Systèmes avancés

Mémoire virtuelle

Grégoire Pichon

Lyon 1 / Laboratoire de l'Informatique du Parallélisme

MIF18 - M1. 2022/2023

Ce cours est librement inspiré du cours de B. Goglin, avec des extraits du cours de LIFASR5 (L. Gonnord et N. Louvet).

Plan

Contexte

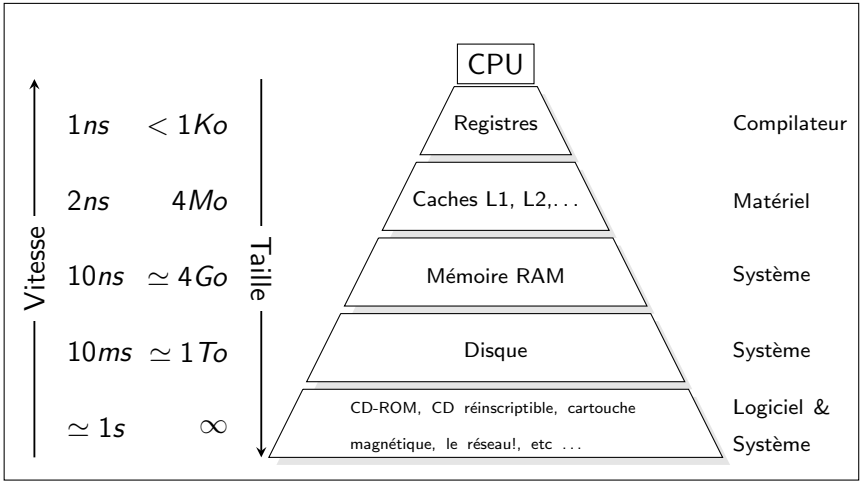
A quoi sert la mémoire ?

- Fournir une abstraction pour le programmeur
- Séparer les différents processus et utilisateurs
- Supporter l'organisation hiérarchique du matériel

Performance de la mémoire

- Maximiser l'utilisation du matériel
- Privilégier la localité spatiale et temporelle

Hiérarchie des mémoires



Problème de la gestion mémoire

Contexte

- Chaque accès à une variable, une fonction... est un accès à la mémoire
- Le compilateur fait la correspondance entre nom et adresse
- Pour que le code soit portable : translation d'adresse ou mémoire virtuelle

Mémoire centrale

- Adressage 32 ou 64 bits, imposé par le processeur
- Pointeurs sur 4 ou 8 octets
- En pratique, les processeurs supportent moins de 64 bits (39 bits mémoire physique, 48 bits mémoire virtuelle sur mon PC)

Plan

Programmation modulaire

Pourquoi faire ?

- Ne pas inclure plusieurs fois les mêmes bibliothèques : gain d'espace disque et de mémoire
- Changement de bibliothèque sans recompiler si même Application Binary Interface (ABI)

Calcul des adresses mémoire

- Chargement absolu : par le programmeur, le compilateur ou l'assembleur
- Chargement dynamique : adresses relatives calculées à l'exécution

Avec l'édition de liens, le partage de code est facile !

Emplacement d'un programme en mémoire

- Historiquement, tous les programmes et bibliothèques étaient chargés dans le même espace
- Un programme n'est pas toujours chargé au même endroit
- Idem pour les bibliothèques : nombre variable, chargement multiple possible...

Code indépendant de la position

- Uniquement des adresses relatives
- Compilation avec le flag Position-Independent Code : -fPIC
- Uniquement pour des références internes au programme

La pile et le tas

La pile

- Références au cadre précédant (programme appelant)
- Programme récursif : plusieurs cadres
- Augmentation imprévisible

Le tas

- Pas forcément contigu
- Augmentation imprévisible

Pour des raisons de sécurité, la position en mémoire de la pile et du tas doivent être variables !

Partitionnement (1/2)

État des lieux

- Une seule mémoire physique
- Plusieurs programmes, bibliothèques, modules..
- Comment les répartir ?
  - Découper en segments contigus ?
  - Allouer par morceaux ?

Partitionnement fixe

- Division de la mémoire en N partitions de taille fixe
- Nombre de partitions limité
- Fragmentation interne : mémoire non utilisée
- Programmes limités en taille
- Facile à gérer : les adresses finales correspondent à un décalage. Il faut faire attention à la taille maximale de la partition !

Partitionnement (2/2)

Partitions de taille différente

- Découpage fixe en ensemble de taille différente
- On prend la partie qui correspond le mieux
- Support matériel : décalage pour la traduction + limite à gérer pour ne pas déborder

Partitionnement dynamique

- Allocation de partitions selon la taille du programme
- Pas de fragmentation interne
- Fragmentation externe : résidus inutilisés entre les partitions
- Compactage dynamique ? Potentiellement cher !

Pagination

Principe

- Au lieu d'allouer une zone mémoire contiguë, on alloue plusieurs pages à un même programme
- Pas de fragmentation externe
- Un peu de fragmentation interne
- Gestion de la mémoire à plus faible granularité : charger, swapper, protéger...

Support matériel

- Complexe : un programme est réparti dans différentes pages physiques, potentiellement discontinues
- On doit traduire toutes les adresses à la granularité de la page

Finalement assez similaire à un système de fichiers !

Plan

Besoins et intérêts

Besoins

- Chaque processus voit un espace linéaire
- Taille non limitée par la mémoire physique (tout n'est pas forcément utilisé en même temps cf. swap)
- Protection, partage

Intérêts

- Plus de mémoire virtuelle que de mémoire physique disponible
- Ce qui n'est pas utilisé peut rester sur le disque
- Cohabitation des processus, indépendamment de leur taille
- Isolation des processus : chacun accède uniquement à son espace virtuel

Différents contextes

Types d'adresses

- Virtuelle : pointeurs déréférencés par le programme
- Physique : dans la mémoire physique de la machine
- Et aussi de bus, manipulables par les périphériques pour le DMA par exemple

Adresses

- Tout est octet. Accès à un bit : instruction spéciale après avoir chargé un octet dans un registre
- Le processeur manipule des adresses virtuelles, la traduction est effectuée au dernier moment (par la MMU)
- Partage facile : plusieurs @virt vers même @phys
- Protection avec des droits pour chaque @virt

Pagination (1/5)

Principe

- Division de la mémoire physique en blocs de taille fixe, les pages (4kio généralement)
- Division de l'espace virtuel en blocs de taille fixe (cadre de page)
- Association entre page virtuelle et page physique : tables de pages

Pagination (2/5)

Traduction des @virt

- @virt = numéro de cadre virtuel + décalage
- Le numéro est un index dans la table des pages
- Pour chaque cadre de page, on a un PTE : Page Table Entry
- Cela permet de faire la correspondance avec une page physique
- Pour cela, on parle de PFN : Page Frame Number
- @phys = PFN + décalage

Pagination (3/5)

Protection @virt : bits stockés dans le PTE

- bit valide ou non : pour les segfault
- bit présent ou non : défauts de page, swap..
- bits read-only, write...
- bit Dirty pour une page modifiée (cf. la suite)

Page Table Entries (PTE)

- PFN pour la page physique
- Les droits

On peut définir des droits page par page ! (utile pour le tas, la pile etc...)

Pagination (4/5)

Avantages

- Allocation de pages facile
- Swap de pages facile (même taille, bits de présence...)
- Partage et protection faciles

Inconvénients

- Fragmentation interne
- Coûteux : on doit faire la traduction @virt/@phys
- Coût mémoire important pour stocker la table des pages

Pagination (5/5)

Exemple simple

- @virt 32 bits, @phys 24 bits, page 4 Kio
- Décalage dans la page de 4 Kio nécessite 12 bits
- @virt : $32 - 12 = 20$ bits de numéro de cadre $+$ 12 bits de décalage
- table des pages pour traduire 2^{20} @virt avec PFN sur $24 - 12 = 12$ bits

Plan

Problématique

De l'espace gaspillé

- Cf. l'exemple précédent, pour un espace 32 bits avec des pages de 4 Kio, on a 2^{20} PTE : la table des pages complète prend beaucoup de place
- Une grande partie de l'espace adressable n'est pas utilisé

Solution : table des pages à plusieurs niveaux

- Table normale : un seul niveau
- Table hiérarchique à deux niveaux :
  - Un tableau de niveau 1 avec N pointeurs vers des tableaux de niveau 2
  - N tableaux de niveau 2 avec M PTE
- Généralement, on fait des tableaux de taille la taille d'une page pour plus d'efficacité.

Table hiérarchique des pages (1/2)

Avantages

- On n'alloue pas les tableaux d'entrées non valide, on marque le pointeur comme invalide
- Un peu de gaspillage pour stocker les pointeurs
- Mais on alloue qu'une partie de l'espace adressable

Exemples

- Processus qui utilise un seul cadre virtuel : on alloue une page par niveau
- Processus qui utilise tout l'espace virtuel : coût identique au coût d'une table à un seul niveau + coût de stockage des pointeurs intermédiaires. Mais très peu probable en pratique !

Table hiérarchique des pages (2/2)

Gestion des adresses

- @virt = agrégation d'adresses pour chaque niveau
- l'OS choisit le nombre de bits en faisant tenir chaque sous-table dans une page exactement
- La traduction @virt/@phys se fait en parcourant la table de haut en bas

Exemple dans Linux sous x86_64 avec @virt 48 bits

- 9 bits de PGD (Page Global Directory)
- 9 bits de PUD (Page Under Directory)
- 9 bits de PMD (Page Middle Directory)
- 9 bits de PTE (Page Table Entry)
- 12 bits de décalage dans la page

Pagination à la demande

L'OS est fainéant

- Une page n'est pas chargée en mémoire avant d'être accédée
- Idem pour les pages allouées
- On ne duplique pas une page mémoire tant qu'elle n'est pas modifiée

Premier accès à une page : défaut de page

Demand paging

- Au début et à la fin, tout est sur le disque
- Au cours de l'exécution, on charge ce qui est accédé, et on peut évincer des pages
- Mouvement de page entre disque et mémoire, bien sur transparent pour les applications

Défaut de page (1/2)

Principe

- Accès qui ne peut pas être réalisé par le matériel
- Le processeur ne peut pas faire, il envoie une exception

Exemples de défauts de page

- @virt invalide
- @virt valide mais page absente en mémoire physique
- @virt valide mais lecture/écriture interdite

Défaut de page (2/2)

Comme un appel système

- L'OS a un traitant pour l'exception défaut de page
- Ce traitant analyse le problème
  - S'il peut être résolu (par ex chargement d'une page depuis le disque), la même instruction est relancée
  - Sinon le processus est tué

Un défaut de page n'est pas forcément une erreur

- Page non présente en mémoire
- Copy-On-Write (cf. la suite de ce CM)
- Malloc pas encore utilisé (pagination fainéante)
- Mais parfois segfault !

Allocation d'une page

Où la mettre

- Liste de pages libres (attention cas NUMA)
- Si pas de page libre, il faut en libérer une

Swap d'une page

- Choisir une page peu utilisée (principe de localité temporelle)
- Si le bit Dirty est positionné, des modifications ont été effectuées par rapport à la version originale présente sur le disque, il faut donc sauvegarder ces modifications !
- Lorsqu'une page libre est récupérée, il faut charger les données et adapter les flags de la page

Projection de fichiers

Sous Unix, tout est fichier

- Le code du programme, les bibliothèques
- Les variables globale, la pile, le tas

L'OS ne fait que manipuler des pages de fichiers

Différents types de projections (avec mmap cf. TP)

- Publique avec MAP_SHARED : les modifications effectuées sont répercutées dans le fichier
- Privée avec MAP_PRIVATE : les modifications sont perdues à la fin du processus
- Anonyme avec MAP_ANONYMOUS : sans fichier de support. Par exemple pour le tas ou la pile

Copy-on-Write - CoW (1/2)

Coût des copies

- Les copies mémoires sont coûteuses, on les évite au maximum
- Retarder la duplication au plus tard, par exemple pour des pages partagées et privées mais pas encore modifiées

Mécanisme

- On met la page en lecture seule dans le matériel : il y aura un défaut de page en cas de tentative de modification
- Le traitant alloue une nouvelle page, copie le contenu de la première
- La copie devient privée

Copy-on-Write - CoW (2/2)

Exemple

- Deux fork() successifs : 4 processus
- Les zones privées sont marquées CoW avec 4 références
  - 1ere modification : copie et la page originale reste partagée avec CoW à 3
  - 2eme modification : copie et la page originale reste partagée avec CoW à 2
  - 3eme modification : copie et la page originale n'est plus partagée : accès normal !

Résumé des défauts de page

En pratique

- Accès invalide : segfault
- Accès valide mais page absente : chargement depuis le disque
- Accès en écriture et page privée : créer une copie avec mapping privé ou CoW

Défauts majeurs versus mineurs

- Majeur : chargement de page depuis le disque ou swap (très coûteux)
- Mineur : les autres
- Voir /usr/bin/time en TP

Heuristiques de gestion mémoire (1/2)

Quand charger les pages ?

- Quand elles sont accédées lors du défaut de page
- Chargement à l'avance des pages suivants (localité spatiale) : Prepaging

Où charger ?

- Dans n'importe quelle page physique en général
- Attention au cas NUMA

Quand sauvegarder ?

- On sauve uniquement les pages Dirty

Heuristiques de gestion mémoire (2/2)

Qui swapper ?

- On prévoit les pages non utilisées à l'avenir
- Politiques de remplacement comme pour les caches
  - LRU : Least Recently Used (mais des limites par ex. N+1 pages en tourniquet avec N pages physiques)
  - LFU : Least Frequently Used

Quand swapper ?

- Au dernier moment ? Risque de blocage...
- En pratique, démon Swapper qui swap en avance quand la mémoire dépasse un certain seuil

Si aucune page swappable, le système se garde de la mémoire et tue un processus peu important pour libérer de la mémoire

Plan

Gestion dans le matériel

Contexte

- Le code manipule uniquement des @virt
- Le processeur doit les traduire en @phys
- La table des pages doit être accessible au matériel
  - Directement par une MMU
  - Par l'OS via une exception

Memory Management Unit (MMU)

- Circuit dédié du processeur pour traduire les @virt
- Obtient le PFN à partir du PTE et ajoute le décalage

Translation Lookaside Buffer (TLB)

Traduction des @virt lente

- On a un accès mémoire par niveau de table, donc plusieurs accès mémoire matériels lors d'un seul accès du point de vue du programme !
- Le TLB est un cache dans la MMU, qui contient les dernières traductions (sans le décalage)

Fonctionnement similaire à un cache

- Totalement associatif : on teste toutes les entrées en parallèle
- Très rapide mais cher donc petit
- Si TLB miss, récupération de la traduction @virt/@phys dans la MMU si elle existe, via l'OS sinon

Comme un cache L1, il faut vider le TLB lors d'un changement de contexte, lorsqu'un autre processus est ordonnancé !

Résumé de la traduction @virt/@phys

Ordre des tests

1. Le programme passe une @virt dans une instruction
2. Le TLB est consulté pour traduire rapidement
   - Si TLB miss
     - Si MMU, la MMU parcourt la table des pages
     - Sinon, exception gérée par l'OS
   - Sinon (hit), le TLB contient le PTE et valide la protection
3. La mémoire est accédée

Plan

Le noyau, un espace mémoire particulier

Espace virtuel spécifique au noyau

- Accessible uniquement en mode privilégié
- Directement déréférençable depuis le noyau
- Non modifié lors des changements de contexte

Espace virtuel utilisateur

- Pas toujours déréférençable depuis le noyau
- Modifié lors des changements de contexte

Mémoire du noyau (1/3)

A quoi le noyau peut-il accéder ?

- Potentiellement toute la mémoire physique
- Son code et ses données propres sont petits (moins de 1 Go)
- La plupart des données en mémoire ne sont jamais déréférencées par le noyau
- Besoin de mapper des pages quelconques de temps en temps

Mémoire du noyau (2/3)

Espace virtuel

- Divisé entre espace virtuel noyau et espace virtuel du processus courant
- Espace noyau virtuellement partagé entre tous les processus, mais une seule fois en mémoire physique
- En 32 bits, espace noyau de 1 Go, un processus ne peut donc utiliser que 3 Go de mémoire virtuelle
- En 64 bits, il y a largement de la place pour tout le monde

Mémoire du noyau (3/3)

Optimisation

- Représentation à l'identique de la mémoire physique
- On parle d'identity mapping
- Pas de traduction @virt/@phys nécessaire pour la partie noyau

Plan

Exemple de traduction (1/2)

- Chaque adresse est codée sur 6 bits
- Une adresse de 6 bits correspond à 4 octets
- Chaque page contient 8 mots mémoires
- Il y a une indirection : pour calculer l'adresse réelle à partir d'une adresse "virtuelle" donnée, on regarde la table des pages du processus en cours
- Une adresse logique est composée de 6 bits, de gauche à droite : 3 pour la page et 3 pour le décalage dans la page.
- Dans les tables de pages, pour chaque page, 3 bits sont utilisés pour détailler les propriétés de la page, dans l'ordre :
  1. pour signaler l'existence de la page
  2. pour signaler le droit d'écriture
  3. pour signaler le droit d'exécution

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVIL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct I	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	I Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

$0b011110 = 0o36$

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVlL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct l	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	l Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On regarde la table des pages à l'adresse 17

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVIL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct I	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	I Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On cherche l'indirection d'indice 3

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVlL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct l	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	l Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On regarde la page 3

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVlL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct l	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	l Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On applique un décalage de 6

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVIL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct I	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	I Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On lit 8 octets

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVlL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct l	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	l Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On regarde la page suivante

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVlL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct l	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	l Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On regarde la page 0

Exemple de traduction (2/2)

- Table des pages du processus 1 à l'adresse 0o17
- Pour le processus 1, donnez les 16 octets de la chaîne de caractères stockée à l'adresse 0b011110 = 0o36

	00	01	02	03	04	05	06	07
00	cta_	est_	B7K	mVf	nWs4	Nfi2	RRUU	7tE0
01	qSFc	dB90	uUou	cUcx	LkPT	8elm	Q7hU	mGRw
02	6csB	XnzW	WuzX	FoH4	gg2	j6Tp	eirC	6mWK
03	sbj2	6y75	7xng	CH81	4P0b	VISQ	ale_	a_ja
04	4Dyr	FVIL	hC6	FRxx	dYrM	K1Yv	uJqg	eg7H
05	MIt	PTs	EJCg	O7OW	zBcs	ct I	usu	A5Ag
06	dzPN	LmbP	rs0l	wnLg	jze0	M9tO	Z85	x p
07	qUK5	Dvs4	Ed5s	goF7	20Jk	9ZeN	tsE	tYxa
10	eOs7	Oki	supo	csan	ivid	S5gD	HkwC	I Yc
11	0Nqu	MxnY	AfAb	JCzq	RTkf	I P4	TzN4	CPp2
12	Md i	Lrba	NFCV	muth	JG5X	xO8g	qaJu	jT39
13	Nv0b	XX3q	n mF	eDxs	yHQv	nbib	tceT	FSW0
14	mQQf	N3mf	qOOj	XSX9	Cw B	n2cq	yw9	2PcV
15	tKgu	8GAT	N217	tz 6	wYxW	Hw44	WUWt	G8Hn
16	101.10	001.17	000.00	110.02	101.15	000.12	111.13	001.30
17	100.00	001.23	000.00	110.03	110.00	010.10	101.10	010.21

On lit 8 octets

Conclusion

A retenir

- L'intérêt de la pagination
- Le fonctionnement des tables de pages hiérarchiques
- Les différents droits et modes de fonctionnement des pages
- Le support matériel (MMU, TLB, swap)
- Le cas particulier du noyau