8. Speicherverwaltung

Alle Beispiel-Quellen mittels SVN unter:

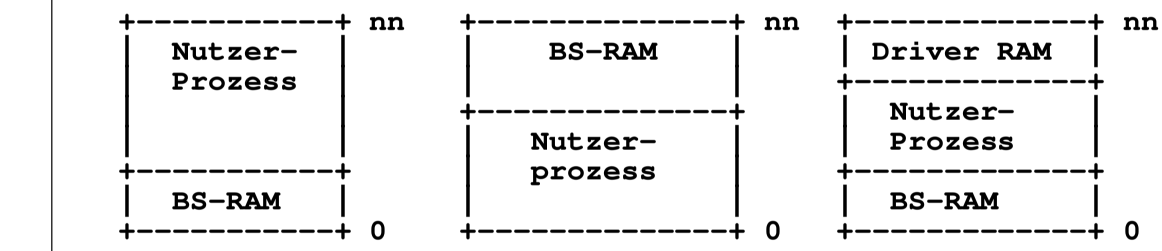
https://svn.informatik.hu-berlin.de/svn/unix-2014/Memory

8.1 Speicherverwaltung - Vorbemerkungen

8.1.1. Monoprogrammierung

- einfachste HS-Verwaltung (für kleine Speicher)
- wird von einfachen Mikrocomputer und Betriebssysteme (CP/M)
- nur ein Prozess kann laufen

Möglichkeiten der Realisierung



8.1.2. Multiprogrammierung

bei größeren Speichern zur besseren Nutzung der Prozessorzeit
(höhere Auslastung oder (besser) höherem Durchsatz)
HS wird in Partitionen aufgeteilt.

Einige akademische Betrachtungen zur Multiprogrammierung

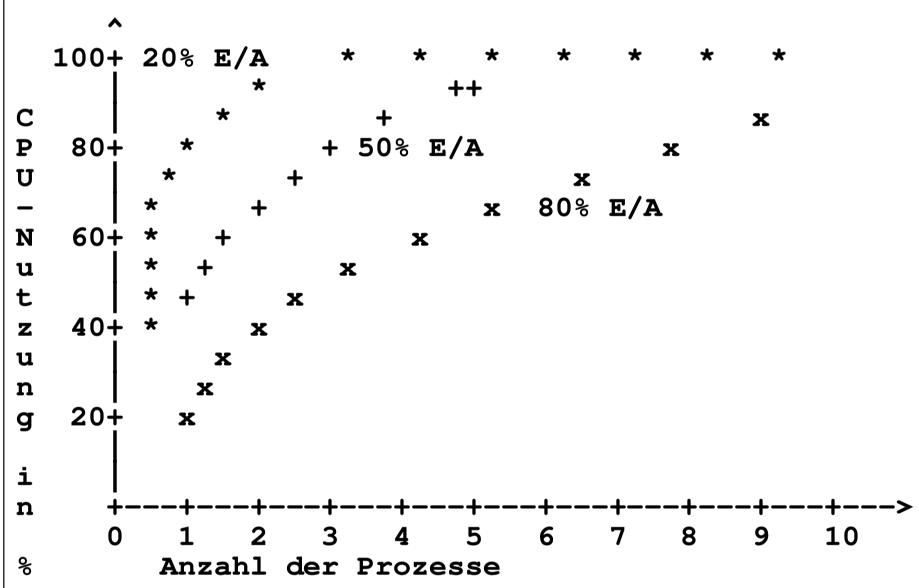
A) Prozessorauslastung

Berechnung:

a) 20% CPU, 80% E/A (100%:20%=5 --> 5 Prozesse == volle Auslastung)
unrealistisch; setzt voraus, daß immer ein Prozess arbeiten kann
und nie alle 5 gleichzeitig warten

b) 0<=p<=1 - p Wahrscheinlichkeit mit der ein Prozess auf eine E/A-Operation wartet,
n ist die Anzahl der Prozess, die den Prozessor benutzen

CPU-Auslastung == 1- P1*P2*P3*P4*...*Pn oder für gleiche Prozess
== 1- P (hoch) n



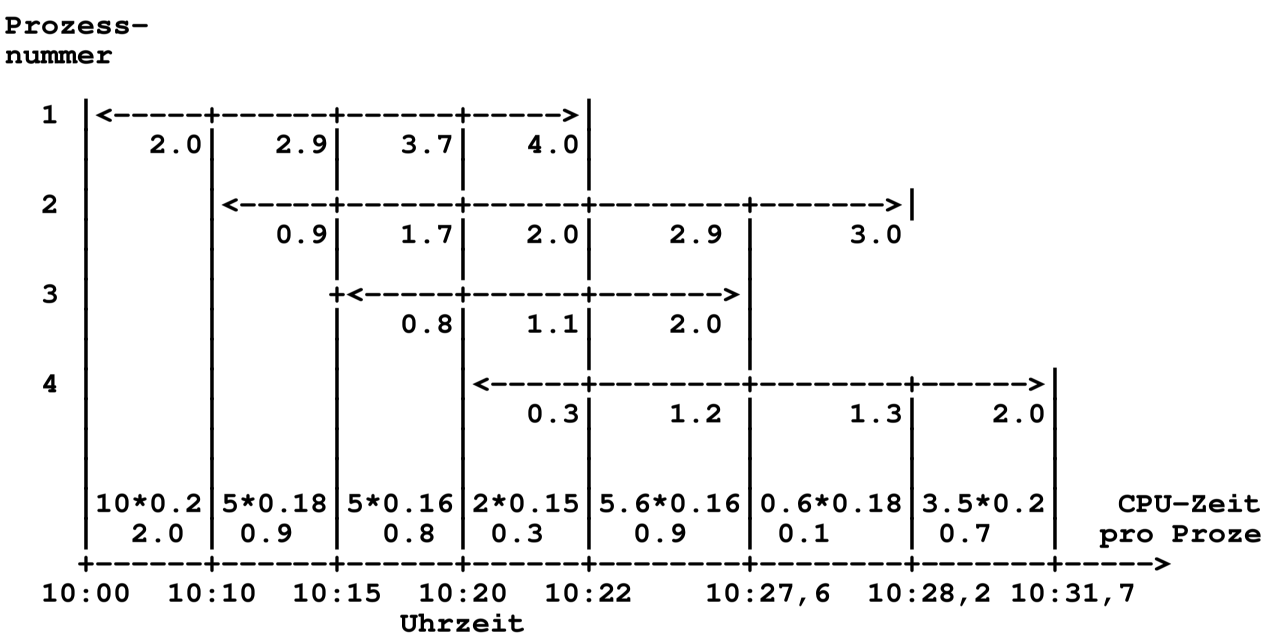
B) Verweilzeitberechnung (Interessant für Nutzer)

Beispiel:

4 Prozess mit je 20% CPU-Belastung

	Startzeit	CPU-Minuten	Endzeit
1	10:00	4	10:22
2	10:10	3	10:28.2
3	10:15	2	10:27.6
4	10:20	2	10:31.7

Zahl der Prozesse	1	2	3	4
CPU wartet	0.80	0.64	0.51	0.41
CPU belegt	0.20	0.36	0.49	0.59
CPU/Prozess	0.20	0.18	0.16	0.15



8.1.3.Multiprogrammierung mit fester Zahl von Partitionen

Hauptspeicher wird in Partitionen fester Größe aufgeteilt.
Einfachste Form der Multiprogrammierung. Mit fester oder variabler Priorität der einzelnen Partitionen.
Die einzelnen Tasks werden entsprechend ihres Speicher- und CPU-Zeitbedarfs in Klassen aufgeteilt. Die Klassen werden einzelnen Partitionen zugeordnet.

Unterscheidungen:

a) mit gleichgroßen Partitionen
   gut für Verwaltung
   schlecht für HS-Auslastung, Partitionsgröße == Bedarf der größten Task

b) mit unterschiedlichen Partitonsgrößen
   (DOS, OS/MFT)
   a) mit einer Eingabewarteschlange für alle Partitionen
      OS/MFT
      Nachteil: Hoher Verwaltungsaufwand
      Vorteile: gute Auslastung des HS, kleine Tasks können überall laufen
   b) mit einer Eingabewarteschlange für jede Partition
      DOS
      Nachteil: schlechte Auslastung des HS
      Vorteile: einfache Verwaltung

Probleme:

a) Die Verschieblichkeit der Tasks

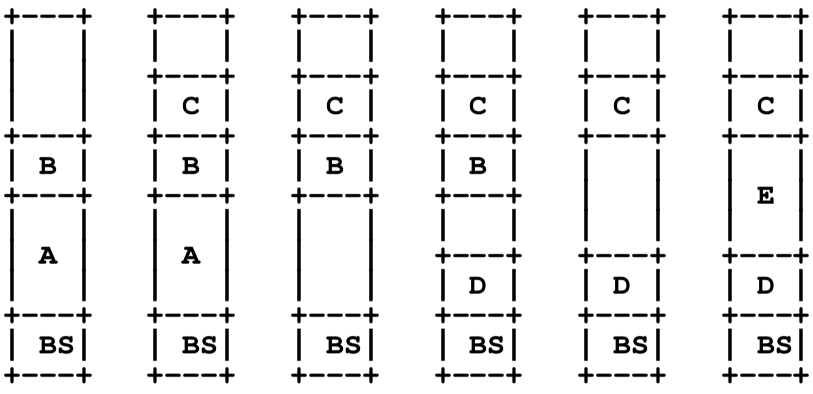
notwendig bei mehreren Warteschlangen, nicht notwendig bei mehreren Warteschlangen
Lösung:
a) selbstverschiebliche Programme
b) Loader (linken zur Ausführungszeit)

b) Speicherschutz

mehrere Möglichkeiten:
a) Speicherschutzschlüssel (IBM)
b) Adressmapping (PDP 11, MMU)

8.1.4.Multiprogrammierung mit variabler Zahl von Partitonen

Hauptspeicher wird während der Arbeit dynamisch in verschieden große Partitionen aufgeteilt, je nach Bedarf.



z.B.: OS/MVT, UNIX in Gründerzeit

Probleme:

a) Hauptspeicheranfangszuweisung
Wieviel Speicher bekommt eine Partition/Prozess bei seinem ersten Eintritt?
– nicht zu viel – Verschwendung
– nicht zu wenig – Verwaltungsaufwand

b) Vergrößerung von Prozessen
   Normalerweise dehnt sich ein Prozess bei der Arbeit aus.
   Woher den zusätzlichen Speicher nehmen?
   a) er bekommt zu Beginn bereits den maximalen Speicher
   b) kleine Erweiterungen werden bereits vorher eingeplant
      und profilaktisch freigehalten
   c) Auslagern und mit größerem Speicher Einlagern
c) Möglichkeiten der Speicherverwaltung
   a) Bitmaps
      Der Speicher wird in kleine einzeln zu vergebende, gleiche Einhe
      geteilt, pro Einheit mindestens ein Eintrag (Bit) in einer Tabel
      Je kleiner die Einheiten, je länger die Bitmap und je länger
      die Suchzeiten und umgekehrt.
   b) Listen
      Speicher wird wie bei a) geteilt. Einheiten werden zu einer
      Gruppe zusammengefaßt, wenn sie zusammenhängend im Speicher
      liegen. Pro Einheitengruppe wird ein Eintrag in einer Liste
      geführt. Eintrag: Kennzeichen, Anfangsadresse, Länge,
      Verweis zum Nutzer(Prozess).
      Probleme: Zusammenlegen, finden eines passenden Stückes im
                Speicher: first-fit - erste
                          next-fit   - nächste
                          best-fit   - passende
                          quick-fit - schnellstes zu findende
   c) Buddy-Systems
      Aufteilung des Speichers entsprechend 2er Potenzen und
      daraus passende Stücken auswählen.
d) Swappingalgorithmen
   a) Bereitstellung des Swapspace auf dem externen Medium? Wann? Wo?
   b) Wann wird ein Prozess ausgelagert?

8.1.5.Virtuelle Speicher

Ein Prozess wird ein virtueller Adressraum zur Verfügung gestellt, der größer oder kleiner als der reale Speicher sein kann. Ein spezieller Algorithmus sorgt dafür, daß immer die benötigten Adressbereiche im Hauptspeicher sind.

a) Paging
   Dazu wird der HS in Frames eingeteilt (z.B. 2KB), der in der Lage ist genau eine Seite aufzunehmen. Virtuell wird mittels Seitennumme adressiert, die in eine Framenummer beim Zugriff umgerechnet wird. Dies kann soft- oder hardwaremäßig geschehen. Die Zahl von Frames, die ein Prozess mindestens bei der Arbeit benötigt wird durch die Hardware bestimmt: z.B. IBM370 8 Frames

```
       EX  Adresse
        1      2
           MOV    Adresse1     Adresse2
            3                   4
              Feld1           Feld2
              5   6           7   8
```

b) Segmentation
   ähnlich Paging, aber mehrere Seiten werden logisch zu einem Segment zusammengefaßt. Den Segmenten kann eine spezielle Bedeutung bzw. Zugriffsrechte gegeben werden.

Probleme:

Replacementalgorithmus:

Wann kann welche Seite im Speicher ersetzt werden?
- **Verweildauer**
- **Nutzungshäufigkeit**
- **Einfachheit der Auslagerung**

Algorithmen:
- **FIFO - first in first out**
- **LRU - least recently use**

Pagesize?
- **512, 1 K, 2 K, 4 K**

8.2 Systemrufe zur Speicherverwaltung

Implizite Speicherverwaltung

- **fork()**
- **exec()**
- **exit()**
- **mmap()**
- **munmap()**

Explizite Speicherverwaltung

- **brk()**
- **sbrk()**
- **getpagesize()**
- **mprotect()**
- **mcntl()**
- **plock()**

```
#include <unistd.h>
int  brk(void *end_data_segment);

Der Systemruf  brk()  setzt das Ende des Datensegments auf die durch
end_data_segment   spezifizierte Adresse. Der Wert  end_data_segment
zuvor auf das nächste Vielfache der Seitengröße gesetzt. end_data_segm
muß größer als das aktuelle Ende des Datensegments sein.

Rückkehrwert:
  0  -  ok
 -1  -  Fehler
        ENOMEM  -  kein Speicherplatz mehr

-------------------------------------------------------------------------

#include <unistd.h>
void *sbrk(ptrdiff_t increment);

Der Systemruf  sbrk()  setzt das Ende des Datensegments um  increments
Bytes herauf. Es werden mindestens die geforderte Anzahl von Bytes
bereitgestellt. Wenn  increments  negativ ist, wird die Zahl der Bytes
im Datensegment verrringert.

Rückkehrwert:
  !=NULL  -  Adresse des Beginns der  increment  Bytes
    NULL  -  kein Speicherplatz mehr
```



```
#include <unistd.h>
size_t getpagesize(void)

Der Systemruf  getpagesize()  liefert die Zahl der Bytes in einer
Seite.

Rückkehrwert:
  Anzahl der Bytes in einer Seite.

-------------------------------------------------------------------------

#include <sys/mman.h>
int mprotect(const void *addr, size_t len, int prot);

Der Systemruf  mprotect()  dient zum Ändern der Zugriffsrechte eines
mit dem Systemruf  mmap()  eingebundenen Adressbereiches.
addr  gibt die Adresse des Bereiches an,   len  die Lanege und
prot  die Zugriffsrechte.

prot:
  PROT_READ   /* page can be read */
  PROT_WRITE  /* page can be written */
  PROT_EXEC   /* page can be executed */
  PROT_NONE   /* page can not be accessed */

Rückkehrwert:
    0  -  ok
   -1  -  Fehler
          EACCES, EINVAL, ENOMEM
```

```
#include <sys/types.h>
#include <sys/mman.h>      /* BSD */
int mctl(caddr_t addr, size_t len, int function, void *arg);

Der Systemruf  mctl()  dient zur Ausführung einer Reihe von Steuer-
operationen über die Seiten im Speicher. Er ist Programmen mit
dem UID 0 vorbehalten.
addr  spezifiziert den Beginn des Speicherbereiches,  len  die
Lanege.  function  und  arg  beschreiben die Steueroperation selbst.

Steueroperationen:
      MC_LOCKAS   -   festhalten aller Seiten im Hauptspeicher (lock)
                      (arg==MCL_CURRENT,MCL_FUTURE)
      MC_UNLOCKAS -   freigeben aller Seiten
                      zum Swappen (unlock)(arg==0)
      MC_LOCK     -   der angegebene Speicherbereich wird im Speicher
                      fixiert
      MC_UNLOCK   -   aufheben der Fixierung
      MC_SYNC     -   Synchronisieren der Seiten mit dem Hintergrund-
                      speicher (bei mmap())

Rückkehrwert:
  0  -  ok
 -1  -  Fehler
        EAGAIN, EINVAL, EPERM
```



```
#include <sys/lock.h>       /* AT&T V.4 */
int plock(int operation);

Der Systemruf  plock()  ermöglicht es Prozessen mit effktivem UID 0
die Auslagerung des Datensegment bzw. Textsegments zu verbieten bzw.
zu erlauben. operation  spezifiziert die entsprechende Funktion.

       PROCLOCK   - Prozesslock (lock Text- und Datensegment)
       TXTLOCK    - lock Textsegment
       DATLOCK    - lock Datensegment
       UNLOCK     - aufheben der vorangegangenen Locks

Rückkehrwert:
    0 - ok
   -1 - Fehler
        EPERM, EINVAL, EAGAIN
```