La mémoire virtuelle de VMS

Quelques
pages des mémoires
d'une
page de mémoire

Jean-Pierre PETIT

ESME-Sudria

23 Janvier 1991

Principes généraux

But:

S’affranchir dans une très large mesure des classiques problèmes de limitations très difficilement contournables introduits par l’adressage d’un espace physique.

Principe:

- Le programmeur *voit* un espace adressable virtuel contigü arbitrairement grand
- Une partie de cet espace virtuel se trouve contenu dans la mémoire physique, le reste se trouvant *ailleurs*.
- Le système d’exploitation assure une gestion transparente de cet espace de façon à amener en mémoire physique toute information exigée par l’exécution de la tâche en cours qui ne serait pas déjà résidente.

- Dans bien des cas, ceci suppose de faire de la place en renvoyant *ailleurs* des informations non immédiatement utiles.
- Pour des raisons de performance, l'*ailleurs* évoqué ci-dessus ne peut raisonnablement être qu'un disque magnétique.

Avantages:

Ils paraissent aujourd'hui tellement évidents qu'on les passe souvent sous silence. Pourtant, il existe un outil de mesure efficace: le pourcentage de programmeurs ayant lu le manuel de l'éditeur de liens (à comparer avec un système à adressage physique).

- Chaque programmeur travaille dans un espace virtuel contigu dont la borne supérieure pratique est souple, c'est-à-dire généralement modifiable par le responsable du système (à condition de ne pas atteindre les limites de l'architecture).

- Une inadéquation de la taille de la mémoire physique par rapport aux besoins d'une application se traduit par une diminution de performances et non une impossibilité d'exécution.
- La coexistence d'autant d'espaces adressables indépendants que de processus élimine une bonne fois pour toute le problème de recalcul de toutes les adresses absolues lors du chargement de l'exécutable.
- Il est possible de se passer complètement de chargeur de programmes, la gestion de la mémoire virtuelle pouvant amener le programme en mémoire au fur et à mesure des besoins.

Inconvénients:

- La gestion de la mémoire par le système d'exploitation peut prélever une fraction non négligeable de la puissance du processeur.

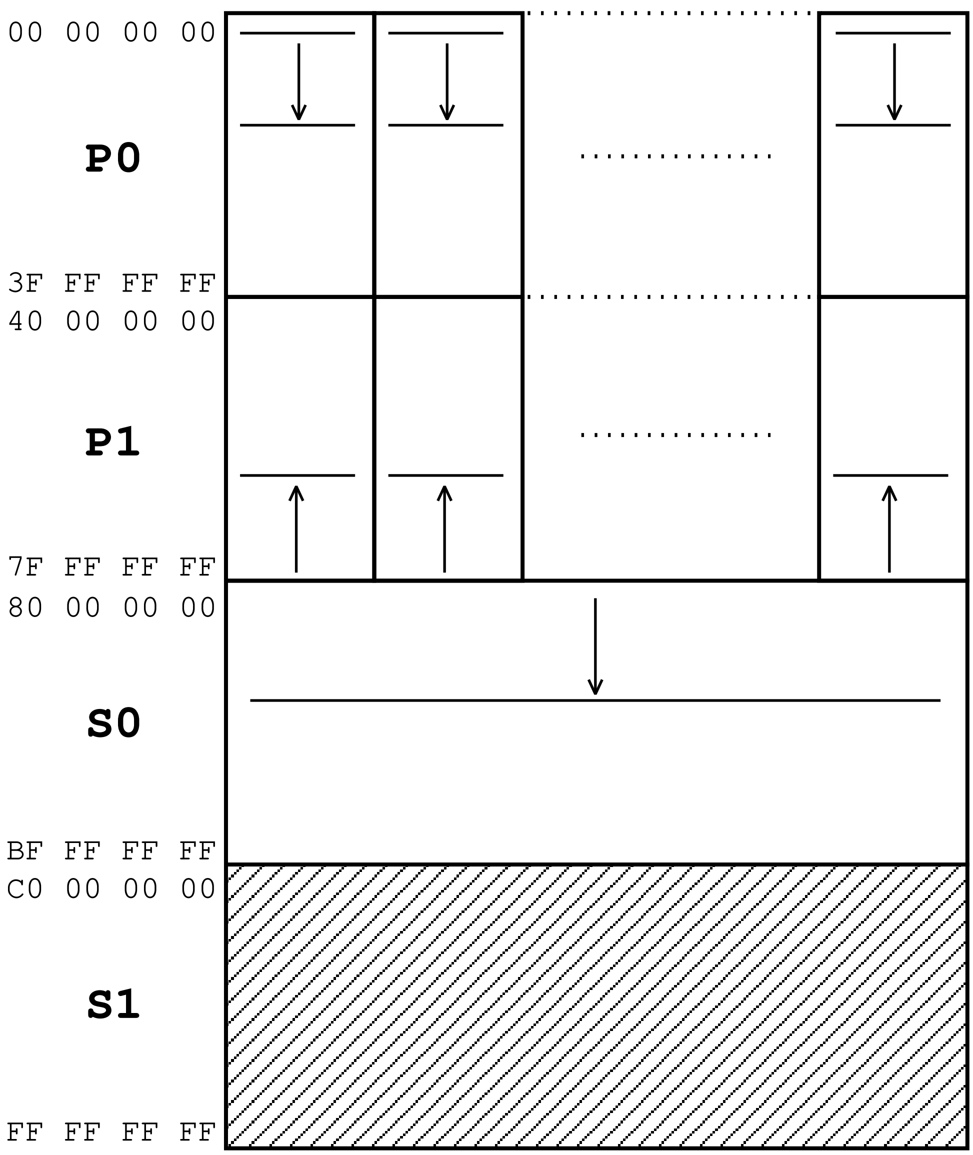
- Cette gestion nécessite de maintenir beaucoup d’informations sur les différents espaces virtuels et sur l’occupation de la mémoire physique, ce qui prélève une partie non négligeable de la mémoire disponible.
- L’ignorance du comportement de la mémoire virtuelle peut conduire à des implémentations qui pénalisent énormément le comportement du système.
- La souplesse de la mémoire virtuelle est un inconvénient ! Avec elle, quelle que soit l’implémentation, l’application fonctionne (**mal mais elle fonctionne**). On accuse donc le système (ou son responsable) mais pas le concepteur.

Architecture VAX et adressage virtuel

31 30	29 … 9	8 … 0
Re-gion	Virtual Page Number	Byte Offset

- L'espace virtuel est composé de pages de taille fixe égale à 512 octets.
- Il est subdivisé en sous-espaces:
  - -- la première moitié ou **espace processus** existe en autant d'exemplaires qu'il y a de processus
  - -- la deuxième moitié ou **espace système** n'existe qu'en un seul exemplaire commun à tous les espaces virtuels de tous les processus
- Chaque sous-espace est à son tour divisé en deux régions de même taille:
  - -- **P0** et **P1** pour chaque espace processus

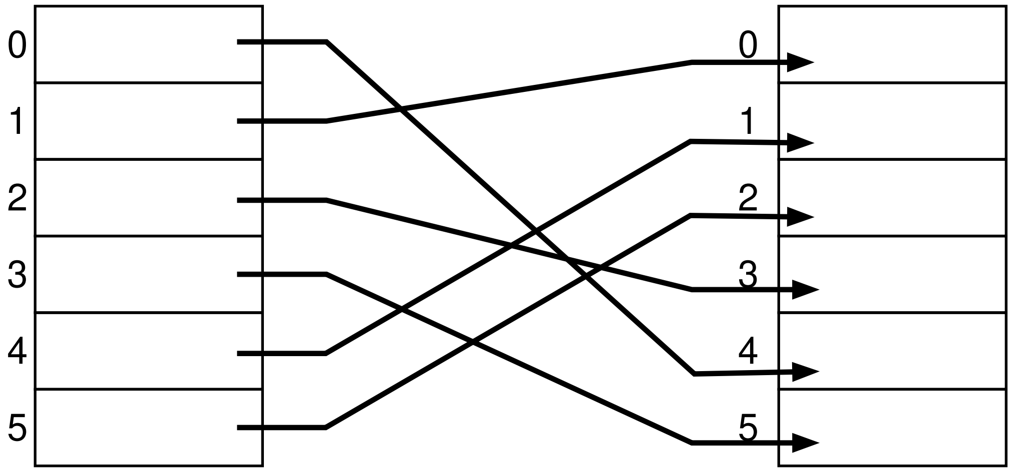
-- **S0** (et **S1** qui n'a pas d'existence physique) pour l'espace système



- La mémoire physique est également découpée en page de 512 octets dénommées *page frames*.

Correspondance virtuel -> physique

- La manière la plus simple d’effectuer cette correspondance (*mapping)* consiste à créer pour chaque région une table appelée *page table*. Chaque élément de cette table ou *page table entry (PTE)* contient le numéro de page physique associé à la page virtuelle selon le schéma suivant:



Espace Virtuel

Espace Physique

0	**4**
1	**0**
2	**3**
3	**5**
4	**1**
5	**2**

Table de Pages

- Lorsque la page n'est pas résidente, la PTE contient de quoi la localiser sur disque.
- La taille d'une PTE est de 32 bits soit 1/128 de la mémoire qu'elle décrit. Une table de page peut donc représenter un volume assez important. Pour éviter de gaspiller de la place, seule la partie utilisée de chaque région est décrite. Les tables de pages sont donc de longueur variable.
- Les régions **P0** et **P1** étant propres à chaque processus, il y a autant de paires de tables de pages que de processus. Par contre une seule table dite *System Page Table* est utilisée pour décrire la région **S0**.
- Le processeur VAX maintient des descripteurs de tables de pages sous formes de paires de registres:

  -- P0BR P0 base register
  P0LR P0 length register

-- P1BR P1 base register
P1LR P1 length register

-- SBR S base register
SLR S length register

- Lors du changement de contexte, l'instruction LDPCTX recharge les 2 paires P0xR et P1xR, ce qui assure automatiquement le changement d'espace virtuel. La paire SxR n'est modifiée que lors du démarrage du système juste avant de passer en adressage virtuel.
- Les tables décrivant P0 et P1 sont repérées par leur adresse virtuelle. La table décrivant S0 est repérée par son adresse physique.
- La PTE contient également des informations de protection de la page, c'est-à-dire les modes d'accès (Kernel, Executive, Supervisor, User) les moins privilégiés autorisant la lecture et la modification de la page.

Mécanisme de pagination

Notion de Working Set:

- Le *Working Set* est le sous-ensemble des pages d'un processus qui à un instant donné sont *valides*, c.à.d. peuvent être référencées par une instruction sans provoquer d' exception du type "Translation Not Valid".
- Il y a donc un working set par processus décrivant les pages des sous-espaces P0 et P1. De plus, il existe un working set pour l'espace S0, il est connu sous le nom de *system working set.*
- Cette notion va permettre de contrôler la quantité de mémoire physique utilisée par un processus.

Notion de Page Fault:

- Il y a *page fault* si, lors d'une tentative d'accès à la page, le mécanisme de traduction d'adresse détecte que la page n'est pas **valide** (bit de poids fort de la PTE).
  Ceci génère une exception **Translation not valid** avec les arguments suivants sur la pile:

Reason Mask
Virtual Address
PC
PSL

Bit #1	0	VA not valid
	1	PTE not valid
Bit #2	0	Read access
	1	Write access

- Cette exception sera traitée par un *exception handler* particulier: le *page fault handler* ou *pager* (routine MMG$PAGEFAULT).
- Le *pager* va utiliser le contenu de la PTE pour localiser la page et entreprendre l'action appropriée pour la rendre valide.

- Il va ensuite positionner le bit "valid" dans la PTE et exécuter une instruction REI (retour d'exception ou interruption).
- A ce moment l'instruction ayant provoqué l'exception reprend son cours et s'exécute normalement puisque la page est devenue valide.
- Il est à noter qu'il peut y avoir plusieurs exceptions consécutives pendant une même instruction (opérandes sources et destination, données occupant plusieurs pages).

Localisation des pages

Au fur et à mesure de l’exécution des programmes, les pages peuvent se trouver en différents endroits. A chaque cas de figure va correspondre une forme différente de PTE.

Page valide:

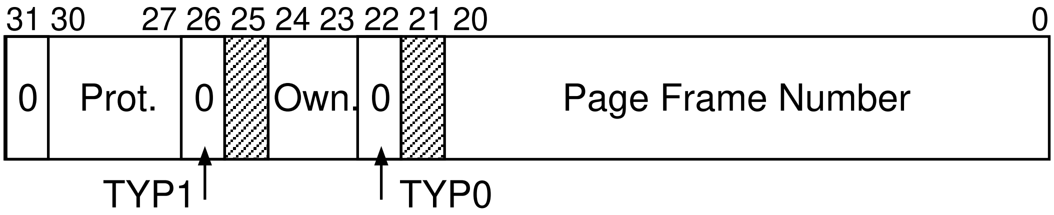
- C’est le cas le plus simple correspondant à une page valide appartenant à un working set:

31	30 … 27	26	25	24 23	22	21	20 … 0
1	Prot.	M		Own.		W	Page Frame Number

- Le *modified bit* de la PTE est automatiquement mis à jour par le processeur VAX lorsqu’il modifie une quelconque donnée à l’intérieur de la page.

Page en transition:

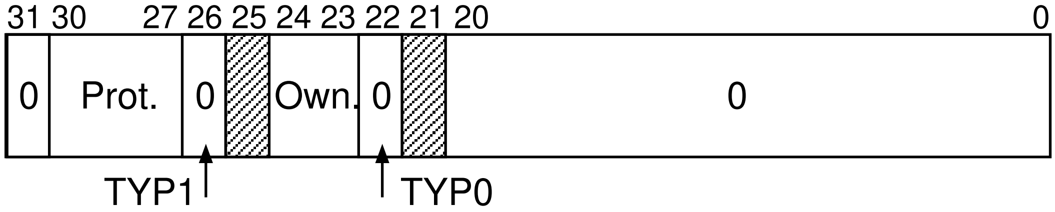
- Lorsque qu’une page fault a lieu alors que le working set a atteint sa taille limite, il devient nécessaire de retirer une page du working set. Selon que cette page a été ou non modifiée, cette page est reportée sur la *modified page list* ou la *free page list.*



- Il y a donc une probabilité non nulle de retrouver en mémoire une page déjà utilisée, plutôt que d'être obligé d'aller la chercher sur disque.
- On parle de *soft page fault* dans ce cas, par opposition à une *hard page fault* qui nécessite un accès au disque et va entraîner la suspension de l'exécution du processus placé en état PFW jusqu'à retour de la page en mémoire.
- Pendant la durée de l'accès disque, la page virtuelle en cours de lecture est également considérée comme une page en transition.

Page de type *Demand Zero*:

- Par opposition à beaucoup d’autres systèmes d’exploitation, VMS ne stocke pas dans le fichier exécutable les pages correspondant à des variables statiques non initialisées. La section de mémoire correspondante est simplement décrite par une adresse de début et un nombre de pages.
- A l’activation de l’exécutable, une PTE du format ci-dessous est créée pour chaque page de type *demand zero.*



- Le *pager* va allouer une page de mémoire physique libre, la remplir de zéros et placer le numéro de page physique (PFN) dans la PTE.

Page située dans un *Page File*:

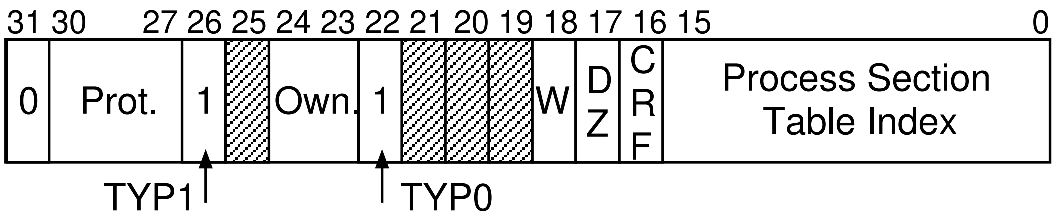
- Lorsque la *modified list* devient trop importante, elle est réécrite sur disque. Il n’est cependant pas possible de réécrire les pages modifiées dans l’exécutable à moins de vouloir crééer des effets de bords intéressants.
- Les pages sont écrites dans un ou plusieurs fichiers de pagination *(page file)* et la PTE modifiée en conséquence pour aboutir au format suivant:

31	30 27	26	25	24 23	22	21 20	19 0
0	Prot.	1		Own.	0	Page File Index	Page File Virtual Block Number
		TYP1			TYP0		

- Depuis VMS V5.0, il est possible d’avoir jusqu’à 4 fichiers de pagination par processus. Pour implémenter ceci, deux bits ont été prélevés sur le VBN stocké dans la PTE et servent d’indice dans un tableau de 4 indices de fichier de pagination situé dans le *process header* (PCB$B_PRCPGFL).

Page décrite par un indice pointant vers la *Process Section Table*:

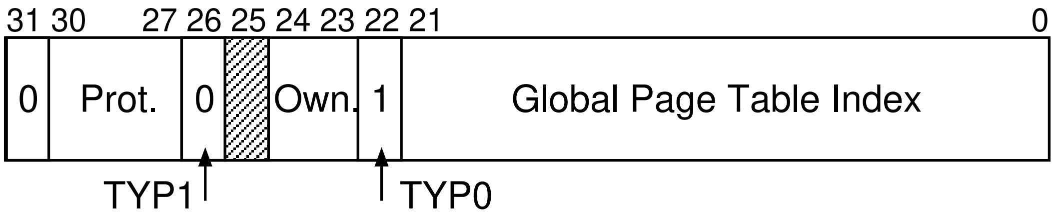
- Lorsque qu'une page d'un exécutable n'a pas encore été appelée en mémoire, sa PTE pointe vers une entrée de la table des sections du process (PSTE). Sa forme est alors la suivante:



- La table des sections comporte pour chaque section:
  - -- l'adresse virtuelle de début
  - -- le nombre de pages
  - -- le VBN de début
  - -- l'adresse du WCB décrivant le fichier
- Si la page n'est pas de type *copy on reference*, la PTE retrouvera cette forme après élimination de cette page de la *free list* ou de la *modified list.*

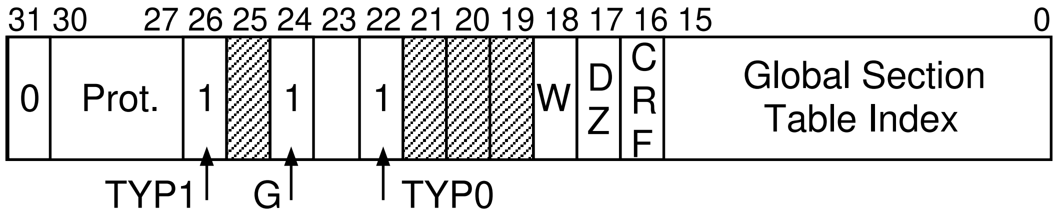
Page décrite par un indice pointant vers la *Global Page Table*:

- VMS permet le partage de pages de mémoire entre différents processus. Ceci permet de faire des économies très substantielles de mémoire et de diminuer le taux de *hard page faults* puisqu’il y a une probabilité importante qu’une page demandée par un processus se trouve déjà en mémoire à la suite d’une demande d’un autre processus.
- Dans ce cas, le format initial de la PTE est le suivant:

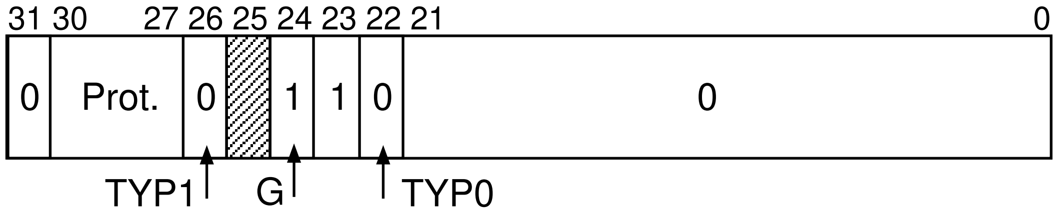


- La PTE pointe vers une entrée de la table des pages globales (GPTE). Cette table est située dans l’espace S0 entre MMG$GL_GPTE et MMG$GL_MAXGPTE.

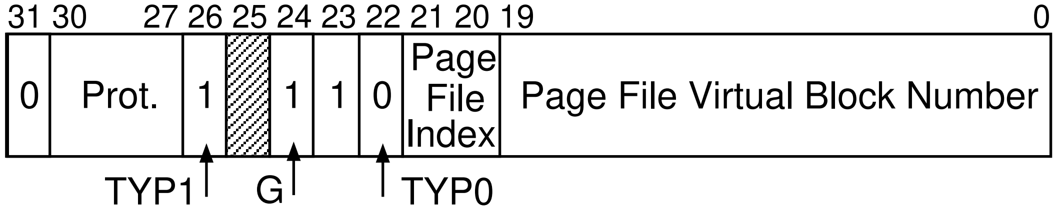
- La GPTE peut correspondre à l’un des trois formats suivants déjà vus:
  - -- page valide
  - -- page en transition
  - -- page de type *demand zero*
- Elle peut aussi avoir l’un des trois formats spécifiques suivants:
  - -- page située dans un *global section file*



  - -- page de type *demand zero* dans une *global page file section*



  - -- page en cours d’utilisation dans une *global page file section*



Gestion des fichiers de pagination

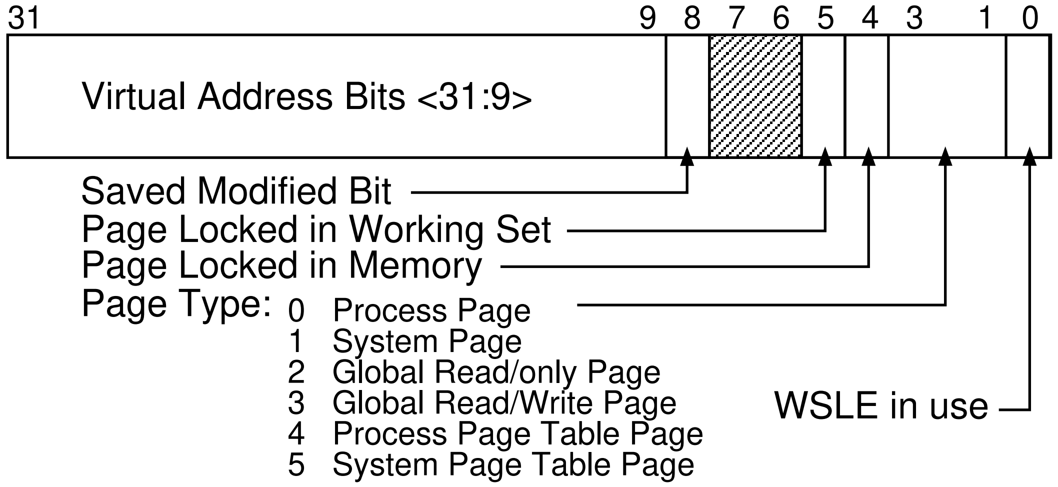
- A la création d'un processus, le *page file* possédant le meilleur rapport blocs réservables / taille totale lui est affecté et RSRVPAGCNT pages lui sont réservées (nombre stocké dans PHD$W_PRCPGFLPAGES).
- Lorsque le processus référence une nouvelle page nécessitant une sauvegarde éventuelle dans un *page file*, ce nombre est décrémenté.
- Lorsque ce nombre atteint zéro, la routine MMG$SWITCH_PRCPGFL est appelée pour rechercher à nouveau un *page file* optimal et une nouvelle réservation est faite.
- Le nombre maximum de *page files* affectés à un processus ne peut en aucun cas dépasser 4.

Gestion du Working Set

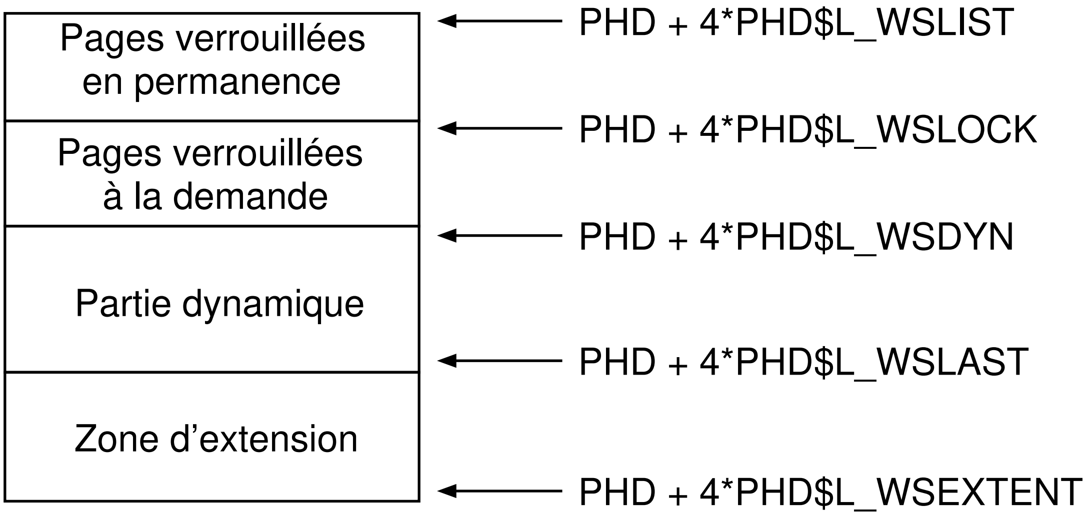
- Le but de la notion de working set est de permettre une gestion de la mémoire physique disponible au mieux de l'intérêt des différents processus.
- Un certain nombre de mécanismes permettent une régulation automatique de la taille des différents working sets de façon à conserver les meilleures performances possible.
- Lorsque la demande de mémoire est très importante, il se peut malgré tout que l'ensemble des working sets ne puissent résider en mémoire. Un processus particulier, le *swapper* est chargé de remédier au déficit de mémoire en expulsant des working sets entiers de processus convenablement choisis. Ces working sets sont écrits dans un *swap file*.

Description du Working Set:

- Le working set d'un processus est décrit par une structure située dans le *process header*: la *working set list* ou *WSL*. Chaque élément de cette table ou *WSLE* possède le format suivant:



- La *WSL* est divisée en trois régions auxquelles s'ajoute une zone d'extension:



Adjonction de pages au *Working Set*:

- Si lors d'une *page fault*, la partie utilisée du *working set* (PCB$L_PPGCNT+PCB$L_GPGCNT) est inférieure la taille limite (PHD$L_WSSIZE), il est probablement possible d'ajouter une nouvelle entrée au *working set*. Cependant, l'une des conditions suivantes doit être remplie:

  -- WSSIZE < WSQUOTA

  -- free list size > GROWLIM

Recyclage des pages du *Working Set*:

- Lorsqu'une *page fault* se produit alors que le working set a atteint sa taille limite, il devient nécessaire d'en expulser une page avant d'en faire rentrer une nouvelle.
- La routine MMG$FREWSLE va balayer la *WSL* en commençant par l'entrée pointée par PHD$L_WSNEXT.

-- les entrées correspondant à des pages de tables de pages actives sont sautées

-- si la page décrite par la WSLE est encore valide dans le *translation buffer*, l'entrée est également sautée (ceci permet de modifier légèrement l'algorithme de type FIFO pour introduire une notion simplifiée de vieillissement de page)

-- au maximum, TBSKIPWSL entrées valides sont sautées de cette façon.

- Si la page choisie a été modifiée, elle doit être reportée sur la *modified list.* Cependant, ceci ne peut se faire qu' après validation d'au moins une des conditions suivantes:

-- modified list < MPW_WAITLIMIT

-- écriture de la modified list en cours
et
modified list < MPW_LOWAITLIMIT

- Dans le cas contraire, le processus est mis en attente (état RWMPB).

Ceci n'est cependant pas effectué si aucun *page file* n'a été installé, ni pour le *swapper* ou tout processus détenant un *mutex.*

Ajustement automatique du *working set*:

- VMS calcule régulièrement (à l'occasion du *quantum end* mais avec un minimum égal à AWSTIME) le taux de pagination de chaque processus. Celui-ci est défini par un nombre de *page faults* ramenées à un temps CPU consommé.
- Si ce taux de pagination est important (supérieur à PFRATH), il est probablement légitime d'en déduire que le *working set* est trop petit et pénalise l'exécution du processus. VMS va alors augmenter la taille du *working set* de WSINC en respectant les critères suivants:

-- pas d'augmentation si la taille courante est inférieure à 75% de la taille limite

-- si la taille limite est supérieure à WSQUOTA l'augmentation n'a lieu que si
free list size > BORROWLIM

-- la taille du *working set* ne peut en aucun cas dépasser WSEXTENT

- Si le taux de pagination est faible (inférieur à PFRATL), il peut paraître légitime de penser que le processus pourrait abandonner un peu de mémoire physique sans être trop pénalisé et en faire profiter les autres.
  Dans ce cas, le *working set* est réduit de WSDEC sous réserve que
  PCB$L_PPGCNT >= AWSMIN .
  En général, on reproche deux choses à ce mécanisme:

  -- il s'attaque aux processus les plus actifs (ceux qui ne consomment pas de temps CPU ne sont pas victimes du *quantum end*)

-- il y a un risque de phénomène de pompage, le *working set* passant son temps à subir des incrémentations et des décrémentations.

- C'est pourquoi ce mécanisme est généralement **inhibé** par une **valeur nulle** de **PFRATL** (depuis VMS V3.1).
- Un nouveau mécanisme introduit depuis VMS V5.0 a pour but de récupérer de la mémoire sur les processus de basse priorité. Il intervient sans prise en compte de taux de pagination si les trois conditions suivantes sont réunies:

  -- le processus a obtenu une élévation de priorité (mécanisme PIXSCAN) lors des 32 derniers *quantum intervals*

  -- *free list size* < GROWLIM

  -- *working set size* > WSQUOTA

Le Swapper

- Le *swapper* est un processus indépendant de priorité temps-réel 16 et dont tout le code exécutable se trouve en espace S0.
- Il hiberne en permanence et n’est réveillé que par certains événements.
- Il entreprend alors la liste d’actions successives suivante:
  - -- si *free list size* < FREELIM écrire la *modified page list* seulement si elle est supérieure à MPW_THRESH dans le but d’arriver à FREEGOAL
  - -- si *modified list size* > MPW_HILIMIT écrire la *modified list* jusqu’à ce qu’elle retombe à MPW_LOLIMIT
  - -- recherche du processus en état COMO le plus prioritaire, comparaison de la taille de son *working set* à celle de la *modified list* et éventuellement *inswap*

- appel de la routine OUTSWAP qui va tenter de faire de la place par différents moyens

- La routine OUTSWAP fait entrer en jeu un mécanisme relativement complexe dont les principales étapes sont:
  - élimination de PHDs pour 2 catégories de processus:
    - processus détruits
    - processus dont le corps a déjà été écrit dans le *swap file*
  - tentative de réduction du *working set* de processus habilement choisis en fonction de leur improbabilité d'exécution dans un futur proche:
    - dans un 1er temps jusqu'à WSQUOTA
    - dans un 2ème temps, si nécessaire, jusqu'à SWPOUTPGCNT
  - en dernier lieu, si nécessaire, *outswap* du corps du processus dans le *swap file*

Choix des candidats:

La sélection des candidats à la rédution de *working set* et à l'*outswap* routine (SCH$OSWPSCHED) repose sur l'utilisation des informations du *scheduler* et d'une table de décision comportant les informations suivantes:

- état du processus
- E/S en cours (*buffered, direct)*
- priorité
- processus dans son *initial quantum*
- processus en attente > LONGWAIT
- processus en attente > DORMANT-WAIT
- particularités de traitement:
  -- *first level trimming only*
  -- *swap asap* (WSQUOTA*)*

Quelques statistiques

VAX/VMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node BYTE
22-JAN-1991 20:22:51

	CUR	AVE	MIN	MAX
Page Fault Rate	0.00	7.06	0.00	46.53
Page Read Rate	0.00	0.31	0.00	4.96
Page Read I/O Rate	0.00	0.05	0.00	0.33
Page Write Rate	0.00	4.16	0.00	79.20
Page Write I/O Rate	0.00	0.03	0.00	0.66
Free List Fault Rate	0.00	1.85	0.00	18.15
Modified List Fault Rate	0.00	2.36	0.00	23.76
Demand Zero Fault Rate	0.00	0.20	0.00	2.64
Global Valid Fault Rate	0.00	2.51	0.00	18.81
Wrt In Progress Fault Rate	0.00	0.05	0.00	0.99
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	8193.00	8193.63	7875.00	8281.00
Modified List Size	1078.00	1095.47	1078.00	1258.00

Un gag classique

- Version 260 000 page faults

```
program FAULT_WORST_CASE
parameter (N = 512)
integer*4 T(N,N)

call LIB$INIT_TIMER

do I = 1, N
        do J = 1, N
                T(I,J) = 1
        end do
end do

call LIB$SHOW_TIMER

end
```

- Version 2 000 page faults

```
program FAULT_BEST_CASE
parameter (N = 512)
integer*4 T(N,N)

call LIB$INIT_TIMER

do J = 1, N
        do I = 1, N
                T(I,J) = 1
        end do
end do

call LIB$SHOW_TIMER

end
```

?