



Betriebs- systeme II

FH München
WS 2006/07
Hans-Georg Eßer
hans-georg.esser@fhm.edu

Die Folien zu „Betriebssysteme II“ basieren auf den Ausarbeitungen von
Prof. Christian Vogt, <http://www.cs.fhm.edu/~vogt> und
Prof. Claudius Schnörr, <http://www.cs.fhm.edu/~schoerr>

Motivation

- What does a **memory address** consist of?
- What happens when accessing an address?
- How can administrators and developers use their knowledge about memory management?
 - How does **Shared Memory** work?
 - What are **memory-mapped files**?
- How does a **Segmentation Fault** occur?
- How to create a **virtual address space**?

Content Overview

- Contiguous memory allocation
 - fix-sized partitions
 - variably-sized partitions
 - methods for administration of free memory
 - segmentation
- Non-contiguous memory allocation
 - Virtual memory management (paging)
 - multi-level paging
 - segmentation plus paging
- Demand paging
 - Page Faults and what to do about them
 - page replacement strategies
 - further design possibilities
- Swapping

Kinds of memory management

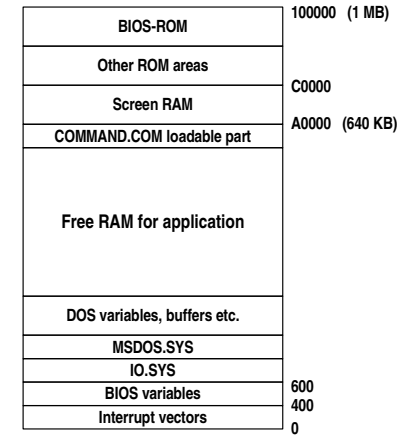
Two principle types

- **Contiguous memory management**
 - Whenever a process requests some memory, OS must satisfy this request with a contiguous memory block
- **Non-contiguous memory management**
 - OS can reserve several smaller memory blocks, giving the requested size when added up
 - Searching the (distributed) process' memory is an OS task.

Kinds of memory management

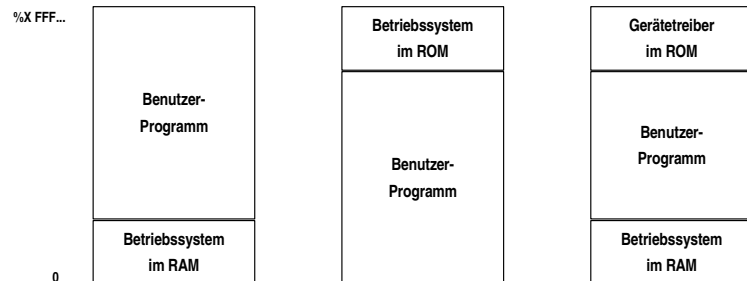
- Today: Main memory almost always **non-contiguous** (virtual memory management).
- Cases of contiguous memory management:
 - disk space management,
 - management of space in page and swap files

Memory usage in MS-DOS



Single tasking without swapping or paging

- Divide memory (RAM and ROM) into partitions for
 - the operating system
 - a (one) user program
- When program finishes: Reuse its memory area for next program

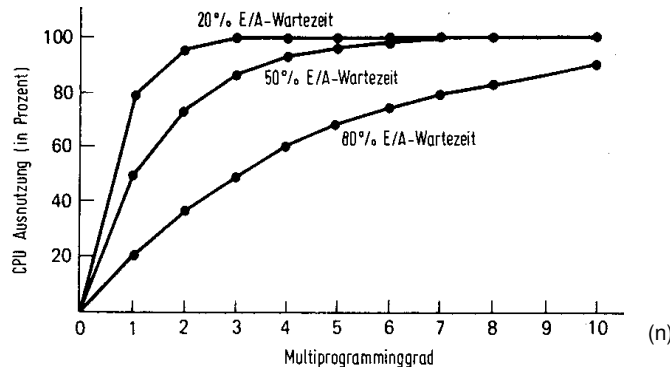


Multiprogramming

- Programs spend a lot of their time waiting (for I/O etc.).
- **Swapping** to disk every time is inefficient.
- Solution: several programs in main memory simultaneously
- Preconditions: **movable (relocatable) code** and **memory protection**

Multiprogramming

- When n programs spend a fraction p of their time waiting, then CPU usage is $= 1 - p^n$.



Relocator for Z80, 19 years old



für 464-664-6128

```

1000 ***** [1164]
1010 *** [51]
1020 *** RELOCATOR V1.0 [51]
1030 *** [51]
1040 *** [51]
1050 *** Geschrieben in *** [2098]
1060 *** Oktober 1987 von *** [860]
1070 *** [51]
1080 *** Hans-Georg Eßer *** [687]
1090 *** [51]
1100 ***** [1164]
1110 [117]
1120 [1209]
1130 [1139]
1140 MODE 2:ZONE 3 [1139]
1150 WINDOW #1,46,76,7,10 [1323]
1160 WINDOW #2,46,76,12,38 [1323]
1170 FOR #1 TO 2:PEN #N,D:PAPER #N,1:NEXT [2184]
1180 PRINT CHR$(24);STRING$(0,216); [6496]
Relocator V1.0 (c) 1987 by Hans-Geor
g Eßer Relocator V1.0 ;STRINGS
(60,218);CHR$(24) [730]
1190 LOCATE 1,22;STRING$(0,216); [8528]
Relocator V1.0 (c) 1987 by Hans-Geor
g Eßer Relocator V1.0 ;STRINGS
(60,218);CHR$(24) [1040]
1210 WINDOW 1,46,5,20 [576]
1220 CLS #1:CLS #2 [2624]
1230 LOCATE #1,2,2:PRINT #1,"Aktuelle Adre [2624]
ese:" [576]
1240 LOCATE #2,2,2:PRINT #2,"Geänderte Ad [3236]
ressen:";PRINT #2 [4640]
1250 LOCATE 5,3:LINE INPUT "Name des Binae [4640]
r-Files > ",file$ [4640]
1260 LOCATE 5,5:INPUT "Ursprungsadresse [4640]
> ",adr$ [3793]
1270 LOCATE 5,7:INPUT "Programm-Laenge (in [3793]
Bytes) > ",length [3793]
1280 LOCATE 5,9:INPUT "Neue Adresse [3384]
> ",adr2$ [3384]
1290 LOCATE 5,11:LINE INPUT "Name des neue [2549]
n Files > ",file2$ [5399]
1300 LOCATE 5,14:PRINT "Relokation wird du [5399]
rchgefuehrt." [1453]
1310 Einlesen [1453]
1320 MEMORY adr2-1 [1408]
1330 LOAD file$,adr1 [568]
1340 diff=adr2-adr1 [4774]
1350 alle Adressen muessen um diff nach
oben verschoben werden. [2180]
1360 FOR #adr1 TO adr1+length [2180]
1370 LOCATE #1,20,2:PRINT #1,"&";HEX$(n,4) [2013]
1380 GOSUB 1480 ' ueberpruefen und ggf. ae [2409]
ndern [2409]
1390 NEXT n [366]
1400 dummy="dummy.dum" [1270]
1410 BAVE dummy,b,adr1,length [1587]
1420 MEMORY adr2-1 [126]
1430 LOAD dummy,adr2 [683]
1440 ERASE dummy [988]
1450 BAVE file2$,b,adr2,length [1590]
1460 LOCATE 5,14:PRINT "Die Relokation wur
de durchgefuehrt." [5148]
1470 END ' Programm-Ende [882]
1480 Aktuelle Adresse ueberpruefen [933]
1490 pr#HEX$(PEEK(n),2) [1242]
1500 byte=VAL("&"+HEX$(PEEK(n+2),2)+HEX$(P
EEK(n+1),2)) [2760]
1510 RESTORE 1830 [920]
1520 rde="" [183]
1530 WHILE rde<>"XX" [1942]
1540 READ rde [483]
1550 IF pr#rde THEN 1840 [760]
1560 WEND [390]
1570 ' Untersuchung auf 2-Byte-Befehl [2528]
1580 pr#HEX$(PEEK(n),2)+HEX$(PEEK(n+1),2) [1841]
1590 byte=VAL("&"+HEX$(PEEK(n+3),2)+HEX$(P
EEK(n+2),2)) [1647]
1600 RESTORE 1840 [886]
1610 rde="" WHILE rde<>"XXXX";READ rde:IF [2416]
rde#> THEN 1730 [1647]
1620 WEND [390]
1630 RETURN [555]
1640 ' Bezug auf Adresse innerhalb des Pro
gramms ? [3010]
1650 IF byte=adr1 OR byte=adr1+length THEN [2412]
RETURN [1814]
1660 Verschieben [815]
1670 byte=byte+diff [564]
1680 b#HEX$(byte,4) [1060]
1690 POKe #2,VAL("&"+LEFT$(b$,2)) [1198]
1700 POKe #1,VAL("&"+RIGHT$(b$,2)) [1323]
1710 PRINT #2," &";HEX$(n,4), [2014]
1720 RETURN [555]
1730 ' 2-Byte Befehl! auswerten [1842]
1740 IF byte=adr1 OR byte=adr1+length THEN [2412]
RETURN [815]
1750 Verschieben [564]
1760 byte=byte+diff [564]
1770 b#HEX$(byte,4) [1060]
1780 POKe #3,VAL("&"+LEFT$(b$,2)) [892]
1790 POKe #2,VAL("&"+RIGHT$(b$,2)) [764]
1800 PRINT #2," &";HEX$(n,4), [2014]
1810 RETURN [555]
1820 ' Daten [442]
1830 DATA CD,DC,FC,D4,C4,F4,EC,E4,CC,C3,DA [6017]
F4,D2,CC,F2,EA,E2,CA,EA,3A,01,11,21,2A,31
,32,22,XX [6017]
1840 DATA DD6,ED6,DD1,DD2A,FD21,FD2A,ED [3306]
7B,ED43,ED53,DD22,FD22,ED73,XXXX [6017]
Listing Relocator
    
```

[no, no, I won't translate this slide...]

(Nicht ganz ernst gemeinter) Literaturhinweis:

H.-G. Eßer: *MC-Relocator – Programmverschiebung leicht gemacht*, Schneider CPC International 03/1988, DMV-Verlag, S. 67 ff.

Code relocation: Z80 assembler

```

#base 0xA000 ; Program starts at 0xA000

ld hl,data
ld a,(hl)
call print
ret

;print
cmp a,' '
jne ende
call OS_PRINT
:ende
ret

;data
char 'x'

A000 ld hl,0xA011 ; data
A003 ld a,(hl)
A004 call 0xA008 ; print
A007 ret

; Function print
A008 cmp a,' '
A00A jne 0xA010 ; ende
A00D call OS_PRINT
; Label ende
A010 ret

; Data
A011 char 'x'
    
```

Code relocation: Z80 assembler

Problems with relocation:

- What is an address?
 - jump (branch) and call targets
 - accesses to data areas
- What just happens to look like an address?
- Multiply indirect addressing

(„load the value from the memory location whos address is in this memory location“)

Better: Loader plans relocation in advance, memorizes where addresses occur

Code relocation and memory protection (1/2)

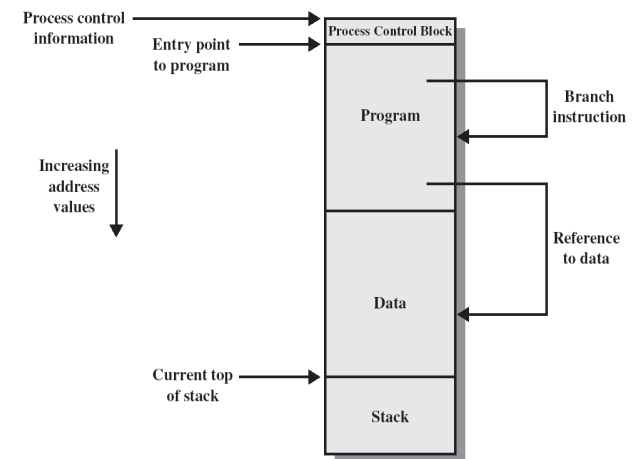
- Program must be able to execute no matter where in the memory it is loaded.

Two possibilities:

- Linker remembers which code blocks contain absolute references. When loading the program it modifies them.
- Machine has a special hardware register, a **basis register**

Every time an address is used (at runtime), the basis register is added to the address.

Code relocation



Code relocation and memory protection (2/2)

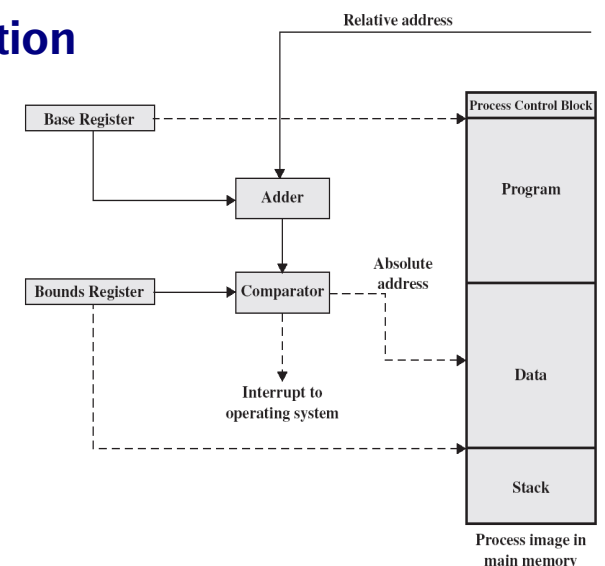
- Programs must not access other processes' memory areas.

Two ways to assure this:

- protection code
- Machine has (yet another) hardware register: **limit register**

By checking the limit register OS can find out if the memory access is valid (i.e. inside the reserved address area).

Support for code relocation and memory protection



```

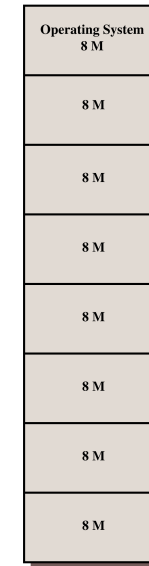
Sep 19 14:20:18 amd64 sshd[20494]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:03 amd64 /usr/sbin/cron[30323]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[6077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10401]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[19781]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5499]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[11888]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 /usr/sbin/cron[14884]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 18:04:05 amd64 sshd[9594]: Accepted publickey for esser from ::ffff:192.168.1.15 port 59772 user [esser]
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12434]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_user: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[6021]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[14884]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[6921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

```

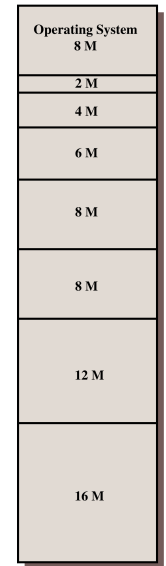
Contiguous Memory Management

Split into fixed partitions

- Equal size
 - Waste memory when there are many small programs
 - Programs fit in any partition
- Unequal size
 - Better memory usage
 - Possibly inapt allocation



(a) Equal-size partitions



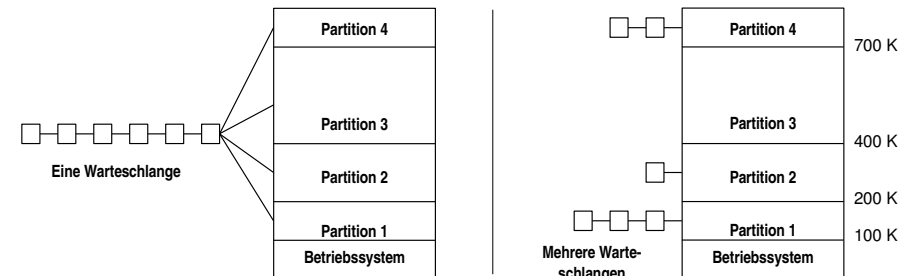
(b) Unequal-size partitions

Split into fixed partitions

- Create memory partitions of fixed (equal or different) size.
 - Assign a process to a free partition.
- Alternatives:
- First program that fits in the free partition (one queue)
 - FIFO for each partition (several queues)
 - Largest program that fits in the free partition
 - Disadvantage: small programs postponed
 - Solution: postpone program no more than k times

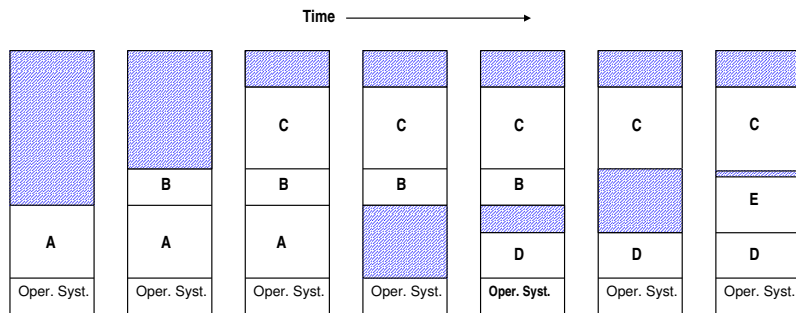
Split into fixed partitions

- Large free spaces within a partition can occur: **internal fragmentation.**



Split into variable partitions (1)

- Configure number and size of partitions dynamically.



Split into variable partitions (3)

What happens when process needs more memory?

- Increase partition size if a neighboring partition is free.
- Allocate a larger (free) partition and move program there.

Split into variable partitions (2)

- Possibly many small areas (holes) in memory remain unused.
external fragmentation.
- Move partitions to get rid of the holes (**memory compaction**).

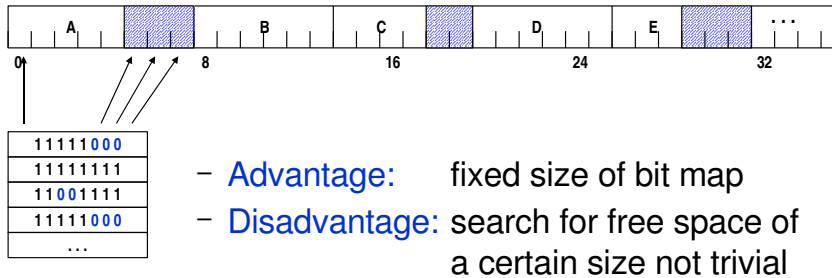
Split into variable partitions (4)

- When there is no sufficiently large partition, one or several process must be **swapped** to disk.
- Alternative: Allocate more memory than was requested by the process. Leads to
 - internal fragmentation
 - If larger memory block is not sufficient either, one of the other methods must be applied.

Which parts of the memory are free?

a) Bit Maps

- partition memory in units (some bytes to several KByte)
- One bit per unit: 0 when unit is free
1 when unit is in use



Which parts of the memory are free?

b) Linked Lists

- Chained list with descriptions of memory areas:
 - used by a process (P) or free (H=hole)
 - start address and length of area
 - pointer to next entry

