1) A goal of memory management is not to provide a convenient abstraction for programming

a) T

b) F *

(Slide 2)

2) A goal of memory management To allocate memory resources among competing processes to

a) T *

b) F

(Slide 2)

3) A goal of memory management is to minimize performance with maximum overhead

a) T

b) F *

(Slide 2)

4) Goals of memory management

a) To provide a convenient abstraction for programming

b) To allocate memory resources among competing processes

c) to maximize performance with minimal overhead

d) All of the above *

(Slide 2)

5) Mechanisms of memory management

a) partitioning

b) paging

c) segmentation

d) All of the above *

(Slide 2)

6) The abstraction that the OS will provide for managing memory is

a) Virtual Box

b) Virtual Reality

c) Virtual Memory *

d) Augmented Reality

(Slide 3)

7) virtual memory (VM) Enables a program to execute without need for complete data in physical memory

a) T *

b) F

(Slide 3)

8) A program can run on a machine with less memory than it “needs”

a) T *

b) F

(Slide 3)

9) Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it

a) T *

b) F

(Slide 3)

10) without sharing methods Processes cannot see the memory of others

a) T *

b) F

(Slide 3)

11) OS will adjust amount of memory allocated to a process based upon its behavior

a) T *

b) F

(Slide 3)

12) VM doesn't require hardware support or OS management algorithms to pull it off

a) T

b) F *

(Slide 3)

13) Programs use physical addresses directly, OS loads job, runs it, unloads it

a) old days *

b) Multiprogramming

(Slide 4)

14) Want multiple processes in memory at once and Can do it a number of ways

a) old days

b) Multiprogramming *

(Slide 4)

15) Fixed and variable partitioning, paging, segmentation

a) old days

b) Multiprogramming *

(Slide 4)

16) Overlap I/O and CPU of multiple jobs

a) old days

b) Multiprogramming *

(Slide 4)

17) updating memory hardware on context switch

a) Need protection

b) Fast translation

c) Fast change *

(slide 4)

18) lookups need to be fast

a) Need protection

b) Fast translation *

c) Fast change

(slide 4)

19) restrict which addresses jobs can use

a) Need protection *

b) Fast translation

c) Fast change

(slide 4)

20) To make it easier to manage the memory of processes running in the system, we're going to make them use virtual addresses

a) T *

b) F

(Slide 5)

21) virtual addresses are physical addresses

a) T

b) F *

(Slide 5 .. logical)

22) Virtual addresses are dependent of the actual physical location of the data referenced

a) T

b) F *

(Slide 5 .. independent)

23) OS determines location of data in physical memory

a) T *

b) F

(Slide 5)

24) Instructions executed by the CPU issue virtual addresses

a) T *

b) F

(Slide 5)

25) physical addresses are translated by hardware into virtual addresses (with help from OS)

a) T

b) F *

(Slide 5 .. swap)

26) The set of virtual addresses that can be used by a process comprises its virtual address space

a) T *

b) F

(Slide 5)

27) happens when there's a sufficient quantity of area within the memory to satisfy the memory request of a method

a) Internal Fragmentation

b) External Fragmentation *

(Slide 8)

28) the process's memory request cannot be fulfilled because the memory offered is in a noncontiguous manner

a) Internal Fragmentation

b) External Fragmentation *

(Slide 8)

29) happens when the memory is split into fixed-sized blocks

a) Internal Fragmentation *

b) External Fragmentation

(Slide 7)

30) because the difference between memory allocated and required space or memory

a) Internal Fragmentation *

b) External Fragmentation

(Slide 7)

31) because the memory is not contiguous

a) Internal Fragmentation

b) External Fragmentation *

(Slide 8)

32) can be solved by re-mapping between VA (Virtual Address) and PA (Physical Address)

a) Internal Fragmentation

b) External Fragmentation *

(Slide 9)

33) can be solved if the page size is relatively small

a) Internal Fragmentation *

b) External Fragmentation

(Slide 9)

34) Hardware requirements: base register and limit register

a) Fixed Partitions

b) Variable Partitions *

(Slide 12)

35) Hardware requirements: base register

a) Fixed Partitions *

b) Variable Partitions

(Slide 10)

36) Physical address = virtual address + base register

a) Fixed Partitions

b) Variable Partitions

c) Both of them *

(Slide 12)

37) Base register loaded by OS when it switches to a process

a) T *

b) F

(Slide 10)

38) we need the limit register for Protection

a) T *

b) F

(Slide 12)

39) If (physical address < base + limit) then exception fault

a) T

b) F *

(Slide 12)

40) job loading and unloading produces empty holes scattered throughout memory

a) Internal fragmentation

b) External fragmentation *

(Slide 12)

41) allocate just enough for process

a) Internal fragmentation

b) no Internal fragmentation *

(Slide 12)

42) Size of each partition is the same and fixed

a) Fixed Partitions *

b) Variable Partitions

(Slide 10)

43) Easy to implement, fast context switch

a) Fixed Partitions *

b) Variable Partitions

(Slide 10)

44) memory in a partition not used by a process is not available to other processes

a) Internal fragmentation *

b) External fragmentation

(Slide 10)

45) Physical memory is broken up into fixed partitions

a) Fixed Partitions *

b) Variable Partitions

(Slide 10)

46) physical memory is broken up into variable sized partitions

a) Fixed Partitions

b) Variable Partitions *

(Slide 10)

47) problems of Variable Partitions

a) External fragmentation *

b) Internal fragmentation

c) Partition size

d) b and c

(Slide 12)

48) problems of Fixed Partitions

a) External fragmentation

b) Internal fragmentation

c) Partition size

d) b and c *

(Slide 10)

49) No internal fragmentation is an advantage of

a) Fixed Partitions

b) Variable Partitions *

(Slide 12)

50) Easy to implement, fast context switch are advantages of

a) Fixed Partitions *

b) Variable Partitions

(Slide 10)

51) disk de-fragmentation can improve your system performance

a) T *

b) F

(Slide 14)

52) Compact memory by copying, first Swap a program out then Re-load it, adjacent to another finally Adjust its base register

a) T *

b) F

(Slide 15)

53) Processes must be suspended during compaction

a) T *

b) F

(Slide 16)

54) compaction Need be done only when fragmentation gets very bad

a) T *

b) F

(Slide 16)

55) paging Solve the …. fragmentation problem by using fixed sized units in both physical and virtual memory

a) internal

b) external *

(Slide 17)

56) paging Solve the … fragmentation problem by making the units small

a) internal *

b) external

(Slide 17)

57) in paging Processes view memory as a contiguous address space from bytes 0 through N (a virtual address space)

a) T *

b) F

(Slide 18)

58) in paging N is dependent of the actual hardware

a) T

b) F *

(Slide 18)

59) in paging virtual pages are scattered across physical memory frames – not contiguous as earlier

a) T *

b) F

(Slide 18)

60) in paging mapping is physical-to-virtual mapping

a) T

b) F *

(Slide 18)

61) in paging mapping is visible to the program

a) T

b) F *

(Slide 18)

62) in paging there's Efficient use of memory, because very little internal fragmentation

a) T *

b) F

(Slide 18)

63) in paging there's No external fragmentation at all

a) T *

b) F

(Slide 18)

64) in paging No need to copy big chunks of memory around to coalesce free space

a) T *

b) F

(Slide 18)

65) Page size is always a power of 3

a) T

b) F *

(Slide 19 .. 2)

66) Multiplication or division by 2^x can be replaced by bitwise left shift or bitwise right shift

a) T *

b) F

(Slide 19)

67) bitwise left shift, bitwise right shift are extremely fast, in comparison to regular multiplication or division by arbitrary integer

a) T *

b) F

(Slide 19)

68) A virtual address consists of two parts: the page and an offset into that page

a) T *

b) F

(Slide 20)

69) a virtual address has two parts: virtual page number & offset

a) T *

b) F

(Slide 22)

70) virtual page number (VPN) is index into a page table

a) T *

b) F

(Slide 22)

71) page table entry contains page frame number (PFN)

a) T *

b) F

(Slide 22)

72) physical address is VPN+offset

a) T

b) F *

(Slide 22 .. PFN+offset)

73) Page tables • map virtual page number (VPN) to page frame number (PFN)

a) T *

b) F

(Slide 22)

74) VPN is simply an index into the page table

a) T *

b) F

(Slide 22)

75) Since the page-tables are under the control of the operating system, if the virtual-address doesn't exist in the page-table then the operating system knows the process is trying to access memory that has not been allocated to it and the access will not be allowed

a) T *

b) F

(Slide 24)

76) says whether the page has been accessed

a) Modify bit

b) Reference bit *

c) Valid bit

d) Protection bits

e) page frame number

(Slide 25)

77) says whether or not the PTE can be used

a) Modify bit

b) Reference bit

c) Valid bit *

d) Protection bits

e) page frame number

(Slide 25)

78) say what operations are allowed on page

a) Modify bit

b) Reference bit

c) Valid bit

d) Protection bits *

e) page frame number

(Slide 25)

79) determines physical page

a) Modify bit

b) Reference bit

c) Valid bit

d) Protection bits

e) page frame number *

(Slide 25)

80) says whether or not the page has been written

a) Modify bit *

b) Reference bit

c) Valid bit

d) Protection bits

e) page frame number

(Slide 25)

81) It is set when a write to the page occurs

a) Modify bit *

b) Reference bit

c) Valid bit

d) Protection bits

e) page frame number

(Slide 25)

82) It is checked each time the virtual address is used

a) Modify bit

b) Reference bit

c) Valid bit *

d) Protection bits

e) page frame number

(Slide 25)

83) Read, write, execute

a) Modify bit

b) Reference bit

c) Valid bit

d) Protection bits *

e) page frame number

(Slide 25)

84) It is set when a read or write to the page occurs

a) Modify bit

b) Reference bit *

c) Valid bit

d) Protection bits

e) page frame number

(Slide 25)

85) Modify bit, Reference bit, Valid bit are of size .. bits

a) 0

b) 1 *

c) 2

d) determined by the size of physical memory

(Slide 25)

86) Protection bits are of size .. bits

a) 0

b) 1

c) 2 *

d) determined by the size of physical memory

(Slide 25)

87) page frame number bits are of size .. bits

a) 0

b) 1

c) 2

d) determined by the size of physical memory *

(Slide 25)

88) Single level page table size is too large

a) T *

b) F

(Slide 26)

89) Hierarchical Page Tables is to Break up the logical address space into multiple page tables • A simple technique is a two-level page table • We then page the page table

a) T *

b) F

(Slide 27)

90) Two-level page tables Virtual addresses (VAs) have three parts: Master page number, secondary page number, and offset

a) T *

b) F

(Slide 28)

91) in Two-level page tables Master page table maps VAs to secondary page table then Secondary page table maps page number to physical page then Offset indicates where in physical page address is located

a) T *

b) F

(Slide 28)

92) in 32 bit Two-Level Paging A logical address is divided into a page number consisting of 10 bits and a page offset consisting of 22 bits

a) T

b) F *

(Slide 31 .. 22,10)

93) the page number is further divided into: a 10-bit page number and a 12-bit page offset

a) T

b) F *

(Slide 31 .. 12,10)

94) in Two-level page tables p1 is an index into the outer page table, and p2 is the displacement within the page of the inner page table • Known as forward-mapped page table

a) T *

b) F

(Slide 31)

95) in 64-bit Two-Level paging isn't sufficient

a) T

b) F *

(Slide 33)

96) in 64 bit Two-Level Paging A logical address is divided into a page number consisting of 52 bits and a page offset consisting of 12 bits

a) T *

b) F

(Slide 33)

97) paging Can still have internal fragmentation

a) T *

b) F

(Slide 36)

98) paging can lead to Memory reference overhead

a) T *

b) F

(Slide 36)

99) we can use a hardware cache of lookups for solving Memory reference overhead problem

a) T *

b) F

(Slide 36)

100) External fragmentation not a problem with paging

a) T *

b) F

(Slide 35)

101) in paging Allocating a page is just removing it from the list

a) T *

b) F

(Slide 35)

102) in paging Memory comes from a free list of fixed size chunks

a) T *

b) F

(Slide 35)

103) in paging All chunks are the same size

a) T *

b) F

(Slide 35)

104) in paging Use valid bit to detect references to swapped pages

a) T *

b) F

(Slide 35)

105) in paging Pages are a convenient multiple of the disk block size

a) T *

b) F

(Slide 35)

106) in paging

a) Easy to allocate memory

b) Easy to swap out chunks of a program

c) both of them *

d) Neither

(Slide 35)

107) Move one/several/all pages of a process to disk

a) context

b) swapping *

(Slide 37)

108) "....." is the unit of swapping

a) bit

b) byte

c) page *

d) word

(Slide 37)

109) The freed physical memory can be mapped to other pages

a) T *

b) F

(Slide 37)

110) Processes that use large memory can be swapped out (but can't later back in)

a) T

b) F *

(Slide 37 .. can back)

111) is a technique that partitions memory into logically related data units

a) paging

b) Segmentation *

(Slide 42)

112) In Variable-sized partitions process is shared with 1 segment only

a) T *

b) F

(Slide 42)

113) In Segmentation process is shared with many segments

a) T *

b) F

(Slide 42)

114) Virtual addresses become <segment #, offset>

a) paging

b) Segmentation *

(Slide 42)

115) view an address space as a linear array of bytes

a) paging *

b) Segmentation

(Slide 42)

116) divide it into pages of equal size

a) paging *

b) Segmentation

(Slide 42)

117) use a page table to map virtual pages to physical page frames

a) paging *

b) Segmentation

(Slide 42)

118) page (logical) => page frame (physical)

a) paging *

b) Segmentation

(Slide 42)

119) partition an address space into logical units

a) paging

b) Segmentation *

(Slide 42)

120) a virtual address is <segment #, offset>

a) paging

b) Segmentation *

(Slide 42)

121) More “logical”

a) paging

b) Segmentation *

(Slide 42)

122) a segment is a natural unit of sharing

a) T *

b) F

(Slide 44)

123) segmentation = 1 segment/process

a) T

b) F *

(Slide 44)

124) variable-sized partition = many segments/process

a) T

b) F *

(Slide 44)

125) segments are really independent; segmentation treats them as such

a) T *

b) F

(Slide 44)

123) segments named by segment #, used as index into table

a) T *

b) F

(Slide 45)

124) offset of virtual address added to base address of segment to yield physical address

a) T *

b) F

(Slide 45)

125) modern architectures support both segments and paging

a) T *

b) F

(Slide 47)

126) segments vary in size, but are typically large (multiple pages)

a) T *

b) F

(Slide 47)

127) manage in chunks from user's perspective

a) Paging

b) Segmentation *

(Slide 48)

128) with combining paging with segmentation there’s no contiguous allocation, no external fragmentation

a) T *

b) F

(Slide 47)