Systèmes d'exploitation centralisés

1SN

2023 - 2024

Présentation du cours

Objectifs

Culture essentielle

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

Compétences essentielles

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

Page de l'enseignement :
`https://moodle-n7.inp-toulouse.fr/course/view.php?id=606`
Contact : jerome.ermont@toulouse-inp.fr

Sources et références

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

Calendrier

Semaine	Cours	TDs	TPs	Séances en autonomie	Projet	Autres Échéances
10 (4/3)	Introduction			Page shell : - présentation (pdf seul) - TP filtres		
11 (11/3)	Mécanismes de base			Page shell : - TP langage de scripts		
12 (18/3)	Processus	Processus			Début minishell	Rendus TP Shell
13 (25/3)		Signaux	Processus			Rendus TP Processus
14 (1/4)	Fichiers		Signaux			Rendus TP Signaux
17 (22/04)		Fichiers (E/S)	Fichiers (E/S)			Rendus TP Fichiers
18 (29/04)		Fichiers (Tubes)	Fichiers (Tubes)			Rendus TP Tubes
19 (6/5)	Mémoire	Contrôle des interactions			Fin minishell	Rendus final minishell
20 (13/5)	Mémoire et virtualisation		Fichiers (select)			
21 (20/05)		Mémoire virtuelle	Coupolage mémoire	Virutalisation		
22 (27/05)						Examen

- 30h encadrées (6 CM, 6 TD, 6 TP)
- projet individuel
- Examen final : 27/05

Première partie

Quelques jalons

Contenu de cette partie

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

Plan

1. Anatomie d'un système informatique
2. Traiter la complexité des systèmes informatiques
3. Fonctions d'un SX
4. Evolution des SX
   - Améliorer le taux d'utilisation des ressources
   - Améliorer le service rendu

TOULOUSE INP N7

Anatomie d'un système informatique (1)

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

Les systèmes informatiques sont complexes

Diversité des composants

- ordinateur/terminal : alimentation, processeur, chipset, carte mère, cache, bus, RAM, carte graphique, réseau...
- périphériques : écran, scanner, imprimante, DVD, disque dur, souris, enceintes, clavier...

Chaque composant a ses particularités

- fonctionnement différent
- technologies spécifiques

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

Parallélisme des composants

→ besoin de protocoles pour

- gérer les interactions entre composants
- coordonner l'action des composants pour réaliser un objectif commun

Pour le concepteur du système : imprévisibilité (en général)

- des usages effectifs du système
- des évolutions de l'environnement d'utilisation

→ besoin de flexibilité, et d'adaptabilité

Plan

1. Anatomie d'un système informatique
2. Traiter la complexité des systèmes informatiques
3. Fonctions d'un SX
4. Evolution des SX
   - Améliorer le taux d'utilisation des ressources
   - Améliorer le service rendu

TOULOUSE INP N7

Comment traiter la complexité des systèmes informatiques ?

→ Adapter le système aux capacités humaines

- limites cognitives
- efficacité faible

Deux stratégies classiques

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

TOULOUSE INP N7

Version informatique de ces stratégies : notion de module

Principe

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

  formel (textuel) → graphique (métaphore)

Exemples (dans le domaine informatique)

mail, ftp, éditeur graphique, bureau...

TOULOUSE INP N7

Exemple : anatomie d'un système informatique (2)

Abstraction des services par des modules

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

- portabilité
- maintenance
- standardisation

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

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

Combiner (composer) les modules (1/2)

Superposition (pelures d'oignon, poupées russes)

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

Exemple : langages de programmation

Générateur d'applications
Langage de haut niveau
Assembleur
Langage machine

Exemple : Système THE (Dijkstra, 1968)

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

Combiner (composer) les modules (2/2)

Fédération de services (construction de systèmes ouverts)
Le système informatique est conçu comme

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

Exemples

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

Plan

1. Anatomie d'un système informatique
2. Traiter la complexité des systèmes informatiques
3. Fonctions d'un SX
4. Evolution des SX
   - Améliorer le taux d'utilisation des ressources
   - Améliorer le service rendu

Rôle d'un SX

Interface des ressources matérielles pour les applications

(→ virtualisation)

Comment simplifier l'accès aux ressources matérielles ?

Virtualisation

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

Remarque : principe d'interposition

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

Outillage

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

Services système : un panorama

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

environnement de base

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

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

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

Interface utilisateur, ou interface de commande

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

Algorithme de principe de l'interpréteur

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

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

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

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

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

Exemple d'usage des interfaces (Unix)

But : recopier un fichier dans un autre

Interface programmatique (en C) :

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

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

Interface de commande :

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

```
cp fich1 fich2
```

Documentation en ligne : commande `man`

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

Plan

1. Anatomie d'un système informatique
2. Traiter la complexité des systèmes informatiques
3. Fonctions d'un SX
4. Evolution des SX
   - Améliorer le taux d'utilisation des ressources
   - Améliorer le service rendu

TOULOUSE INP N7

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

$$\left.\begin{array}{l}\text{Fonctions}\\ \text{Modèle}\end{array}\right\}\ \text{simples} \longrightarrow \left\{\begin{array}{l}\text{Fonctions évoluées}\ \left\{\begin{array}{l}\text{efficacité}\\ \text{confort d'utilisation}\end{array}\right.\\ \text{Modèle... aussi simple que possible}\end{array}\right.$$

Modèle initial ≈ ordinateur individuel

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

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

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

Traitements par lots *(temps différé, batch)*

Idée : déléguer la saisie des programmes et des commandes

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

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

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

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

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

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

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

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

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

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

Remarques

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

Conclusion sur la deuxième génération

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

TOULOUSE INP N7

Équilibrer l'utilisation des ressources (1/3)

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

Situation

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

→ un traitement ne peut utiliser *toutes* les ressources *en permanence*

→ mise en place d'activités concurrentes (système multiprogrammé)

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

Équilibrer l'utilisation des ressources (2/3)

Mener plusieurs traitements de front ⇒ partager les ressources

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

Exemple : *partage physique de la mémoire centrale*

Équilibrer l'utilisation des ressources (3/3)

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

Idée

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

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

Raffinement de l'idée : pagination

(va-et-vient sur des fragments d'image mémoire (pages))

Conclusion sur la troisième génération

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

Améliorer le service rendu : interfaces utilisateur

- Multiprogrammation « interactive » : temps partagé (MULTICS ; TSO — Time Sharing Option)
  - possibilité d'exécuter un interpréteur de commandes parallèlement aux applications

    → retour au contrôle interactif de l'utilisateur sur l'avancement de ses programmes
  - glissement du calculateur au processeur de données
- Amélioration des modèles et interfaces utilisateur
  - interfaces graphiques : Alto, MacOS...
  - langages de scripts, filtres : Unix (processus, fichiers, filtres)

Améliorer le service rendu : interconnexion et répartition des systèmes

Quatrième génération (après 1980) : Unix

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

Améliorer le service rendu : systèmes répartis à large échelle

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

Processeurs et réseaux

- de plus en plus performants
- de moins en moins coûteux

→ croissance très rapide

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

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

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

→ mobilité et variabilité des environnements d'exécution

→ reprise et extension du modèle des systèmes classiques :

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

→ informatique comme ressource consommable : fermes de calcul et de stockage accessibles à distance (Cloud) : traitements de masses de données, calcul intensif, facturés au volume.

Conclusion : éléments de prospective

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

  Ex: Multics → UNIX → Linux

  ou : MS-DOS → Windows → Linux

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

Bilan

Des programmes complexes qui ont un impact

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

Notions importantes (et/ou difficiles)

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

TOULOUSE INP N7

Deuxième partie

Mécanismes de mise en œuvre des SX

Le SX fournit une interface (abstraite) d'accès aux ressources matérielles pour un ensemble de traitements indépendants

⇒

- superviser l'exécution des applications en fonction de la disponibilité des ressources
- gérer le partage
- contrôler l'accès aux ressources
- gérer efficacement les ressources
  - parallélisation (découplage temporel)
  - localité (découplage spatial)

Contenu de cette partie

Mécanismes matériels, protocoles et schémas utilisés par le SX pour

- gérer des traitements concurrents
- protéger/contrôler l'accès aux ressources
  - processeur
  - mémoire
  - périphériques
- utiliser les ressources de manière plus efficace
  - parallélisation
  - localité

Annexe : mécanismes et services de transfert du contrôle

Contenu des parties suivantes

Algorithmes et politiques utilisés par le SX pour gérer les processus et les différentes ressources

Plan

5. Architecture d'un ordinateur : modèle, terminologie
6. Exécution d'un programme : notion de processus
7. Partage des ressources entre processus concurrents
   - mémoire
   - processeur
   - périphériques
8. Protection des ressources
   - cœur : processeur, mémoire
   - encapsulation : mode superviseur, amorce
9. Implantation efficace du SX
   - délégation des E/S
   - hiérarchie de mémoires

TOULOUSE INP N7

Architecture d'un ordinateur : modèle (simple), terminologie

Cycle processeur (UC)

```
répéter
  lire données (MC→UC)
  lire instruction(MC→UC)
  exécuter intruction(données)
  écrire résultat (UC→ MC)
sans fin
```

Plan

5. Architecture d'un ordinateur : modèle, terminologie
6. Exécution d'un programme : notion de processus
7. Partage des ressources entre processus concurrents
   - mémoire
   - processeur
   - périphériques
8. Protection des ressources
   - cœur : processeur, mémoire
   - encapsulation : mode superviseur, amorce
9. Implantation efficace du SX
   - délégation des E/S
   - hiérarchie de mémoires

TOULOUSE INP N7

Exécution d'un programme : notion de processus

processus (*activité*) $\triangleq$ *exécution* d'un programme par un processeur

- Analogie :
  - *livre* $\sim$ programme (statique) ;
  - (activité de) *lecture* d'un livre $\sim$ processus (dynamique)
- Un programme peut être exécuté par plusieurs processus en même temps, chaque processus travaillant sur ses propres données
  Exemple : traitement de texte

Point de vue du programmeur

Plan

5. Architecture d'un ordinateur : modèle, terminologie
6. Exécution d'un programme : notion de processus
7. Partage des ressources entre processus concurrents
   - mémoire
   - processeur
   - périphériques
8. Protection des ressources
   - cœur : processeur, mémoire
   - encapsulation : mode superviseur, amorce
9. Implantation efficace du SX
   - délégation des E/S
   - hiérarchie de mémoires

TOULOUSE INP N7

Mise en œuvre du partage de ressources entre activités concurrentes

Objectif

Répartir à tout instant les ressources entre les différents processus, de sorte à
- permettre l'exécution simultanée de plusieurs processus
- fournir à chaque processus les ressources nécessaires à son exécution

Structures de données

→ gérer l'état d'allocation des ressources aux différents processus

- par processus : ressources { attendues / obtenues } (*descripteurs de processus*)
- par ressource : processus utilisateurs (élus), processus en attente

Algorithmique

- Mécanismes matériels utiles à la réalisation du partage proprement dit
  - mémoire → partage physique
  - processeur → partage temporel
  - périphériques « autonomes » → gestion des interactions (échange des requêtes et des résultats)
- Politiques d'allocation : choix des processus élus, selon les besoins utilisateur (priorités, équité...), la nature du périphérique...
  → chapitres suivants

Mémoire centrale

- Partage physique
- Une partie de la mémoire espace système est attribuée au SX. Cet espace contient notamment les données utiles à la gestion
  - des processus (descripteurs de processus),
  - des ressources (requêtes en attente)
  - des échanges avec les périphériques (tampons d'E/S...)

Processeur

Partage physique (ex : multicœur) : *vrai parallélisme*

Partage temporel (*pseudoparallélisme*) (cas le plus fréquent)

Le processeur est alloué à tour de rôle à chacun des processus.

⇒ mécanisme pour

- interrompre le processus en cours et sauvegarder son état
- restaurer l'état du processus suivant, puis reprendre son exécution

La notion de processus permet d'abstraire la gestion physique des processeurs

Mécanisme pour le partage temporel du processeur : commutation de contexte

Contexte $\triangleq$ ensemble des informations à sauvegarder
pour pouvoir poursuivre un traitement interrompu entre 2 instructions

Etat d'un processus (en général)

- valeur des données utilisées
- code exécuté
- état (contexte) du processeur

Données et instructions restent souvent inchangées
→ il suffit de sauver l'*état du processeur* :

- registres généraux (données et instructions)
- mot d'état programme (MEP, ou PSW) :
  compteur ordinal (CO), code condition, masque d'IT, contexte mémoire accessible, mode (programme/superviseur)...

Commutation de contexte

sauvegarde du contexte courant
restauration du prochain contexte
} par une instruction indivisible

Interaction entre le SX et les périphériques

Problème

Informer le SX dès qu'un événement nouveau (fin de traitement, incident...) se produit sur l'un des périphériques.

Solutions

- Attente active (scrutation) :
  le SX teste en permanence l'état des différents périphériques
  → simple et fiable
  → coûteux et inadapté aux applications temps-réel
  → E/S sur systèmes simples/anciens
- Interruptions (mécanisme matériel)
  - le SX n'attend pas, mais poursuit son activité
  - un périphérique signale au SX le moment où l'événement survient
  - à la réception du signal, le SX interrompt le traitement en cours, pour traiter l'événement signalé
  - l'événement traité, le SX reprend le traitement interrompu
  → réponse « rapide » au « signal » dePi
  → plus complexe ; requiert un support matériel (ITs + commutation)

Prise en compte des interruptions : cycle de l'UC avec ITs

RI : registre instruction
RD : registres données
as: adresse sauvegarde CO
ai : adresse 1ère instruction à exécuter sur interruption
IP : booléen vrai ssi interruption présente
IA : booléen vrai ssi traitement interruption autorisée

```
répéter
    RI := Mem[CO];
    RD := charger(Mem[CO]);
    CO :=CO + 1;
    exécuter (RI);
     si (IP et IA) alors
     début -- commuter(traitement en cours,traitant IT)
          IP := IA := faux;
          Mem[as] :=CO;
          CO := ai;
      fin si;
fin répéter;
```

```
ai :  < traiter l'interruption >;
       -- commuter(traitant IT, traitement interrompu)
       IA:=vrai;
       CO:=Mem[as]
```

Les ITs permettent au SX de réagir aux périphériques à tout moment

Exemples

- calcul et E/S simultanés (systèmes de 3ème génération)
- intervention externe : utilisateur, chaînes de mesures
- délai de garde

Terminologie

- Niveaux d'interruption : causes d'interruption possibles, identifiées
  - Chaque niveau peut être traité séparément, par une routine (procédure) de traitement (traitant, handler)
  - Les adresses de ces routines sont souvent conservées dans une zone fixée de l'espace système : vecteur d'interuption
  - Priorités possibles entre niveaux
- Masque d'interruption : ensemble des niveaux pouvant provoquer une IT
- Niveau désarmé (vs armé) $\rightarrow$ l'IT est ignorée
- Niveau masqué (vs démasqué) $\rightarrow$ le traitement de l'IT est retardé

Que se passe t-il lorsqu'une IT est reçue
alors qu'une autre IT est déjà en cours de traitement ?

Selon le matériel, ou le niveau, la nouvelle IT peut

- être ignorée (IT désarmée)
- provoquer l'abandon du traitement d'IT en cours (IT perdue)
- être traitée selon une politique de niveaux de priorité
  - les niveaux inférieurs ou égaux à l'IT en cours sont masqués
  - l'arrivée d'une IT de niveau supérieur interrompt le traitement en cours qui sera repris ensuite : traitements en cascade (empilés)

Plan

5. Architecture d'un ordinateur : modèle, terminologie
6. Exécution d'un programme : notion de processus
7. Partage des ressources entre processus concurrents
   - mémoire
   - processeur
   - périphériques
8. Protection des ressources
   - cœur : processeur, mémoire
   - encapsulation : mode superviseur, amorce
9. Implantation efficace du SX
   - délégation des E/S
   - hiérarchie de mémoires

TOULOUSE INP N7

Contrôle du processeur : interruptions

Problème

Le SX doit pouvoir reprendre le processeur aux traitements en cours

- utilisation des interruptions d'E/S (2e génération)

- horloge (3e génération)
  périphérique générant des IT à des instants programmables
  → le SX peut reprendre la main à intervalles réguliers

Protection de la mémoire (1/2)

Problème

fixer les opérations permises (lire, écrire...) { pour chaque traitement / sur chaque mot mémoire

Manière directe (données)

- la mémoire est divisée en blocs
- à chaque bloc sont associés
  - un verrou physique fixant (et permettant de contrôler) les droits d'accès pour l'exécution en cours :
    - accès interdit (-),
    - accès en exécution (X),
    - accès en lecture (L),
    - accès en lecture et écriture (E)
  - un verrou logique, définissant un propriétaire pour le bloc, et ses droits d'accès

Protection de la mémoire (2/2)

Manière détournée (encapsulation) : via le mécanisme d'adressage

- les programmes sont implantés dans des zones contiguës et utilisent des adresses relatives (relogeables) commençant à 0
- utilisation de 2 registres

- base (RB) : registre de translation (utilisé par le SX)
  $\rightarrow$ évaluation de l'adresse réelle
- limite (RL) : taille de l'espace du processus lors de l'accès

Encapsulation des ressources (1/2)

Mode superviseur et instructions privilégiées

instructions mode programme
Calculs logiques, arithmétiques…

instructions privilégiées
Accès aux ressources : manipulation du MEP
halt
E/S…

Protocole d'accès aux ressources contrôlées par le SX

- accès aux ressources obligatoirement en mode superviseur
- changement de mode : programmme → superviseur
  possible seulement via une instruction d'appel superviseur :
  `trap(n)` (ou `SVC(n)`...)
  - commutation de contexte
    (avec la routine système identifiée par le paramètre (n))
    → passage en mode superviseur
  - exécution de la routine système, qui se termine par une
  - restauration (commutation) du contexte de l'appelant
    → retour au mode utilisateur

→ l'accès aux ressources passe nécessairement par les procédures système

Encapsulation des ressources (2/2)

Installer le SX dès le démarrage : amorce

Un (petit) programme permet d'installer en mémoire le SX, avant tout autre programme : chargeur initial (ou amorce, ou bootstrap).

- IBM 360 : une seule instruction (IPL (E/S)) + pupitre
- sur minis vers 1965 : saisir un petit programme aux clés
- sur ordinateurs actuels : programme en ROM

Chargement téléscopique

Le chargeur initial peut en charger un autre plus complet, etc...

ROM → bootblock → fichier boot → UNIX

- ROM : tests matériels, puis chargement du bloc d'amorce (bootblock) (ex : MBR)
- bloc d'amorce (≈ 1Ko) : programme (chargement) + table (adresse fichier d'amorce)
- fichier d'amorce (≈ 100 Ko) (exemples (Linux) : LILO, GRUB) : choix et chargement du noyau
- les tables sont modifiables (instruction *installboot*)

Plan

5. Architecture d'un ordinateur : modèle, terminologie
6. Exécution d'un programme : notion de processus
7. Partage des ressources entre processus concurrents
   - mémoire
   - processeur
   - périphériques
8. Protection des ressources
   - cœur : processeur, mémoire
   - encapsulation : mode superviseur, amorce
9. Implantation efficace du SX
   - délégation des E/S
   - hiérarchie de mémoires

TOULOUSE INP N7

Parallélisation de l'utilisation des ressources : délégation des E/S

Moyen : périphériques autonomes + IT

Délégation des E/S : principe

- les interactions avec chaque périphérique sont gérées par un composant du SX (pilote), spécifique à chaque périphérique
- l'interface d'un périphérique est un ensemble de registres accessibles
  - par un jeu d'instructions spécifique (peu flexible/portable)
  - ou comme des mots mémoires «ordinaires» (Memory mapped I/O)

Réduire le nombre des IT

→ ajout de contrôleurs intermédiaires chargés de « regrouper » les IT

Par rapport au schéma précédent, les contrôleurs gèrent :

- des opérations plus complexes
- vérification des données
- plusieurs périphériques similaires (mais un seul transfert à la fois)

Réduire encore le nb d'IT : périphériques à ADM et processeurs canaux

- Le SX prépare et place en mémoire centrale le programme canal, puis lance le canal
- Accès direct à la mémoire centrale (ADM) :
  les commandes canal échangent directement les données par blocs avec la mémoire centrale
- Le canal signale par une IT la fin de l'exécution de son programme

Canal : processeur pouvant exécuter des séquences d'E/S

Découplage des activités (1/2)

But : éviter que le SX ou le périphérique restent en attente d'un résultat ou d'une nouvelle requête

Découpler les requêtes

→ associer à chaque périphérique une file de demandes en attente

- nouvelle demande d'E/S → la demande est déposée dans la file (si le périphérique est déjà actif), le demandeur est mis en attente, et un nouveau processus actif est choisi.
- IT de fin d'E/S → le demandeur est réveillé, et une nouvelle E/S est choisie dans la file, puis lancée

Découplage des activités (2/2)

Découpler l'accès aux résultats/données

Même schéma (interaction producteurs-consommateurs) que pour les requêtes : les données en attente de traitement sont conservées dans des tampons FIFO

→ variations de vitesse (temporaires) entre « processeurs » absorbées : si leurs vitesses moyennes sont égales, le producteur et le consommateur pourront progresser indépendamment.

Découplage des données : hiérarchie de mémoires

Niveaux de mémoire

Idée : un processus n'utilise que peu de données à la fois
→ ranger les données prochainement utilisées en mémoire rapide

Heuristique : principe de localité

ce qui a été accédé récemment le sera prochainement
(le passé récent est une bonne image du futur proche)

Mécanismes matériels de transfert du contrôle

Transfert de contrôle $\triangleq$

- arrêt du traitement en cours C
- commutation de contexte entre C et un autre traitement A
- exécution/reprise de A

Variante structurée éventuelle (et fréquente) :
A se termine par un transfert de contrôle vers C $\rightarrow$ pile d'exécution

Transfert asynchrone indépendant du flot de contrôle défini par le programmeur (programmation événementielle)

- *Interruptions* (matérielles)

Transfert synchrone intégré (attente) au flot de contrôle du traitement

- *Appel superviseur*
- *Déroutement* : commutation (et traitement) causés par l'exécution du processus en cours (division par zéro, protection mémoire...)

Services logiciels de transfert du contrôle

- Commutation de contexte
  - API système : sauvegarde/restauration de contexte (exemple UNIX : setjmp, longjmp)
  - structures/bibliothèques des langages de programmation : coroutines (Modula 2), méthodes (périmées !) suspend/resume de la classe Java Thread
- Transfert asynchrone (interaction « asynchrone »)
  - API système : « interruptions » logicielles (ex : signaux UNIX)
  - langages de programmation : support à la programmation par événements (schéma publier/s'abonner) (Java : Swing, Beans)
- Transfert synchrone
  - appels systèmes, appel procédural
  - exceptions (Ada, Java, Modula 3 ...)

TOULOUSE INP N7

Troisième partie

Processus

Le SX fournit une interface d'accès aux ressources matérielles pour un ensemble de traitements indépendants (processus)

⇒ gérer la progression des processus suivant la disponibilité des ressources
≜ *ordonnancement*

Contenu de cette partie

Processus

- représentation selon différents points de vue : utilisateur, programmeur, SX
- interfaces système
- mise en œuvre par le SX
- politiques d'ordonnancement
- *aperçu : architecture d'une application parallèle*

Plan

10 Modèles de processus

11 Mise en œuvre de la gestion des processus

12 Ordonnancement des processus
- Définitions
- Ordonnancement à court terme

13 Conception d'applications parallèles

TOULOUSE INP N7

Notion de processus

processus $\triangleq$ activité d'exécution d'un programme par un processeur

Un processus n'est pas un programme

- Analogie :
  - *livre* $\sim$ programme (statique) ;
  - (activité de) *lecture* d'un livre $\sim$ processus (dynamique)
- 2 exécutions d'un même programme = 2 processus différents (chaque processus travaille sur ses propres données)
  Exemple : traitement de texte

Point de vue du programmeur

processus $\triangleq$ activité d'exécution d'un programme par un processeur

- Etat de la machine
  = état du processeur (registres) + état de la mémoire (données)
- Exécution d'une instruction $\leftrightarrow$ changement d'état de la machine
- Exécution d'un programme = exécution d'une suite d'instructions

$\rightarrow$ Processus = suite d'actions = suite d'états obtenus = trace

Point de vue SX

processus = utilisateur de ressources

Le SX doit gérer le partage des ressources, afin que chaque processus finisse par disposer des ressources nécessaires à son exécution.

→ pour le SX,

- l'activité d'un processus est abstraite (réduite) aux opérations d'allocation/restitution de ressources
- état d'un processus
  = état par rapport à l'allocation des différentes ressources

Modèle fourni par le SX

Contrôle des ressources utilisées par un processus : environnement d'exécution

- processeur et mémoire vive sont gérés entièrement par le SX
  → transparent pour l'utilisateur/le programmeur

- l'utilisation des ressources périphériques est souvent demandée explicitement, via une interface spécifique

  Remarque
  Unix propose une interface unifiée (flots (fichiers)) pour les échanges de données entre un processus et son environnement

- les variables d'environnement fournissent un moyen général et souple pour contrôler et échanger les informations et les données relatives aux ressources disponibles lors de l'exécution :
  - identifiant de l'utilisateur courant, de la machine,
  - protocole utilisé par le terminal,
  - chemins d'accès aux bibliothèques et exécutables...

Interface programmatique (API) de gestion des processus

Créer un processus

Windows

```
BOOL CreateProcess (
 LPCTSTR lpApplicationName,    // programme exécutable
 LPTSTR lpCommandLine,         // ...ou ligne de commande
 LPSECURITY_ATTRIBUTES lpProcessAttributes
 LPSECURITY_ATTRIBUTES lpThreadAttributes
 BOOL bInheritHandles,        // indicateurs d'héritage
 DWORD dwCreationFlags,       // priorité, nouvelle fenêtre...
 LPVOID lpEnvironment,        // → environnement
 LPCTSTR lpCurrentDirectory,
 LPSTARTUPINFO lpStartupInfo, // fenêtre, redirections
 LPPROCESS_INFORMATION lpProcessInformation // résultat
);
```

```
typedef struct _PROCESS_INFORMATION {
  HANDLE hProcess ;
  HANDLE hThread ;
  DWORD dwProcessId ;
  DWORD dwThreadId ;
} PROCESS_INFORMATION, * LPPROCESS_INFORMATION ;
```

API processus

Créer un processus

Unix

Décomposition en 2 primitives

- Création d'un processus fils : `fork()`
  - Hérite de l'environnement construit par le processus père
  - Exécute le même programme que le père
- Commutation de programme : `exec(...)`
  Le fils charge un nouveau programme à exécuter

Exemple

```
... /* code exécuté par le père (seul) */
if (fork()==0) {
     /* code exécuté par le fils */
     exec("prog_fils",...);
} else {
     /* code exécuté par le père */
}
```

API processus

Terminer un processus

- *Windows* : `VOID ExitProcess(UINT uExitCode) //code de retour`
- *Unix* : `VOID exit(int ret) // code de retour`

Comment le SX garantit-il la terminaison «propre» des processus ?

```
int main(int arc, *char argv[]) {
    ...
}
```

API processus

Réponse : interposition (enveloppe)

appel d'une procédure enveloppant le programme principal
et se terminant systématiquement par un appel à `exit(...)`

Autres opérations

- lister les informations de gestion d'un processus :
  ressources utilisées, identifiant, programme, utilisateur...
- suspendre/reprendre un processus :
  masquées dans d'autres primitives : `wait()`, `sleep()`, `read()`...

Plan

10 Modèles de processus

11 Mise en œuvre de la gestion des processus

12 Ordonnancement des processus
- Définitions
- Ordonnancement à court terme

13 Conception d'applications parallèles

TOULOUSE INP N7

Représentation des processus

Point de vue SX

processus = utilisateur de ressources
→ état d'un processus = état d'allocation des ressources

- pour chaque processus, les informations d'allocation des ressources (obtenues/demandées) sont conservées dans un descripteur de processus (ou PCB : Process Control Block)
- la table des processus regroupe les différents descripteurs de processus

Descripteur de processus

- identifiant du processus
- copie du contexte processeur (à la dernière commutation) :
  - mot d'état programme (CO,CC...),
  - registres (généraux,adresses piles et segments...)
- informations de gestion des ressources
  - UC : état du processus (prêt, bloqué...), priorité... mémoire : adresse de la table des pages (zones allouées, droits d'accès...)
  - E/S : périphériques accessibles (descripteurs de fichiers ouverts : droits d'accès, tampons...)
  - statistiques d'utilisation (utilisé pour les algorithmes d'allocation des ressources)
- liens vers les processus créateur (père)/créés(fils)

Le contexte du processus comprend le descripteur de processus et les structures qu'il référence (tables, piles...)

Exemple (Unix)

Descripteur de processus = Structure utilisateur + Structure de processus

Mise en œuvre des opérations sur les processus

Création

- Création du descripteur de processus initialisé
  - à partir des paramètres d'appel pour Windows
  - à partir du descripteur de processus père pour Unix
- Le processus est prêt ou suspendu

Destruction/Terminaison

- Libération des ressources utilisées par le processus
- Libération du descripteur de processus

Plan

10 Modèles de processus

11 Mise en œuvre de la gestion des processus

12 Ordonnancement des processus
- Définitions
- Ordonnancement à court terme

13 Conception d'applications parallèles

TOULOUSE INP N7

Ordonnancement des processus

But : gérer l'allocation des ressources aux processus

Situation

Certaines ressources (UC, imprimante...)
n'admettent qu'un nombre limité d'utilisateurs simultanés

→ une file est associée à chaque ressource, qui
contient les (descripteurs des) processus en attente de la ressource
Par la suite, pour être concis, « mettre un processus en attente » sera utilisé pour : « intégrer le descripteur d'un processus à une file d'attente ».

→ lorsque la ressource est disponible,
le choix du prochain processus auquel allouer la ressource dépend

- de la nature de la ressource
- de critères définissant la politique d'allocation appliquée
  - *priorités* : privilégier certains processus pour l'accès à la ressource
  - *équité* : garantir un accès à tout processus
  - *contraintes de temps* : garantir une borne sur le temps d'attente
  - ...

La vie des processus : terminologie (1/2)

Niveaux d'ordonnancement

Définis selon la fréquence des décisions d'allocation :

- ***à long terme*** (régulation) :
  choix des programmes à lancer
  (cf traitement par lots)
- ***à moyen terme*** (synchronisation) :
  choix des processus prêts
  (coordination, régulation par
  va-et-vient, attente d'E/S...)
- ***à court terme*** (exécution) :
  choix du processus actif
  (allocation du processeur (UC))

La vie des processus : terminologie (2/2)

Etats d'un processus (pour l'ordonnancement)

Allocation du processeur : politiques d'ordonnancement court terme

Caractéristiques des algorithmes

- priorité associée aux processus, ou non
  - pas de priorité
    → simple, service équitable (tout processus finit par être servi)
  - priorités
    - contrôle fin de l'allocation
    - risque de famine pour les processus non prioritaires
      remède : faire croître la priorité avec le temps passé (ancienneté)
- réquisition (préemption), ou non
  - pas de réquisition : simple, pas d'hypothèses sur le matériel
  - réquisition
    → possibilité de garantir des temps de réponse
    → nécessaire aux systèmes interactifs et/ou temps-réel

Remarque : pas d'algorithme convenable pour tous les critères. . .

Premier arrivé, premier servi (politique FIFO)

Principe : servir les requêtes par ordre chronologique d'arrivée

- simple
- sans priorités (équitable)
- sans réquisition
  - → forte variance du temps de service : le temps de service d'un processus dépend du comportement des processus qui le précèdent
    - → inadapté au temps partagé

Tourniquet (Round-Robin)

Principe

quantum $\triangleq$ durée maxi. d'activité continue pour le processus actif (élu)

- les processus prêts sont rangés dans une file
- le processeur est alloué au processus en tête de file
- le processeur est réquisitionné (préempté) au profit du processus suivant dans la file
  - soit en fin de quantum ;
  - soit sur blocage du processus actif (attente d'une synchronisation, d'une E/S...)

- $\approx$ FIFO, avec réquisition $\rightarrow$ adapté au temps partagé
- Paramètre essentiel : valeur du quantum (usuel : 10-100 ms)

Allocation à deux niveaux

Principe

- Les processus prêts sont divisés en catégories (processus interactifs, tâches de fond...)
- Chaque catégorie a sa politique d'allocation (FIFO, tourniquet...)
- Une politique d'allocation est définie entre catégories (FIFO, tourniquet, priorités...)

Exemples

- Files multiniveaux
  - gestion interne aux catégories : FIFO
  - ordonnancement entre catégories : priorités statiques
- FSS (Fair Share Scheduling)
  - Chaque catégorie reçoit un nombre de quantums
  - Le nombre de quantums alloués peut être ajusté dynamiquement
- Tourniquet multiniveaux
  - gestion interne aux catégories : tourniquet
  - ordonnancement entre catégories : priorités statiques
  - adaptatif : un processus peut changer de niveau selon qu'il épuise son quantum ou non

Ordonnancement temps-réel (1/2)

Objectif *(temps réel « dur »)* : garantir aux traitements le respect de contraintes de dates « précises » de terminaison (échéances)

Modèle : Un processus ordonnancé est vu comme exécutant une série de tranches de calcul successives (*tâches*)).
Les tâches exécutées par un processus sont caractérisées par

- leur date d'arrivée
- leur (pire) temps d'exécution
- leur échéance (date de fin d'exécution au plus tard)

→ *contrôle d'admission* : l'ordonnanceur peut rejeter une tâche a priori, si son exécution compromet le respect des contraintes de date

Cas courant : tâches périodiques

Les tâches exécutées par chaque processus arrivent à intervalles réguliers.
Elles peuvent alors être caractérisées par

- leur période p (ou leur fréquence : $1/p$)
- leur (pire) temps d'exécution t (avec $t \leq p$)

→ calculs d'ordonnancement simplifiés

Ordonnancement temps-réel (2/2)

Stratégies de base

Ingrédients courants : priorités + préemption

- *ordonnancement à taux monotone* (RMS : Rate Monotonic Scheduling)
  - tâches périodiques
  - priorité fixe : ordre inverse des périodes
  - simple, prévisible (contrôle d'admission, impact d'une surcharge)
- *échéance la plus proche* (EDF : Earliest Deadline First)
  - priorité dynamique : échéance la plus proche
  - optimal
  - complexe (priorités dynamiques) : calcul des priorités, contrôle d'admission, impact d'une surcharge

TOULOUSE INP N7

Plan

10 Modèles de processus

11 Mise en œuvre de la gestion des processus

12 Ordonnancement des processus
- Définitions
- Ordonnancement à court terme

13 Conception d'applications parallèles

TOULOUSE INP N7

Démarche de conception d'une application parallèle

1. définir des *activités élémentaires*, exécutées en parallèle
2. composer (*coordonner*) ces activités élémentaires

Exemple 1 : enregistrement d'un flux de données reçu à haut débit

La réception et l'enregistrement des données doivent être parallèles
→ 2 activités : RD (réception données) et ED (écriture données), qui doivent

- être aussi autonomes que possible
- échanger des données

→ utilisation d'un tampon en mémoire centrale

ED et RD doivent se coordonner pour accéder au tampon

- ED doit attendre RD si le tampon est vide
- RD doit attendre ED si le tampon est plein

⇒ primitives pour communiquer (tampons...) et synchroniser (attente...)

Exemple 2 : structuration « classique » de l'activité d'un SX

Structuration type

- un processus par activité utilisateur
- un processus (démon) par périphérique
- un ou plusieurs processus (démons) pour la gestion mémoire
- les services système peuvent être réalisés par des processus ou bien comme des procédures

Structure de base de l'activité d'un SX : démons (processus cycliques)

Caractère permanent $\rightarrow \begin{cases} \text{simple(conception)} \\ \text{efficacité de mise en œuvre} \end{cases}$

$\rightarrow$ utilisé dans les couches basses (SX)

Mécanismes d'interaction pour les processus

Communication

- Synchrone : lecture/écriture vers des flots/fichiers
  → prochain cours + TD (Unix)
- Couplée : tampons
  → tubes (pipes) + TD (Unix/Shell)
- Asynchrone :
  - communication par événements, schéma s'abonner/publier
    → service d'interruptions logicielles (signaux UNIX → TD)
  - E/S non bloquantes (→ TD Unix)
  - mémoire partagée (→ Cours + TD mémoire virtuelle)

Synchronisation (attente contrôlée par le programmeur)

- couplage induit par l'accès aux tubes
- schéma fork/join : possibilité pour un processus père d'attendre la terminaison d'un fils (→ TD)

Quatrième partie

Fichiers

Contenu de cette partie

Gestion des données conservées en mémoire secondaire

- Fichiers
  - point de vue utilisateur : interface, modèle
  - mise en œuvre : implantation, accès
- Organisation et désignation des fichiers
  - point de vue utilisateur : répertoires, liens
  - mise en œuvre
- Annexes : points techniques

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers
- Interface et modèle
- Mise en œuvre du système de gestion de fichiers

16 Organisation des fichiers
- Interface
- Mise en œuvre du service de gestion de répertoires

17 Annexes
- Le sous-système d'E/S d'UNIX BSD
- Système de fichiers standard Linux (FSSTND)
- Sûreté de fonctionnement
- Méthodes d'accès

TOULOUSE INP N7

Gestion des données rémanentes

Intérêt de la mémoire secondaire (disques, bandes...)

- Rémanence (les données peuvent être conservées)
- Capacité de stockage

Aide à l'utilisation des données stockées en mémoire secondaire :

- Abstraction
  notion de fichier : modèle simple d'accès aux données stockées
- Partage et protection des fichiers
  droits d'accès, contrôle de concurrence
- Organisation logique des fichiers
  service de gestion de répertoires (SGR)
- Autres services
  - Contrôles et services liés au « type », au contenu des fichiers (binaires, textes...)
  - Accélération des accès (index...)
  - Sauvegarde

Structure des services de gestion des données rémanentes

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers
- Interface et modèle
- Mise en œuvre du système de gestion de fichiers

16 Organisation des fichiers
- Interface
- Mise en œuvre du service de gestion de répertoires

17 Annexes
- Le sous-système d'E/S d'UNIX BSD
- Système de fichiers standard Linux (FSSTND)
- Sûreté de fonctionnement
- Méthodes d'accès

TOULOUSE INP N7

Gestion des échanges avec les périphériques

Abstraction des E/S

- périphériques blocs : disques...
  - données structurées (regroupées) dans des blocs (souvent de taille fixe)
  - chaque bloc est identifié par une adresse
  - l'accès aux données est déterminé par le numéro du bloc
- périphériques caractères : clavier. . .
  - données non structurées, vues comme des suites (flots) d'octets
  - l'accès aux données doit suivre l'ordre du flot

Caches

- *But* : optimiser les échanges en conservant dans un tampon (cache) en mémoire rapide une copie des données échangées
- *Difficulté* : gestion de la cohérence entre copie en cache et copie sur le périphérique

Notion de fichier

Abstraction pour un ensemble de données

- conservées indépendamment de l'exécution des applications (rémanence)
- ayant un lien logique (pour l'utilisateur)
- différenciées par un identifiant (clé) ou un indice (position)

$$\rightarrow \text{fichier} \triangleq \text{suite} \begin{cases} \text{d'octets} \\ \text{d'enregistrements} \end{cases}$$

Opérations

Abstraction : support, implantation, opérations de bas niveau

Métaphore : bande magnétique

- créer/détruire (Unix : `creat`.../`unlink`...)
- ouvrir/fermer (Unix : `open`/`close`)
- accès (lire, écrire, naviguer) (Unix : `read` ,`write`, `lseek`)

Utilisation des fichiers

Fichiers ordinaires

Sous Unix, le contenu d'un fichier est une suite d'octets, sans autre structure. L'interprétation de ce contenu dépend de l'utilisation :

Programmes exécutables

Commandes du système ou programmes créés par un usager

Exemple

`gcc -o prog prog.c`	produit le programme exécutable (fichier prog)
`./prog`	exécute le programme prog

Fichiers de données

- Documents, images, programmes sources, etc.
- *Convention* : il est pratique d'ajouter au nom un suffixe indiquant la nature du contenu (usage en Unix, nécessaire sous Windows)
  - *Exemples* : .c (programme C), .o (binaire translatable), .gif (format d'images), .ps (PostScript), .pdf...
  - *Remarque :* la commande Unix `file` fournit dans tous les cas une indication sur la nature du fichier.

Utilisation des fichiers

Autres fichiers

La plupart des ressources se présentent sous Unix comme des fichiers afin d'offrir une interface (un modèle) unique à l'utilisateur :

- tubes (pipes) : accès à un tampon mémoire
  vu comme une paire de fichiers (dépôt/retrait)
- répertoires
- liens (alias)
- fichiers spéciaux : les périphériques appraissent sous Unix comme des fichiers (souvent situés dans le répertoire `/dev`)
  - Les périphériques sont désignés par des noms de fichiers
  - Interface d'usage (commandes, primitives) : celle des fichiers
    - *open/close* : connexion/déconnexion au périphérique
    - *read* : lecture (si cela a un sens par rapport au périphérique)
    - *write* : écriture (si cela a un sens par rapport au périphérique
    - *ioctl* : action spécifique.
  - Les mécanismes de protection sont les mêmes.
  - À un fichier spécial correspond un pilote de périphérique

(Davantage de détails sont fournis en annexe)

Protection et partage des fichiers

La protection (ou sécurité) recouvre plusieurs aspects

- confidentialité :
  informations accessibles aux seuls usagers autorisés
- intégrité : pas de modifications non désirées
- contrôle d'accès :
  seuls certains usagers sont autorisés à faire certaines opérations
- authentification :
  garantir qu'un usager est bien celui qu'il prétend être

Note : quelques compléments sur la mise en œuvre de l'intégrité et du contrôle d'accès sont fournis en annexe.

Sécurité des fichiers (sous Unix) : droits d'accès

On définit

- des types d'opérations sur les fichiers : lire (`r`), écrire (`w`), exécuter (`x`)
- des classes d'usagers :
  usager propriétaire du fichier (`u`), groupe propriétaire (`g`), autres (`o`)

Fichiers ordinaires

Exemple (fichier `fich`) :

`rwx r-- r--` : tout accès pour le propriétaire, lecture seule pour tous les autres

`chmod go+w fich` : droit w donné au groupe et aux autres

`chmod o-w fich` : retire le droit w aux autres

Répertoires

Même chose, mais le droit x signifie « recherche dans le répertoire »

Mécanisme de délégation

- *Problème* : partager un programme dont l'exécution nécessite des droits d'accès que n'ont pas les utilisateurs du programme
- *Moyen* (`setuid` ou `setgid`) : accorder les droits du propriétaire à l'utilisateur (uniquement) pour la durée de l'exécution du programme

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers
- Interface et modèle
- Mise en œuvre du système de gestion de fichiers

16 Organisation des fichiers

17 Annexes

TOULOUSE INP N7

Système (logique) de fichiers

Implante

- un ensemble (non structuré) de fichiers
- vus comme des tableaux d'octets (sans structure interne)
- dans un espace physique vu comme un disque (logique) (tableau de blocs de taille fixe)

Structure type d'un disque logique

- blocs d'amorce
  UNIX : `bootblocks (BB)`
- informations sur la structure du disque : adresse, type des ≠ zones qui le constituent, localisation des blocs libres
  UNIX : `superbloc (SB)`
- catalogue du disque : liste des *descripteurs* des fichiers présents sur le disque logique
  UNIX : `table des I-nœuds`
- fichiers

Structure type d'une entrée de catalogue (i-nœud Unix)

- propriétaires(s)
- droits d'accès
- date(s) de dernier accès, de création...
- taille
- type
- référence(s) vers l'implantation du fichier sur disque
- données pour gérer la fiabilité : checksum, référence vers une copie...

Interface avec l'organisation physique : partitions (disques virtuels)

But : indépendance par rapport à la configuration physique

- un disque virtuel peut utiliser une portion d'un disque physique,
- ou un groupe de disques physiques
- chaque disque virtuel a la structure décrite précédemment

Terminologie

minidisks (VM), volumes (MS-DOS), partitions (MacOS), disques logiques

Exemple : Unix propose 2 types de disques logiques

- swap : utilisé pour la régulation et la gestion mémoire
- systèmes de fichiers : utilisé pour... le stockage des fichiers

Implantation des partitions : structure type d'un périphérique

- bloc d'amorce : code d'amorce + table des partitions
- table des partitions : taille et adresse des différentes partitions
- réserve de blocs (remplacement des blocs devenant véreux)

Interconnexion des disques logiques

≈ greffe d'arborescences contenues dans des disques logiques

- au niveau de la racine : Windows
- insertion d'une sous-arborescence quelconque en un point quelconque d'une autre arborescence (montage)

  Exemple : Unix (et NFS)
  - un système de fichiers (SF) est privilégié : le SF « système »
  - les autres SF peuvent s'y greffer (instruction `mount` (montage logique))
  - mécanisme utilisé par NFS pour la connexion de fichiers/répertoires distants

Représentation de l'espace libre (1/2)

But

garder une trace des blocs qui peuvent être (ré)alloués
→ « liste » des blocs libres

Vecteur de bits

carte de l'occupation du disque logique : pour chaque bloc du disque, un bit indique si le bloc est libre

- Avantages :

  simple et efficace

- Inconvénient :

  ... à condition que le vecteur tienne en mémoire centrale

→ utilisation

  - par les systèmes/micro-ordinateurs (MacOS)
  - lorsque la partition comprend relativement peu de blocs (groupes de cylindres BSD 4.x)

Représentation de l'espace libre (2/2)

Liste chaînée

carte de localisation de l'espace libre :

- L'espace libre est vu comme une suite de blocs.
- Chaque bloc libre contient l'adresse du suivant

Problème

inefficace (allouer n blocs → n E/S)

Amélioration 1

le premier bloc libre contient les adresses des n suivants (dont n-1 sont directement utilisables)

Amélioration 2

si n blocs libres contigus, stocker n et l'adresse du premier

Implantation des fichiers sur le disque

Allocation contiguë

Chaque fichier est rangé dans des blocs d'adresses successives

- L'entrée du catalogue pour le fichier contient l'adresse du premier bloc et la taille du fichier
- Stratégies de placement : First fit, Best fit, Worst fit

Avantages

- permet l'accès direct et l'accès séquentiel
- temps d'accès minimal, pour un fichier

Inconvénients

- fragmentation externe (blocs libres nombreux mais dispersés) $\rightarrow$ recompactage coûteux
- gestion lourde, lorsque la taille des fichiers augmente

Implantation des fichiers sur le disque

Allocation fragmentée

Pas de relation entre les adresses (la localisation) des blocs du fichier

Table d'allocation (FAT) (∼ carte de l'occupation du disque)

La table d'allocation est indexée par les numéros de blocs disque.

- *Idée* : regrouper toutes les adresses→ moins d'E/S
- Le catalogue contient le numéro du 1er bloc alloué à chaque fichier.
- Chaque entrée de la table d'allocation contient
  - le numéro du bloc suivant, si le bloc correspondant est alloué à un fichier
  - 0 si le bloc correspondant est libre
  - EOF si le bloc correspondant est le dernier du fichier
- Le chaînage des entrées exprime celui des blocs alloués

Implantation des fichiers sur le disque

Allocation chaînée

Les blocs alloués à un fichier sont chaînés

- Le catalogue contient l'adresse du premier bloc et celle du dernier bloc de chaque fichier
- Chaque bloc contient l'adresse de son successeur

Avantages

- pas de fragmentation externe
- pas de problème lié à l'évolution de la taille d'un fichier

Inconvénients

- accès direct très inefficace
- fiabilité : perdre un bloc ≈ perdre tout ce qui suit dans le fichier
  → chaînage double → surcroît de gestion

Implantation des fichiers sur le disque

Allocation indexée

Le premier bloc de chaque fichier est (le premier bloc de) la table des adresses des blocs alloués au fichier

Avantages

- Cf allocation chaînée
- Accès direct possible

Inconvénient : perte de place pour les petits fichiers

→ Idée : gérer un ensemble de blocs d'index (de taille réduite)

- plus compact
- mais temps d'accès accru

Méthodes

- Liste chaînée
- Arborescence (le bloc d'index référence des blocs d'index qui...)
- Méthode hybride (Unix) : l'entrée du catalogue pour un fichier (i-nœud) contient 15 adresses de blocs :
  - 12 adresses directes (48K adressables)
  - 3 adresses indirectes
    indirections simple (1 Mo adressable), double (1 Go), triple (1 To)

I-nœud UNIX

mode
(bloc/caractère)
propriétaires
(u,g)
dates
(création, accès,
modification)
taille
nb de blocs
adresses directes
(12)
indirection simple
indirection double
indirection triple
données
index

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers
- Interface et modèle
- Mise en œuvre du système de gestion de fichiers

16 Organisation des fichiers
- Interface
- Mise en œuvre du service de gestion de répertoires

17 Annexes
- Le sous-système d'E/S d'UNIX BSD
- Système de fichiers standard Linux (FSSTND)
- Sûreté de fonctionnement
- Méthodes d'accès

TOULOUSE INP N7

Interface utilisateur pour la manipulation des fichiers

Service fourni

- permettre une désignation « familière » (noms externes)
- structurer l'espace de noms/d'objets défini par l'utilisateur

→ notion de répertoire :
ensemble de fichiers nommés et regroupés au choix de l'utilisateur

Structuration minimale : répertoire « plat »

Simple liste de noms de fichiers

Difficultés

- accumulation des fichiers → allongement de la liste
- conflits entre noms

Structuration classique : arborescence

Principe : inclure des répertoires dans les répertoires

Organisation purement logique, choisie par l'utilisateur

Exemple

Intérêt

- organisation définie par l'utilisateur : fichiers regroupés par sujet/usage/utilisateur
- règle les conflits de noms (un fichier est désignable par le chemin (unique) le reliant à la racine)
- contrôle des droit d'accès

Modèle d'utilisation : navigation dans l'arborescence

- Nom unique : chemin absolu (chemin reliant la racine au fichier)
  Exemple (Unix) : `/utilisateurs/lulu/pg1`
- Raccourcis
  - chemins relatifs :
    utilisation d'un préfixe implicite (répertoire de travail/courant)
    - *Exemple* : si le répertoire courant est `/utilisateurs/lulu`, le fichier précédent peut être désigné par `pg1`
    - *Généralisation* :
      liste de préfixes (chemins) permettant l'emploi de noms locaux
      Exemple : variable `PATH` en Unix
    - Opérations sur le répertoire courant
      - initialisation : répertoire privé (ou répertoire de connexion)
      - affectation : `cd`
      - affichage : `pwd`
  - abréviations spécifiques à une application
    Exemple (shell Unix) : `.`, `..`, `~xxx`
  - synonymes : liens

Structure de graphe : liens logiques

But

désigner (sans dupliquer) un même fichier depuis plusieurs répertoires

- le fichier/le répertoire apparait comme une entrée « ordinaire » dans chacun des répertoires
- structure de graphe

Exemple

Non unicité des désignations/des chemins
→ navigation moins naturelle (cycles, prédécesseurs multiples. . .)

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers

16 Organisation des fichiers
- Interface
- Mise en œuvre du service de gestion de répertoires

17 Annexes

Mise en œuvre des répertoires

répertoire = table de correspondance :
nom externe (utilisateur) ↔ nom interne (SX)

pour Unix, un répertoire est juste un fichier contenant cette table

Mise en œuvre des liens (Unix)

- liens physiques (nom interne) (commande `ln f ./f_bis`) : ajout d'une entrée `f_bis` au répertoire courant, associée au même i-nœud que `f`
- liens symboliques (chemin d'accès) (`ln -s f ./f_bis`) : le chemin du fichier `f` est stocké dans le fichier `f_bis` du répertoire courant
  → problème de mise en cohérence du contenu des fichiers de référence, en cas d'évolution

Exemple (sous UNIX, dans l'arborescence précédente)

création d'un lien symbolique vers binaires, sous le nom `programmes`, depuis le répertoire `lulu`.

```
>pwd
/utilisateurs/lulu
>ls
essai pg1
>ln -s /binaires programmes
>ls
essai pg1 programmes
# un fichier "/utilisateurs/lulu/programmes" a été créé,
dont le contenu est "/binaires"
>cd programmes
>ls
count find more sort
```

Contrôle d'existence

Problème

maintenir la cohérence des liens vers un objet,
si celui-ci est supprimé/déplacé

- Politique minimale : évaluation paresseuse
  lien vers un objet disparu = référence erronée
  Exemple : liens symboliques Unix
- Politique classique : conserver l'objet tant qu'il est référencé
  - Conserver le nombre de références (liens) vers chaque objet
    Exemple : liens physiques Unix
    → un fichier se peut être supprimé que si son compteur est nul
    Problème : test invalide en cas de cycles dans les références
  - Ramasse-miettes (périodique)
    - marquer les objets accessibles depuis la racine
    - supprimer les objets non marqués
  - Datation : supprimer les objets non accédés depuis longtemps

Interprétation des chemins d'accès par le service de répertoires

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers
- Interface et modèle
- Mise en œuvre du système de gestion de fichiers

16 Organisation des fichiers
- Interface
- Mise en œuvre du service de gestion de répertoires

17 Annexes
- Le sous-système d'E/S d'UNIX BSD
- Système de fichiers standard Linux (FSSTND)
- Sûreté de fonctionnement
- Méthodes d'accès

TOULOUSE INP N7

Le sous-système d'E/S d'UNIX BSD

But : proposer une interface homogène,
pour masquer les particularités des divers périphériques

→ Unix présente les périphériques

- à un niveau abstrait comme des `fichiers séquentiels` avec lesquels s'échangent des flots.
  - leur interface recouvre celle des fichiers
  - prise en compte des particularités
    → opération supplémentaire : `ioctl()`
- à un niveau fin, via une table de points d'entrée qui normalise les opérations réalisées par chaque pilote de périphérique

Identification d'un périphérique

- classe (bloc/caractère)
- numéro majeur : identifiant du pilote (type du périphérique)
- numéro mineur : identifie un périphérique parmi ceux de son type

Interface pilote/noyau

- routines d'initialisation
- routines de traitement des interruptions
- points d'entrée (opérations d'E/S)
  2 tables, en fonction de la classe :
  - `bdevsw` : périphériques mode bloc
  - `cdevsw` : périphériques mode caractère

  Chaque entrée de la table correspond à un pilote (numéro majeur) et contient une série de pointeurs (points d'entrée) vers les routines de service du périphérique (accès, contrôle...)

Communication avec le système

tampons (blocs/caractères)

Structure et interface

<table>
<tr><td colspan="7">Interface logicielle</td></tr>
<tr><td rowspan="2">sockets</td><td rowspan="2">fichier simple</td><td rowspan="4">interface corrigée</td><td rowspan="4">interface brute</td><td rowspan="4">interface données brutes</td><td>interface données corrigées</td><td>mémoire virtuelle</td></tr>
<tr><td rowspan="2">gestion du terminal</td><td rowspan="2">gestion de l'espace de swap</td></tr>
<tr><td>protocoles</td><td>SGF</td></tr>
<tr><td>interface réseau</td><td colspan="2">pilotes de périphériques mode bloc</td><td colspan="3">pilotes de périphériques mode caractère</td></tr>
<tr><td colspan="7">Interface matérielle</td></tr>
</table>

Périphériques bloc

- disques, bandes... (taille courante d'un bloc : 1Ko-8Ko)
- utilisent une antémémoire (un cache) : « tampons de blocs »

Périphériques caractère

- tous les périphériques (sauf réseau) n'utilisant pas le tampon de blocs
  - `/dev/mem`, `/dev/null`
  - mémoire virtuelle
  - interface « brute » pour périphériques blocs (ex : `/dev/rdsk`)
- échanges par (flots d') octets
- gestion des tampons par le pilote lui-même (*C-listes*)

Système de fichiers standard Linux (FSSTND) (`man hier`)

Cette arborescence définit une organisation type pour les fichiers sous Linux qui se retrouve, avec quelques variantes, sur la plupart des systèmes Unix. Principaux répertoires de cette arborescence :

- **/bin** → programmes et commandes utilisateur de base : cat, cp, mv, rm...
- **/boot** → copie des données utiles au démarrage (blocs d'amorce...)
- **/etc** → fichiers de configuration locaux, comme passwd, group, ou utilisés parTCP/IP, comme services, inetd.conf, exports
  - **/etc/skel** → répertoire type d'un nouvel utilisateur
  - **/etc/rc.d** → scripts appelés par init au démarrage du système
  - **/etc/X11** → fichiers de config. X11 (Xconfig, Xmodmap, xinitrc...)
- **/conf** → fichiers de configuration. Si ce répertoire est présent, les fichiers de /etc sont des liens vers les fichiers de /conf
- **/dev** → pilotes de périphériques : console (console système), mouse, hda ((1er) disque IDE), ...,lp0 ((1ère) imprimante), ..., null (poubelle (sans retour)), sda (1er) disque SCSI)), sda1 (1ère partition de sda), sda2 (2e partition), ..., sdb ((2e) disque SCSI), ..., tty (terminal virtuel)...
- **/home** → répertoires des utilisateurs. Souvent sur une partition distincte.
- **/install** → informations sur les programmes installés
- **/lib** → code des bibliothèques partagées
- **/mnt** → montage de disquettes, de systèmes de fichiers via NFS...

- **/sbin** → commandes de base de démarrage et d'administration système : fdisk, fsck, getty, init, update, ifconfig, ping, lilo...
- **/tmp** → espace de manœuvre par tous les utilisateurs
- /var → fichiers variant souvent : traces, journaux, spoule (/var/spool/), verrous (/var/lock/)
- **/usr** → principaux programmes non nécessaires durant le démarrage. L'idée est de permettre de partager ce répertoire entre machines.
- **/usr/bin** → commandes utilisateur et programmes système. La séparation entre /usr/bin,/bin, et /sbin n'est pas toujours claire : il est préférable que ces répertoires contiennent des liens réciproques
- **/usr/bin/X11** → programmes du système de fenêtrage X
- **/usr/doc** → documentations qui ne sont pas au format man
- **/usr/include** → fichiers d'include des bibliothèques C
- **/usr/lib** → librairies statiques de divers langages, talons de liaison avec les bibliothèques partagées, et divers fichiers de configuration
- **/usr/lib/X11** → données X11 (polices, tables de couleur...)
- **/usr/man** → pages du manuel en ligne
- **/usr/src** → codes sources des programmes système
- **/usr/local** → installation de programmes supplémentaires. Comprend souvent des répertoires bin, etc, include, sda, et man
- **/usr/TeX** → installation du logiciel d'édition TeX

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers

16 Organisation des fichiers

17 Annexes
- Le sous-système d'E/S d'UNIX BSD
- Système de fichiers standard Linux (FSSTND)
- Sûreté de fonctionnement
- Méthodes d'accès

TOULOUSE INP N7

Sûreté de fonctionnement

But : garantir l'intégrité des données conservées

- fiabilité (résistance aux pannes)
- sécurité(contrôles d'accès, intégrité, authentification, confidentialité)

Fiabilité

Moyen : redondance

Sauvegarde périodique

- complète/de reprise
- différentielle/par incréments :
  limitée aux fichiers modifiés depuis la dernière sauvegarde

Principe : combiner plusieurs fréquences de sauvegarde

Exemple

fréquence	nature	conservation
quotidienne	incrémentale	semaine
hebdomadaire	complète	mois
mensuelle	complète	an

Fiabilité : redondance interne

Principe : fournir plusieurs moyens d'établir une information

Exemples

- dupliquer les descripteurs
- stocker l'identifiant du fichier dans chacun des blocs du fichier
- chaînage double des blocs d'un fichier

Remarques

- Il est souvent préférable que les copies soient « éloignées »
- Un principe général est de combiner
  - une information centralisée (efficace) (ex : table d'allocation)
  - et une information « dispersée » (robuste)
    (ex : informations au niveau des blocs)
- La redondance peut être exploitée
  - statiquement (en cas de défaillance)
  - dynamiquement (calculs d'accès en double)

TOULOUSE INP N7

Sécurité

Moyens

- cryptage + clés (session, fichiers, répertoires, opérations)
- listes d'accès
  matrice : opérations permises, par utilisateur, et par fichier
  - raccourcis : droits par défaut, groupes d'utilisateurs
  - opérations élémentaires (lire/écrire/exécuter...)
- capacités
  - clé autorisant l'accès, attribuée en fonction des identificateurs de l'objet/de l'utilisateur
  - structure d'une capacité

droits d'accès	identificateurs	bits de contrôle

  - pour être robuste, le contrôle doit être fait à chaque accès (et non à l'ouverture/au premier accès seulement)

Mise en œuvre des listes d'accès sous UNIX : ACLs (Access Control Lists)

Documentation complémentaire : `https://www.usenix.org/legacy/publications/library/proceedings/usenix03/tech/freenix03/full_papers/gruenbacher/gruenbacher_html/main.html`

Généralisation du mécanisme de droits d'accès Unix (`ugo/rwx`)

Objectif

Mise en place de politiques de sécurité plus fines :

- Possibilité d'allouer des droits distincts au sein d'un même groupe
- Allocation de droits sur une base individuelle, pour chaque fichier

Interface

- ACL = suite d'Access Control Entries (ACE)
- Forme d'une ACE : entrée : [uid/gid] : permissions
- Deux commandes :
  - Lecture : getfacl *fichier*...
  - Mise-à-jour : setfacl -x|-m|-s [ACL] *fichier*...
- Politique de sécurité : seuls le propriétaire et l'usager root peuvent modifier l'ACL d'un fichier

ACLs : ACE

Permissions classiques

Propriétaire :	user : :rwx
Groupe du propriétaire	group : :rwx
Autres	other : :rwx

Permissions étendues

Usager nommé	user : nom :rwx
Groupe nommé	group : nom : rwx
Masque	mask : :rwx

Rôle du masque : limiter les permissions des usagers/groupes nommés

Héritage d'ACL entre répertoires

default : <déclaration d'ACE>

Sémantique

- Un répertoire hérite de la liste *default* à la fois en tant que liste d'ACL effective et liste *default* pour lui-même ;
- La liste *default* inhibe le filtre *umask*.

ACLs : exemple

```
$ umask 027
$ mkdir dir
$ ls -dl dir
drwxr-x-- ... agruen suse ... dir

$ getfacl dir
# file: dir
# owner: agruen
# group: suse
user::rwx
group::r-x
other::--

$ setfacl -m user:joe:rwx dir
$ getfacl -omit-header dir
user::rwx
user:joe:rwx
group::r-x
mask::rwx
other::--

$ ls -dl dir
drwxrwx--+ ... agruen suse ... dir
```

Plan

14 Gestion des données rémanentes

15 Gestion des fichiers

16 Organisation des fichiers

17 Annexes
- Le sous-système d'E/S d'UNIX BSD
- Système de fichiers standard Linux (FSSTND)
- Sûreté de fonctionnement
- Méthodes d'accès

TOULOUSE INP N7

Méthodes d'accès

Objectif

organiser un ensemble de données pour améliorer l'efficacité des accès, en fonction des patrons (motifs) d'accès aux éléments de cet ensemble

- Parcours de l'ensemble complet → accès séquentiel
- Accès « ponctuel »
  - à des enregistrements identifiés → accès indexé
  - par le contenu → accès associatif (listes inverses)

Accès séquentiel

Métaphore : bande magnétique

Structure de données ≈ file

- Les enregistrements sont vus comme une suite
- Un curseur (ou index) désigne l'enregistrement accessible

- Opérations de base :
  - ouverture/revenir_au_début : curseur sur position 1
  - lecture/passer_au_suivant
    - renvoie la valeur de l'enregistrement désigné par le curseur
    - l'index progresse d'une position
  - ajout/suppression (écriture)
    - normalement en fin de fichier
    - certaines organisations offrent la possibilité de fixer la position du curseur et d'écrire à partir de celle-ci (flots Unix)

Accès direct

But : retrouver un enregistrement à partir (d'une partie) de son contenu

Clés ordonnées : accès indexé

Clé $\triangleq$ partie de l'enregistrement qui permet de l'identifier de manière unique

$\rightarrow$ accès à l'enregistrement via une table : < clé, adresse >

Accélérer la recherche d'une clé $\rightarrow$ structuration arborescente

arbres équilibrés (B-arbres) : toutes les branches restent de hauteur voisine, quelle que soit l'évolution de l'arbre

Faciliter les accès séquentiels : séquentiel indexé

- structuration arborescente
- niveau logique « réduit » (adresses physiques)
- un bloc contient des enregistrements dont les clés se suivent (par rapport aux valeurs existant dans le fichier)
- nombre de niveaux réduit/fixe → zones (blocs) de débordement

Adressage dispersé (hachage, «hash-code»)

But : éviter la gestion/le stockage d'une table ordonnée <clé,adresse>

→ déterminer l'adresse à partir de la clé :
adresse = f(clé) (f : fonction de dispersion (de hachage))

Idéal

Pour un fichier de N enregistrements,
bijection entre les valeurs de clés présentes et 1,2...,N

Réalité : collisions (f non injective) → f renvoie vers une (tête de) liste

Remarque : modulo N est une fonction facile à mettre en œuvre

Accès associatif : listes inverses

Généralisation de l'accès indexé : permettre de retrouver les enregistrements ayant une valeur donnée, pour un champ donné

Multiliste

Pour chacun des champs pour lequel un accès par la valeur est souhaité

- ajouter un champ « pointeur » à chaque enregistrement
  $\rightarrow$ une liste regroupe les enregistrements de même valeur
- ajouter une table (index «secondaire»)
  $<$ valeur, pointeur vers la tête de liste associée à la valeur $>$

Fichier inversé

Pour chacun des champs pour lequel un accès par la valeur est souhaité

- un index regroupe, pour chaque valeur,
  la liste des clés (ou des adresses) des enregistrements dont le champ possède cette valeur
- l'enregistrement ne contient pas de valeur,
  mais un pointeur vers l'entrée de l'index contenant la valeur

Cinquième partie

Mémoire

Contenu de cette partie

Gestion des images mémoire des processus

- Gestion de la mémoire physique
- Mémoire virtuelle
  - principe
  - amélioration des performances
  - mémoire virtuelle comme medium de communication
    - partage de mémoire entre processus
    - couplage de fichiers en mémoire virtuelle
- Conclusion : synthèses
  - hiérarchie de mémoires
  - exécution des programmes en mémoire
- Exemples : Unix BSD, Solaris, Linux, Windows NT

TOULOUSE INP N7

Motivation : image mémoire d'un processus (Unix)

Pour le programmeur : image mémoire = zone mémoire (RAM)

- privée
- contiguë
- commençant à l'adresse 0
- bornée par la taille des adresses

Objectif SX

Faire coexister plusieurs images mémoire dans une RAM

- partagée
- limitée en capacité

Plan

18 Gestion de la mémoire physique
- Allocation contiguë
- Allocation fragmentée

19 Mémoire virtuelle
- Pagination et va-et vient
- Mise en œuvre d'un espace virtuel de grande taille
- Gestion de la mémoire virtuelle
- Utiliser la mémoire virtuelle pour échanger des données

20 Synthèses
- Hiérarchies de mémoires
- Étapes de production d'un programme exécutable

21 Exemples
- Unix 4.3 BSD
- Solaris
- Windows NT

TOULOUSE INP N7

Partage physique de la mémoire entre processus

Allocation de blocs de mémoire contiguë

Solution directe

allouer à chaque processus un bloc (distinct) de mémoire, de taille suffisante pour contenir son image mémoire

Partitions fixes *(OS 360 MFT)*

- La mémoire est divisée en zones de taille fixe (*partitions*).
- Une file d'attente est associée à chaque partition
- Quand un processus est lancé, le SX lui alloue la plus petite partition pouvant contenir son image mémoire (*best fit*)

Avantage

Permet un va et vient efficace (coexistence de processus prêts)

Problème : fragmentation interne

En général un processus utilisera seulement une partie de sa mémoire allouée

Eviter la fragmentation interne

→ ***Partitions variables***

- Allouer à chaque processus une partition de la taille exacte de son image mémoire
- En fin d'exécution, l'espace occupé est libéré

Politiques d'allocation de l'espace libre

Exemple : en ❷, P5 arrive, de taille 60K

- Plus grande zone disponible (*Worst-fit*)
  → implanter P5 à l'adresse 100K
- Plus petite zone possible
  («meilleur ajustement» ou *Best-fit*)
  → implanter P5 à l'adresse 940K
- Première zone possible (*First-fit*)
  → implanter P5 à l'adresse 100K

Partitions variables

Difficulté : *fragmentation externe* (émiettement)

L'espace libre est suffisant, mais divisé en nombreux fragments de petite taille, inutilisables séparément (*miettes* (*garbage*))

Exemple : en ❸, que faire si P5 arrive, de taille 150K ?

→ *(Re)compactage* : regrouper les zones libres en un seul bloc

⇒ recopie, réimplantation...

Bilan (provisoire)

Difficultés

- zones de taille fixe : manque de souplesse
  (tailles prédéfinies → fragmentation interne)
- zones de taille variable : fragmentation externe
  - recompactage : simple, mais coûteux en temps
  - traitement algorithmique :
    gérer les zones libres (recherche ; fusion de zones libres voisines)
    - lourd/complexe en général
      - table d'occupation de la mémoire (recherche lente, fusion ok)
      - listes de zones libres, classées par tailles (fusion lente)
    - possible lorsque l'ensemble des tailles demandées est régulier (puissances de 2) ou fixe (évite l'émiettement)
      Exemples : VM 370 (IBM) ; structures de données noyau Solaris

Intérêt

- d'éviter la fragmentation externe (zones fixes)
- de disposer d'un mécanisme d'allocation non contiguë...

Plan

18 Gestion de la mémoire physique
- Allocation contiguë
- Allocation fragmentée

19 Mémoire virtuelle

20 Synthèses

21 Exemples

Allocation fragmentée

Pagination (Allocation fragmentée de zones fixes)

Idée : fractionner l'image mémoire d'un processus
⇒ gérer une carte d'implantation de l'image mémoire (*table des pages*)

- L'image mémoire (virtuelle) est découpée en *pages*
- La mémoire physique est découpée en *cases*
- Cases et pages ont une taille P fixée et identique

Evaluation de l'adresse virtuelle (Av)

A_v= (n° de page,déplacement dans la page) = (p,d),
avec p = A_v div P et d = A_v mod P (efficace avec P= 2^k)

Exemple

Avantages

- pas de fragmentation externe
- fragmentation interne : 1/2 page par processus, en moyenne
  $\rightarrow$ ok si la taille des pages est assez petite (souvent 2K-8K)
- (vu plus loin) interprétation des adresses (adressage indirect)
  $\rightarrow$ possibilité de
  - déplacer une page (réimplantation dynamique)
  - partager une page
  - contrôler l'accès à une page

Mise en œuvre efficace des tables des pages

Difficulté

En général, une table des pages par (image mémoire de) processus, référencée dans le descripteur du processus

Difficulté

Table des pages en RAM ⇒ temps d'accès mémoire doublé
→ mécanismes accélérateurs nécessaires

Idée naïve

conserver la table des pages en mémoire rapide (tableau de registres)

La taille d'une table des pages est importante
(adresses de 32 bits → espace de 4Gio, pages de 4Kio → 10^6 pages)
→ coût prohibitif

- matériel
- commutation de contexte

Mise en œuvre efficace des tables des pages (suite)

TLB (translation lookaside buffer)

→ mécanisme matériel spécifique :

registre associatif (TLB : translation lookaside buffer)

- contient les derniers accès (page → case) (LRU) : *principe de localité*
- recherche en parallèle
- en cas d'échec dans la recherche associative
  - On consulte la table complète (en mémoire)
  - On met le registre associatif à jour, avec la nouvelle référence
- LRU est une très bonne heuristique (mesures sur des applications types)

  16 entrées → 80% de succès, 32 (80486) → 98%, 512 → 99%, 4096 → 99,99%

Segmentation (Allocation fragmentée de zones variables)

Mémoire vue comme un ensemble de zones (*segments*)

- de longueurs variables
- sans ordre relatif

Idée

Un segment est destiné à contenir des objets de même « type »

→ possibilité de contrôles et protection adaptés au type :
Pour chaque segment

- contrôle des accès
  opérations possibles limitées aux opérations spécifiques au type des objets du segment
- contrôle de l'adressage
  Ex : si le segment contient un tableau, on conserve sa dimension

Mise en œuvre de la segmentation

Analogue à la pagination :

- Adresse segmentée : (id. segment, dépl. dans le segment)
- Mécanisme

- Mise en œuvre de la table des segments

  cf table des pages : table en mémoire, registres associatifs

Différence : fragmentation externe

le recompactage est facilité (indirections), mais reste coûteux.

Segmentation paginée (Multics, 680x0)

- conserver l'abstraction offerte par la segmentation,
- . . . sans la fragmentation externe

Idée : paginer les segments

- une table des segments pour chaque processus
- une table des pages pour chaque segment

680x0 →

- 16 tables, pour 16 processsus (maximum)
- Chaque table de segments contient 64 entrées
- Chaque table de pages contient 16 entrées

Plan

18 Gestion de la mémoire physique
- Allocation contiguë
- Allocation fragmentée

19 Mémoire virtuelle
- Pagination et va-et vient
- Mise en œuvre d'un espace virtuel de grande taille
- Gestion de la mémoire virtuelle
- Utiliser la mémoire virtuelle pour échanger des données

20 Synthèses
- Hiérarchies de mémoires
- Étapes de production d'un programme exécutable

21 Exemples
- Unix 4.3 BSD
- Solaris
- Windows NT

TOULOUSE INP N7

Mémoire virtuelle

Problème : implanter un espace virtuel plus grand que l'espace réel

Solution directe : recouvrements (« overlays »)

Principe : indiquer (pragma) les procédures mutuellement indépendantes d'un programme (pas d'appel, ni d'imbrication)

Exemple : initialisation et reste du programme

→ de telles procédures peuvent être implantées à la même adresse (*se recouvrir*)

→ l'espace mémoire nécessaire au programme est réduit d'autant

Remarques

- A tout moment, un seul chemin de l'arbre est présent en mémoire
- Inconvénients : gestion explicite, ad hoc → limité

Va-et-vient (multiplexage temporel de la mémoire)

Une copie de l'image mémoire de chaque processus est conservée en mémoire secondaire

Pour changer l'occupant de la mémoire centrale :

1. Sauvegarde de l'image mémoire de l'élu courant
2. Restauration de l'image mémoire du nouvel élu

Intégration du va-et-vient et de la pagination

Idée : appliquer aux pages le principe du va-et-vient

→ *indicateur de présence* en mémoire pour chaque entrée de la table des pages

Principe de fonctionnement

Remarques

- L'emploi de zones fixes simplifie le placement : cases équivalentes
- Fonction topographique = table des pages

Mécanisme de défaut de page (1/3)

Chargement

- ajout d'une colonne (*invalide*) à la table des pages, indiquant pour chaque page si elle est chargée en mémoire
- ajout d'une *table de couplage* <n° page, n° bloc> (colonne *bloc*), qui localise la copie de chaque page en mémoire secondaire (taille blocs (disque) = taille pages)

Mécanisme de défaut de page (2/3)

Remplacement

Va-et-vient « page à page » s'il n'y a plus de cases libres

Remarque : le nombre d'E/S causées par un défaut de page double

Mécanisme de défaut de page (3/3)

Optimisation : mémoire associative (Translation Lookaside Buffer–TLB)

- *invalide* : bit de présence (faux si page en mémoire)
- *modifié* (“dirty bit”) : indique si une écriture a été effectuée sur la page (utilisé pour le remplacement)
- *mode d'accès* : opérations autorisées / la page (utilisé pour la protection)

Plan

18 Gestion de la mémoire physique

19 Mémoire virtuelle
- Pagination et va-et vient
- Mise en œuvre d'un espace virtuel de grande taille
- Gestion de la mémoire virtuelle
- Utiliser la mémoire virtuelle pour échanger des données

20 Synthèses

21 Exemples

TOULOUSE INP N7

Mise en œuvre d'un grand espace virtuel (1/3)

Difficulté : taille potentielle de la table des pages

Solution1 : liste inverse de pages (liste de cases)

Liste par case : < id de processus, n° de page >

- liste des cases réduite → accès séquentiel
- sinon, accès calculé (fonction de dispersion)
- usage courant d'une mémoire associative

Mise en œuvre d'un grand espace virtuel (2/3)

Solution 2 : pagination hiérarchique (hyperpages)

Idée : paginer la table des pages

- la table des pages est découpée en *hyperpages*
- la table des hyperpages (THP) permet de ne désigner que les parties de la table des pages effectivement utilisées
  → un compteur "nombre de pages désignées"
  est associé à chaque entrée de la THP
- adresse virtuelle = <n° hyperpage, n° page, déplacement>

Efficacité

- Mémoire associative nécessaire
- La table des hyperpages doit toujours rester en mémoire

Mise en œuvre d'un grand espace virtuel(3/3)

Pagination à 2 niveaux (S. Krakowiak, *principes des systèmes d'exploitation* (Dunod), p362))

Ce principe peut être réitéré → hiérarchie d'hyperpages (Linux : 3 niveaux prévus)

Plan

18 Gestion de la mémoire physique

19 Mémoire virtuelle
- Pagination et va-et vient
- Mise en œuvre d'un espace virtuel de grande taille
- Gestion de la mémoire virtuelle
- Utiliser la mémoire virtuelle pour échanger des données

20 Synthèses

21 Exemples

TOULOUSE INP N7

Gestion de la mémoire virtuelle

Motivation : coût des défauts de page

Temps d'accès effectif à la mémoire

$$\boxed{\text{TAE} = (1\text{-p}) \times \text{TAM} + \text{p} \times \text{TDP}}$$

avec :

- p : probabilité d'un défaut de page ;
- TAM : temps d'accès mémoire
- TDP : temps d'accès, en cas de défaut de page (gestion de l'IT + va et vient)

Ordres de grandeur

- TAM $\approx$ 100 ns
- TDP $\approx$ 1 ms + 2*8ms $\approx$ 17 ms

$\rightarrow$ TAE $\approx 100 + 17\ 10^6\ p$

Pour avoir une dégradation des performances $\leq$ 10% de TAM,
on doit avoir : $100 + 17\ 10^6\ p \leq 110$
soit : $p \leq 1\ /\ 1{,}7\ 10^6$

Comment améliorer les performances de la mémoire virtuelle ?

Peaufiner le mécanisme

- éviter les lectures/écritures disque superflues
  - *bit d'utilisation* (*dirty bit*) : indique si la page victime a été modifiée (auquel cas elle doit effectivement être recopiée)
  - pool de *pages libres et sales* (utilisées auparavant)
    → mémoriser le numéro des pages libérées
    → (localité) peut éviter un va-et-vient lors d'un défaut de page
- anticiper
  - pool de pages libres : évite d'attendre la sauvegarde de la victime
  - préchargement de pages : souvent possible et efficace (localité/ensemble de travail)
  - restitution volontaire de page : ok si automatique (compilateur)

Choisir la bonne victime

→ algorithmes de remplacement

Ecarter les processus causant trop de défauts de pages

→ régulation basée sur un modèle de comportement des processus

Choix des pages victimes : algorithmes de remplacement

Évaluation du comportement d'un processus en fonction de la mémoire disponible

- c(m,Ω) : *nombre de chargements de pages* causés par Ω
  - Ω : suite de références (numéros de pages virtuelles demandées par le processus),
  - m : nombre de pages mémoire disponibles pour le processus

- *taux de défaut de pages* F= c(m,Ω)/|Ω|
  avec |Ω| : longueur de la suite de références

Algorithmes de remplacement

Algorithme aléatoire

Victime = page choisie aléatoirement

FIFO

Victime = première page chargée (parmi les pages présentes)

Remarque : l'allure de $c(m,\Omega)$ peut être contre-intuitive pour FIFO

Exemple (anomalie de Belady) :

- gestion FIFO des requêtes de remplacement
- Pour $\Omega = 1,2,3,4,1,2,5,1,2,3,4,5$ on a : $c(3,\Omega) < c(4,\Omega)$

Algorithme optimal

Victime = page dont la prochaine utilisation est la plus tardive

$\Rightarrow$ évaluer a priori la chaîne des références

$\rightarrow$ irréalisable en général

Approximation de la solution optimale

Ordre chronologique d'accès (LRU , Least Recently Used)

Heuristique : localité temporelle

Le passé récent est une bonne image du futur proche

→ base pour évaluer le futur

→ LRU : victime = page inutilisée depuis le plus longtemps

Remarques

- LRU ne présente pas l'anomalie de Belady
- Mise en œuvre directe coûteuse (activée à chaque référence)
  - gérer une *liste* doublement chaînée.
    Fonctionnement
    - à chaque nouvelle référence, placer la page référencée en tête
    - victime = queue
  - numéroter les références successives
    → associer un compteur (*date logique*) à chaque page
    Difficultés : débordement du compteur, arithmétique, tri

Mises en œuvre approchées de LRU

utilisent un indicateur (bit) de référence associé à chaque page
(0 initialement, mis à 1 lors de l'accès)

Représentation « économique » des dates logiques

Date logique = historique des bits de référence sur une période de temps

- le bit de gauche est le bit de référence courant
- le mot est décalé vers la droite, à intervalles réguliers
- la plus petite valeur correspond à l'accès le moins récent

0	0	1	1	
0	1	0	0	
0	1	1	0	
0	0	0	1	
0	0	0	0	

Algorithme de la deuxième chance

- Index parcourant la table des pages de manière circulaire
- L'entrée pointée de la table des page a un bit de référence valant
  - 1 → bit remis à 0 ; l'index progresse
  - 0 → c'est la victime

Conclusion

Évaluation des différents algorithmes

Point saillant : existence d'un seuil M_0 à partir duquel F croît très vite

- avant ce seuil, les performances s'améliorent nettement avec m
- après ce seuil, augmenter m est peu rentable

Moralité

L'algorithme compte moins que la taille de la mémoire allouée

Stratégies de régulation

Modèle de comportement d'un programme : ensemble de travail [Denning]

Essentiel : fournir un espace suffisant à chaque processus

Espace suffisant déterminé à partir de 2 heuristiques de localité

- *temporelle* : sur une (courte) période d'observation, l'ensemble des pages utilisées est stable
- *spaciale* : les références se concentrent sur peu de pages
  valeurs usuelles : 75% des accès portent sur 20% des adresses

→ notion d'*ensemble de travail* ("*working set*") :
pages référencées par un processus, sur une période de temps

T_i : transition entre deux ensembles de travail
WS : ensemble de travail

Illustration expérimentale [Hatfield 1972 – IBM Syst. Journal]

Illustration empirique : écroulement du système ("trashing")

Phénomène observé

Explication

Dès qu'un processus a moins de mémoire que son ensemble de travail
→ il engendre des défauts de pages nombreux et fréquents
- → s'il n'y a plus de cases libres, retrait de pages à d'autres processus qui, à leur tour, ne disposent plus de leur ensemble de travail. . .
  - → boule de neige

Algorithmes basés sur le modèle de comportement

Idée

Prévenir l'écroulement en limitant le nombre de processus prêts (actifs)

Algorithme de l'ensemble de travail (WS)

- l'ensemble de travail de chaque processus est mémorisé
- en cas de défaut de page
  - on remplace une page qui n'est dans aucun ensemble de travail
  - s'il n'y en a pas, on réquisitionne les pages détenues par le processus le moins prioritaire (qui est suspendu)

  → tout processus actif dispose des pages de son ensemble de travail

Exemple : VMS (VAX)

relevé périodique des bits de référence pour estimer l'ensemble de travail

Algorithmes basés sur le modèle de comportement (suite)

Algorithme du taux de défaut de page par processus (PFF, page fault frequency)

Idée

Utiliser le taux de défauts de page pour détecter si un processus dispose de son espace de travail

Exemple : VM 370

Remarques

- PFF est plus aisé à réaliser que WS
- Les capacités mémoire actuelles permettent (et ont pour objectif) de réduire très fortement le va-et-vient, et donc l'écroulement

Plan

18 Gestion de la mémoire physique

19 Mémoire virtuelle
- Pagination et va-et vient
- Mise en œuvre d'un espace virtuel de grande taille
- Gestion de la mémoire virtuelle
- Utiliser la mémoire virtuelle pour échanger des données

20 Synthèses

21 Exemples

TOULOUSE INP N7

Partage de pages entre processus

Principe

Référencer une même case mémoire à partir de pages distinctes de processus distincts

Protection

- contrôle de l'espace d'adressage
  - indicateur spécifique (bit d'« invalidité »)
  - valeur limite associée à la table des pages
- indicateurs pour chaque page, précisant le mode d'accès (lecture, écriture, ajout, exécution...)

Transfert efficace de valeurs : copie sur écriture *(c.o.w.)*

But

Fournir à un processus (B) une copie des données d'un autre processus (A)

Contrainte : les copies doivent être indépendantes (locales à (A) et à (B))

Observation

inutile de dupliquer les données si elles ne sont pas modifiées

→ partager la page des données d'origine en lecture,
et ne dupliquer qu'à la première écriture

Exemple : mécanisme UNIX de création de processus

duplication sur écriture de l'image mémoire du processus père

Intégration (*couplage*) des fichiers dans l'espace virtuel

Idée : au chargement, les fichiers binaires de code exécutable sont liés (via la table de couplage) à l'image mémoire des processus.
[i.e. les blocs du fichier sont associés à des pages de l'image mémoire]

→ utiliser le même mécanisme pour coupler les fichiers de données

- plus d'E/S : contenu du fichier = variables en mémoire
- plus de tampons → moins de recopies → transferts plus efficaces

Intégration (*couplage*) des fichiers dans l'espace virtuel (suite)

Couplage (*mapping*) du fichier en mémoire virtuelle

- fixer 1 adresse de base B en mémoire virtuelle pour le contenu du fichier
- associer le contenu du fichier aux adresses successives à partir de B (coupler les blocs du fichier aux pages successives à partir de B)

Remarques

- couplage $\equiv$ l'ouverture
- accès au fichier $\equiv$ accès à l'image mémoire
- transferts réalisés via le mécanisme de défaut de page
- possibilité de partager un fichier (cf TD)

Systèmes proposant ce mécanisme

Multics (1965) (réalisation systématique des E/S par couplage des fichiers)
Appolo Domain, OS/2, Windows NT, Mach, Linux.

Plan

18 Gestion de la mémoire physique
- Allocation contiguë
- Allocation fragmentée

19 Mémoire virtuelle
- Pagination et va-et vient
- Mise en œuvre d'un espace virtuel de grande taille
- Gestion de la mémoire virtuelle
- Utiliser la mémoire virtuelle pour échanger des données

20 Synthèses
- Hiérarchies de mémoires
- Étapes de production d'un programme exécutable

21 Exemples
- Unix 4.3 BSD
- Solaris
- Windows NT

TOULOUSE INP N7

Hiérarchie de mémoires

Caractéristiques de la pagination avec défaut de page

Deux niveaux de mémoire

- un niveau rapide, mais coûteux et restreint (haut)
- un niveau lent, mais économique et de grande capacité (bas)

Les données sont

- conservées sur le disque (capacité = 1000 fois celle de la RAM)
- traitées en mémoire centrale (RAM)

Idéal : amener dans le niveau haut les informations dont la fréquence / la probabilité d'utilisation sont les plus grandes

→ mémoire

- comparable au niveau haut pour les temps d'accès
- comparable au niveau bas pour les capacités

Comment ? *(rappel)*

heuristiques de localité

- spatiale (ensemble de travail) et
- temporelle (LRU)

On dit que le niveau haut (mémoire centrale) joue un rôle d'*antémémoire* (*cache*) pour le niveau bas (mémoire secondaire).

Les idées développées dans le cadre de la pagination sont utilisées pour la hiérarchie des différentes mémoires existantes

Fonctionnement d'une mémoire cache(niveau matériel)

- une ligne du cache contient, outre les données, un indicateur de validité des données, et une étiquette (*tag*) identifiant l'adresse mémoire centrale à laquelle correspondent les données du cache.
- toutes les informations à lire passent d'abord dans le cache
- écriture dans le cache avec recopie vers la mémoire centrale
- la mémoire cache est découpée en *blocs* (p. ex. 64 octets)
- le cache gère les N dernières références dans une « pile » LRU

Les colonnes du cache

(cache à correspondace prédéfinie)

Pour réduire le nombre de comparateurs nécessaires,

- La mémoire centrale est divisée en *colonnes* correspondant à une partition de la mémoire, espérée équiprobable.
- Chaque colonne est divisée en *rangées*.
  Une rangée d'une colonne a la taille d'un bloc.
- De même, le cache est divisé en colonnes, puis en rangées.

Chaque colonne du cache joue le rôle de cache
pour une colonne de la mémoire.

Les colonnes du cache (suite)

Chaque colonne du cache joue le rôle de cache
pour une colonne de la mémoire.

Analyse d'une adresse

b: numéro de rangée dans la colonne k (ici 512 blocs)
k : numéro de colonne (32 colonnes)
d : déplacement dans le bloc (64 octets)

Exemple : cache de 8 Kio

- 8 Ki / (64*32) = 4 blocs par colonne dans le cache,
- donc 4 comparateurs (au lieu de 128) (128=8 Ki/64)

Si ce cache correspond à une mémoire de 1 Mio,
celle-ci comportera 512 rangées par colonne.

Plan

18 Gestion de la mémoire physique

19 Mémoire virtuelle

20 Synthèses
- Hiérarchies de mémoires
- Étapes de production d'un programme exécutable

21 Exemples

TOULOUSE INP N7

Etapes de l'implantation d'un programme en mémoire

- chargement souvent dynamique → gain de place mémoire : les procédures sont stockées sur disque au format binaire relogeable, et chargées en mémoire à leur premier appel
- L'édition de liens peut être dynamique (bibliothèques)
  - partage et mise à jour automatique du code
  - les procédures à lier dynamiquement sont représentées par un talon qui gère la liaison et se remplace par la procédure effective au premier appel.

Mise en œuvre d'un maillon de la chaîne d'implantation

- *substitution*/inclusion : remplacer chaque occurence de l'objet abstrait (symbole) par sa correspondance concrète (adresse)

  - simple, statique, efficace
  - une *table de traduction*, indiquant la position des occurrences de chacun des objets substitués dans le code d'origine doit être conservée, si l'on veut pouvoir refaire l'étape
- *chaînage*/indirection : remplacer chaque occurence de l'objet abstrait par l'adresse d'une référence vers l'objet concret.
  - plus souple, mais moins efficace

Format du binaire relogeable [1]

1. Source : S. Krakowiak

Edition de liens [2]

2. Source : S. Krakowiak

Chaîne d'implantation d'un programme en mémoire : résumé

avec gcc [3]

3. Source : S. Krakowiak

Exemple : image mémoire d'un processus Unix

Interface du service de gestion mémoire :

(niveau haut) → `malloc/free (stdlib.h)`

(niveau bas) → manipulation de `brk` :

```
#include <unistd.h>
int brk (const void *ptr)
```
fixe la valeur de `brk`

```
void * sbrk (int increment)
```
- ajoute `increment`
- renvoie la valeur précédente de `brk`

Plan

18 Gestion de la mémoire physique
- Allocation contiguë
- Allocation fragmentée

19 Mémoire virtuelle
- Pagination et va-et vient
- Mise en œuvre d'un espace virtuel de grande taille
- Gestion de la mémoire virtuelle
- Utiliser la mémoire virtuelle pour échanger des données

20 Synthèses
- Hiérarchies de mémoires
- Étapes de production d'un programme exécutable

21 Exemples
- Unix 4.3 BSD
- Solaris
- Windows NT

TOULOUSE INP N7

Gestion mémoire sous UNIX 4.x BSD

Va et vient

- le va et vient est appliqué, mais dans des cas peu fréquents (prévention de l'écroulement, processus inactifs) : les processus sont suspendus par le processus scheduler
  - lorsque la charge est élevée
  - et que la mémoire libre est en dessous d'un seuil minimum (lotsfree $\approx$ 1/64mémoire totale), durant trop longtemps
- le scheduler suspend alors (et répète jusqu'à avoir plus de lotsfree mémoire libre)
  - le processus inactif depuis le plus longtemps, parmi ceux inactifs depuis plus de 20s
  - le plus gros processus inactif depuis le plus longtemps
- la gestion « courante » de la mémoire est basée sur la pagination

Gestion mémoire sous UNIX 4.x BSD (suite)

Pagination

- Pagination à 2 niveaux
- taille des pages indépendante du matériel ($\geq$ 512 octets)
- pages verrouillables (non éjectables) (attente E/S, va-et-vient)
- les pages d'un processus sont généralement préchargées (et mises dans la liste des pages libres, étant marquées récupérables)
- une *table des cases* est gérée (liste inverse)
- et parcourue circulairement, s'il faut libérer de la place (proc. *pagedaemon*)
  - les cases déjà libres ou verrouillées ne sont pas touchées
  - sinon, si le processus auquel la case est allouée a suffisamment d'espace mémoire
    - la page est éventuellement sauvée sur disque (bit « modifié »)
    - la case est marquée comme libre mais récupérable
    - la page correspondante est marquée comme invalide
- pagedaemon est réveillé
  - lorsque le processus scheduler en a besoin
  - quand le nombre de cases libres est au dessous d'un seuil (lotsfree) (1/64 mémoire totale)
- pagedaemon utilise moins de 10% du temps processeur.

Solaris

Gestion de la mémoire virtuelle : segmentation paginée

Solaris (suite)

Gestion de la mémoire noyau : allocation de plages (slabs)

Principe

- une *plage* (slab) est une suite de cases contiguës
- un *cache* est composé d'une ou plusieurs plages
- chaque structure de données (SdD) du noyau (descripteurs de processus, de fichiers...) est associée à un cache, qui contiendra les instances (objets) de cette SdD
- lorsqu'un cache est créé, il est peuplé d'un nombre déterminé d'objets marqués comme *libres*
- les objets d'un cache sont alloués et marqués *utilisés* au fur et à mesure des besoins

- une demande d'allocation est servie en cherchant d'abord une plage partiellement remplie, puis une plage vide
- si toutes les plages sont pleines, une nouvelle plage est créée et associée au cache

Avantages

- pas de fragmentation : ensemble de tailles prédéterminé (structures du noyau), extension dynamique des plages, allocation aux plages par pages de taille unique
- efficacité : allocation (objets libres pré-créés) et libération (indicateur libre)

Windows NT

Structure

- Mémoire à deux niveaux : segmentation et pagination
- Adresses virtuelles sur 32 bits ⇒ 4G octets adressables
- Découpage logique :
  - Intelx86 : 1024 segments de 1024 pages de 4K
  - Digital Alpha : 256 segments de 2048 pages de 8K

Page directory = table des segments, implantée à une adresse fixe

Règles de couplage

- Une zone interdite : 0x0 à 0xFFFF
  (64Ko de début, référencée par les pointeurs "nuls")
- Zone programme applicatif : 0x10000 à 0x7FFEFFFF (2G - 128K)
- Une zone interdite : 0x7FFF0000 à 0x7FFFFFFF (64Ko)
- Zone noyau : 0x80000000 à 0xFFFFFFFF (2G)

Programmation du couplage en 2 étapes

- réservation/libération d'un espace d'adressage (région) ;
- couplage/découplage proprement dit d'un contenu.

Primitives

- *VirtualAlloc/VirtualFree* : réservation/libération
  (et (dé)couplage de pages)
- *VirtualLock/VirtualUnlock* : (dé)verrouillage en mémoire
- *CreateFileMapping/MapViewOfFile/UnmapViewOfFile* :
  définition/couplage/découplage de fichiers

Couplage de fichiers sous Windows NT : primitives

Réservation/libération d'une région et (dé)couplage de pages

```
LPVOID VirtualAlloc (
   LPVOID lpAddress, // adresse de base
   DWORD dwSize, // taille en octets
   DWORD flAllocationType, // type d'opération
   DWORD flProtect, // attributs de protection
);
LPVOID VirtualFree (
   LPVOID lpAddress, // adresse de base
   DWORD dwSize, // taille en octets
   DWORD dwFreeType, // type d'opération
);
```

avec :

```
flAllocationType = {MEM_RESERVE, MEM_COMMIT, MEM_RESET}
dwFreeType={MEM_RELEASE, MEM_DECOMMIT}
```

Couplage de fichiers sous Windows NT : primitives

Couplage de contenu

Etapes

1. Créer ou ouvrir le fichier : *CreateFile, OpenFile* ;
2. Définir le couplage du fichier : *CreateFileMapping* ;
3. Associer une région au fichier : *MapViewOfFile*.

```
HANDLE CreateFileMapping (
   HANDLE hFile, // le fichier
   LPSECURITY_ATTRIBUTE lpFileMappingAttributes,
   DWORD flProtect, // attributs de protection
   DWORD dwMaximumSizeHigh, // forts poids
   DWORD dwMaximumSizeLow, // faibles poids
   LPCTSTR lpName
);
```

avec :

`flProtect = {PAGE_READONLY,PAGE_READWRITE,PAGE_WRITECOPY}`

☞ `PAGE_WRITECOPY` : voir `FILE_MAP_COPY`

Couplage de fichiers sous Windows NT : primitives

Réserver/Libérer une région en mémoire virtuelle

```
LPVOID MapViewOfFile (
   HANDLE hFileMappingObject, // l'objet couplage
   DWORD dwDesiredAccess,
   DWORD dwFileOffsetHigh, // forts poids
   DWORD dwFileOffsetLow, // faibles poids
   DWORD dwNumberOfBytesToMap
);
BOOL UnmapViewOfFile(
   LPCVOID lpBaseAddress
);
```

avec :

```
dwDesiredAccess =
    {FILE_MAP_WRITE,FILE_MAP_READ,FILE_MAP_COPY}
```

☞ FILE_MAP_COPY : fichier original non modifié

Sixième partie

Virtualisation

Plan

22 L'idée

23 Les problèmes à résoudre

24 Les approches

L'idée

La vision habituelle

L'idée

Plusieurs machines sur une seule !

Idée fondamentale d'objet virtuel

- Origine :VM/CMS (Virtual Machine/Cambridge Monitor System), IBM (1967)
- Remise au goût du jour ;
- Plusieurs implantations possibles ;
- Mécanismes matériels facilitant la mise en œuvre.

Objectifs

- Cohabitation de plusieurs systèmes différents ;
- Isolation : l'arrêt de l'un n'arrête pas les autres
  - → facilite l'administration : cohabitation entre serveurs en exploitation et serveurs en déploiement, test ...
- Personnalisation :chaque application/utilisateur dispose d'un environnement (ressources, système) adapté à ses besoins

L'idée

La vision des précurseurs

Types de solutions

Objectifs

- transparence vis à vis des applications
  $\rightarrow$ maintien des performances
- limiter les modifications à apporter sur les systèmes invités

Types de solutions

- La virtualisation par simulation complète avec plusieurs variantes :
  - Emulation de chaque instruction ;
  - Compilation de blocs d'instructions à la volée.
- La paravirtualisation ;
- La virtualisation à hyperviseur ;

Plan

22 L'idée

23 Les problèmes à résoudre

24 Les approches

Les problèmes à résoudre

Principe de base

Notion de machine virtuelle

Un système d'exploitation « invité » ne s'exécute plus sur la machine réelle mais sur une machine « virtuelle ».

☞ Un logiciel de virtualisation est donc nécessaire.

Virtualisation des ressources allouées à une machine

- Partage du temps processeur entre machines virtuelles ;
- Une unité de disque virtuel : réalisé par un « gros » fichier ;
- Une carte graphique de base : réalisée par une fenêtre ;

Plan

22 L'idée

23 Les problèmes à résoudre

24 Les approches

La virtualisation par simulation complète

Avantages et inconvénients

- Avantage : pas de modification du SE « invité »
- Inconvénient : performances faibles : il faut tout émuler.

Exemple

Projet Bochs

Site Web : http ://bochs.sourceforge.net/

- Emulateur de la famille de PC à base de IA-32(x86) ;
- Développé en C++ ;
- Emulation du processeur, d'un BIOS et des périphériques de base (console, disque, ...) ;
- Supporte plusieurs SE : Linux, DOS, Windows NT/XP,...
- Problème : performances faibles mais bon support de test.

La vision actuelle : par simulation (émulation)

Matériel émulé, mais cœur (RAM, UC) accessible aux invités

Exemple de configuration PC émulée par Parallels Desktop

- Un processeur Intel Pentium.
- Jusqu'à 1500 MB de mémoire.
- une carte graphique VGA.
- Jusqu'à 4 disques d'interface IDE (de 20 à 128 GB représenté par un fichie dont un de démarrage.
- Jusqu'à 5 interfaces réseaux , dont une carte virtuelle Ethernet.
- 4 ports séries (COM) ports.
- 3 ports parallèles (LPT)
- un controleur USB 2.0 controller.
- une carte son.
- un clavier PC générique.
- une souris PS/2.

La vision actuelle : par hyperviseur

La vision actuelle : par hyperviseur

- l'hyperviseur est un noyau léger, multiplexant les accès au matériel des systèmes invités.
- les systèmes invités doivent être modifiés pour utiliser l'API de l'hyperviseur
- les appels systèmes invités s'appuient sur les pilotes d'un système « privilégié »
- exemples : Xen (Citrix), KVM, vSphere (VMWare), Hyper-V (MS)

Des produits commerciaux

Exemple

Famille VMware

Site Web : http ://www.vmware.com/fr/

- Famille de logiciels de virtualisation.
- Versions serveurs ;
- Approche par hyperviseur ;
- Limite : processeur Intel (Pas d'émulation)