Was ist Swapping?

Als Swapping bezeichnet man die Auslagerung von Daten aus dem physikalischen Arbeitsspeicher auf die Festplatte um den nutzbaren Arbeitsspeicher virtuell zu vergrößern.

Welche Vor- und Nachteile bietet Swapping?

Durch Swapping wird der von den Prozessen nutzbare Adressraum erweitert und es können mehr und vor allem speicherintensivere Programme ausgeführt werden. Der größte Nachteil besteht darin, daß der Zugriff auf einen Swapbereich der Festplatte wesentlich langsamer ist als der Zugriff auf einen Arbeitsspeicher.

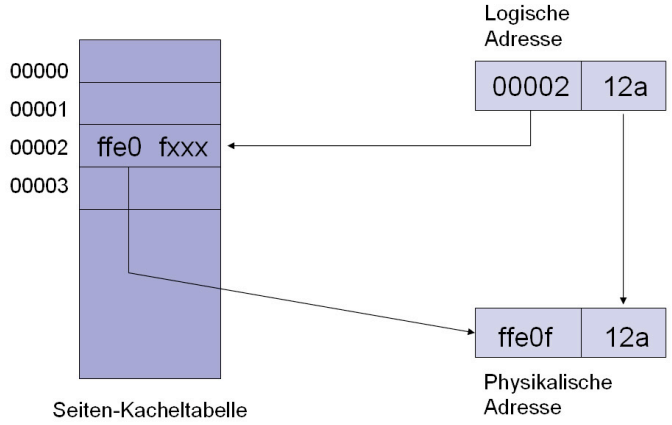
Der Swap-Bereich

Der Swapbereich eines Betriebssystems kann in Form einer oder mehrerer separaten Partitionen sowie in einer oder mehreren Dateien realisiert werden.
Unter Unix besteht der Auslagerungsbereich in der Regel aus einer Partition, da der Zugriff auf diese schneller ist als auf eine Datei, weil in diesem Fall der Umweg über das Dateisystem gegangen werden muß.
Der Nachteil bei der Auslagerung in einer Partition ist, daß man diesen Bereich bereits bei der Installation festlegen muß und wenige Möglichkeiten hat ihn nachträglich zu verändern.

Manche Unixsysteme arbeiten jedoch mit einer Swappartition UND einer Swapdatei.
Dies hat den Vorteil, daß man eine vollkommen dynamische Lösung zur Verfügung hat und man auch kurzfristige Speicherengpässe einfach kompensieren kann.

Ein System kann auch über mehrere getrennte Swapbereiche verfügen. Dies bringt den Vorteil mit sich, daß es zur gleichen Zeit auf mehrere Laufwerke schreiben und so Daten schneller aus- und wieder einlagern kann. Außerdem kann man so im laufenden Betrieb Swapspeicher hinzufügen und entfernen.

Rückblick: Paging

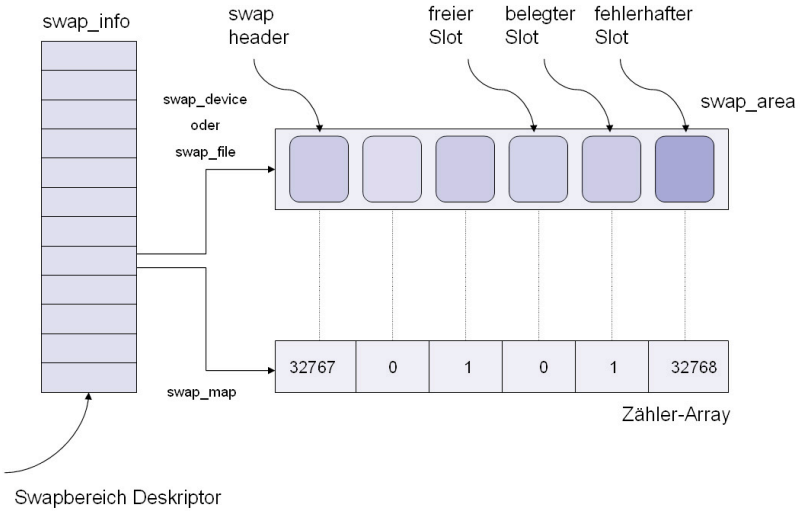


Beim Paging werden Seiten des logischen Adressraums über die Seiten-Kachelntabelle in Kacheln im physikalischen Arbeitsspeicher umgesetzt.

Aufbau eines Swap-Bereichs:

Der Swapbereich ist ein einzelne Slots mit jeweils 4096 Bytes Größe unterteilt. In jeden dieser Slots kann genau eine Speicherseite eingelagert werden.

Die folgende Grafik verdeutlicht den Aufbau eines Swapbereiches:



Der erste Slot beinhaltet den so genannten swap_header, in dem Informationen über den Swapbereich gespeichert sind. Der swap_header besteht aus zwei Teilen, den `info`- und den `magic`-Bereich.

Der `info`-Bereich beeinhaltet unter anderem folgende Felder:

`info.last_page`
: Angabe des letzten benutzbaren Slots

`info.nr_badpages`
: Anzahl der fehlerhaften Slots

`info.padding`
: Padding Bytes (Auffüll-Bytes)

`info.badpages`
: Angabe der fehlerhaften Slots (bis 637)

Der `magic`-Bereich ist deutlich einfacher aufgebaut, er besteht nur aus einem einzigen Feld. In diesem ist ein String gespeichert, der Aufschluß über die verwendete Version des Swap-Algorithmus gibt. SWAP-SPACE für Version 1 bzw. SWAPSPACE2 für Version 2. Version 2 kann im Gegensatz zur ersten Version einen größeren Swapbereich ansprechen.

Der Swapbereich Deskriptor

Jeder Swap-Bereich hat seinen eigenen `swap_info_struct` Deskriptor, der weitere Informationen beeinhaltet. Er gliedert sich wie folgt:

`flags`
: dieses Feld besteht aus zwei, sich überlappenden Unterbereichen:

  `SWP_USED`
  1, wenn der Swap-Bereich aktiv ist, 0 wenn nicht
  `SWP_WRITEOK`
  bestimmt, ob der Swapbereich beschreibbar ist

`swp_map`
: deutet auf ein Zähler-Array, ein Zähler für jeden Slot.
  0: Der Slot ist frei
  >0: Der Slot ist bereits benutzt
  32767: ist die Seite im Slot als permanent gekennzeichnet und kann nicht entfernt werden
  32768: ist die Seite im Slot als fehlerhaft gekennzeichnet und somit unbenutzbar

`prio`
: Priorität, mit der auf den Swap-Bereich zugegriffen wird.

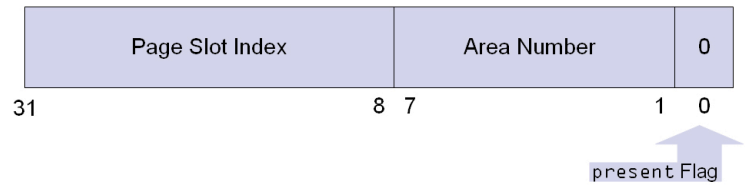
`sdev_lock`
: spin lock, der den Deskriptor gegen simultane Zugriffe in SMP Systemen schützt

`max`
: Angabe der Größe des Swap-Bereiches in Slots (je 4 KB)

`pages`
: Angabe der verwendbaren Slots
(wie max nur ohne den ersten und den fehlerhaften Slots)

`next`
: Zeiger auf den nächsten Swapbereich Deskriptor

Der Swapped Out Page Identifier



Der Swapped out Page Identifier gibt Auskunft darüber, ob sich eine Seite im Speicher oder im Swapbereich befindet. Ist das letzte Bit 0, so wurde die Seite ausgelagert, ist es 1, so befindet sich die Seite im Arbeitsspeicher.

Erstellen, Aktivieren und Deaktivieren von Swapbereichen

Die Swap-Partitionen werden mit Hilfe von `mkswap` erstellt, wobei die beschriebene Struktur angelegt wird, außerdem wird nach fehlerhaften Slots gesucht. Die swap-Partition bleibt nach der Initialisierung mit `mkswap` inaktiv, sie wird erst durch ein beim Neustart aufgerufenes Skript, bzw. im laufenden Betrieb durch die Funktion `swapon()` aktiviert.
Bei ihrem Aufruf wird zunächst überprüft, ob der aktuelle Benutzer berechtigt ist diese Aktion auszuführen. Dann wird die Priorität festgelegt, mit der auf den Swapbereich zugegriffen wird. Auslagerungsbereiche mit höherer Partition werden bevorzugt beschrieben, erst wenn diese voll sind werden auch Bereiche mit niedrigerer Priorität verwendet.
Des Weiteren wird der `swap_header` angelegt und die einzelnen Slots im Swapbereich wie folgt gekennzeichnet: Ein Slot erhält den Zählerwert null, wenn er beschreibbar ist, oder den Wert 32768, wenn er fehlerhaft ist. Der Deskriptor wird dann in die Swapliste eingetragen.

Deaktivieren einer Swappartition:

Ein aktiver Swapbereich wird mittels der Funktion `swapoff()` deaktiviert.

Die ausgelagerten Seiten werden von der Festplatte in den Hauptspeicher zurück geschrieben, wobei die entsprechenden Zugehörigkeiten zu ihren Prozessen von der Funktion `try_to_unuse()` bestimmt werden. Diese Funktion arbeitet sich Slot für Slot durch den Auslagerungsbereich was sehr aufwendig ist.

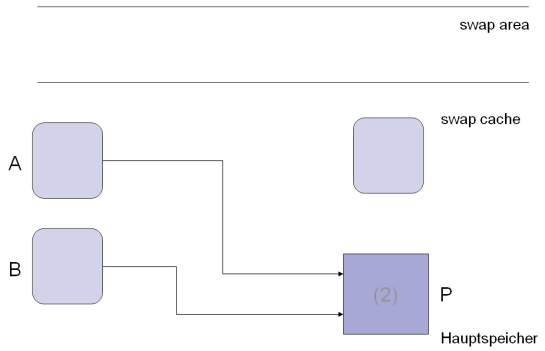
Anfordern und Freigeben eines Slots

Beim Anfordern eines leeren Slots sind zwei Möglichkeiten zu unterscheiden. Zum einen kann nach einem freien Slot beginnend am Anfang des Swapbereichs gesucht werden. Es wird bei jedem Slot überprüft, ob dieser frei ist, wenn nicht wird auf den nächsten Slot übergesprungen. Auf diese Art ist der Auslagerungsbereich immer voll ausgenutzt und es befinden sich kaum leere Slots im belegten Swapbereich.

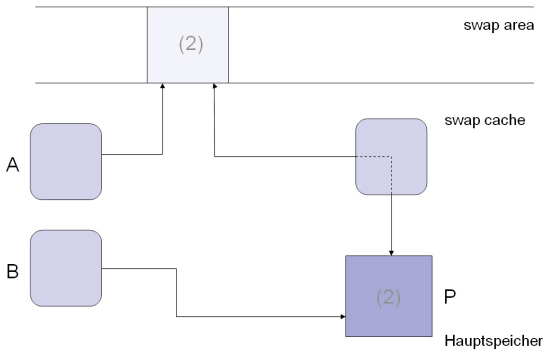
Zum anderen kann die Suche beim zuletzt angeforderten Slot begonnen werden. Diese Methode hat eine stärkere Fragmentierung des Swapbereichs zur Folge, andererseits dauert die Suche nicht so lange wie bei der ersten Möglichkeit.
Linux verwendet aus Gründen der Geschwindigkeit die zweite Variante.

Der Swapcache

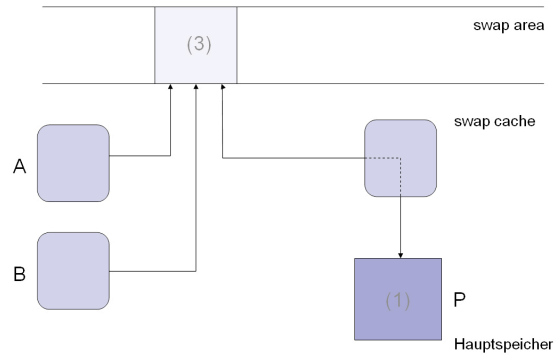
Da meist mehrere Prozesse auf die gleiche Speicherseite zugreifen ergeben sich Probleme, wenn ein Prozess versucht diese gemeinsam genutzte Seite in den Swapbereich auszulagern.
Man könnte dies umgehen, indem man Seiten grundsätzlich entweder nur für keinen oder für alle Prozesse auslagert, was wiederum sehr aufwändig wäre. Linux löst dieses Problem jedoch anders. Es wird ein so genannter Swap Cache eingesetzt, der Seiten vor ihrer endgültigen Auslagerung in den Swapbereich zwischenlagert und alle zugreifenden Prozesse auf die ausgelagerte Speicherseite verweist.



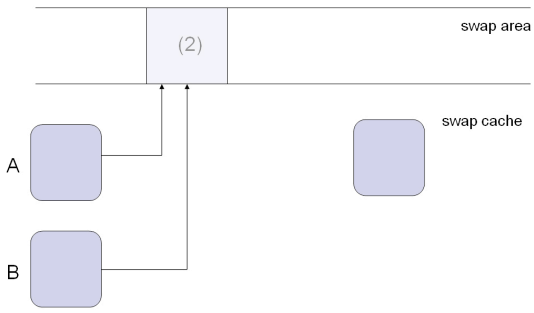
Die Prozesse A und B greifen beide auf die, sich im Hauptspeicher befindliche Seite P zu.



Für den Prozess A nun die Seite in den Swapbereich ausgelagert.
Prozess B greift nach wie vor auf die Seite im Arbeitsspeicher zu.
Der Swap Cache verweist zum einen auf die Seite im Swapbereich als auch auf die Seite im Arbeitsspeicher



Durch den Swapcache ist die Adresse der bereits vom Prozess A ausgelagerten Seite bekannt und die Seiten-Kachelntabelle von Prozess B kann nun ebenfalls auf diese verweisen.



Die Zeigers des Swap Caches werden gelöscht und der Speicher wird freigegeben.

Der Swap Cache kann auf folgende Helferfunktionen zugreifen:

`look_swap_cache()`
findet die entsprechende Seite im Swapbereich, auf die der Identifier passt

`add_to_swap_cache()`
fügt die angegebene Seite in den Swapcache ein

`delete_from_swap_cache()`
löscht die angegebene Seite aus dem Swapcache

`free_page_and_swap_cache()`
entfernt Seiten aus dem Swapcache, wenn diese von keinem Prozess mehr verwendet werden

Swapping Out (Auslagerung)

Hier soll nun der eigentliche Vorgang der Speicherauslagerung verdeutlicht werden. Um freien Arbeitsspeicher zu schaffen wird die Funktion `try_to_swap()` aufgerufen. Sie sucht geeignete Seiten im Arbeitsspeicher und markiert diese mittels der Funktion `mark_page_accessed()`. Anschließend wird die markierte Seite gesperrt und der zugehörige Eintrag aus der Seitentabelle gelöscht. Die Seite wird nun in den Swapcache geschrieben und bei Bedarf werden ausreichend freie Slots im Swapbereich angefordert. Die nun ausgelagerte Seite wird wiederum in der Seitentabelle vermerkt.

Swapping In (Zurückschreiben)

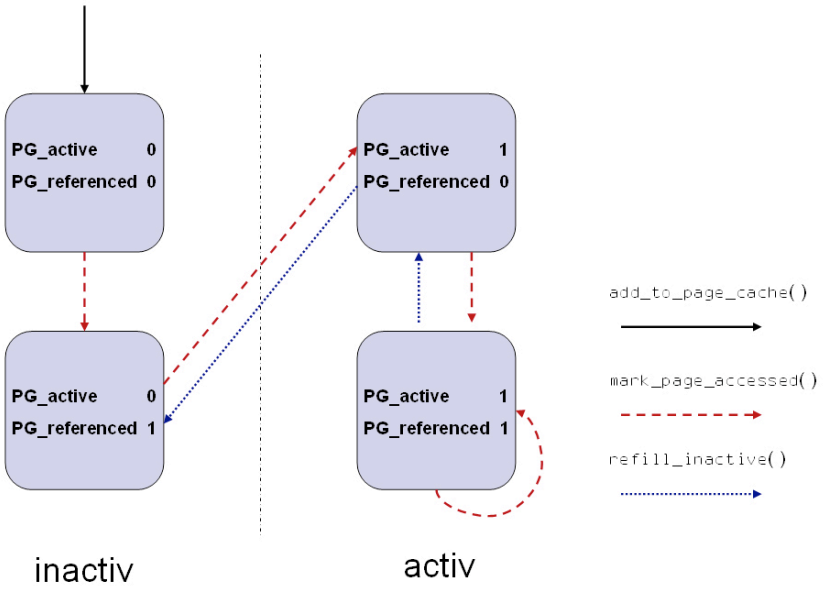
Den Vorgang, bei dem eine ausgelagerte Seite wieder zurück in den Arbeitsspeicher geschrieben wird bezeichnet man als Swapping In. Dies ist notwendig, wenn Prozesse wieder auf die ausgelagerte Seite zugreifen wollen. Das erneute Einlagern der Seite in den Arbeitsspeicher wird durch die Funktion `do_swap_page()` realisiert.
Diese ruft die folgenden Unterfunktionen auf. Zum einen `swapin_readahead()`, sie ist notwendig, um aus dem Swapbereich lesen zu können. Dann folgt die Funktion `read_swap_cache_async()`, welche Voraussetzung ist um eine Seite einlagern zu können. Zum Schluß wird die Seite im Swap Cache mit `mark_page_accessed()` gekennzeichnet, gesperrt und anschließend gelöscht, so fern kein anderer Prozess sie mehr referenziert. Die Seitentabelle wird aktualisiert und der Slot wieder freigegeben

Speicherrückgewinnung

Es gibt zahlreiche Möglichkeiten zu entscheiden, welche Seite im Arbeitsspeicher bleiben darf und welche auf einen Swapbereich ausgelagert werden soll.

Unter Linux wird diese Entscheidung mit Hilfe der LRU (least recently used) Strategie getroffen.
Ausgehend davon, daß es Seiten gibt, die häufiger abgefragt werden als andere wird eine Liste geführt, welche Speicherseiten zuletzt in Verwendung waren. Diese Seiten werden im Arbeitsspeicher behalten, Seiten, die nicht so oft verwendet werden kommen in den Swapbreich.
Die LRU-Liste besteht wiederum aus zwei getrennten Listen (`activ` und `inactiv`)

Eine sich nicht im Arbeitsspeicher befindliche Seite wird zunächst auf die `inactiv` Seite geschrieben. (`add_to_page_cache()`)Bei ihrem ersten Aufruf wird das `referenced` Bit gesetzt (`mark_page_accessed()`), die Seite bleibt jedeoch in der `inactiv`-Liste. Wird nun nochmals auf die Seite zugegriffen, so wird die Seite in die `activ` Liste geschrieben, wird das `activ` Bit gesetzt und das `referenced` Bit wiederum gelöscht. Solange auf sie zugegriffen wird, wird bleibt die Seite in der `activ`-Liste.
Wird die Seite über einen längeren Zeitraum nicht mehr referenziert, so wird das `referenced` Bit gelöscht, die Seite bleibt jedoch noch in der `aktiv` Liste. (`refill_inactive()`)
Bleibt die Seite noch längere Zeit inaktiv, so wird das `activ` Bit gelöscht, das `referenced` Bit wieder gesetzt und die Seite in die `inactiv` Liste geschrieben.



Die `try_to_free_page()` Funktion

Um Seiten wieder frei zu geben wird die Kernelfunktion `try_to_free_page()` aufgerufen. Sie wiederum ruft eine Funktion namens `shrink_caches()` mit bei jedem Durchlauf steigender Priorität auf um die angegebene Anzahl von Seiten freizugeben. Schafft es die Funktion nicht, den geforderten Speicher freizumachen, so wird ein Prozess im Usermode beendet
Die Funktion `shrink_caches()` realisiert das eigentliche Freigeben von Seiten, indem sie auf der `inactiv`-Seite nach geeigneten Speicherseiten sucht.

Quellenverzeichnis

Understanding the Linux Kernel (2nd Edition)
Daniel P. Bovet, Marco Cesati, 2002, Sebastopol

Galileo Computing – Swapping
(aus der Reihe „Wie werde ich ein Unix-Guru?“)
www.galileocomputing.de, Galileo Press GmbH, 2003, Bonn

Linux Paging, Caching und Swapping
Timo Schreiber, 2000, Uni Karlsruhe

Understanding The Linux Virtual Memory Manager
Mel Gorman, 2003, University of Limerick