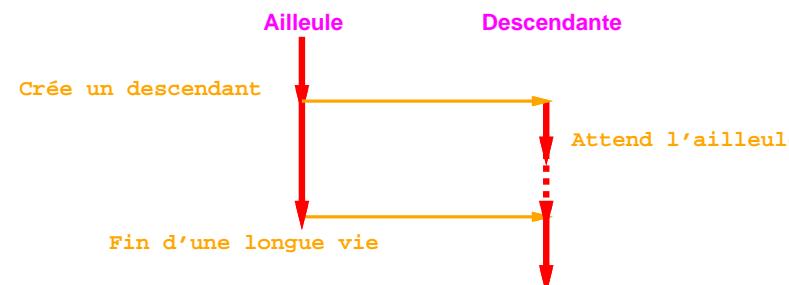
Le plan

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



Vie et mort d'un processus

(I)



Préemption dans le noyau

- 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



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



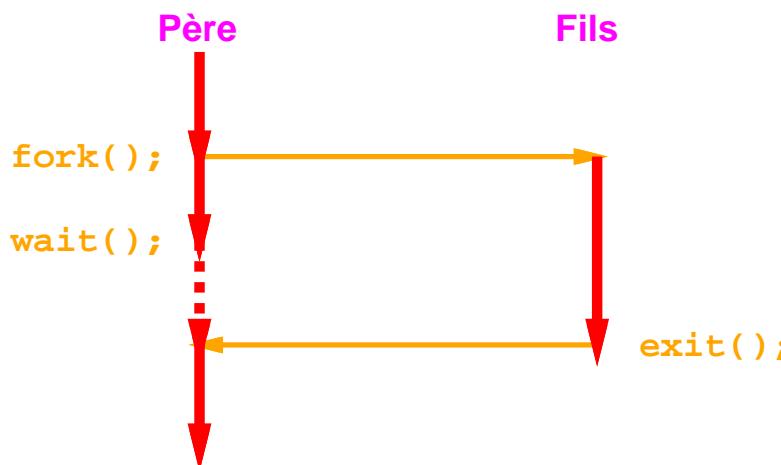
4 états d'exécution possible

(II)

- 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);
    }
    /* On n'exécutera jamais ici. */
}
```



Contexte logiciel processus lourd UNIX

(I)

- État du processus (arrêté, activable,...)
- Date de lancement du processus
- Temps unité centrale utilisée
- 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)
- Identité du groupe *GID* (*group identification*) réel, *GID* effectif (droits temporaires empruntés à un groupe)



Distinguer le père du fils en Unix ?

(II)

```
}
```

- Je suis le père et resultat contient le numéro du fils *
- ...



Contexte logiciel processus lourd UNIX

(II)

- 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



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*



Espace virtuel des processus UNIX

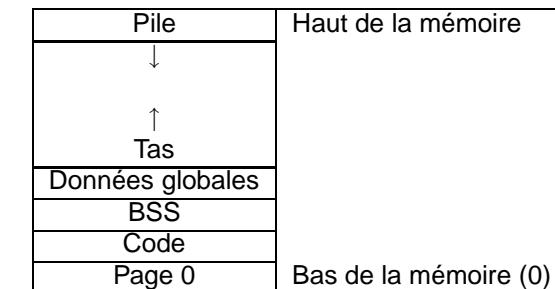
(II)

```
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)
```

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
- Pile contient au départ arguments de



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'adresses 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
```
 - ▶ IPC (InterProcess Communication) : sémaphores, mémoire partagée,...



Processus légers

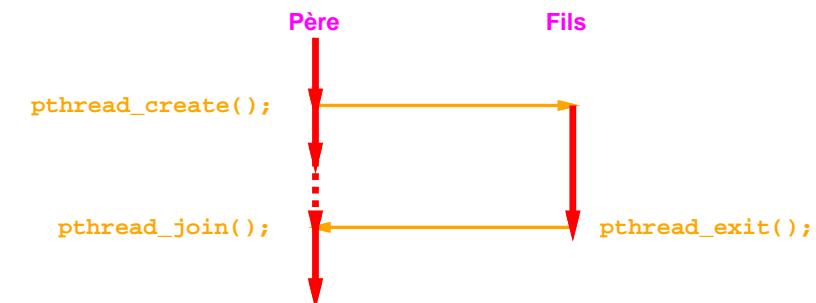
- Besoin de concurrence plus fine dans un programme
- Processus lourd peut contenir plusieurs fibres d'exécutions (*threads of execution*, processus légers)
- Partagent au sein d'un processus lourd Unix
 - ▶ Mémoire (données globales)
 - ▶ Fichiers
 - ▶ Signaux
 - ▶ Données internes au noyau (état processeur,...)
- 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

(I)



Vie et mort d'un processus version thread

Thread POSIX



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

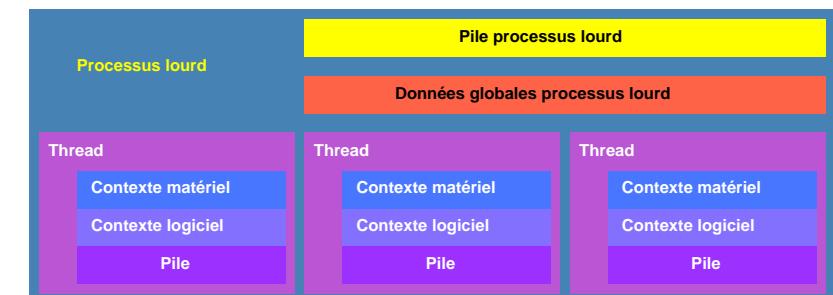
- Attendre la fin d'un fils *thread*

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

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

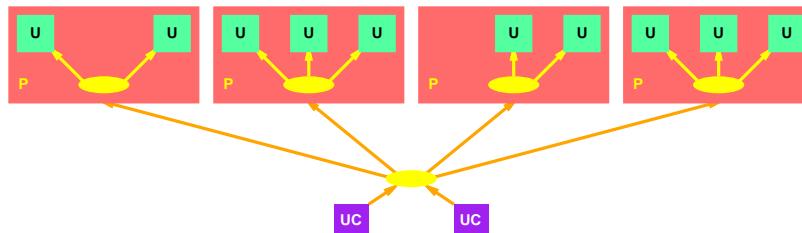


Threads utilisateurs dans un processus



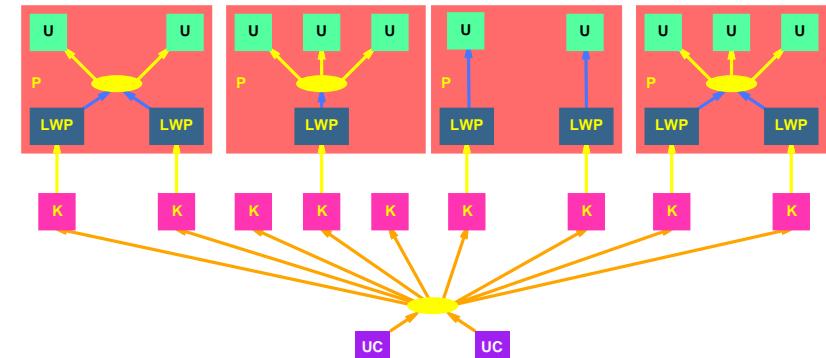
2 niveaux d'ordonnanceur

Où choisir de multiplexer exécution concurrente ?



- Ordonneur du noyau : processus plutôt lourds
- Ordonneur utilisateur dans chaque processus lourds : threads plutôt légères
⚠️ blocages sur E/S... ☺

Avec ordonneur de threads noyau



Les processus légers dans Linux

- Pas de gestion spécifique dans Linux (contrairement à Solaris (LWP) ou Windows)
- Processus léger ≡ 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

Quid entre processus lourd et léger ?

```
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

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 µs	66 µs
LWP	350 µs	? 390 µs
Processus	1700 µs	200 µs

Compromis d'utilisation



Des processus sans système d'exploitation ?

(I)

- Processus et concurrence ≡ 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



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



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
 - Systèmes de fichiers
 - Disques
 - Formatage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Différents usages de clone()

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

```
1 asmlinkage int sys_fork(struct pt_regs *regs)
{
    return do_fork(SIGCHLD, regs.esp, &regs, 0, NULL, NULL);
}
```

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

```
1 asmlinkage int sys_vfork(struct pt_regs *regs)
{
    return do_fork(CLONE_VFORK|CLONE_VM|SIGCHLD, regs.esp, &regs, 0, NULL);
}
```

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

(I)



Différents usages de clone()

```
1 asmlinkage int sys_clone(struct pt_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, &regs, 0, parent_tidptr, child_
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 pt_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, parent_tidptr,
12                      ...
13     if (!(clone_flags & CLONE_STOPPED))
14         wake_up_new_task(p, clone_flags);
15     else
16         if (p->state == TASK_STOPPED)
17             if (clone_flags & CLONE_VFORK)
18                 wait_for_completion(&vfork);
19     return pid;
20 }
```



Différents usages de clone()

(IV)

- 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

- 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 !



(I)

Exemple de liste de processus

• Outils ps, top,...

UUSER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	1584	80	?	S	04:46	0:00	init [5]
root	2	0.0	0.0	0	0	?	SN	04:46	0:00	[ksoftirqd]
root	3	0.0	0.0	0	0	?	S<	04:46	0:00	[events/0]
root	2141	0.0	0.0	2984	176	?	Ss	04:46	0:00	/sbin/klogd
bind	2171	0.0	0.1	29740	736	?	Ssl	04:46	0:00	/usr/sbin/bind
	...									
keryell	9620	0.0	0.1	4612	904	pts/4	R+	08:52	0:00	ps auxww

- Des entrées dans /proc pour chaque processus dont self



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'attentes d'entrées/sorties
 - ▶ Statistiques d'usage
 - ▶ Priorité
 - ▶ Gestion des machines parallèles NUMA



Représentation des tâches en interne

(II)

▶ ...

- 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

(III)

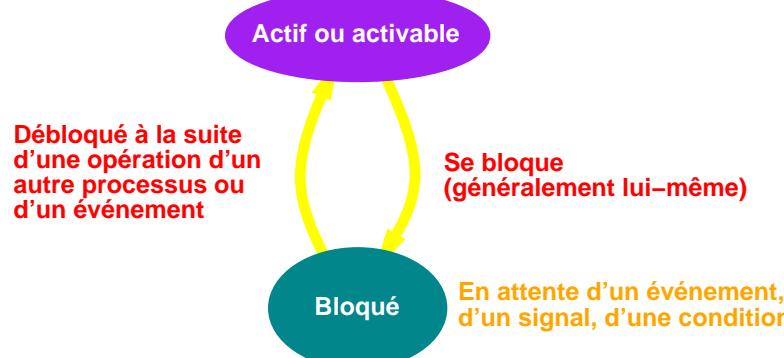
```

1 struct_uthread_info_u{
2     struct_u_task_struct_u *task; /* main_task_structure */
3     struct_uexec_domain_u *exec_domain; /* execution_domain */
4     unsigned_u long flags; /* low_level_flags */
5     unsigned_u long status; /* thread-synchronous_flags */
6     __u32 cpu; /* current_CPU */
7     __s32 preempt_count; /* preemptable, <0=BUG */
8     mm_segment_t addr_limit; /* thread_address_space */
9     struct urestart_block_u restart_block;
10    unsigned_u long previous_esp; /* ESP of the previous stack in
11                                * of nested (IRQ) stacks
12    __u8 supervisor_stack[0];
};
```



Graphe des états d'un processus

(I)



Accès aux tâches

(I)

- `current_thread_info()` pointe vers le descripteur courant

```

1 static_uinline_u struct_uthread_info_u*current_thread_info(void)
2 {
3     struct_uthread_info_u*ti;
4     __asm__ ("andl %esp,%0;":=r"(ti)":"~(THREAD_SIZE-u1));
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

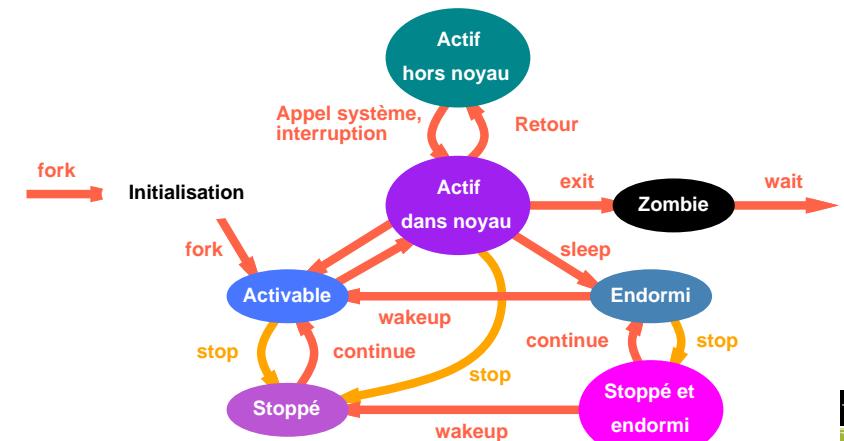
- Fonction de hachage pour retrouver rapidement une tâche à partir de son *pid*

- [kernel/pid.c]



Un processus Unix dans tous ses états

(I)



États internes dans Linux

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

(I)



États internes dans Linux

► #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 SIGSTOP (^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



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 débugger,...)
- #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



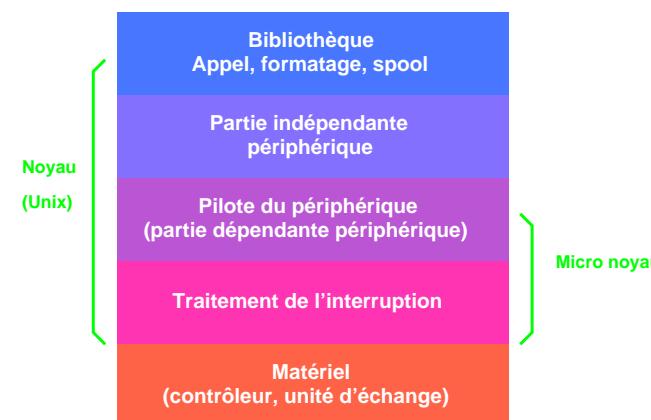
Le plan

- 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 Entrées-sorties
 - Gestion du temps qui passe
 - Ordonnancement

- 5 Virtualisation
- 6 Les entrées-sorties
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
 - NFS
- 7 Systèmes de fichiers distants
- 8 Conclusion



Entrées-sorties physiques (bas niveau)



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
 - Systèmes de fichiers
 - Disques
 - Formatage
 - RAID
 - ZFS
 - Pilote de périphérique
 - Systèmes de fichiers distants
 - NFS
- 7 Conclusion



Déroulement d'une entrée-sortie physique

- 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

- 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
 - Résonnateur 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/echelles_de_temp
<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
 - Systèmes de fichiers
 - Disques
 - Formatage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Priorité

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

- 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



(I)

Types de multitâche

- 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



(I)

Quelques politiques d'ordonnancement

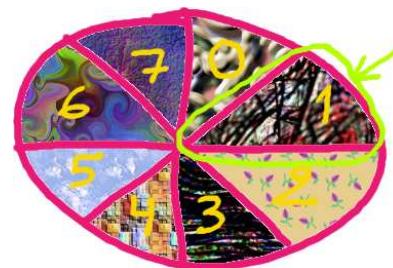
- Premier Arrivé Premier Servi (PAPS) ou *First In First Out* (FIFO)
- Tourniquet (*round robin*) ou partage du temps
- Priorité statique (temps réel) ou dynamique
- *Shortest job first...* Mais nécessite de connaître la durée de la tâche (future)
- *Earliest deadline first...* Mais nécessite de connaître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

(I)



Temps partagé – tourniquet

- 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

Programme ≡ calculs + E/S

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

- Processus orienté calcul
 - ▶ Gros calculs
 - ▶ Préemption fréquente gâcherait du temps (système, cache, swap,...)
 - ▶ Réactivité moins évidente à l'utilisateur
- Processus orienté entrées-sorties
 - ▶ Calculs souvent bloqués par des E/S
 - ▶ Préemption 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

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



Politique par priorité

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 à priorité plus dynamique
- Caractéristiques de chaque périphérique : ne pas ralentir des périphériques qui sont déjà lents,...

(I)



Politique à priorités dans Unix

(II)

- 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})</i>	<i>Exemple</i>	<i>Type</i>
+19 (+ faible)	Useur d'écran	Plutôt utilisateur
⋮	⋮	
+1		
0 (standard)	Jeux vidéo	
-1	Numérisation d'un cours	
⋮	⋮	Plutôt système
-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 \odot (modifie sa base) à l'aide de la commande `nice`
- Le super utilisateur peut augmenter la priorité



Toutes les priorités dans Linux

(I)

[`include/linux/sched.h`]

```

1  /*
2  * Priority of a process goes from 0..MAX_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_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 #define MAX_PRIO ((MAX_RT_PRIO+40)

```

[`kernel/sched.c`]



Toutes les priorités dans Linux

(II)

```

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))
9
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

(II)

```

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	ksoft
5	root	9	0	0	0	0	S 0.0	0.0	59:20.41	kswapd
6	root	9	0	0	0	0	S 0.0	0.0	0:00.00	bdflush



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 ↵ 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

- ▶ 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
- Préemption 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

(II)



Ordonneur Linux 2.6

- 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 (runqueues)

(I)

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

```

1 struct_u runqueue_u{
2     spinlock_t_u lock;
3     unsigned_u long_u nr_running;
4     #ifdef_u CONFIG_SMP
5     unsigned_u long_u cpu_load;
6     #endif
7     unsigned_u long_u long_u nr_switches;
8     unsigned_u long_u nr_uninterruptible;
9
10    unsigned_u long_u expired_timestamp; /* Date de l'échange d'arrays */
11    unsigned_u long_u long_u timestamp_last_tick;
12    task_t_u *curr, u *idle;
13    struct_u mm_struct_u *prev_mm; /* Espace mémoire de la tâche précédente */
14    prio_array_t_u *active, u *expired, u arrays[2];
15    int_u best_expired_prio;

```



File d'exécution (runqueues)

(II)

```

11     atomic_t_u nr_iowait;
12     ...
13     /* Des choses pour le multiprocesseur et des statistiques */
14 }

```



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...
- Idée : 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 $\mathcal{O}(\log \text{bitsizeof}(\text{int}))$, en général 1 cycle

(II)

Tableaux de priorité

- Construire 1 tableau de bits associé aux files avec bit à 1 si file non vide
- Trouver en $\mathcal{O}(\frac{\#prio}{\text{bitsizeof}(\text{int})} \log \text{bitsizeof}(\text{int}))$ 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

(I)

- Idée 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
- Dès qu'une tâche de *active* a usé son quantum, on recalcule son quantum et on la met au bon endroit dans le tableau *expired*



Recalculer les quanta de temps

(II)

- 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     /* ... Switch the active_and_expired_arrays.
4     */
5     schedstat_inc(rq, sched_switch);
6     rq->active = rq->expired;
7     rq->expired = array;
8     array = rq->active;
9     rq->expired_timestamp = 0;
10    rq->best_expired_prio = MAX_PRIO;
11 } else
12     schedstat_inc(rq, sched_noswitch);

```

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



La fonction d'ordonnancement schedule()

- 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

- Un processus utilisateur a par défaut un 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.
14 */
```



Calcul des priorités dynamiques et quanta



Calcul des priorités dynamiques et quanta

- 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 /*
2  * task_timeslice() scales user-nice values [-20...0...19]
3  * to timeslice values:[800ms...100ms...5ms]
4  *
5  * The higher a thread's priority, the bigger timeslices
6  * it gets during one round of execution. But even the lowest
7  * priority thread gets MIN_TIMESLICE worth of execution time.
8  */
9 #define MIN_TIMESLICE max(5*HZ/1000, 1)
10 #define DEF_TIMESLICE (100*HZ/1000)
11 #define SCALE_PRIO(x, prio) \
12     max(x*(MAX_PRIO-prio)/(MAX_USER_PRIO/2), MIN_TIMESLICE)
13 static unsigned int task_timeslice(task_t *p)
```



Calcul des priorités dynamiques et quanta

```

15 {
16     if (p->static_prio < NICE_TO_PRIO(0))
17         return SCALE_PRIO(DEF_TIMESLICE*4, p->static_prio);
18     else
19         return SCALE_PRIO(DEF_TIMESLICE, p->static_prio);
}

```



Sommeil & réveil

(IV)

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]



Sommeil & réveil

(II)

Sommeil & réveil

(III)

```

1 long
2 do_clock_nanosleep(clockid_t which_clock, int flags, struct timespec *tsave)
{
4 ...
5 /* Crée un élément de liste abs_wqueue avec la tâche courante comme
6 valeur, style_abs_wqueue = CONS(current, NIL) */
7 DECLARE_WAITQUEUE(abs_wqueue, current);
8 ...
9 init_timer(&new_timer);
10 new_timer.expires = 0;
11 new_timer.data = (unsigned long)current;
12 new_timer.function = nanosleep_wake_up;
13 abs = flags & TIMER_ABSTIME;
14 ...
15 if (abs && (posix_clocks[which_clock].clock_get !=
16 posix_clocks[CLOCK_MONOTONIC].clock_get))
17 /* En gros fait
18 nanosleep_abs_wqueue = CONS(current, nanosleep_abs_wqueue) */
19 add_wait_queue(&nanosleep_abs_wqueue, &abs_wqueue);
20

```



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)`



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

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 : ordonneur choisit de rentrer actif 1 processus activable *présent en mémoire principale*
- Au niveau haut : ordonneur 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'ordonneur noyau a constamment en mémoire principale une table de l'ensemble des processus avec données nécessaire au fonctionnement de l'ordonneur



Ordonnancement et va et vient

- 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)

(I)



Petite conclusion sur ordonnancement Linux

- De gros progrès dans Linux 2.6 !
- Noyau préemptif
- Rajout de priorités fixes temps réel
- Ordonneur 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)

(I)



Politiques temps réel

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`
 - ▶ Préemption
 - ▶ 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)`



Des ennuis : inversion de priorité

Supposons

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



Des ennuis : inversion de priorité

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 !



(II)

Solutions possibles

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



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

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).



(I)

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.



Tâche martienne de priorité moyenne

(I)

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



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.



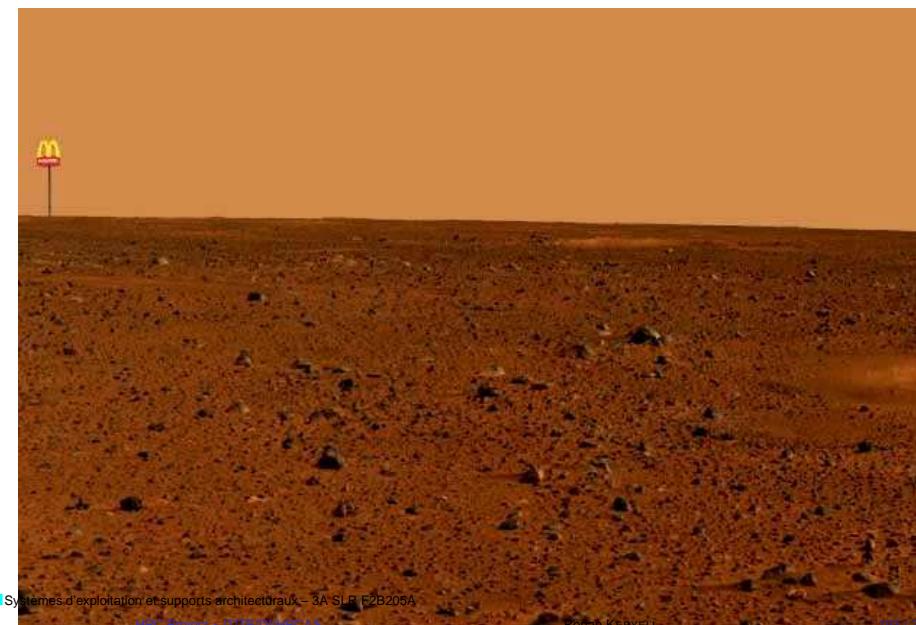
Debug martien

(I)

- Vitesse de la lumière Terre-Mars $\approx 14 \text{ mn } \odot$
- 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 !



Debug martien



(II)

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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion

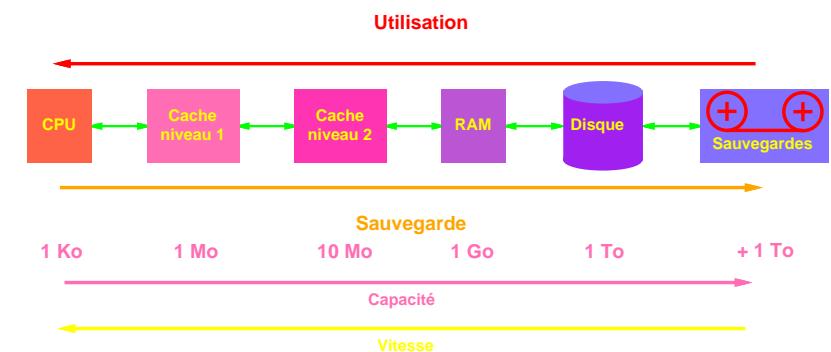
Gestion des mémoires

(I)

Hiérarchie mémoire

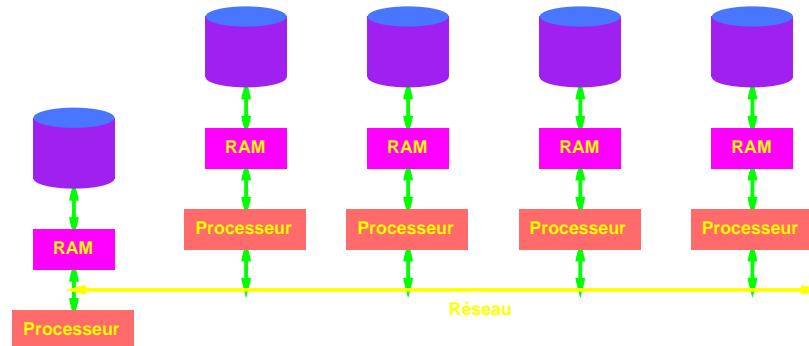
(I)

- 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



Hiérarchie mémoire version réseau

(I)



Dépasser des limitations d'adressage

(II)

- 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

(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
- 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 :

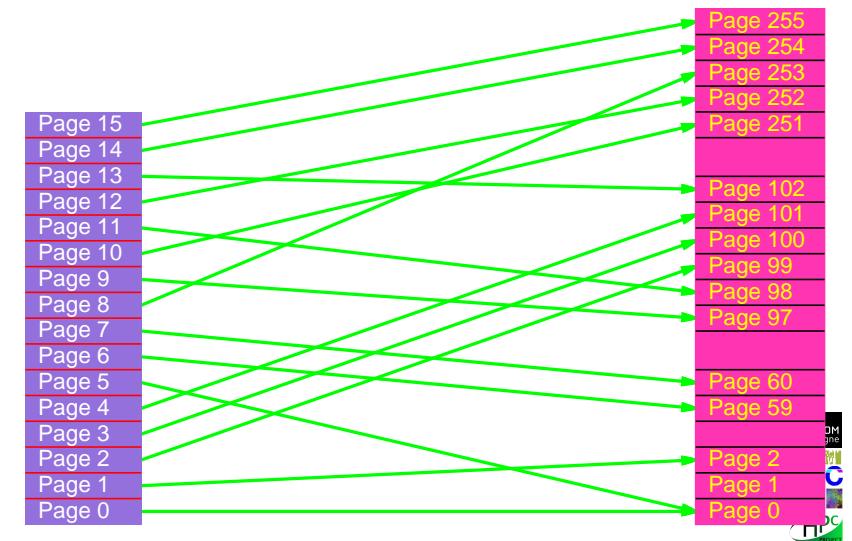
$$a_p = a_v + 16 \times s$$

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



Dépasser des limitations d'adressage

(III)



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 $p \cdot 2^{32}$ et elles peuvent grossir chacune jusqu'à 4 Go



Intérêts de la mémoire virtuelle

(II)

- 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*)



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
 - Code si plusieurs instances du même programme



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 ?

$$f : a_v \longrightarrow a_p \\ \mathbb{N}/2^{64}\mathbb{N} \longrightarrow (\mathbb{N}/2^{40}\mathbb{N})^{2^{64}}$$

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

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

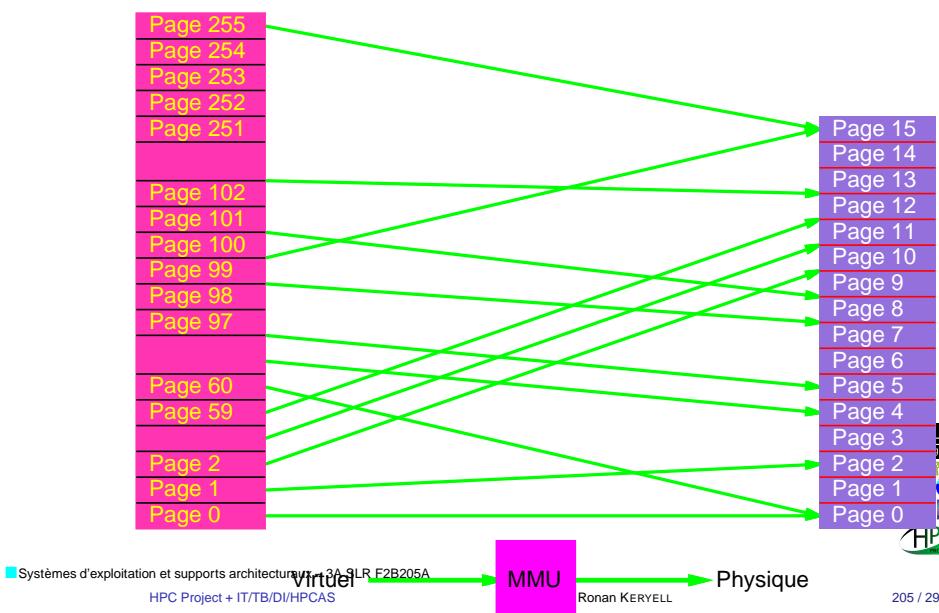
$$a_p = f\left(\frac{a_v}{t}\right) + (a_v \% t)$$

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



Grande mémoire virtuelle

(II)



- 💡 Idée 2 : structure « creuse » : impossible qu'un programme non pervers utilise autant de mémoire physique et donc de pages ↗

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

Traduction des adresses des pages

(I)

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

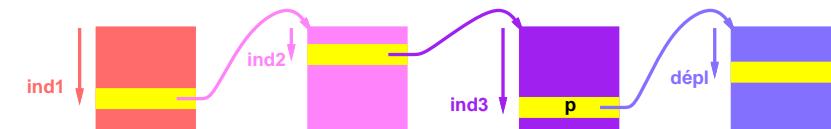
	Existence	Page physique
0	0	
1	1	0x23fe
2	1	0x5fde
3	1	0x2345
4	0	
5	1	0x0
6	0	
	:	:

Traduction de l'adresse virtuelle 0x37890 ($[\underline{3} \quad 0x7890]$) en adresse physique 0x23457890 ($[0x2345 \quad 0x7890]$)

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
- $\boxed{\text{ind1} \quad \text{ind2} \quad \text{ind3} \quad \text{dépl}}$ traduite en $\boxed{\text{p} \quad \text{dépl}}$



Défaut de page

(I)

- Si une page n'existe pas (invalidé) 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
 - ▶ Emulation 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



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



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



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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Machines virtuelles matérielles

- 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,...



Architectures virtuelles

(I)

Différences entre ce que voit l'utilisateur et la réalité

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

Transparence

... ⚠ mais attention parfois aux performances !

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
- SOA (Service-Oriented Architecture)



Intérêts de la virtualisation

(II)

- ▶ 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 ± 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-fenêtre
 - 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, PTlsm (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 \rightsquigarrow transformé en écriture dans une fenêtre de l'écran physique
- L'aspect multiprogrammation reste simple : commuter des machines virtuelles
- ⚡ Difficile à optimiser



Machines virtuelles - fonctionnement

(II)

- ▶ 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
- Nouveaux processeurs rajoutent des instructions pour virtualiser de manière efficace instructions superviseurs
 - ▶ AMD Pacifica, Intel VT



Machines virtuelles - fonctionnement

(III)

- ▶ 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



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)
- OS hôte tourne dans « domaine » 0

Nécessite d'avoir sources de l'OS...



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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



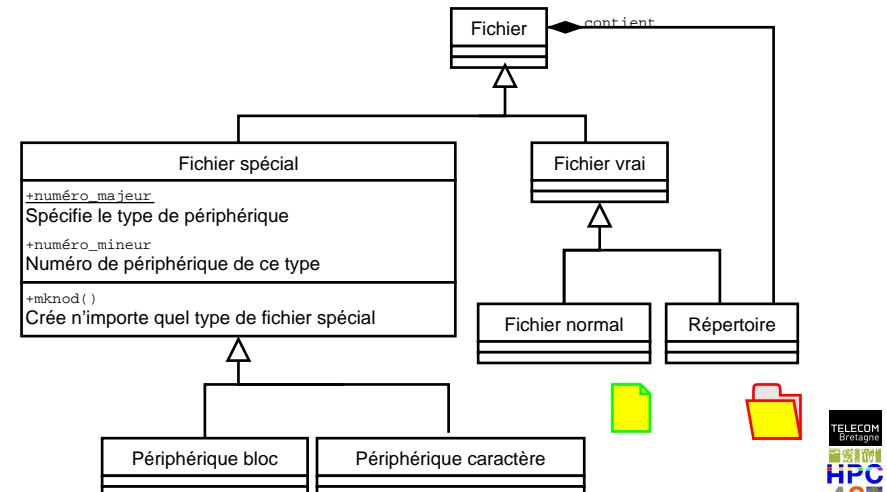
Descripteur de fichier

(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-noeud (*v-node*) du fichier dans son système de fichiers. v-noeud implémenté par exemple par un i-noeud
 - ▶ 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
- Idée 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



Hiérarchie de classes de fichiers Unix simplifiée (I)



Descripteur de fichier

(II)

- 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 exécutables ? Mon voisin a le droit de le lire ?...)
- Désassociation d'un descripteur de fichier lorsqu'on n'en a plus besoin = *fermer* un fichier

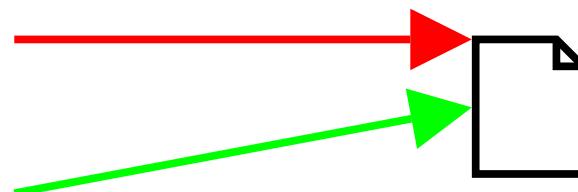

```
int close(int fd)
```



Sémantique des fichiers Unix

- 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



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



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

(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

(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... ☺



Passe un descripteur à ton voisin

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

(I)

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

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...

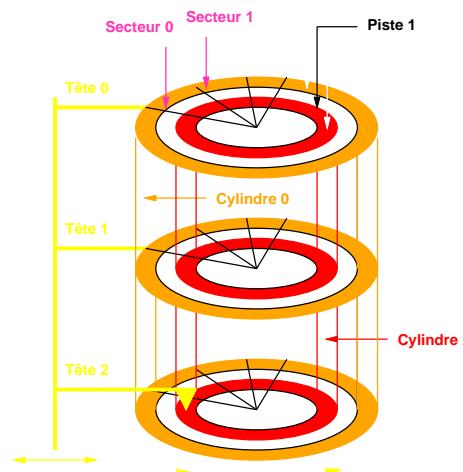
- `read`, `write`, ... : assez bas niveau
- Ces appels systèmes provoquent un passage dans le noyau à chaque fois : lourd
- Besoin de fonctions légères et de plus haut niveau : tamponne les entrées-sorties et envoie le tout par bloc ↗ moins de passage dans le noyau
- `[f]printf()`
- `[f]scanf()`
- `fopen()`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

(I)



(Disques magnétiques)

(I)



Unix FFS optimisé pour les disques

- 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

- 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

(I)



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...



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

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

Les commandes sont abrégables

- 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) 32130

(I)

format à l'œuvre

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
- 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



format à l'œuvre

(III)

- 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

- Partage du disque disque entre plusieurs OS

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

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

SELECT ONE OF THE FOLLOWING:

- Create a partition
 - Specify the active partition
 - Delete a partition
 - Exit (update disk configuration and exit)
 - Cancel (exit without updating disk configuration)
- Enter Selection:

- 1 seule partition Solaris par disque



Étape fdisk sur PC

(II)

- Partition Solaris alignée sur 1 cylindre
- Épargner le *Master Boot Record* sur le cylindre 0
- Subtilité
 - On formate le disque avec `format`
 - On partitionne globalement avec `fdisk` appelable directement depuis `format`
 - 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/rdsck/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

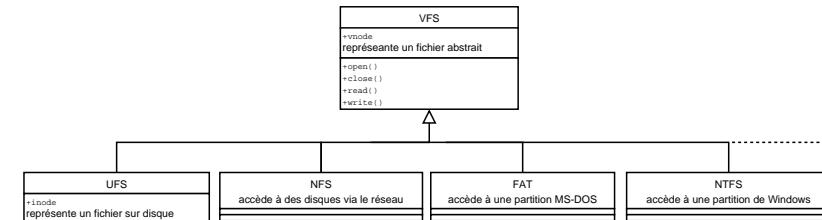


- / est toujours monté (lancement du noyau) et indémontable
- Montage cache les fichiers préexistants dans le répertoire

Interface de système de fichier VFS

(I)

Le Virtual File System permet un héritage au sens objet



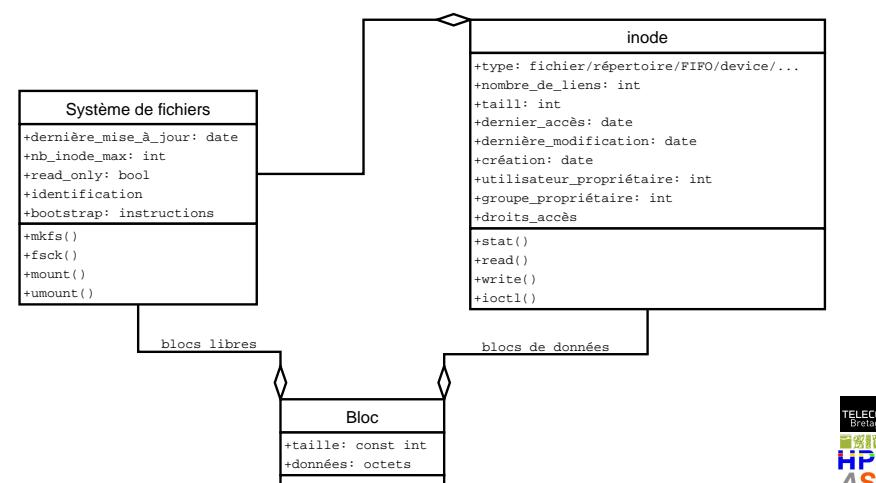
Montage/démontage d'un système de fichiers (II)

- 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

(I)



Structure de répertoire virtuel Unix

(I)

inode	Nom
12345	.
67890	..
2004	toto
4	Schtroumpfette_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 : aplatis 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 (grossiers séquentiels,...)
 - ...



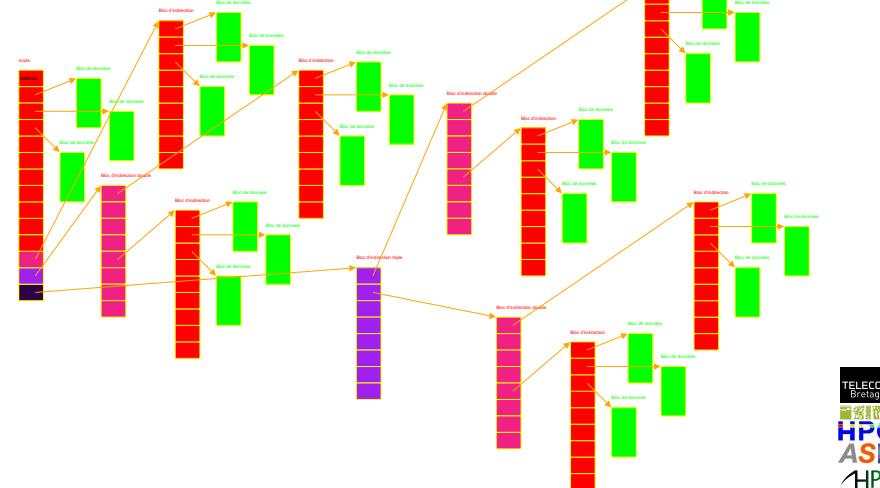
Transformer de la mémoire disque en fichiers Unix

(I)

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



Transformer de la mémoire disque en fichiers Unix (II)



Systèmes 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

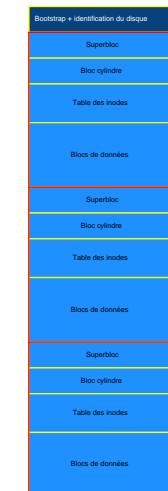
HSFS : High Sierra et ISO-9660 (version officielle de la précédente) pour CD-ROM. Lecture seule. Extension Rock Ridge fournissant la sémantique UFS (sauf les liens durs et... l'écriture !)

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



Structure globale du FFS

(I)



Groupe de cylindre 1

Groupe de cylindre 2

Groupe de cylindre 3



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

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



Systèmes de fichiers sous Solaris

(III)

- 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 ?
 - ▶ Créer de nouveaux systèmes de fichiers
 - ▶ Rendre les ressources locales et distantes accessibles aux utilisateurs
 - ▶ Connexion et ajout de nouveaux disques



Systèmes de fichiers sous Solaris

(IV)

- ▶ Mise en place d'une excellente politique de sauvegarde
- ▶ Vérification et correction des fichiers endommagés
Pour les hackers : fsdb un debogueur 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

(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 nécessite pas de fsck



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 superblocks 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

(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

(V)

```
deauville-root > newfs -v /dev/rdsck/c0t4d0s5
newfs: construct a new file system /dev/rdsck/c0t4d0s5: (y/n)? y
mkfs -F ufs /dev/rdsck/c0t4d0s5 5926944 214 12 8192 1024 64 2 167 6144 t 0 -1 8 128
/dev/rdsck/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, 3122720, 3220528,
3318336, 3416144, 3513952, 3611760, 3709568, 3807376, 3905184, 4002992,
4100800, 4198608, 4296416, 4394224, 4492032, 4589840, 4684064, 4781872,
4879680, 4977488, 5075296, 5173104, 5270912, 5368720, 5466528, 5564336,
5662144, 5759952, 5857760,
```

Il peut être utile de stocker cette information pour avoir l'adresse des superblocs en cas de coup dur.

- tunefs permet de fignoler les paramètres après coup. Moins utile avec tous les caches des disques



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



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 moitiés 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
- ▶ Corrige le superbloc (taille, nombre d'inodes, nombre de blocs et d'inodes libres)



Vérification d'un système de fichiers

(II)

- ▶ Peut récupérer un superbloc de secours. Si le système est trop HS pour savoir où le trouver, chercher si vous n'avez pas la sortie de newfs quelque part sinon faire un newfs -N du disque pour faire un système de fichiers pour de faux
- ▶ Vérification des inodes (nombre de liens vers l'inode, taille, blocs de données référencés 2 fois)
- ▶ Correction des répertoires « . » et « .. » dans les répertoires
- ▶ ...
- ▶ Si fichiers et répertoires (inodes) non référencés dans un répertoire : reliés à lost+found
- Certains problèmes sont insolubles automatiquement : choix ↗ questions à l'utilisateur. Possibilité de faire un fsck à la main (sur un système de fichier inactif!) avec
fsck /dev/rdsk/device-name
- ⚡ fsck n'a aucun moyen de réparer le *contenu* des fichiers...



RAID

(I)

Augmenter débit et capacité mais diminuer coût ↗ 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...



Vérification d'un système de fichiers

(III)

- ⚠ 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)

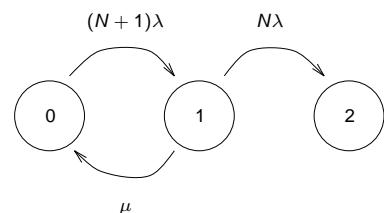
Mettre plus de disques pour compenser les pannes

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 :

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



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.) \rightsquigarrow seule la parité suffit.

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

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

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



Types de RAIDs

(II)

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

	D_1	D_2	D_3	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

\exists autres combinaisons de RAID



ZFS de OpenSolaris Sun

(I)

- Essaye 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
- \rightsquigarrow Zetabyte File System

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



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



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



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
 - Systèmes de fichiers
 - Disques
 - Formatage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



Déploiement de ZFS

(I)

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

Contrôleur ou pilote de périphérique

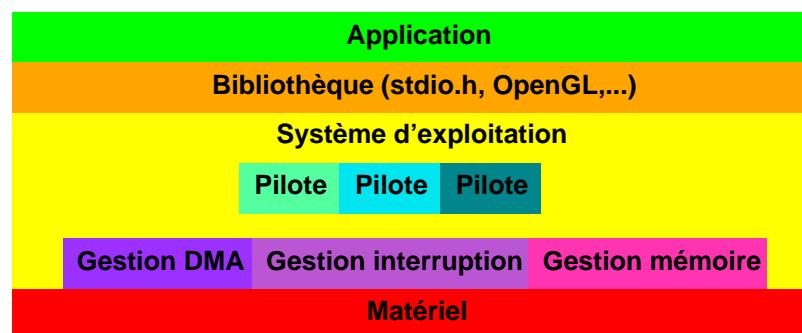
(I)

- 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



Contrôleur ou pilote de périphérique

(II)



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



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 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 caractères 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



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,...)



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



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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



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,...)



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
 - Systèmes de fichiers
 - Disques
 - Formattage
 - RAID
 - ZFS
 - Pilote de périphérique
- 7 Systèmes de fichiers distants
 - NFS
- 8 Conclusion



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



► NFS

Network File System (NFS)

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



- Standard de facto
- Modèle client-serveur
 - Serveur de fichiers exporte un ensemble de fichiers



Network File System (NFS)

(II)

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



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'harchitecte 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`)



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



Le plan

1	Historique
2	Concepts de base
3	Concurrence & Parallélisme
●	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
4	Gestion mémoire

5	Virtualisation
6	Les entrées-sorties
●	Systèmes de fichiers
●	Disques
●	Formatage
●	RAID
●	ZFS
●	Pilote de périphérique
7	Systèmes de fichiers distants
●	NFS
8	Conclusion



Table des matières

Copyright (c)	2
Le cours	3
Problématique	4
Le plan	5
Introduction	6
Machine Virtuelle Étendue	8
Abstraction de fichier	9
Exemples de fichiers et fichiers spéciaux	10
Transparence et opacité	12
Nirvana des systèmes	15
Un ordinateur dans une perspective logicielle	16
DCE — <i>Distributed Computing Environment</i>	17
Évolution logicielle	18
Pourquoi ce cours en 1A/2A... puis en 3A ?	20
Bibliographie	21
Bibliographie Linux	21
Le plan	23

1 Historique

Le plan	25
Historique	26
1945–1955 : ordinateur séquentiel	26
1955–1965 : Jusqu'à l'IOS	28
1965–1985 : Multiprogrammation, temps partagés	30
Unix Story	34
1985– : Micro-ordinateurs	38

2 Concepts de base

Le plan	43
Concepts de base	44
Multiprogrammation	45
Code translatable	46
Base et déplacement	48
Temps partagé – tournequet	50
Blocage dans le tournequet	51
Pagination	52
Segmentation	54

3 Concurrence & Parallélisme

Le plan	56
---------	----

● Introduction

Le plan	57
Concurrence et parallélisme	58
Tâches	59
Le plan	60
Schéma d'un monoprocesseur	61
Mécanisme d'interruptions	63
Schéma d'un multiprocesseur	67
Interruptions & multiprocesseur	68
Classes d'architectures matérielles	69

● Notion de noyau

Le plan	70
Notion de noyau	71
Noyau Unix	72
Notion de micro-noyau	73
Mises en œuvres de la concurrence	74
Fonctions d'un micro-noyau	75
Noyau et interface de programmation	76
Structure Unix traditionnel (ab initio)	77
Séparer politique et mécanisme	78
Structure Unix moderne	79
Centralisation contre distribution	80
Systèmes distribués	81
Quand donner contrôle au noyau ?	82
Informations générées par le noyau	83
Graphe des états d'un processus	84
Graphe des états d'un processus sous Unix	85
Atomicité des actions noyau	86
Contexte matériel	87
Sauvegarde et restitution de contexte	88
Appel système	89
Appel moniteur/BIOS	90
Entrée et sortie du noyau	91
Contexte logiciel d'un processus	92
Processus lourds & légers, <i>threads</i>	93

● Processus lourds & légers, *threads*

Conclusion

- 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



Table des matières

Le plan	94
Vie et mort d'un processus	95
Préemption dans le noyau	96
Synchronisation & concurrence	97
4 états d'exécution possibles	98
Vie et mort d'un processus vers Unix	99
Distinguer le père du fils en Unix ?	101
Contexte logiciel processus lourd UNIX	103
Contexte logiciel UNIX en 2 parties	105
Espace virtuel des processus UNIX	106
Communications inter-processus sous UNIX	108
Processus légers	109
Vie et mort d'un processus version thread	110
Threads POSIX	111
Threads utilisateurs dans un processus	112
2 niveaux d'ordonnanceur	113
Avec ordonnanceur de threads noyau	114
Les processus légers dans Linux	115
Quid entre processus lourd et léger ?	116
Processus et threads : combien ça coûte ?	117
Des processus sans système d'exploitation ?	118

● Ordonnancement

Le plan	140
Temps	140
Génération d'événements	142
L'heure et sa distribution	143
Ordonnancement	145
Priorité	146
Types de multitâche	148
Quelques politiques d'ordonnancement	149
Processus dirigé par le calcul ou les E/S	150
Temps partagé – tournequet	151
Blocage dans le tournequet	152
Politique par priorité	153
Politique à priorités dans Unix	154
Toutes les priorités dans Linux	156
Exemple de processus Unix avec top	158
Choix du quantum de temps dans Linux	160
Ordonnateur Linux 2.6	162
File d'exécution (<i>runqueues</i>)	163
Tableaux de priorité	165
Recalculer les quantas de temps	167
Calcul des priorités dynamiques et quanta	169
Sommeil & réveil	174
Préemption	177
Équilibrage de charge	178
Affinité tâche-processeur	179
Unix : 2 ordonnancement	180
Ordonnancement et va et vient	181
Politiques temps réel	182
Petite conclusion sur ordonnancement Linux	183
Des ennuis : inversion de priorité	184
Solutions possibles	186
Mars Pathfinder Mission on July 4th, 1997	187
Tâche martienne de priorité forte	188
Tâche martienne de priorité faible	189
Bug martien	190
Debug martien	191

● Plus de détail des tâches dans Linux

Le plan	120
Définitions usages de clone()	121
Tâches noyau	125
Exemple de liste de processus	126
Représentation des tâches en interne	127
Accès aux tâches	130
Graphe des états d'un processus	131
Un processus Unix dans tous ses états	132
États internes dans Linux	133

● Entrées-sorties

Le plan	136
Entrées-sorties physiques (bas niveau)	137
Déroulement d'une entrée-sortie physique	138

● Gestion du temps qui passe

Le plan	139
---------	-----



4 Gestion mémoire

Le plan	194
Gestion des mémoires	195
Hierarchie mémoire	196
Hierarchie mémoire versio réseau	197
Dépasser des limitations d'adressage	198
Dépasser la mémoire physique	201
Intérêts de la mémoire virtuelle	202
Grande mémoire virtuelle	204
Traduction des adresses des pages	206
Tailles de pages	207
Pagination à 3 niveaux sur SPARC	208
Défaut de page	209
Vision objet de la mémoire virtuelle	210
Mémoire virtuelle et localité	211
Limiter les défauts de pages	212
Accélérer la traduction d'adresse	213
Contenu d'un TLB	214
Choix des pages à enlever du TLB	215
Donner de l'espace aux processus	216

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

238

Disques

(Disques magnétiques)	239
Unix FFS optimisé pour les disques	240
Partitionnement des disques	241

Formattage

Utilitaire format	244
format à l'œuvre	245
Etape fdisk sur PC	248
Interface de système de fichier VFS	250
Montage/démontage d'un système de fichiers	251
Structure de système de fichier Unix	253
Structure de répertoire virtuel Unix	254
Transformer de la mémoire disque en fichiers	255
Transformer de la mémoire disque en fichiers Unix	256
Structure globale du FFS	258
Systèmes de fichiers sous Solaris	259
UFS journalisé	263
Vérification d'un système de fichiers	268

RAID

271

RAID	271
Redundant Array of Inexpensive Disks	272
Types de RAIDs	273

ZFS

275

ZFS de OpenSolaris Sun	275
Concepts de ZFS	276
Déploiement de ZFS	278

● Pilote de périphérique

Le plan	228
Contrôleur ou pilote de périphérique	279
Types de pilotes de périphériques	280
Quelques méthodes d'un pilote DDI/DKI	282



279 / 298

5 Virtualisation

Le plan	
Architectures virtuelles	
Machines virtuelles matérielles	
Intérêts de la virtualisation	
Machines virtuelles logicielles	
Machines virtuelles - fonctionnement	
Paravirtualisation	

6 Les entrées-sorties

Le plan	
---------	--

● Systèmes de fichiers

Le plan	229
Hiérarchie de classes de fichiers Unix simplifiée	230
Descripteur de fichier	231
Sémantique des fichiers Unix	233
Quelques méthodes associées à des fichiers	234
Passer un descripteur à ton voisin	237

7 Systèmes de fichiers distants

Le plan	286
Pouvoir délocaliser les fichiers	287
Propriétés importantes	288

Considérations de conception

289

Composants de NFS	295
NFS : sans état	296

NFS

Le plan	291
Network File System (NFS)	292
Buts de conception	294

8 Conclusion



298 / 298