OPERATING SYSTEMS
DESIGN AND IMPLEMENTATION
Third Edition
ANDREW S. TANENBAUM
ALBERT S. WOODHULL

Chapter 4
Memory Management

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Monoprogramming without Swapping or Paging

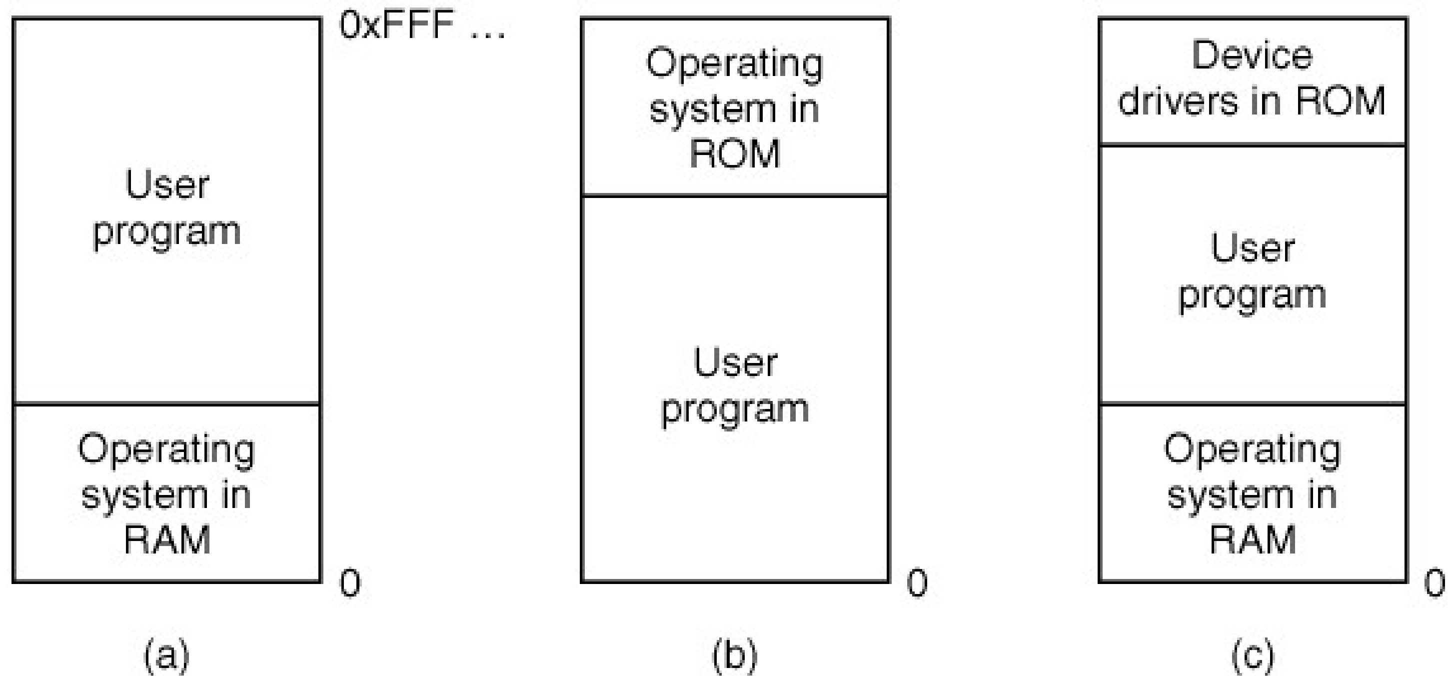
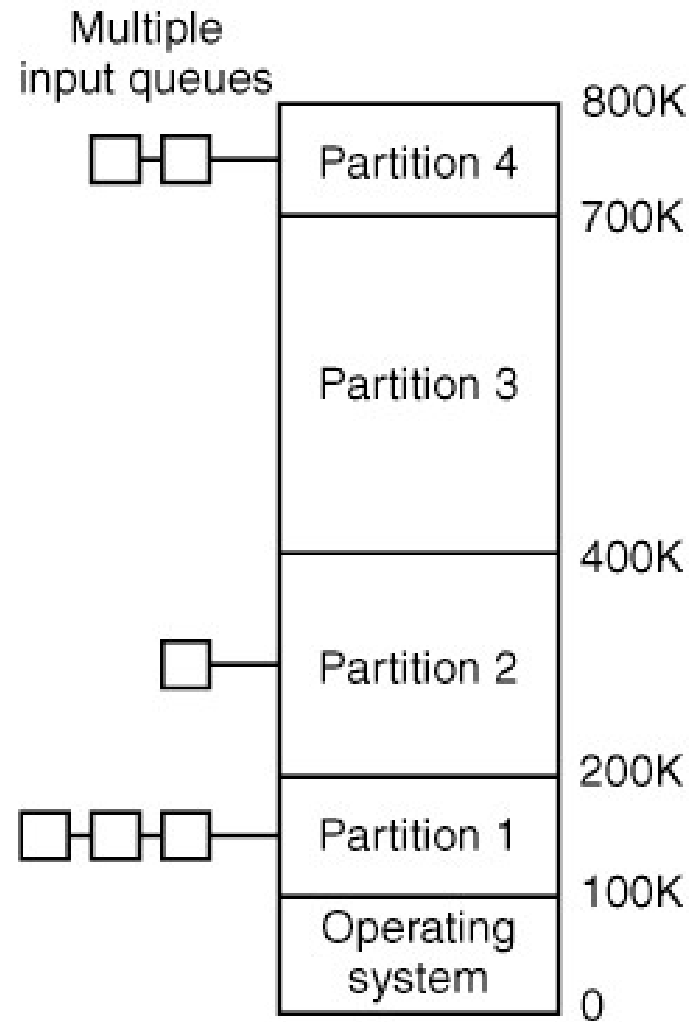


Figure 4-1. Three simple ways of organizing memory with an operating system and one user process. Other possibilities also exist.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Multiprogramming with Fixed Partitions (1)

Figure 4-2. (a) Fixed memory partitions with separate input queues for each partition.



Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Multiprogramming with Fixed Partitions (2)

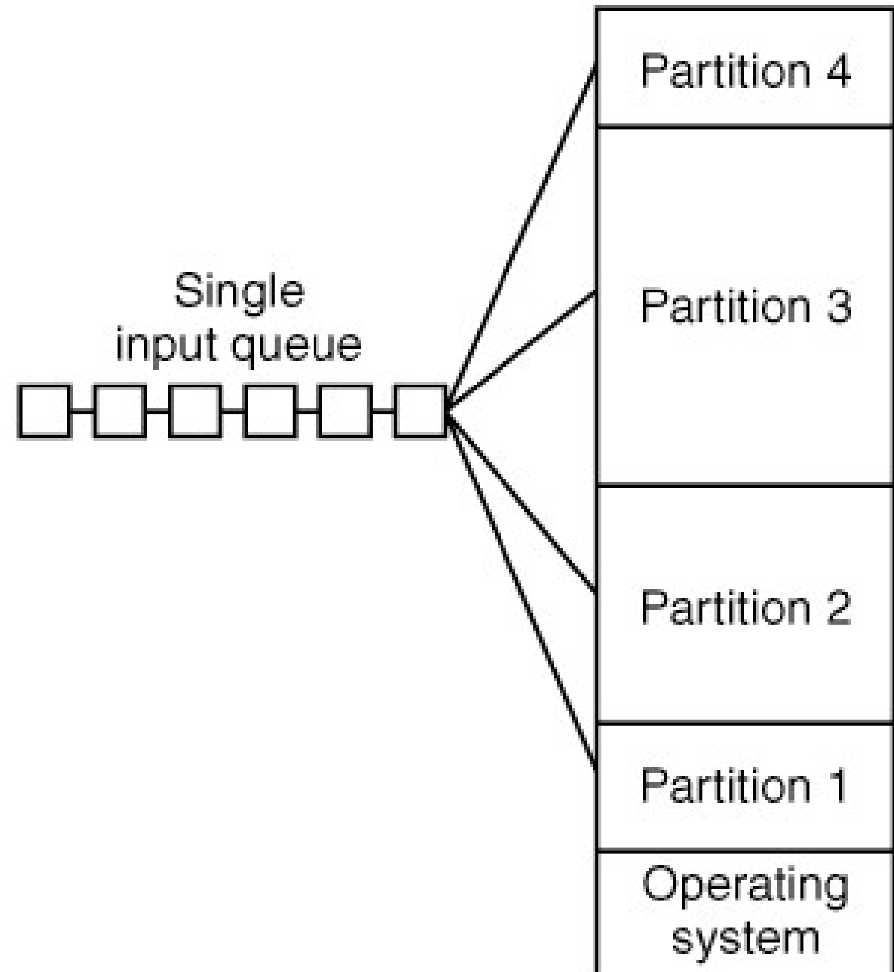


Figure 4-2. (b) Fixed memory partitions with a single input queue.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Swapping (1)

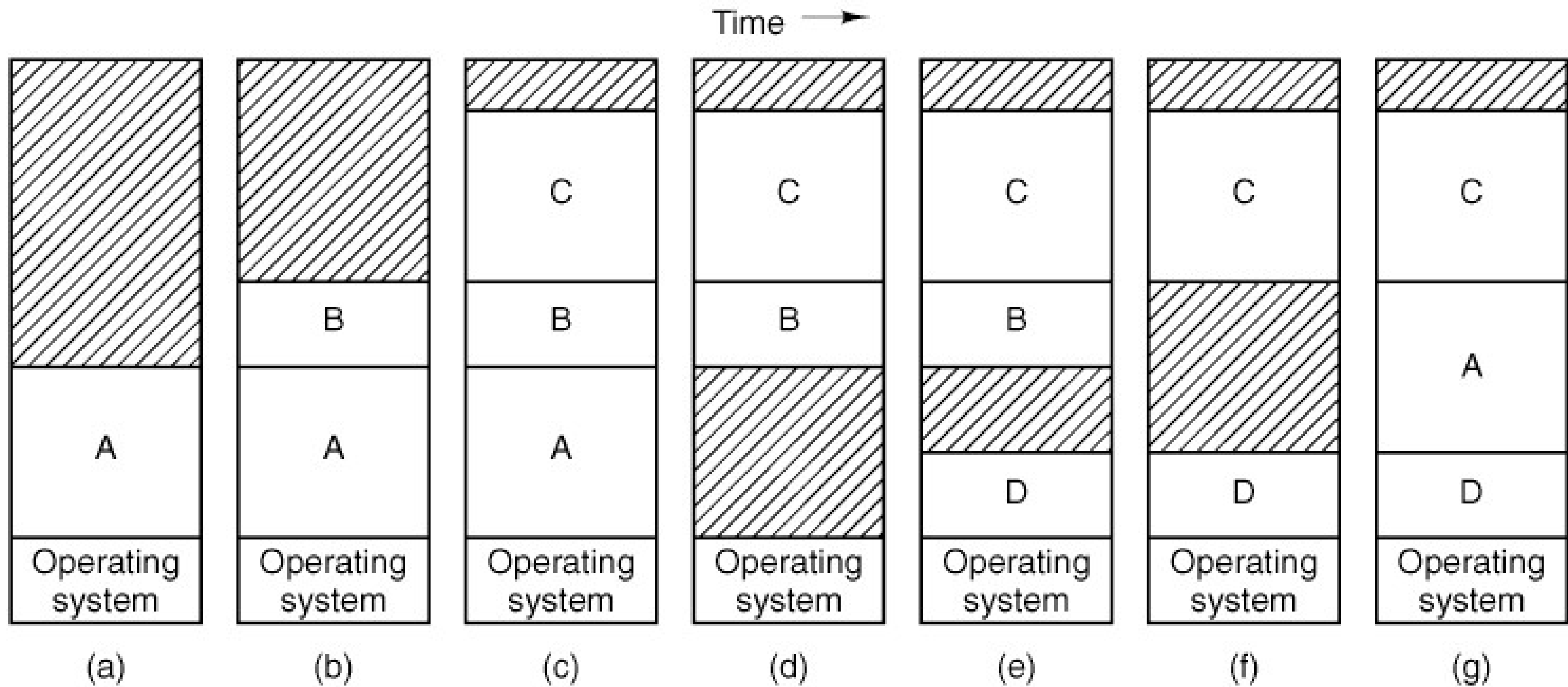
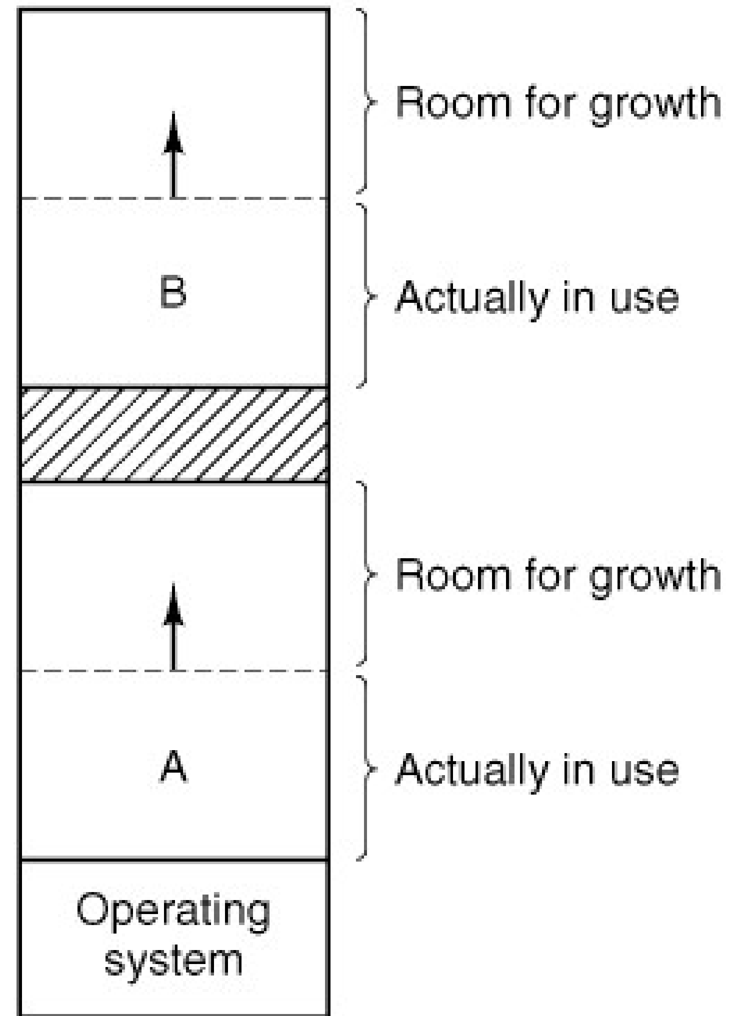


Figure 4-3. Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Swapping (2)

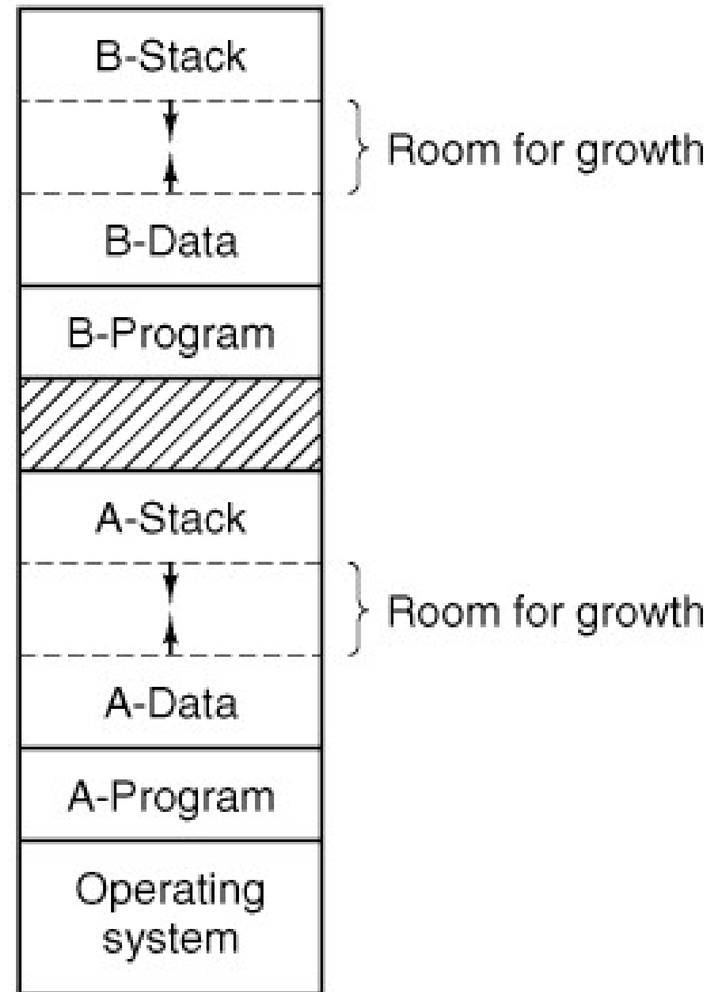
Figure 4-4. (a) Allocating space for a growing data segment.



Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Swapping (3)

Figure 4-4. (b) Allocating space for a growing stack and a growing data segment.



Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Memory Management with Bitmaps

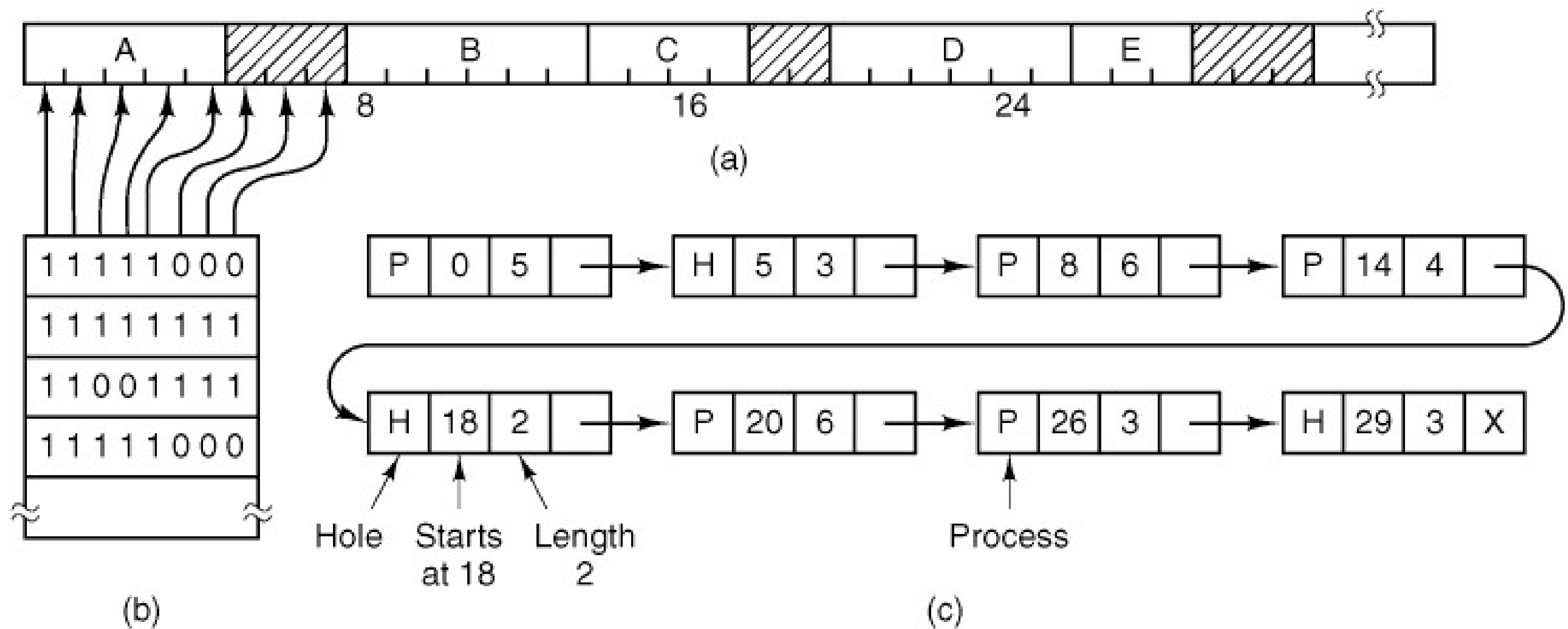


Figure 4-5. (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap. (c) The same information as a list.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Memory Management with Linked Lists

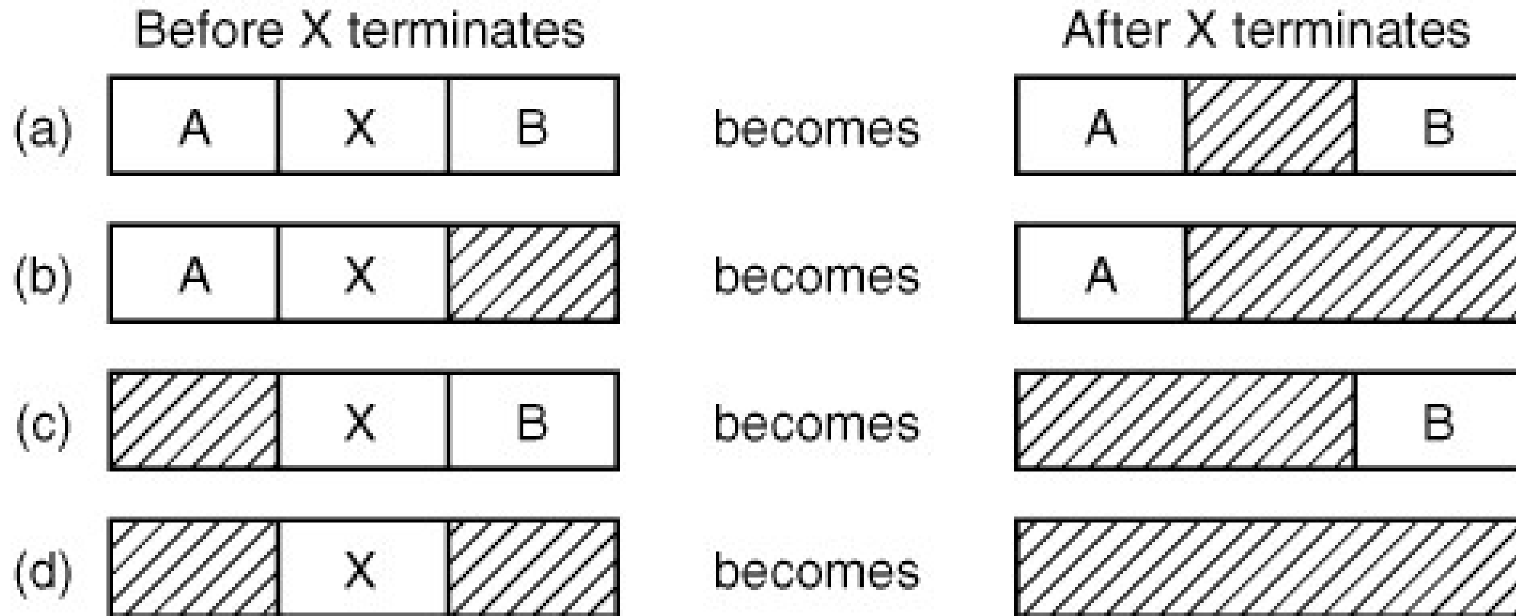


Figure 4-6. Four neighbor combinations for the terminating process, X.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Memory Allocation Algorithms

- First fit
  Use first hole big enough
- Next fit
  Use next hole big enough
- Best fit
  Search list for smallest hole big enough
- Worst fit
  Search list for largest hole available
- Quick fit
  Separate lists of commonly requested sizes

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Paging (1)

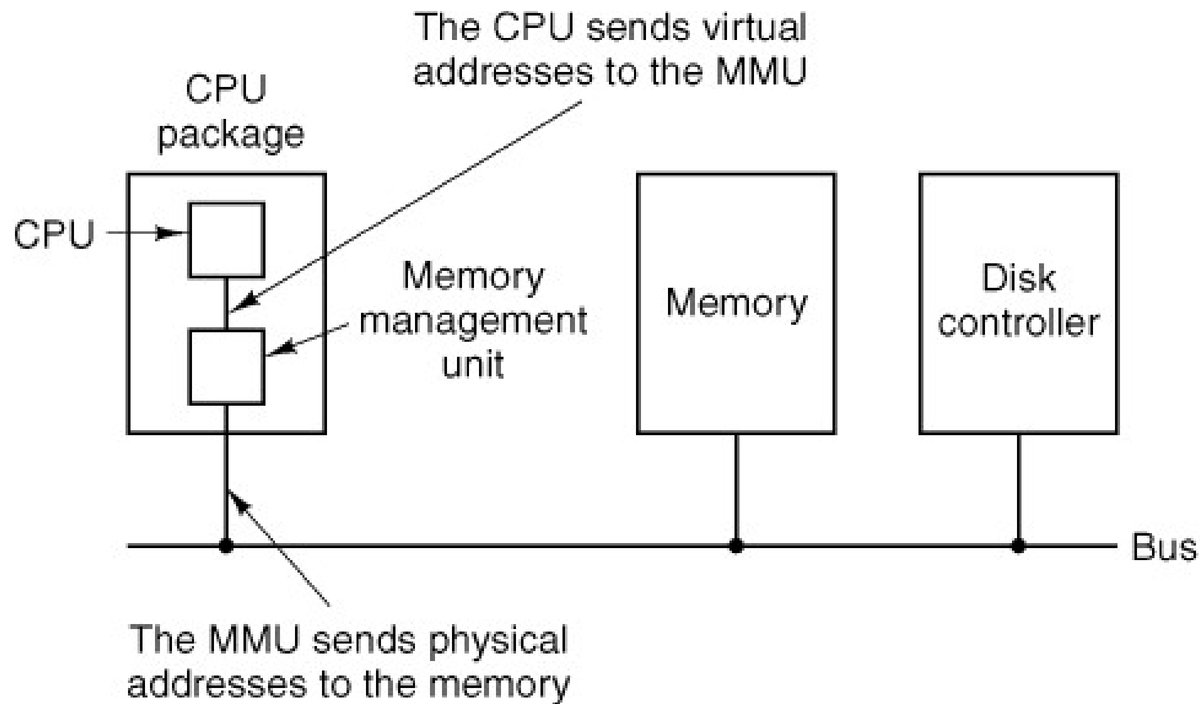
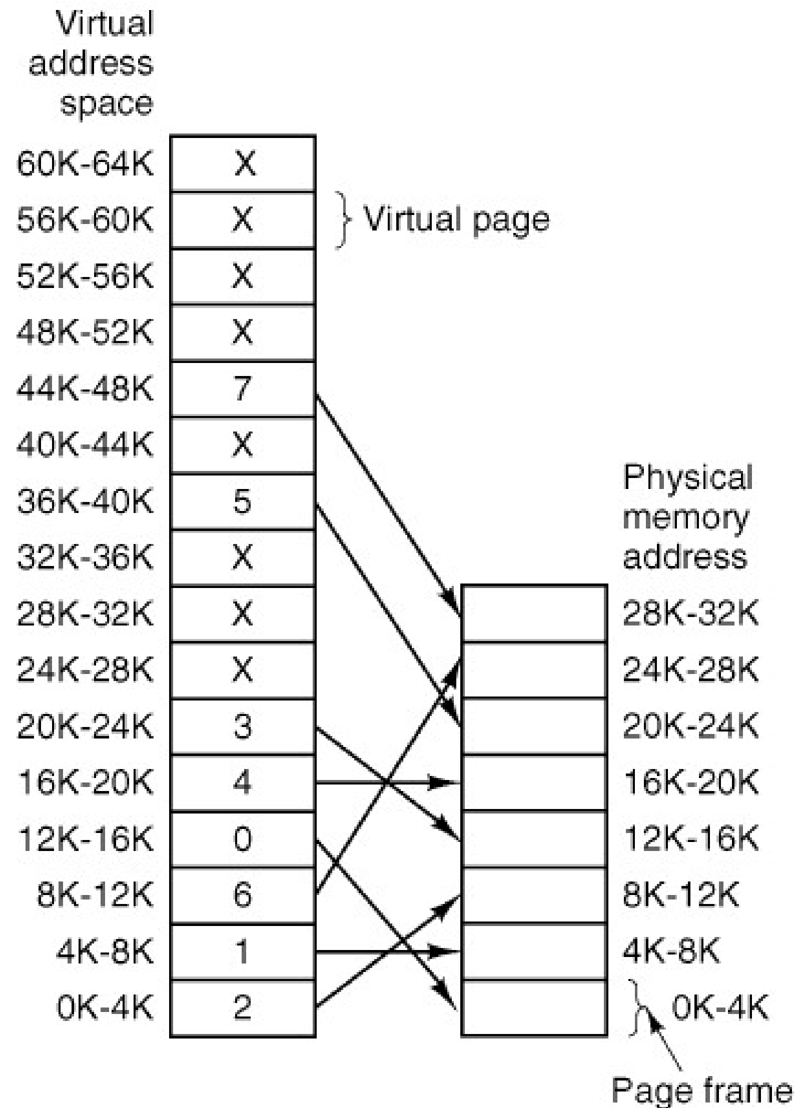


Figure 4-7. The position and function of the MMU. Here the MMU is shown as being a part of the CPU chip because it commonly is nowadays. However, logically it could be a separate chip and was in years gone by.

Paging (2)

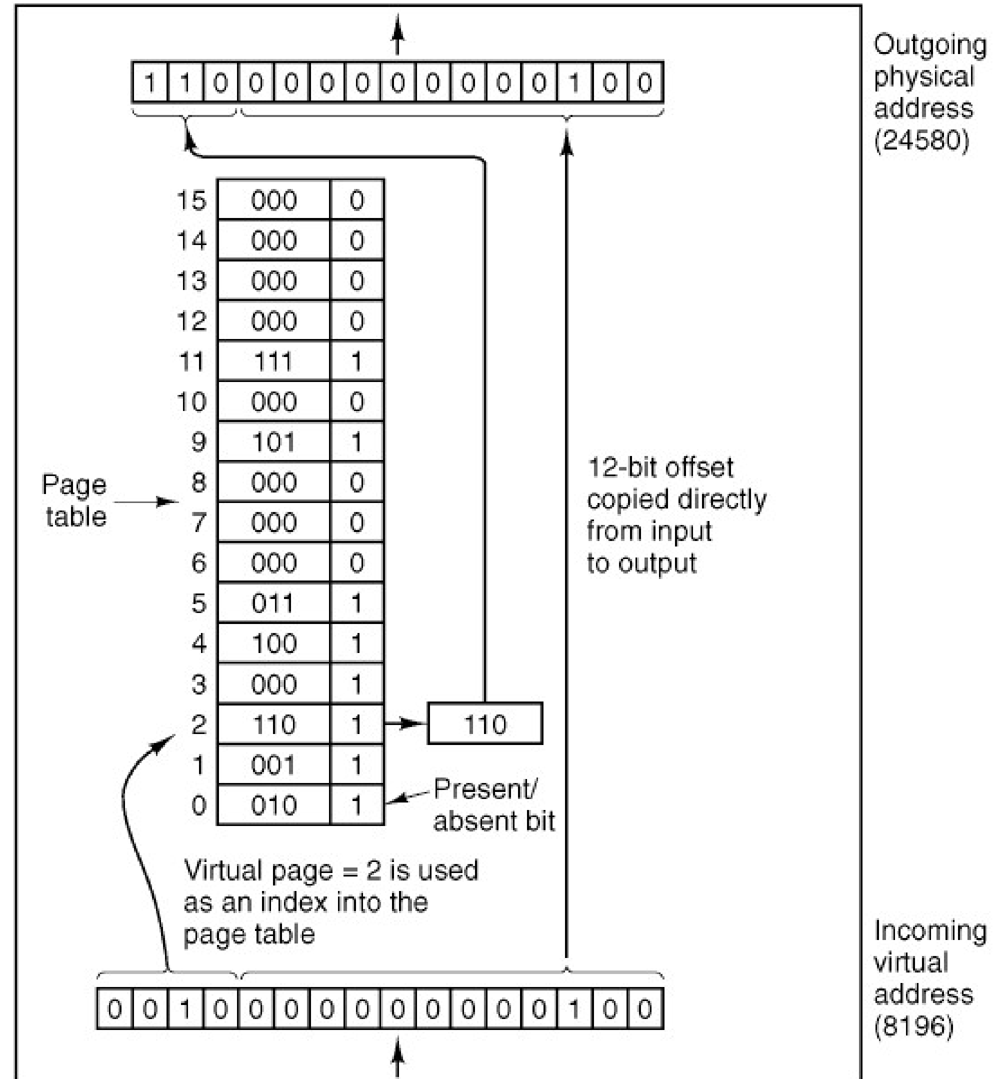
Figure 4-8. The relation between virtual addresses and physical memory addresses is given by the page table.



Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Paging (3)

Figure 4-9. The internal operation of the MMU with 16 4-KB pages.



Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

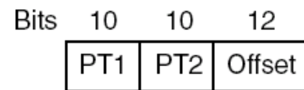
Page Tables

- Purpose : map virtual pages onto page frames

- Major issues to be faced
  1. The page table can be extremely large
  2. The mapping must be fast.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Multilevel Page Tables



Top-level page table: entries 0, 1, 2, 3, 4, 5, 6, …, 1023. Second-level page tables: entries 0, 1, 2, 3, 4, 5, 6, …, 1023. Page table for the top 4M of memory. To pages. (b)

Figure 4-10. (a) A 32-bit address with two page table fields. (b) Two-level page tables.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Structure of a Page Table Entry

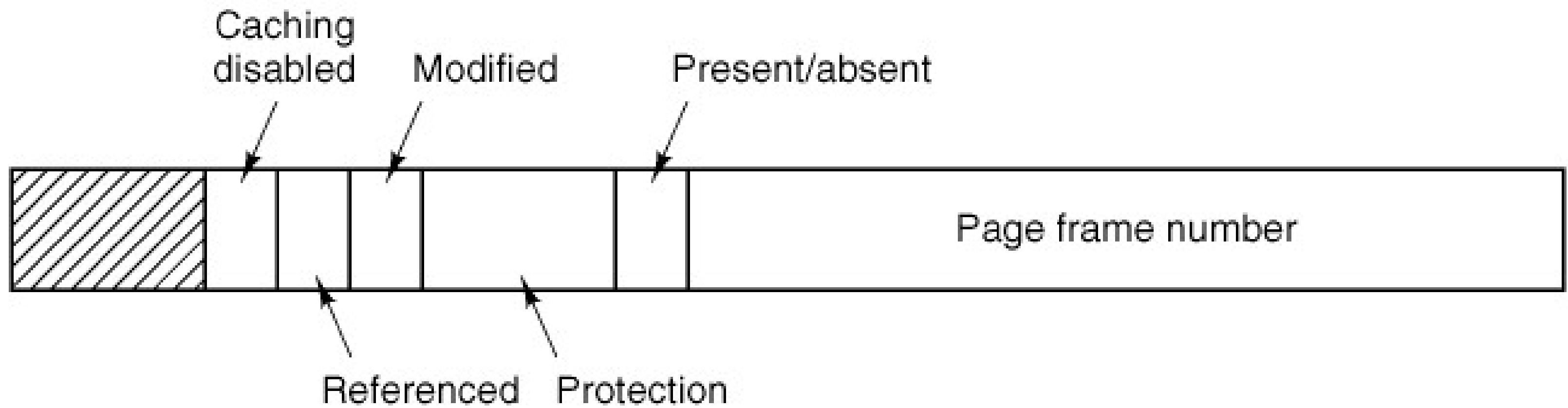


Figure 4-11. A typical page table entry.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

TLBs—Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Figure 4-12. A TLB to speed up paging.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Inverted Page Tables

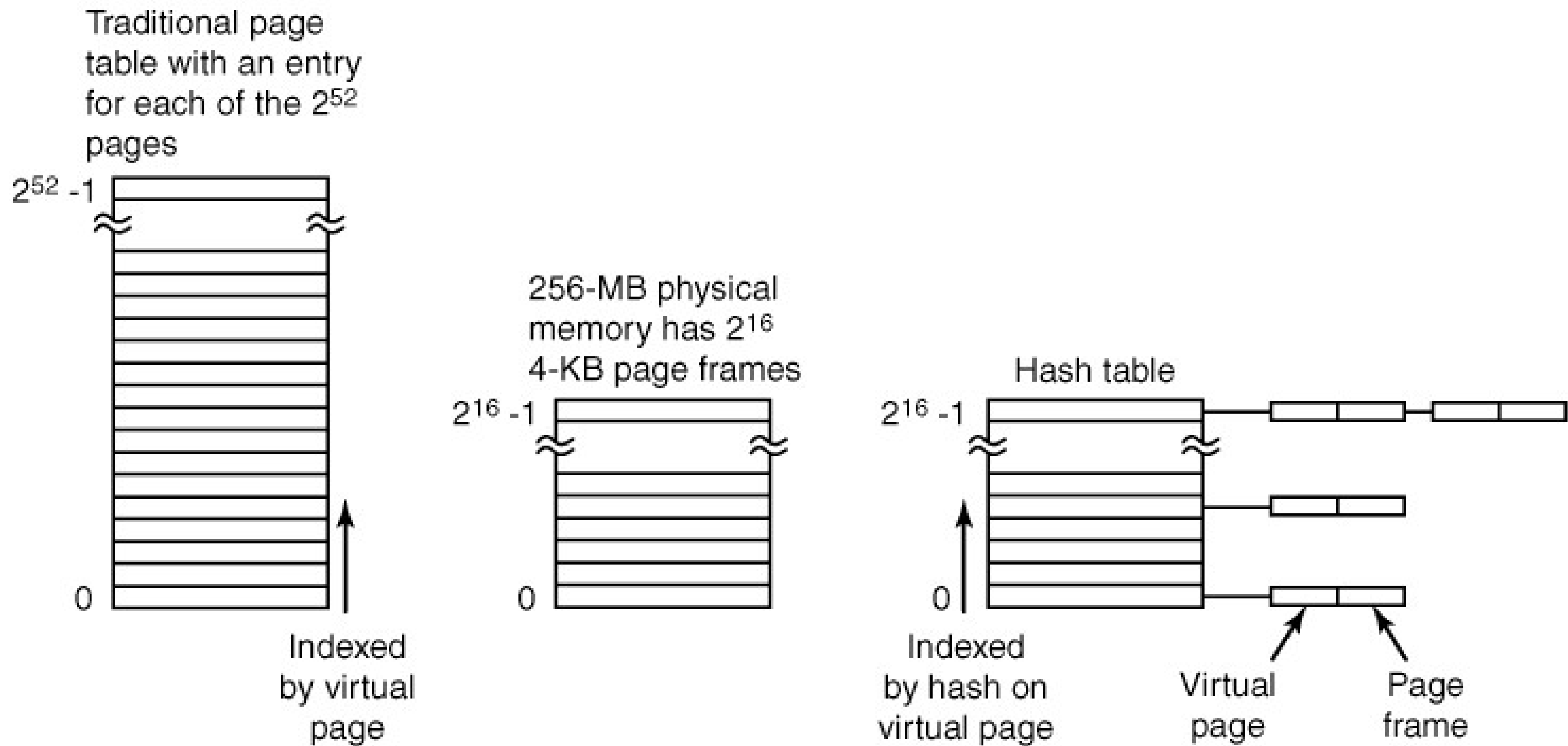


Figure 4-13. Comparison of a traditional page table with an inverted page table.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Page Replacement Algorithms

- Optimal replacement
- Not recently used (NRU) replacement
- First-in, first-out (FIFO) replacement
- Second chance replacement
- Clock page replacement
- Least recently used (LRU) replacement

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Second Chance Replacement

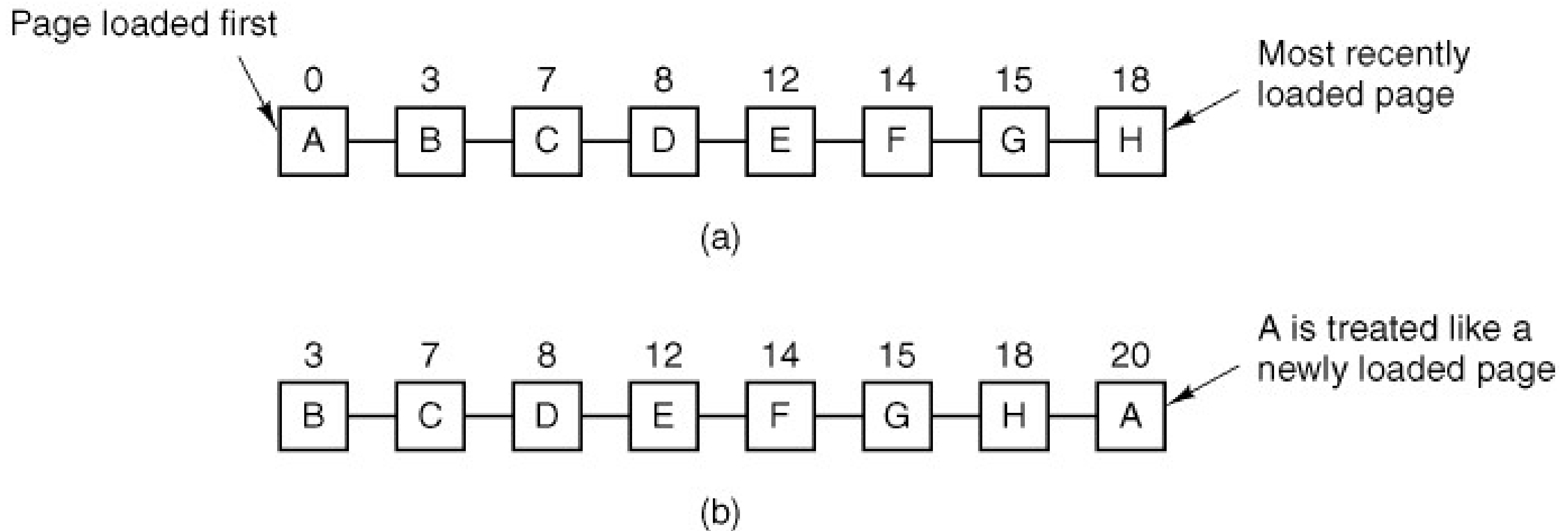


Figure 4-14. Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and A has its R bit set. The numbers above the pages are their loading times.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Clock Page Replacement

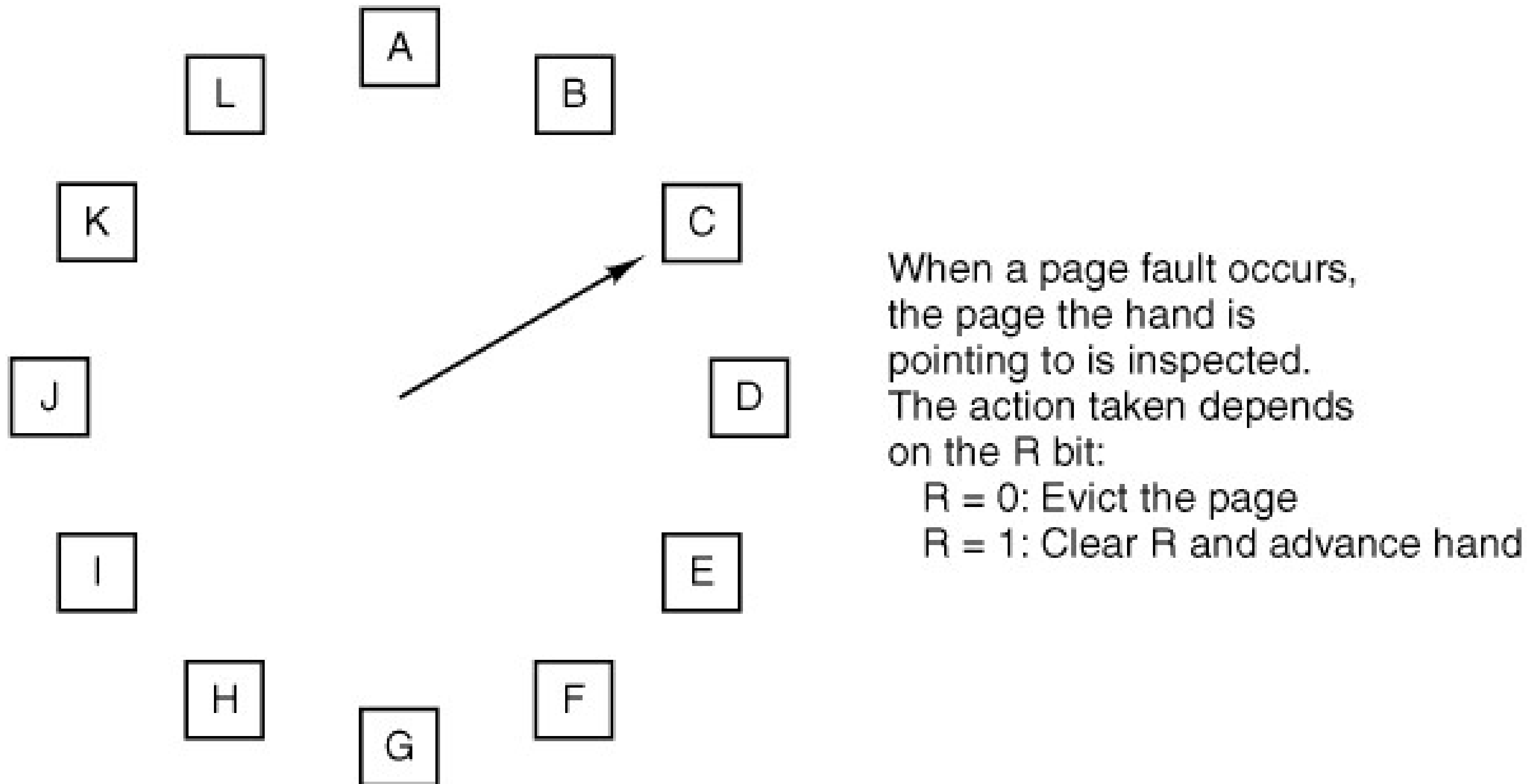


Figure 4-15. The clock page replacement algorithm.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Simulating LRU in Software (1)

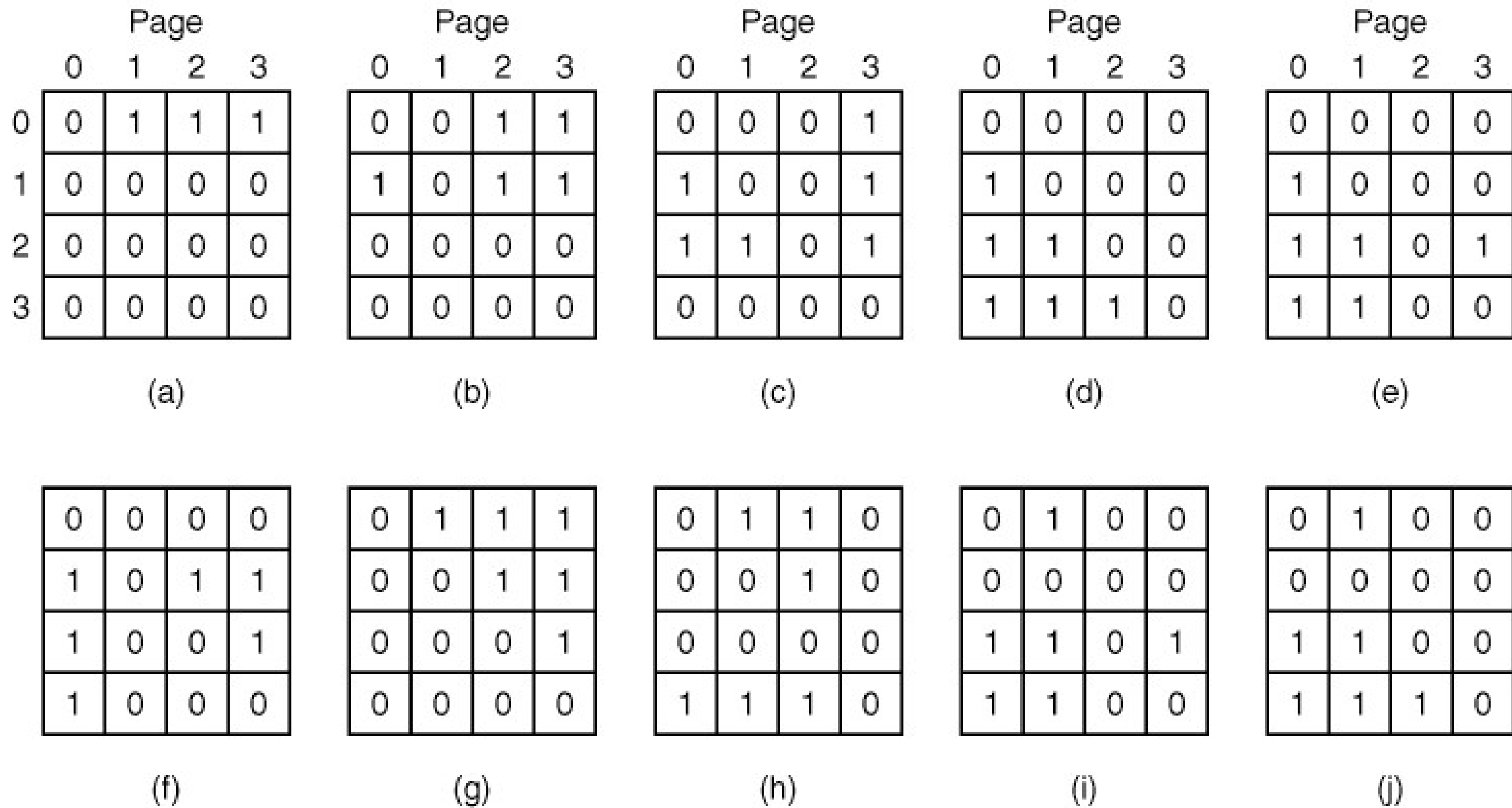


Figure 4-16. LRU using a matrix when pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Simulating LRU in Software (2)

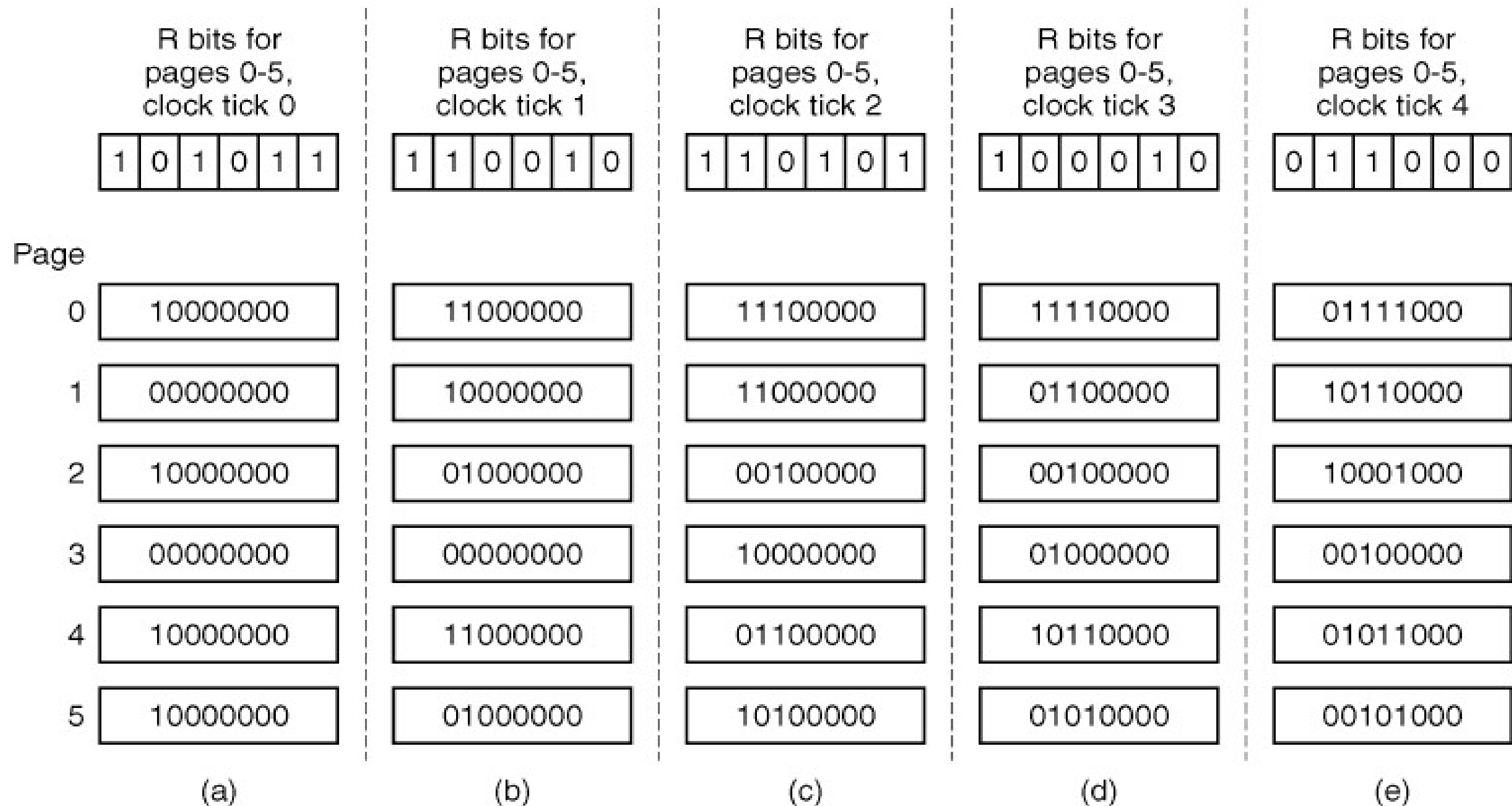


Figure 4-17. The aging algorithm simulates LRU in software. Shown are six pages for five clock ticks. The five clock ticks are represented by (a) to (e).

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

The Working Set Model

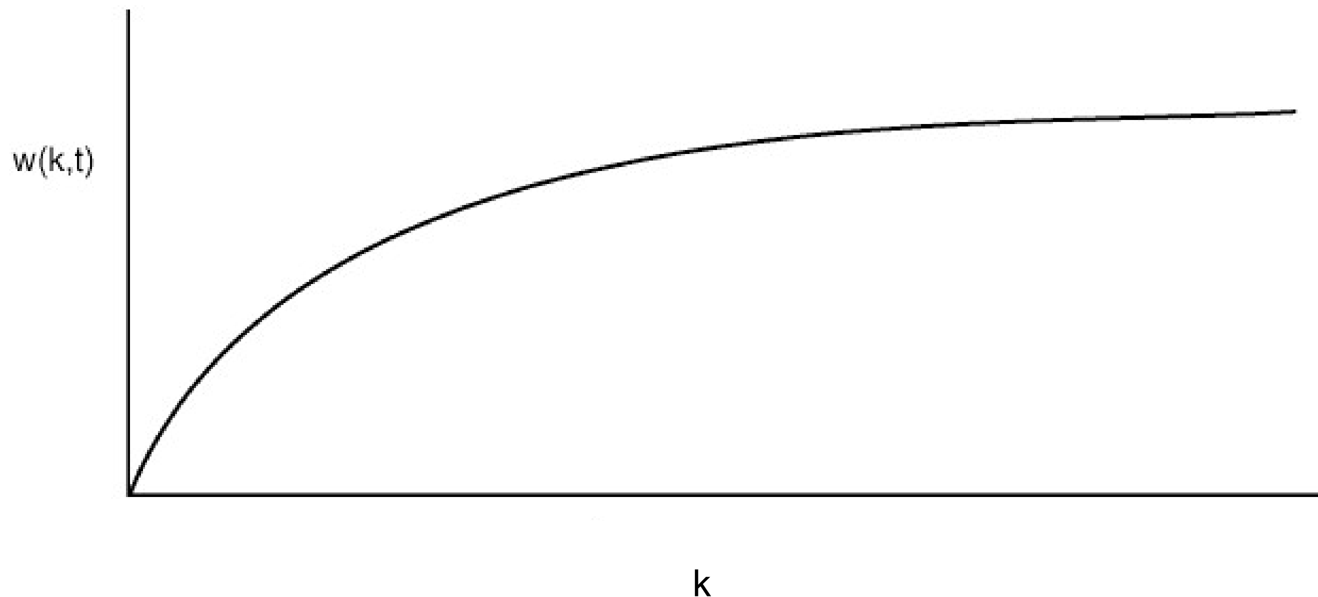


Figure 4-18. The working set is the set of pages used by the *k* most recent memory references. The function *w(k, t)* is the size of the working set at time *t*.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Local versus Global Allocation Policies

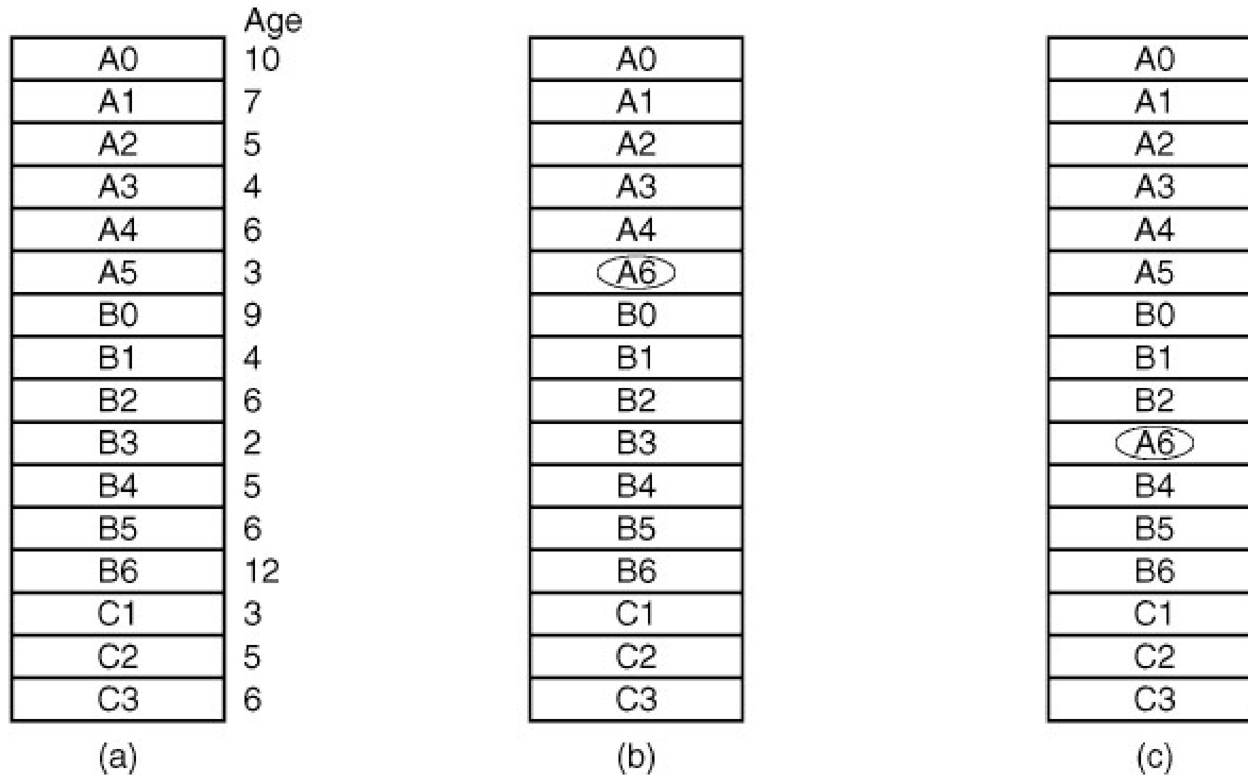


Figure 4-19. Local versus global page replacement. (a) Original configuration. (b) Local page replacement. (c) Global page replacement.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Page Fault Frequency

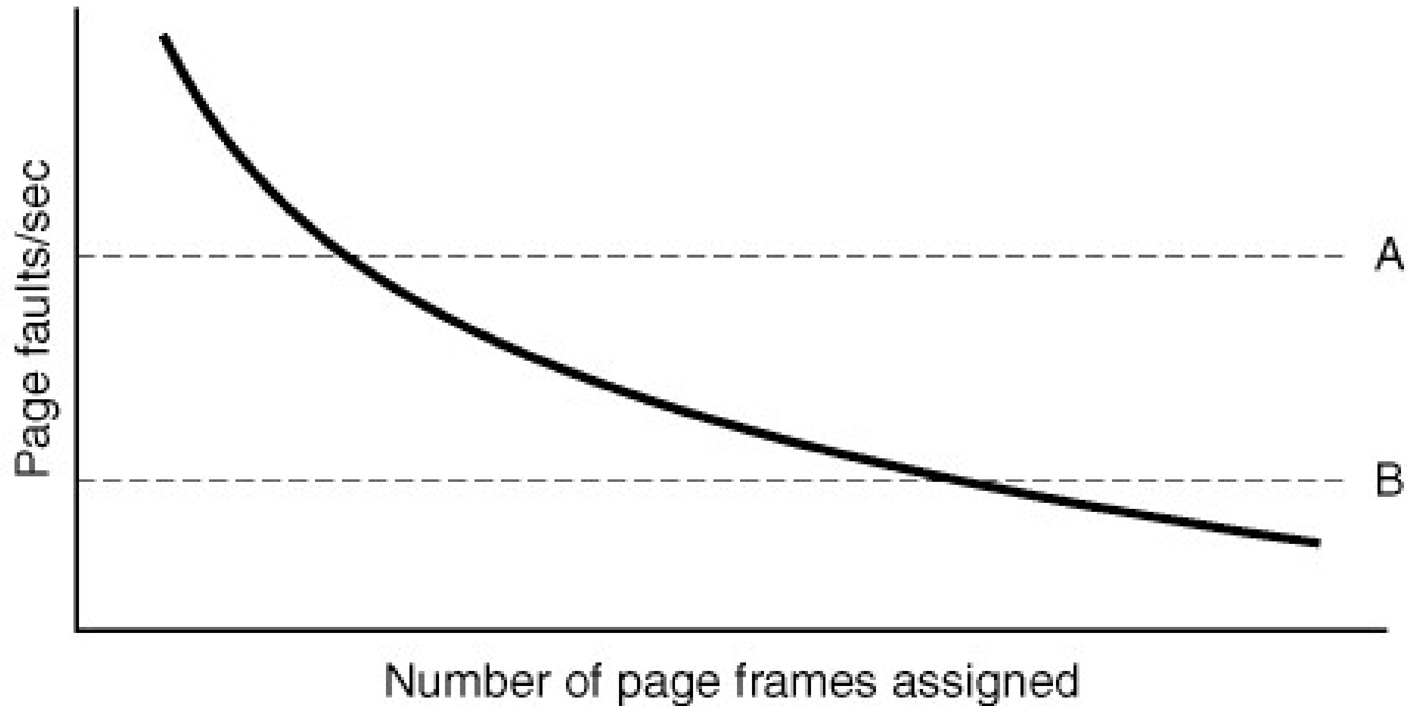


Figure 4-20. Page fault rate as a function of the number of page frames assigned.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation (1)

Examples of tables saved by a compiler …

1. The source text being saved for the printed listing (on batch systems).
2. The symbol table, containing the names and attributes of variables.
3. The table containing all the integer and floating-point constants used.
4. The parse tree, containing the syntactic analysis of the program.
5. The stack used for procedure calls within the compiler.

These will vary in size dynamically during the compile process

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation (2)

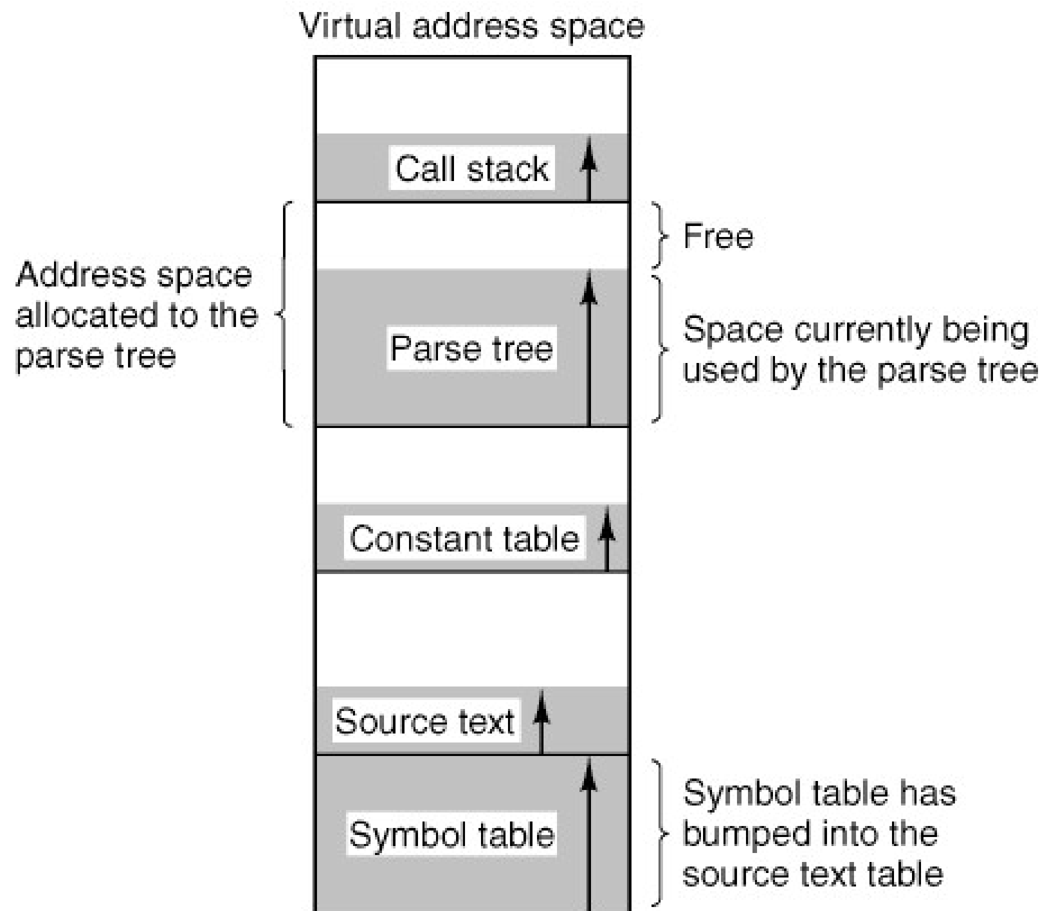


Figure 4-21. In a one-dimensional address space with growing tables, one table may bump into another.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation (3)

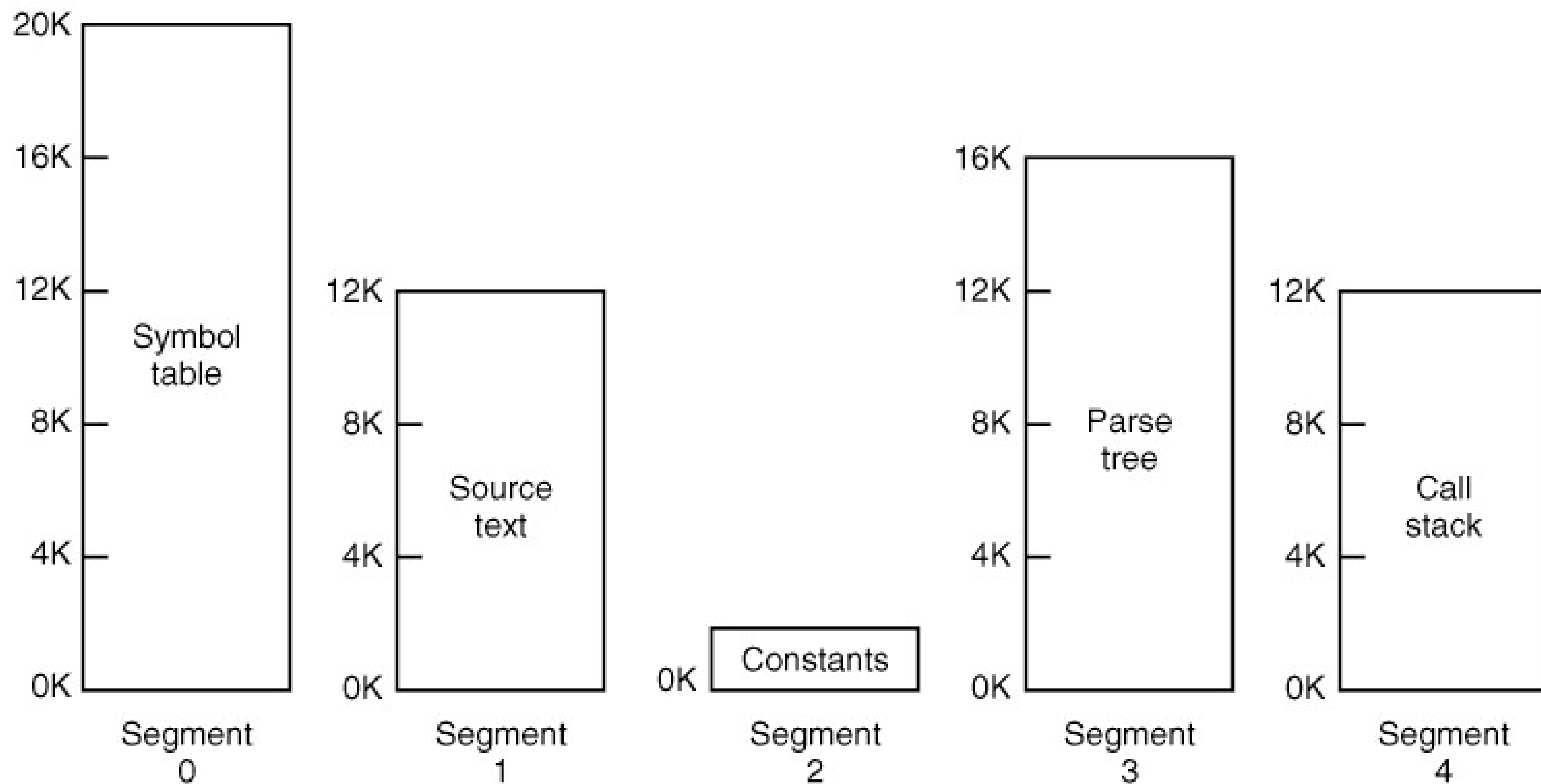


Figure 4-22. A segmented memory allows each table to grow or shrink independently of the other tables.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation (4)

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes

. . .

Figure 4-23. Comparison of paging and segmentation.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation (4)

Consideration	Paging	Segmentation
		. . .
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Figure 4-23. Comparison of paging and segmentation.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Implementation of Pure Segmentation

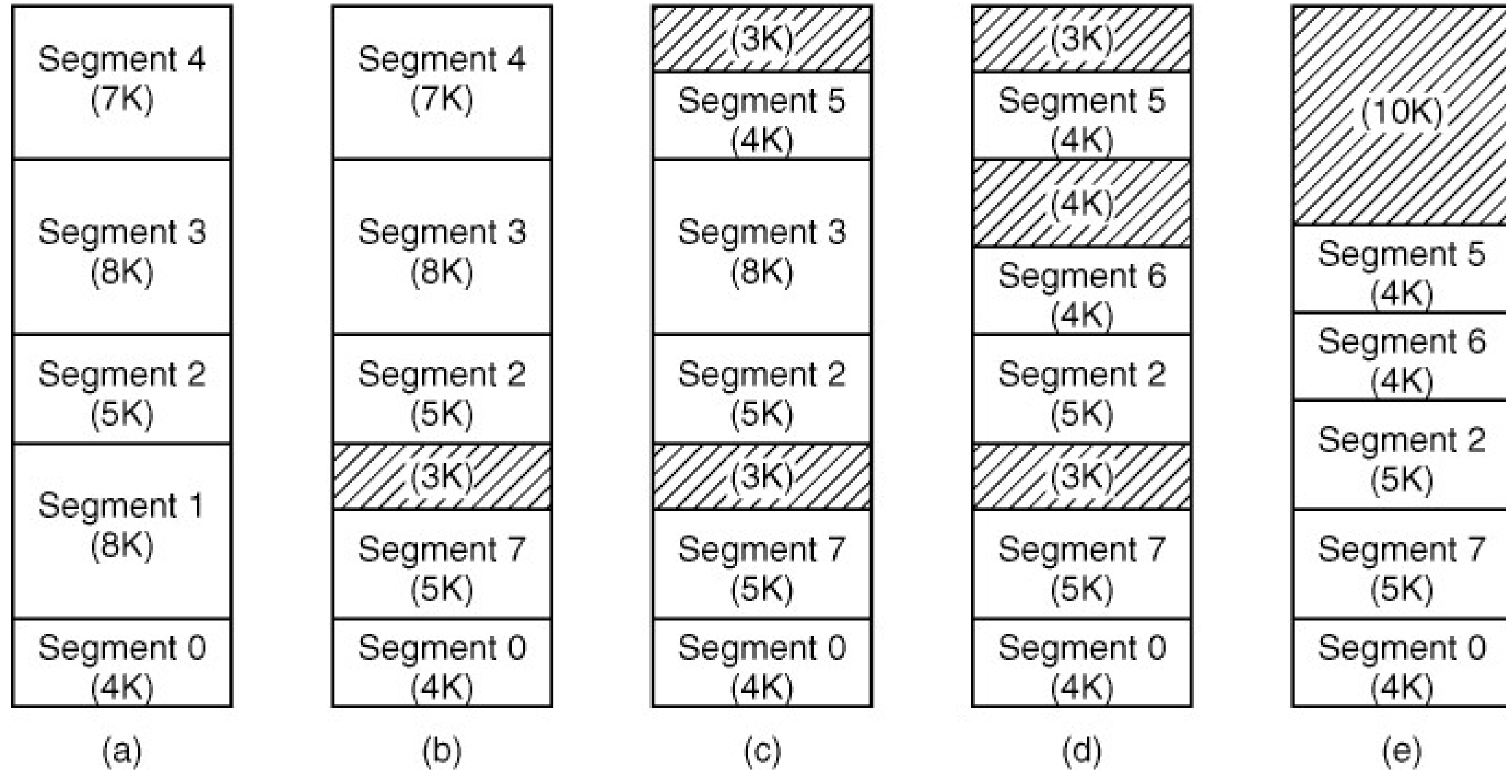


Figure 4-24. (a)-(d) Development of checkerboarding. (e) Removal of the checkerboarding by compaction.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation with Paging: The Intel Pentium (1)

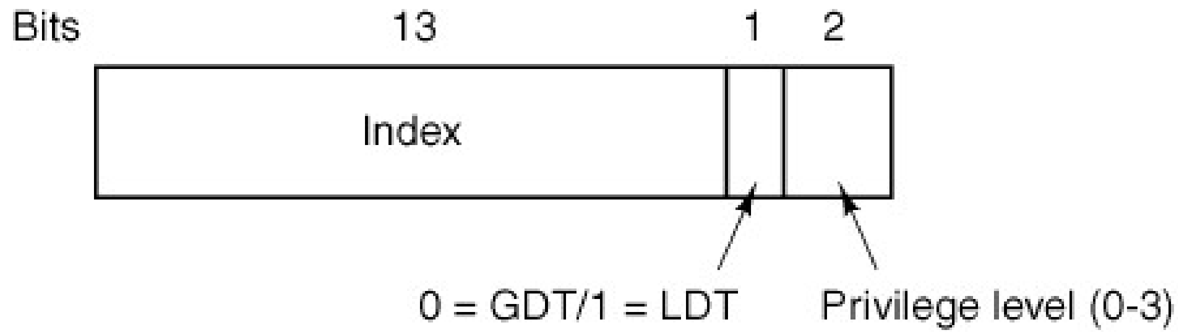


Figure 4-25. A Pentium selector.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation with Paging: The Intel Pentium (2)

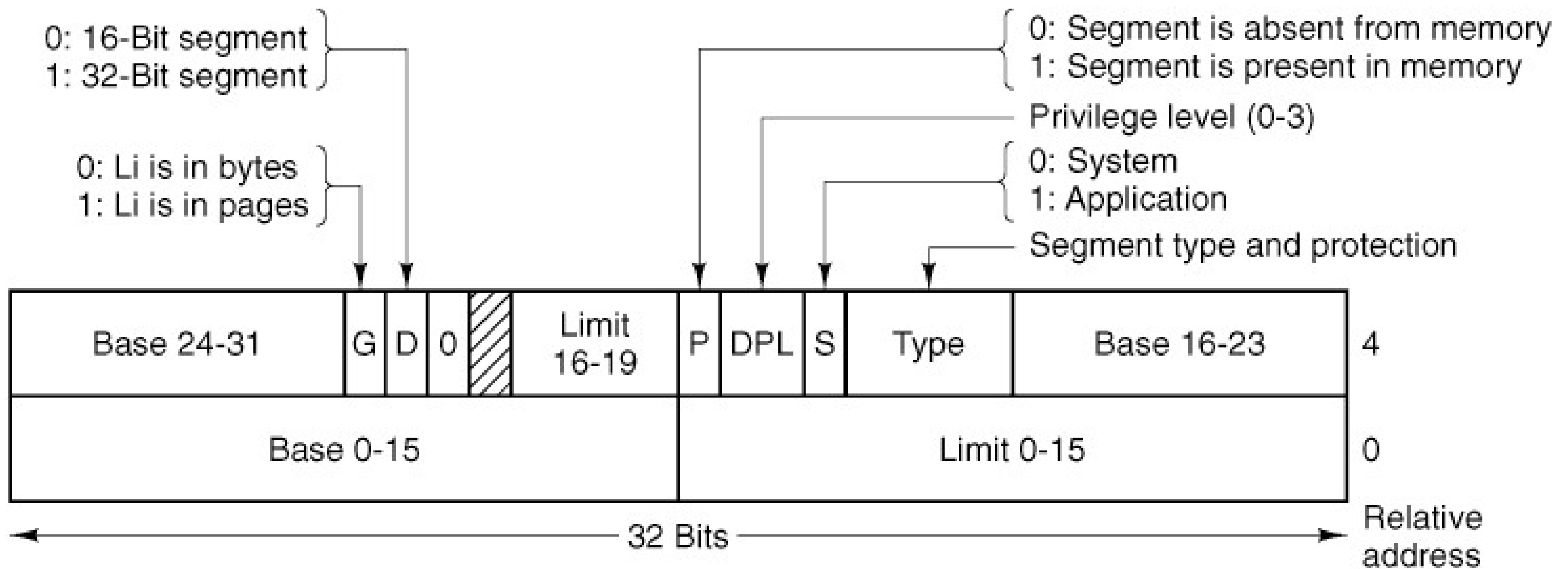


Figure 4-26. Pentium code segment descriptor. Data segments differ slightly.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation with Paging: The Intel Pentium (3)

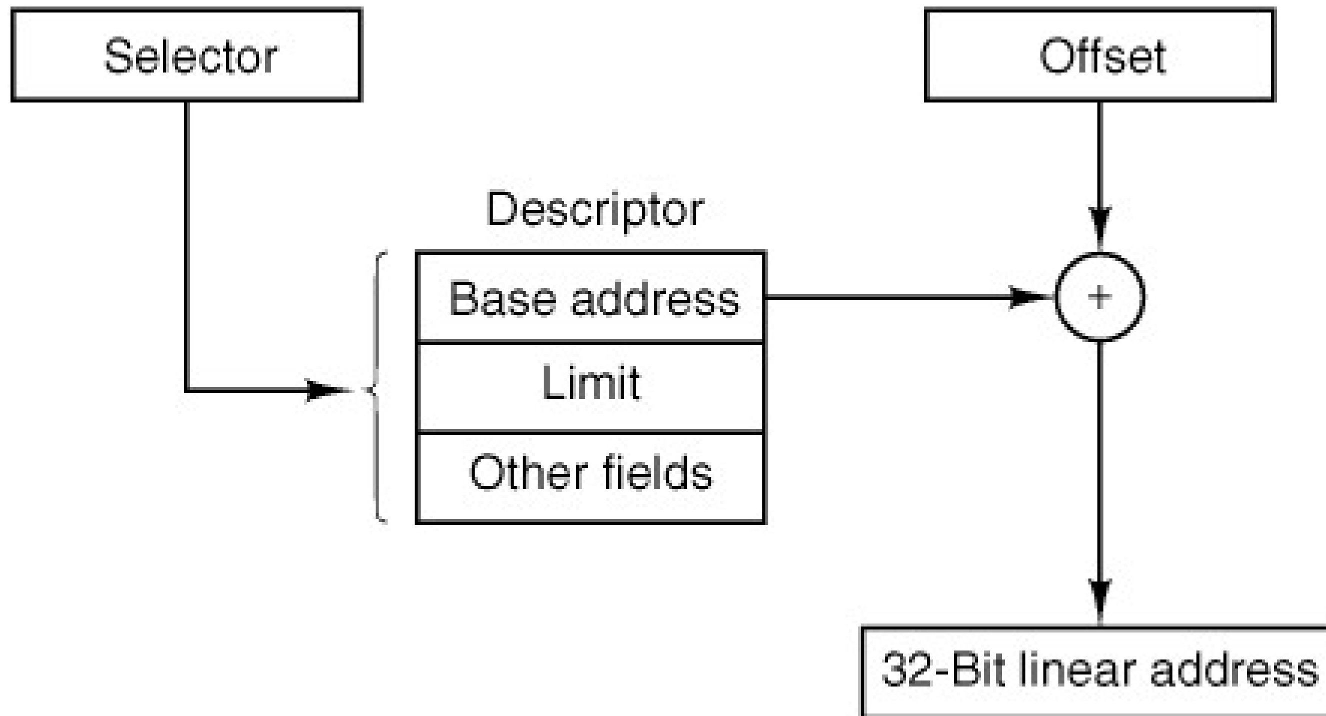


Figure 4-27. Conversion of a (selector, offset) pair to a linear address.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation with Paging: The Intel Pentium (4)

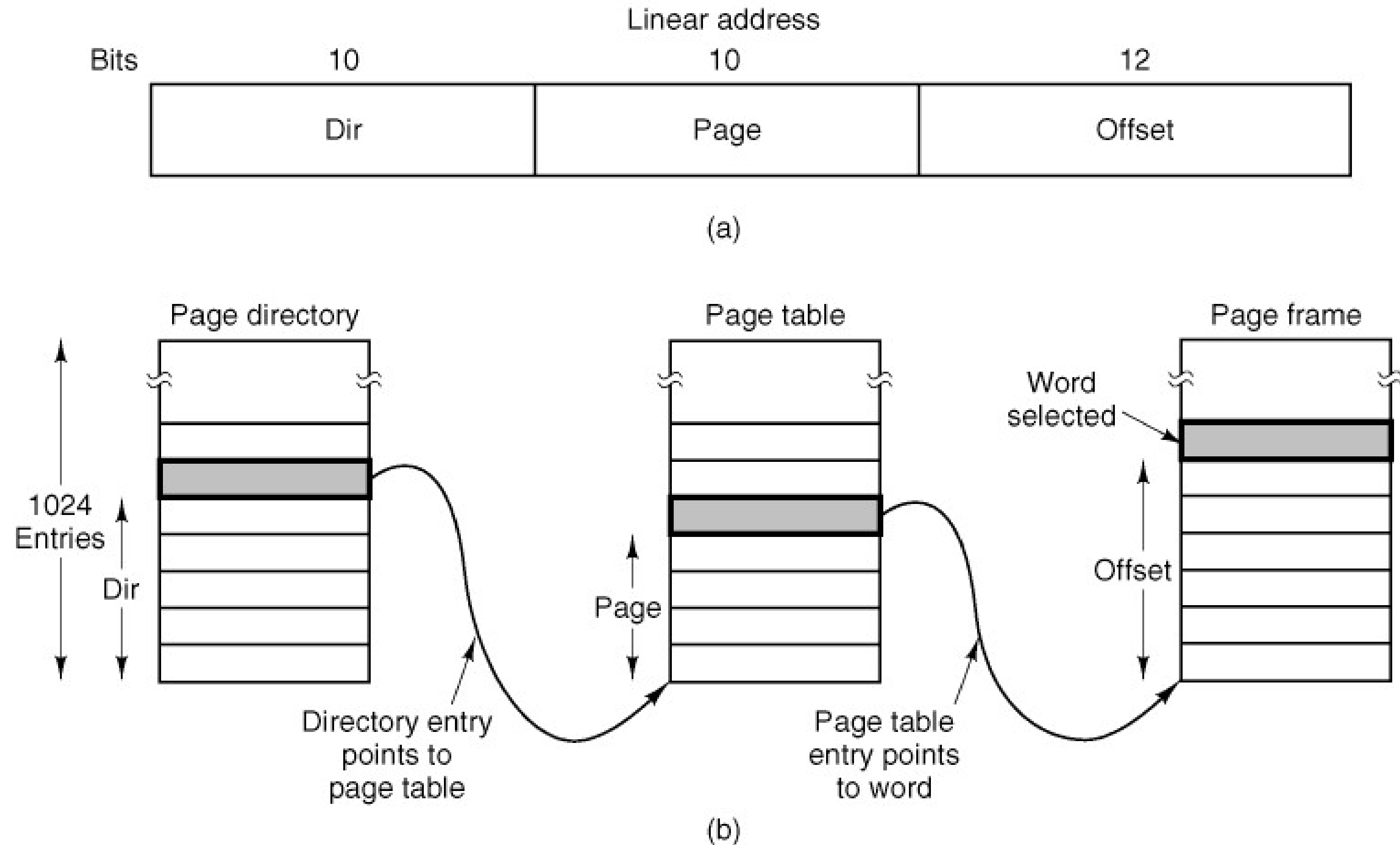


Figure 4-28. Mapping of a linear address onto a physical address.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Segmentation with Paging: The Intel Pentium (6)

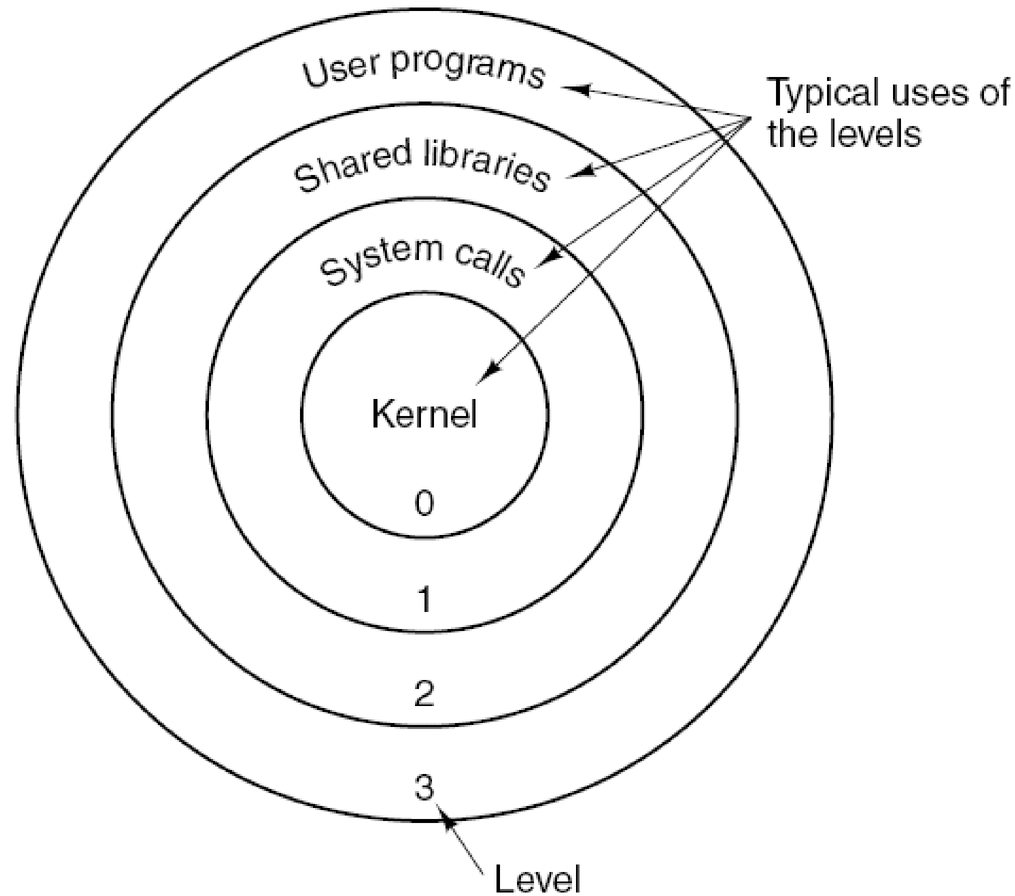


Figure 4-29. Protection on the Pentium.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Memory Layout (1)

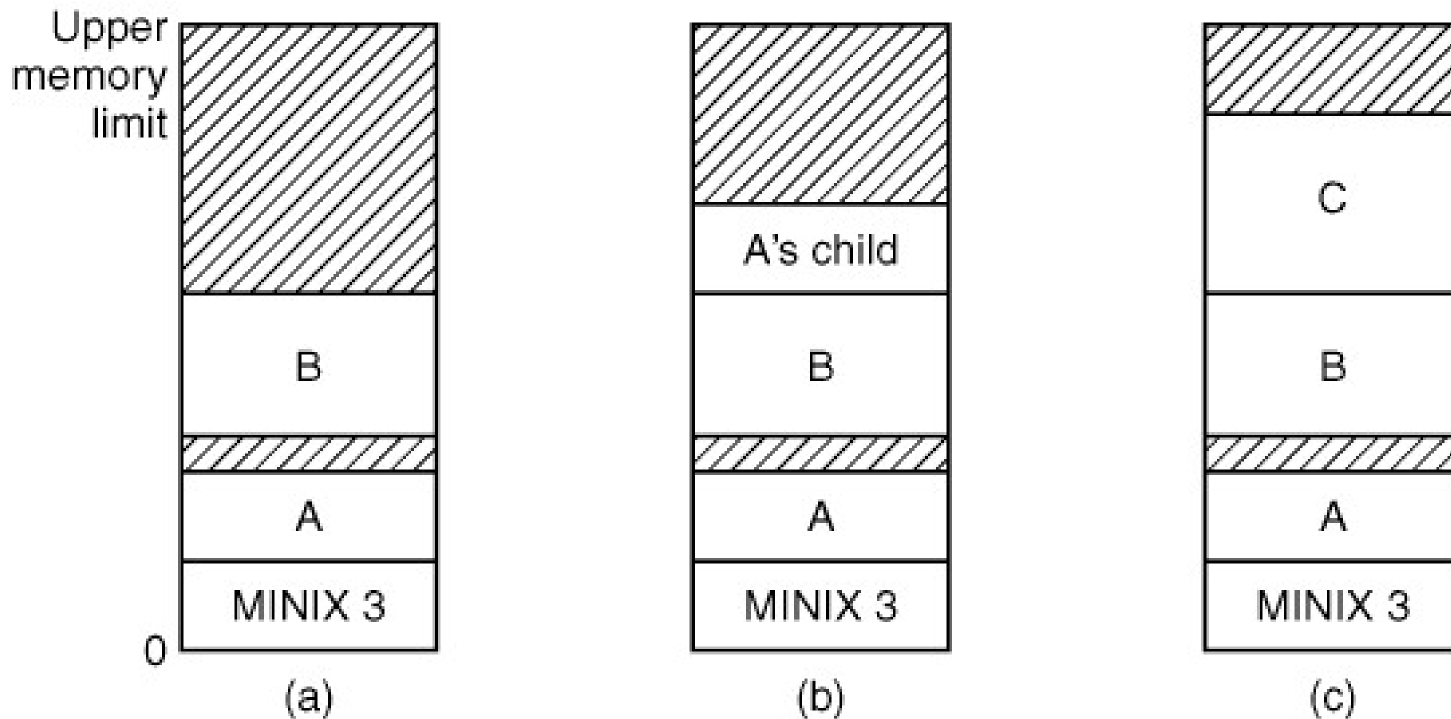


Figure 4-30. Memory allocation (a) Originally. (b) After a fork. (c) After the child does an exec. The shaded regions are unused memory. The process is a common I&D one.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Memory Layout (2)

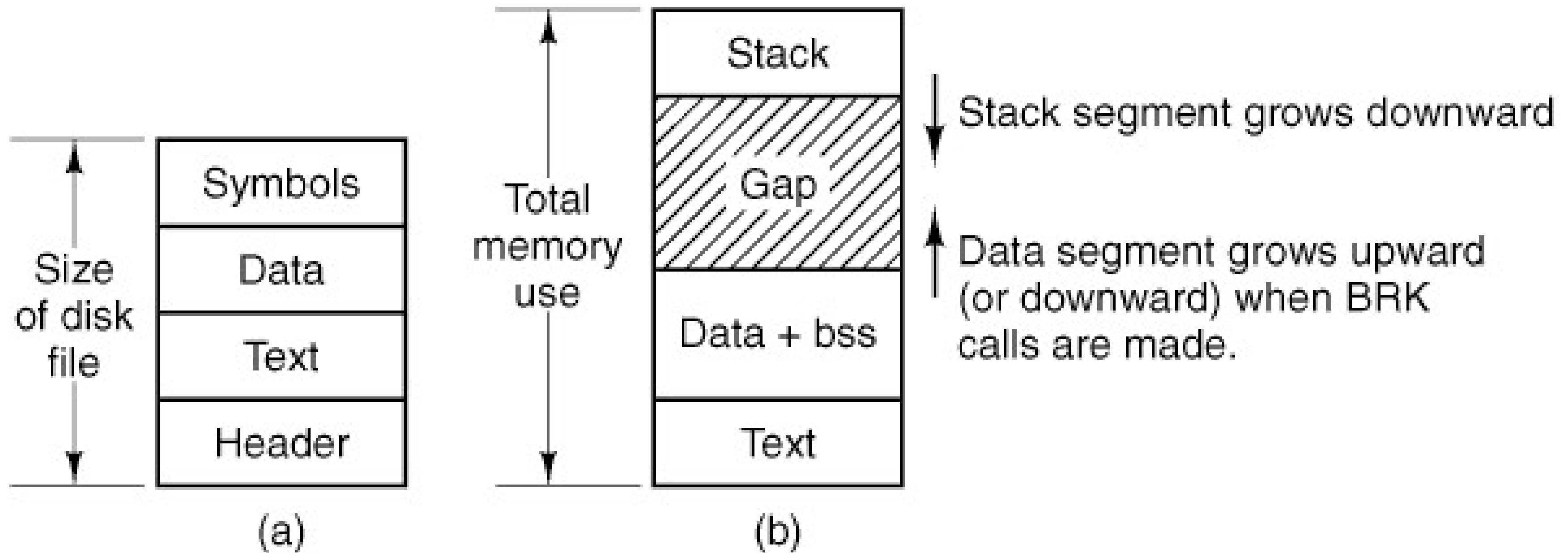


Figure 4-31. (a) A program as stored in a disk file. (b) Internal memory layout for a single process. In both parts of the figure the lowest disk or memory address is at the bottom and the highest address is at the top.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Process Manager Data Structures and Algorithms (1)

Message type	Input parameters	Reply value
fork	(none)	Child's PID, (to child: 0)
exit	Exit status	(No reply if successful)
wait	(none)	Status
waitpid	Process identifier and flags	Status
brk	New size	New size
exec	Pointer to initial stack	(No reply if successful)
kill	Process identifier and signal	Status
alarm	Number of seconds to wait	Residual time
pause	(none)	(No reply if successful)
sigaction	Signal number, action, old action	Status
sigsuspend	Signal mask	(No reply if successful)
sigpending	(none)	Status
sigprocmask	How, set, old set	Status
sigreturn	Context	Status
getuid	(none)	Uid, effective uid
getgid	(none)	Gid, effective gid
getpid	(none)	PID, parent PID

. . .

Figure 4-32. The message types, input parameters, and reply values used for communicating with the PM.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Process Manager Data Structures and Algorithms (2)

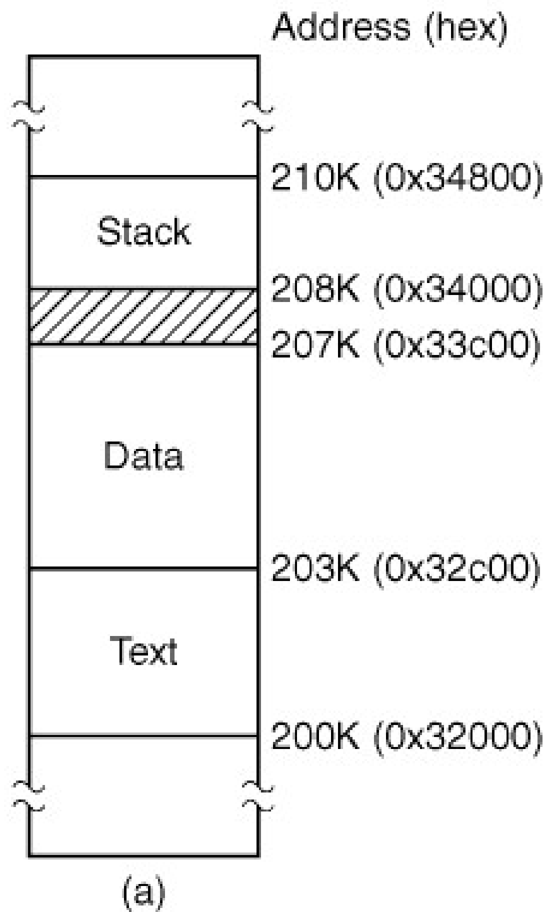
. . .

Message type	Input parameters	Reply value
setuid	New uid	Status
setgid	New gid	Status
setsid	New sid	Process group
getpgrp	New gid	Process group
time	Pointer to place where current time goes	Status
stime	Pointer to current time	Status
times	Pointer to buffer for process and child times	Uptime since boot
ptrace	Request, PID, address, data	Status
reboot	How (halt, reboot, or panic)	(No reply if successful)
svrctl	Request, data (depends upon function)	Status
getsysinfo	Request, data (depends upon function)	Status
getprocnr	(none)	Proc number
memalloc	Size, pointer to address	Status
memfree	Size, address	Status
getpriority	Pid, type, value	Priority (nice value)
setpriority	Pid, type, value	Priority (nice value)
gettimeofday	(none)	Time, uptime

Figure 4-32. The message types, input parameters, and reply values used for communicating with the PM.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Processes in Memory (1)



	Virtual	Physical	Length
Stack	0x8	0xd0	0x2
Data	0	0xc8	0x7
Text	0	0xc8	0

(b)

	Virtual	Physical	Length
Stack	0x5	0xd0	0x2
Data	0	0xcb	0x4
Text	0	0xc8	0x3

(c)

Figure 4-33. (a) A process in memory. (b) Its memory representation for combined I and D space. (c) Its memory representation for separate I and D space

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Processes in Memory (2)

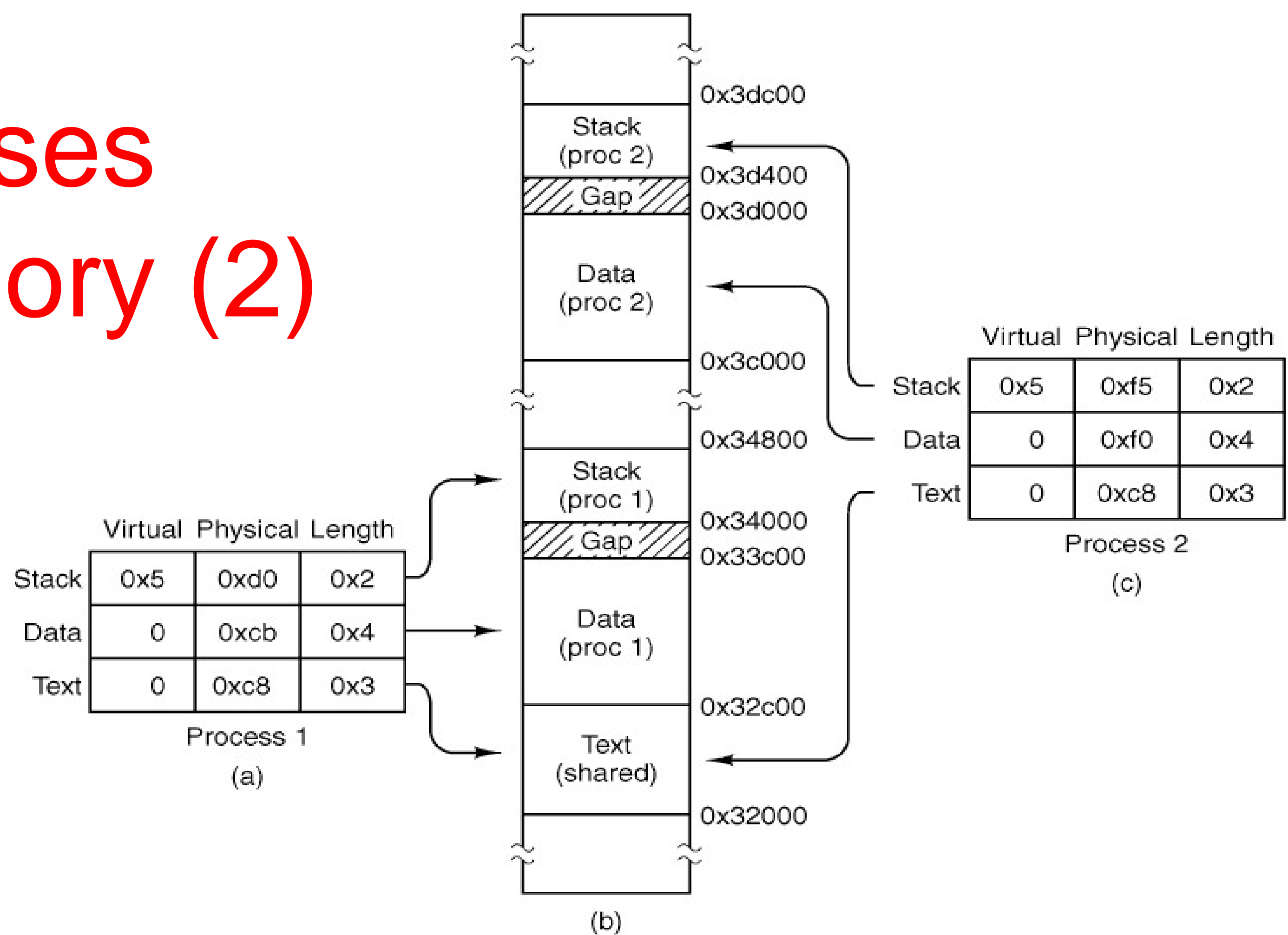


Figure 4-34. (a) The memory map of a separate I and D space process, as in the previous figure. (b) The layout in memory after a second process starts, executing the same program image with shared text. (c) The memory map of the second process.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

The Hole List

```
PRIVATE struct hole {
        struct hole *h_next;         /* pointer to next entry on the list */
        phys_clicks h_base;          /* where does the hole begin? */
        phys_clicks h_len;           /* how big is the hole? */
} hole[NR_HOLES];
```

Figure 4-35. The hole list is an array of struct hole.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

FORK System Call

1. Check to see if process table is full.
2. Try to allocate memory for the child's data and stack.
3. Copy the parent's data and stack to the child's memory.
4. Find a free process slot and copy parent's slot to it.
5. Enter child's memory map in process table.
6. Choose a PID for the child.
7. Tell kernel and file system about child.
8. Report child's memory map to kernel.
9. Send reply messages to parent and child.

Figure 4-36. The steps required to carry out the fork system call.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

EXEC System Call (1)

1. Check permissions—is the file executable?
2. Read the header to get the segment and total sizes.
3. Fetch the arguments and environment from the caller.
4. Allocate new memory and release unneeded old memory.
5. Copy stack to new memory image.
6. Copy data (and possibly text) segment to new memory image.
7. Check for and handle setuid, setgid bits.
8. Fix up process table entry.
9. Tell kernel that process is now runnable.

Figure 4-37. The steps required to carry out the exec system call.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

EXEC System Call (2)

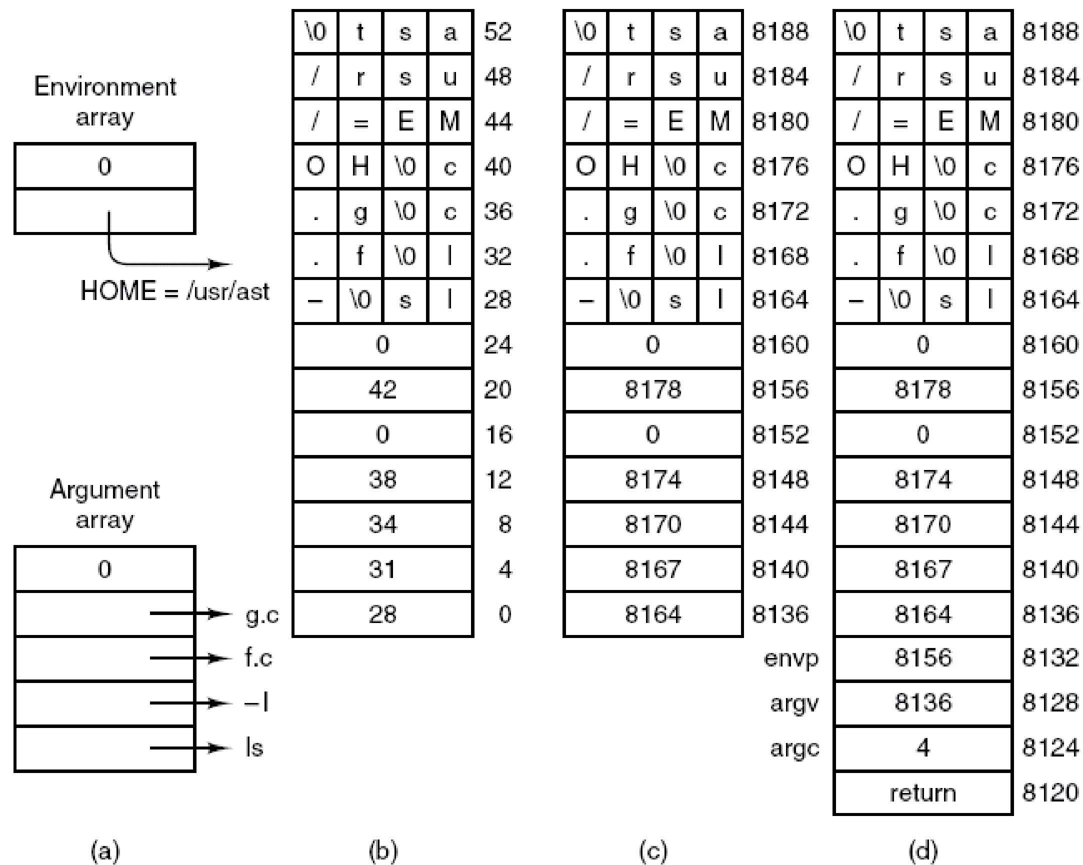


Figure 4-38. (a) The arrays passed to *execve*. (b) The stack built by *execve*. (c) The stack after relocation by the PM. (d) The stack as it appears to *main* at start of execution.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

EXEC System Call (3)

```
push  ecx           ! push environ
push  edx           ! push argv
push  eax           ! push argc
call  _main         ! main(argc, argv, envp)
push  eax           ! push exit status
call  _exit
hlt                 ! force a trap if exit fails
```

Figure 4-39. The key part of *crtso*, the C run-time, start-off routine.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (1)

Preparation: program code prepares for possible signal.
Response: signal is received and action is taken.
Cleanup: restore normal operation of the process.

Figure 4-40. Three phases of dealing with signals.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (2)

```
struct sigaction {
    __sighandler_t sa_handler;   /* SIG_DFL, SIG_IGN, SIG_MESS,
                                    or pointer to function */
   sigset_t sa_mask;             /* signals to be blocked during handler */
   int sa_flags;                 /* special flags */
}
```

Figure 4-41. The sigaction structure.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (3)

Signal	Description	Generated by
SIGHUP	Hangup	KILL system call
SIGINT	Interrupt	TTY
SIGQUIT	Quit	TTY
SIGILL	Illegal instruction	Kernel (*)
SIGTRAP	Trace trap	Kernel (M)
SIGABRT	Abnormal termination	TTY
SIGFPE	Floating point exception	Kernel (*)
SIGKILL	Kill (cannot be caught or ignored)	KILL system call
SIGUSR1	User-defined signal # 1	Not supported
SIGSEGV	Segmentation violation	Kernel (*)
SIGUSR2	User defined signal # 2	Not supported
SIGPIPE	Write on a pipe with no one to read it	FS

Figure 4-42. Signals defined by POSIX and MINIX 3. Signals indicated by (*) depend on hardware support. Signals marked (M) not defined by POSIX, but are defined by MINIX 3 for compatibility with older programs. Signals kernel are MINIX 3 specific signals generated by the kernel, and used to inform system processes about system events. Several obsolete names and synonyms are not listed here.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (4)

Signal	Description	Generated by
SIGALRM	Alarm clock, timeout	PM
SIGTERM	Software termination signal from kill	KILL system call
SIGCHLD	Child process terminated or stopped	PM
SIGCONT	Continue if stopped	Not supported
SIGSTOP	Stop signal	Not supported
SIGTSTP	Interactive stop signal	Not supported
SIGTTIN	Background process wants to read	Not supported
SIGTTOU	Background process wants to write	Not supported
SIGKMESS	Kernel message	Kernel
SIGKSIG	Kernel signal pending	Kernel
SIGKSTOP	Kernel shutting down	Kernel

Figure 4-42. Signals defined by POSIX and MINIX 3. Signals indicated by (*) depend on hardware support. Signals marked (M) not defined by POSIX, but are defined by MINIX 3 for compatibility with older programs. Signals kernel are MINIX 3 specific signals generated by the kernel, and used to inform system processes about system events. Several obsolete names and synonyms are not listed here.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (5)

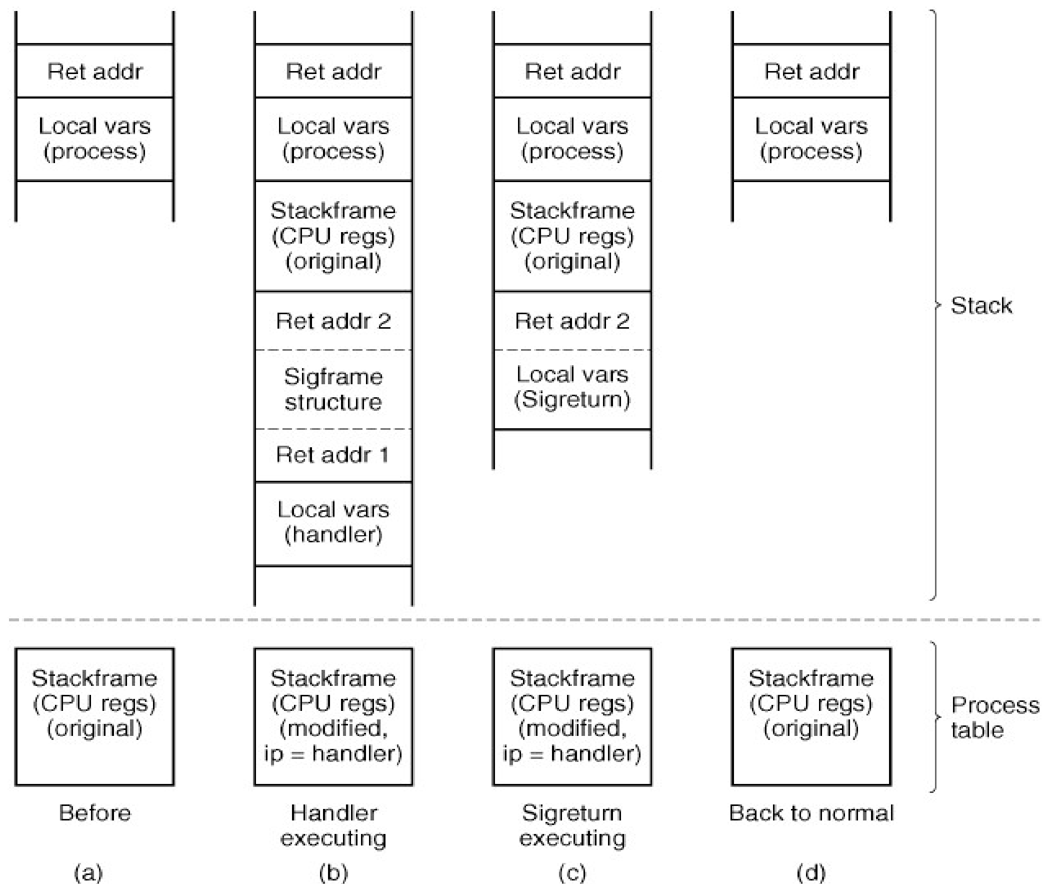


Figure 4-43. A process' stack (above) and its stackframe in the process table (below) corresponding to phases in handling a signal. (a) State as process is taken out of execution. (b) State as handler begins execution. (c) State while sigreturn is executing. (d) State after sigreturn completes execution.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Initialization of Process Manager

cs	ds	text	data	bss	stack	
0000800	0005800	19552	3140	30076	0	kernel
0100000	0104c00	19456	2356	48612	1024	pm
0111800	011c400	43216	5912	6224364	2048	fs
070e000	070f400	4352	616	4696	131072	rs

Figure 4-44. Boot monitor display of memory usage of first few system image components.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Implementation of EXIT

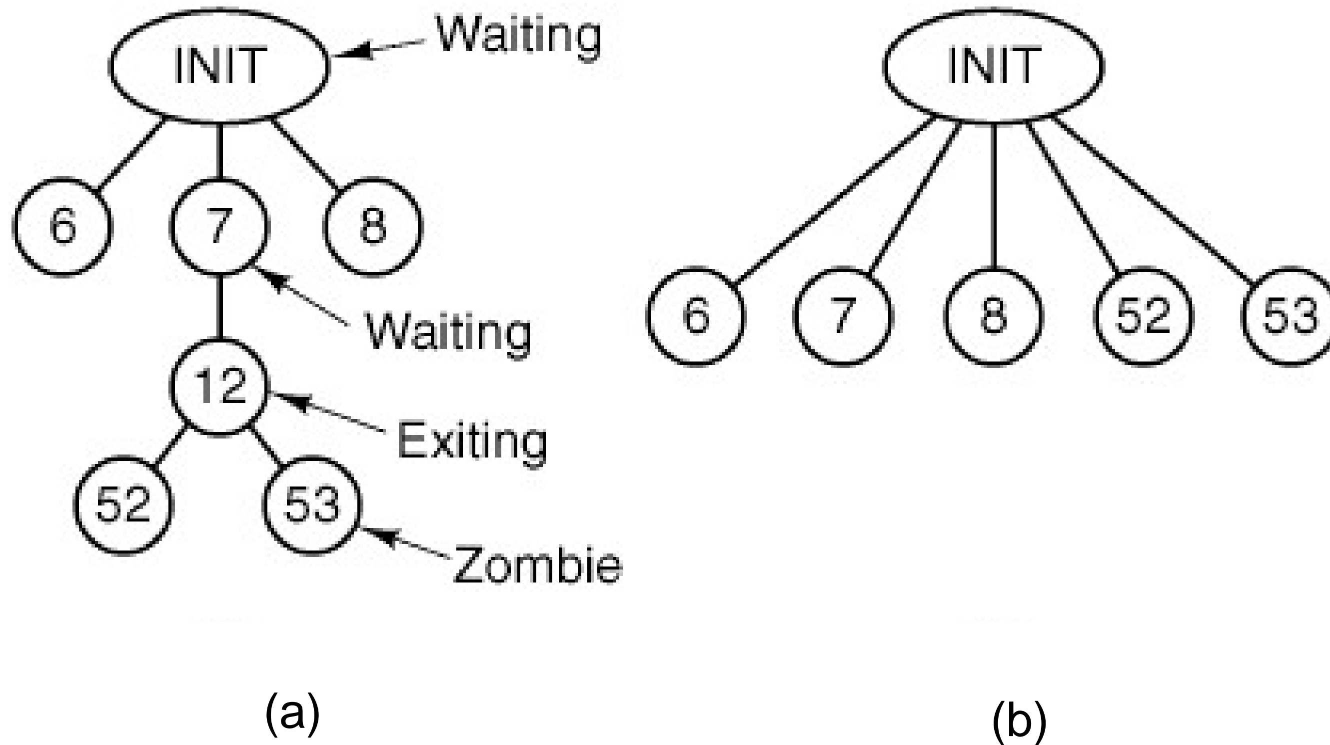


Figure 4-45. (a) The situation as process 12 is about to exit. (b) The situation after it has exited.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Implementation of EXEC

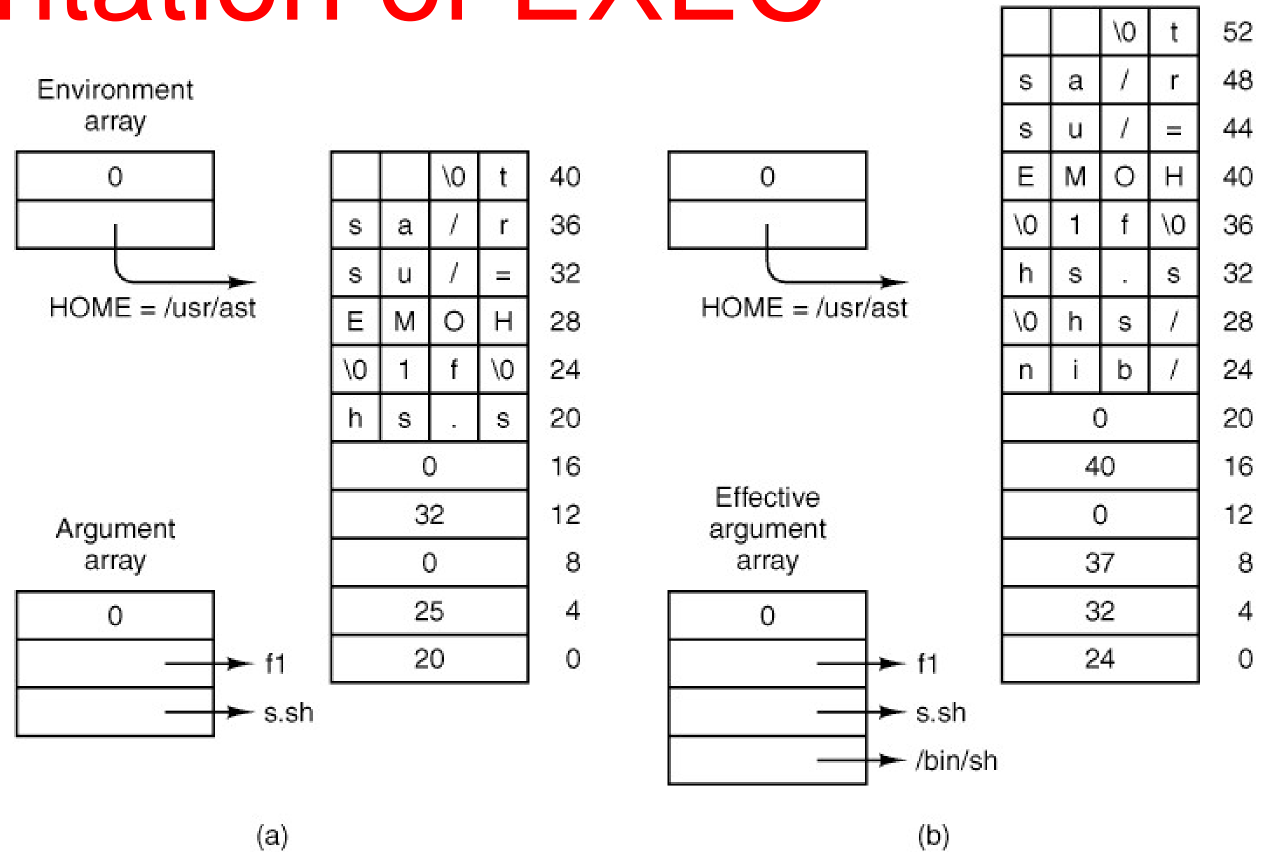


Figure 4-46. (a) Arrays passed to *execve* and the stack created when a script is executed. (b) After processing by *patch_stack*, the arrays and the stack look like this. The script name is passed to the program which interprets the script.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (1)

System call	Purpose
sigaction	Modify response to future signal
sigprocmask	Change set of blocked signals
kill	Send signal to another process
alarm	Send ALRM signal to self after delay
pause	Suspend self until future signal
sigsuspend	Change set of blocked signals, then PAUSE
sigpending	Examine set of pending (blocked) signals
sigreturn	Clean up after signal handler

Figure 4-47. System calls relating to signals.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (2)

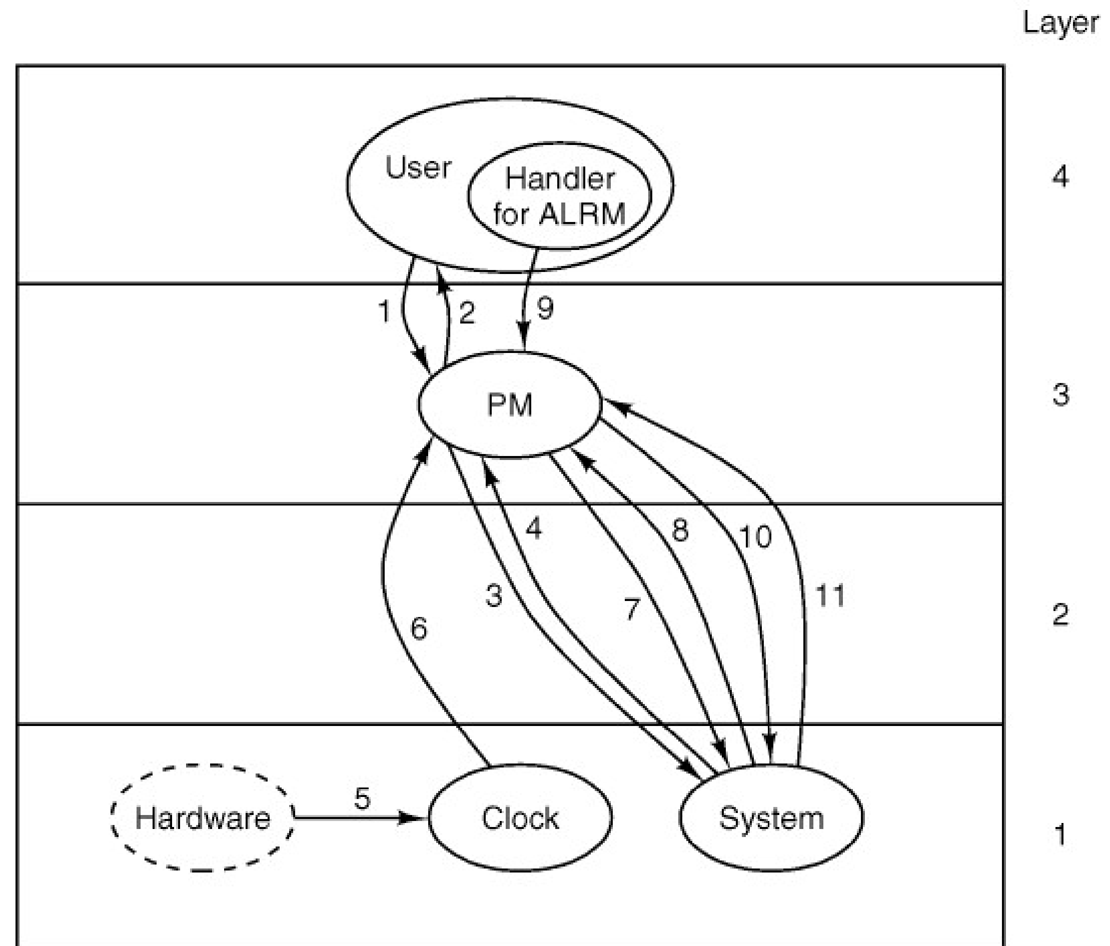


Figure 4-48. Messages for an alarm. The most important are: (1) User does alarm. (4) After the set time has elapsed, the signal arrives. (7) Handler terminates with call to sigreturn. See text for details.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Signal Handling (3)

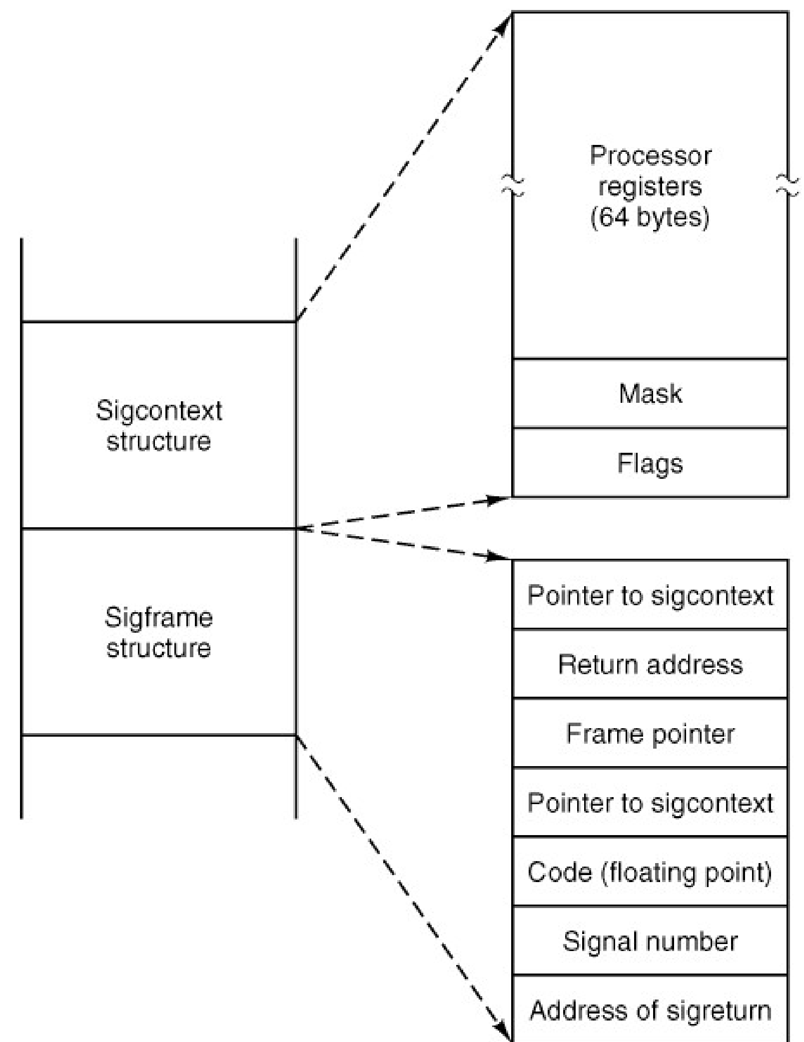


Figure 4-49. The sigcontext and sigframe structures pushed on the stack to prepare for a signal handler. The processor registers are a copy of the stackframe used during a context switch.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Other System Calls (1)

Call	Function
time	Get current real time and uptime in seconds
stime	Set the real time clock
times	Get the process accounting times

Figure 4-50. Three system calls involving time.

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Other System Calls (2)

System Call	Description
getuid	Return real and effective UID
getgid	Return real and effective GID
getpid	Return PIDs of process and its parent
setuid	Set caller's real and effective UID
setgid	Set caller's real and effective GID
setsid	Create new session, return PID
getpgrp	Return ID of process group

Figure 4-51. The system calls supported in *servers/pm/getset.c.*

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Other System Calls (3)

System Call	Description
do_allocmem	Allocate a chunk of memory
do_freemem	Deallocate a chunk of memory
do_getsysinfo	Get info about PM from kernel
do_getprocnr	Get index to proc table from PID or name
do_reboot	Kill all processes, tell FS and kernel
do_getsetpriority	Get or set system priority
do_svrctrl	Make a process into a server

Figure 4-52. Special-purpose MINIX 3 system calls in *servers/pm/misc.c.*

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Other System Calls (4)

Command	Description
T_STOP	Stop the process
T_OK	Enable tracing by parent for this process
T_GETINS	Return value from text (instruction) space
T_GETDATA	Return value from data space
T_GETUSER	Return value from user process table
T_SETINS	Set value in instruction space
T_SETDATA	Set value in data space
T_SETUSER	Set value in user process table
T_RESUME	Resume execution
T_EXIT	Exit
T_STEP	Set trace bit

Figure 4-53. Debugging commands supported by *servers/pm/trace.c.*

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

Memory Management Utilities

Three entry points of alloc.c

1. *alloc_mem* – request a block of memory of given size
2. *free_mem* – return memory that is no longer needed
3. *mem_init* – initialize free list when PM starts running

Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8