Jon Tice

Csci 411

3/3/2018

**TANNENBAUM Book questions**

|  |
| --- |
| **Memory Management** |

1. (4 pts) Consider a logical address space of 64 pages of 1,024 bytes each, mapped onto a physical memory of 32 frames.
   1. How many bits are there in the logical address?

1024=2^10

64=2^6

The logical address is 6+10= 16 bits

* 1. How many bits are there in the physical address?

1024=2^10

32=2^5

The physical address is 15 bits

1. (4 pts) Consider a logical address space of 32 pages of 512 bytes each, mapped onto a physical memory of 64 frames.
2. How many bits are there in the logical address?

512=2^9

32 = 2^5

Logical address = 5+9=14 bits

1. How many bits are there in the physical address?

512 = 2^9

64 = 2^6

Physical Address = 9+6=15 bits

1. (5 pts) Page Address Translation

Assuming a 1 KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

1024 = 2^10 = bits for offset

1. 2375

00000000000000000000100101000111

Offset = 0101000111 = 327

Page Number = 0000000000000000000010 = 2

1. 19366

00000000000000000100101110100110

Offset = 1110100110 = 934

Page Number = 0000000000000000010010 = 18

1. 30000

00000000000000000111010100110000

Offset = 0100110000 = 304

Page Number = 0000000000000000011101 = 29

1. 256

00000000000000000000000100000000

Offset = 0100000000 = 256

Page Number = 0000000000000000000000 = 0

1. 16385

00000000000000000100000000000001

Offset = 0000000001 = 1

Page Number = 0000000000000000010000 = 16

1. (5 pts) Sharing memory pages
   1. What is the effect of allowing two entries in a page table to point to the same page frame in memory? Explain how this effect could be used to decrease the amount of time needed to copy a large amount of memory from one place to another.

This allows user to share code and data, if the code is reentrant space can be saved by using programs like text editors and compilers. Copying large amounts of memory could be affected by this change. Two pages in a single process can reference the same physical page so if one thing is changed at an address the same offset of the other page will change as well, this allows large amounts of to be copied quickly by mapping a page read-only at the two addresses.

1. (12 pts) Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would the best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Best fit makes the most efficient use of the memory.

200 KB

100 KB

500 KB

600 KB

300 KB

Best-Fit

212 KB -> 300 KB with a 88 KB partition

417 KB -> 500 KB with a 83 KB partition

112 KB -> 200 KB with a 88 KB partition

426 KB -> 600 KB with a 174 KB partition

Worst Fit

212 KB -> 600 KB with a 388 KB partition

417 KB -> 500 KB with a 83 KB partition

112 KB -> 388 KB with a 276 KB partition

426 KB could not be allocated because there is no partition large enough

|  |
| --- |
| **Virtual Memory** |

1. (5pts) Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.

A page fault occurs when a page that has not been put into main memory is attempted to be accessed. The OS then verifies the memory access and terminates the program if it is not valid. If it is valid then a free frame is located and the page is read into the free frame then the instruction is restarted once the page and process table is updated.

1. (5 pts) Define Working Set Model.

Working Set Model defines the amount of memory that a process requires in a given amount of time.

**Additional Questions**

*We didn’t talk about all of these in class. Some are in your book. Some you may have to look up.*

1. (5 pts) What is the copy-on-write feature, and under what circumstances is it beneficial to use this feature?

Copy-on-write allows processes to share pages than each having a separate copy of the pages. This is beneficial for the fork() operation because the child must have a copy of the parents address space so the child shares the parents pages until a write is issued then a copy is made.

1. (5 pts) Can you start a program before it is completely loaded in memory? How and why or why not?

If the part of your program that is executable and is loaded into memory along with that the program needs to initially start is also loaded in then yes you can, if there is an access to a page that has not been loaded in yet then a page fault will be generated.

1. (5 pts) What is the difference between *on-demand* page replacement and *pre-fetch* page replacement? Would it benefit our OS to use them in concert?

On demand only loads pages that are demanded by the operating system which allows more space in main memory reducing context switching time, whereas pre fetch gets operations who result is expected to be needed soon but this risks wasting time by fetching data that will not be used.

1. Consider the following page reference string:

1, 2, 3, 4, 2, 1,5,3,6,7,2,1,2,3,7,6,3,5

How many page faults would occur for the following replacement algorithms, assuming, three, frames? Remember that all frames are initially empty, so your first unique pages will cost one fault each.

1. (5 pts) LRU replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **1** | **1** | **4** | **4** | **4** | **5** | **5** | **5** | **7** | **7** | **7** | **7** | **3** | **3** | **3** | **3** | **3** |  |  |
|  | **2** | **2** | **2** | **2** | **2** | **2** | **3** | **3** | **3** | **2** | **2** | **2** | **2** | **2** | **6** | **6** | **6** |  |  |
|  |  | **3** | **3** | **3** | **1** | **1** | **1** | **6** | **6** | **6** | **1** | **1** | **1** | **7** | **7** | **7** | **5** |  |  |

**Page Faults: 15**

1. (5 pts) Second chance replacement, aka clock replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **1** | **1** | **4** | **4** | **4** | **5** | **5** | **5** | **7** | **7** | **7** | **7** | **3** | **3** | **3** | **3** | **3** |  |  |
|  | **2** | **2** | **2** | **2** | **2** | **2** | **3** | **3** | **3** | **2** | **2** | **2** | **2** | **2** | **6** | **6** | **6** |  |  |
|  |  | **3** | **3** | **3** | **1** | **1** | **1** | **6** | **6** | **6** | **1** | **1** | **1** | **7** | **7** | **7** | **5** |  |  |

**Page Faults: 15**

1, 2, 3, 4, 2, 1,5,3,6,7,2,1,2,3,7,6,3,5

1. (5 pts) Optimal or MIN replacement Algorithm

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **5** |  |  |
|  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **6** | **6** | **6** |  |  |
|  |  | **3** | **4** | **4** | **4** | **5** | **3** | **6** | **6** | **6** | **1** | **1** | **3** | **3** | **3** | **3** | **3** |  |  |

**Page Faults:12**

1. (10 pts) How do we execute an operating system on top of another operating system and how can we use that abstraction to introduce new operating system services? (basically summarize section 10.2

You can do this by use of a virtual machine which can run and think as if it is the only one using memory but the memory is being managed by the MMU giving a layer of abstraction to the running operating systems services.

 (20 pts) **Segmented Memory.** Consider the following segmented memory state.

|  |  |
| --- | --- |
| http://www.cs.utexas.edu/users/mckinley/372/homework/hwk4.gif | Given the following memory state, requests, and algorithms show the state of memory after each request. If the current state cannot accommodate a request, perform process relocation (for a growing process) or compaction. For relocation, follow the specified algorithm. For compaction, first find (top down) a process with holes on both sides of it. If the two holes are sufficient to accommodate the new process, move up the middle process in the hole. Otherwise, compact all processes up until there is large enough hole to accommodate the request.   * Process E starts and requests 300 memory units. * Process A requests 400 more memory units. * Process B exits. * Process F starts and requests 800 memory units. * Process C exits. * Process G starts and requests 900 memory units.  1. (6 pts) Describe the contents of memory after each request using first fit (draw a picture or write the range of addresses for each process).   C:\Users\chili\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Memory.jpg   1. (6 pts) Describe the contents of memory after each request using best fit.  1st Request – E will take up 1500-1800 the rest will remain the same   2nd Request – A will take 300-700 the rest will be unchanged  3rd Request – 900-1500 will no longer be taken by B the rest is unchanged  4th Request – F will take 900-1800 the rest is unchanged  5th Request -1800-2000 is no longer taken by C the rest is unchanged  6th Request - 1800-2700 is taken by G the rest is unchanged   1. (6 pts) Describe the contents of memory after each request using worst fit.   1st Request – E will take up 2000-2300 the rest will remain the same  2nd Request – A will take 2300-2700 the rest will be unchanged  3rd Request – 900-1500 will no longer be taken by B the rest is unchanged  4th Request – F will take 300-1100 the rest is unchanged  5th Request -1800-2000 is no longer taken by C the rest is unchanged  6th Request - 1100-2000 is taken by G the rest is unchanged   1. (2 pts) For this example, which is best?   Best fit is the best for this example. |

 (5 pts) **Paging.** Consider a process with a logical address space of 4 pages of 1024 bytes per page, mapped onto a physical memory of 64 frames.

1. (2 pts) How many bits are there in the logical address?

1024=2^10

2=2^2

2+10=12 bits

1. (2 pts) How many bits are there in the physical address?

1024=2^10

64=2^6

6+10 = 16 bits

1. (1 pts) Given the following page table map: page 0 is mapped to frame 3, page 1 is mapped to frame 14, page 2 to frame 6, and page 3 to frame 33, what is the physical address of page 2 byte 256?

256 = 0010000000

6 = 000010

Physical Address: 0000100010000000

 (5 pts) **More Paging.** Consider a paging system with the page table stored in memory.

1. (2 pts) If a memory reference takes 60 nanoseconds, how long does a paged memory reference take?

120 nanoseconds

60 to access the page table + 60 to access the word in memory

1. (3 pts) If we add associative registers (a TLB), what percent of page table references need to hit in the TLB to get an effective access time of 65 nanoseconds?

About 54%

 (10 pts) **TLBs.** What is the advantage of a TLB that has an entry for every page frame? What is the advantage of a TLB smaller than the total number of frames? Which architectural design decisions determine this size?

This allows faster translation of addresses to their actual spot in main memory having a small TLB takes advantage of the principals of locality, spatial locality in this case, which will help keep the page faults to a minimum.

 (10 pts) **Sharing.** Is data sharing easier in paged or segmented memory schemes? Why?

Segmentation distinguishes and separately protects procedures and data whereas paging does not facilitate sharing of procedures or distinguish between process and data separately. Segmentation requires the programmer to be aware of memory limits for risk of a segmentation fault can occur.

 (10 pts) **Virtual Memory.** When processes are allowed to grow larger than memory, page tables also grow very large. How could we organize page tables and TLB to keep access times as quick as possible for codes with good locality? For example, assume physical memory is 512K, each page is 1K, and a TLB of size 128. If we assume most processes are 256K or less, then we could allocate a fixed-size page table with 256 entries. Now in the unexpected case, where the page table grows larger than 256 entries, how should we organize it? What implications does your design have on average access time and on the maximum virtual memory size of a program?

Double the size of each page from 1K to 2K.

 (20 pts) **Page Replacement Algorithms.** Consider the following page reference stream and 3 page frames:   
0 1 2 3 2 4 3 1 1 5 2 4 6 3 3 4 6 3 4 7   
For each algorithm, show the contents of the page frame after each reference, and then compute the total number of page faults, divided in to cold misses and other misses.

1. MIN

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** |  |  |
|  | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **5** | **2** | **6** | **6** | **6** | **6** | **6** | **6** | **6** |  |  |
|  |  | **2** | **2** | **2** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **7** |  |  |

Page Faults: 8

1. FIFO

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **2** | **2** | **2** | **3** | **3** | **3** | **3** | **3** |  |  |
|  | **1** | **1** | **1** | **1** | **4** | **1** | **1** | **1** | **1** | **1** | **6** | **6** | **6** | **6** | **6** | **6** | **6** |  |  |
|  |  | **2** | **2** | **2** | **2** | **4** | **4** | **4** | **5** | **5** | **5** | **4** | **4** | **4** | **4** | **4** | **7** |  |  |

Page Faults: 12

1. LRU

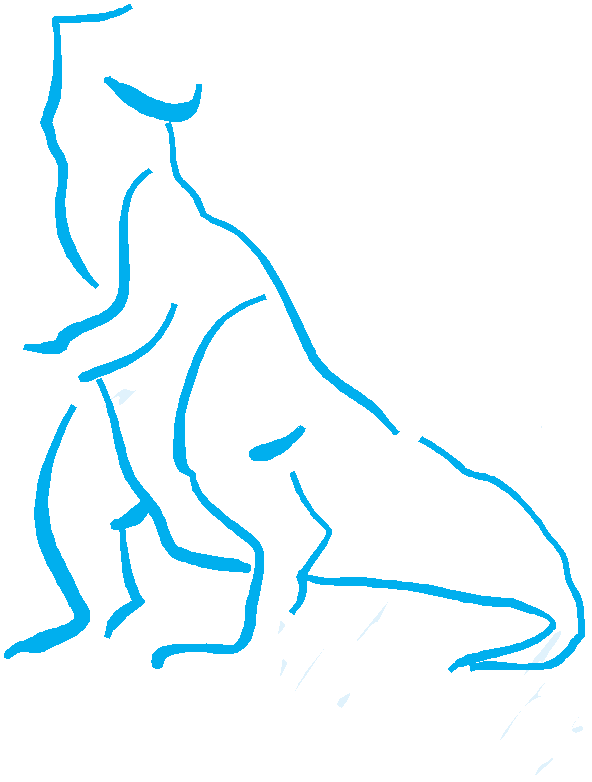
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** |  |  |
|  | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **5** | **2** | **6** | **6** | **6** | **6** | **6** | **6** | **6** |  |  |
|  |  | **2** | **2** | **2** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **7** |  |  |

Page Faults

1. Clock with one bit, where after 5 references, the OS sets all reference bits to 0. Show the clock bits states separately.
2. For 4 page frames, what is the minimum number of misses possible in general, for this stream?

The minimum number is 4 misses using MIN.

Main Memory



Exercises

C H8APTER

**8.9** Explain the difference between internal and external fragmentation.

**Answer:**

a. Internal fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released.

b. External fragmentation is unused space between allocated regions of memory. Typically external fragmentation results in memory regions that are too small to satisfy a memory request, but if we were to combine all the regions of external fragmentation, we would have enough memory to satisfy a memory request.

**8.10** Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linkage editor is used to combine multiple object modules into a single program binary. How does the linkage editor change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linkage editor to facilitate the memory-binding tasks of the linkage editor?

**Answer:**

The linkage editor has to replace unresolved symbolic addresses with

the actual addresses associated with the variables in the final program

binary. In order to perform this, the modules should keep track of

instructions that refer to unresolved symbols. During linking, each

module is assigned a sequence of addresses in the overall program

binary and when this has been performed, unresolved references to

symbols exported by this binary could be patched in other modules

since every other module would contain the list of instructions that

need to be patched.

**8.11** Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit

**53**

algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and

375 KB (in order)? Rank the algorithms in terms of how efficiently they

use memory.

**Answer:**

a. **First-fit**:

b. 115 KB is put in 300 KB partition, leaving (185 KB, 600 KB, 350 KB,

200 KB, 750 KB, 125 KB)

c. 500 KB is put in 600 KB partition, leaving (185 KB, 100 KB, 350 KB,

200 KB, 750 KB, 125 KB)

d. 358 KB is put in 750 KB partition, leaving (185 KB, 100 KB, 350 KB,

200 KB, 392 KB, 125 KB)

e. 200 KB is put in 350 KB partition, leaving (185 KB, 100 KB, 150 KB,

200 KB, 392 KB, 125 KB)

f. 375 KB is put in 392 KB partition, leaving (185 KB, 100 KB, 150 KB,

200 KB, 17 KB, 125 KB)

g. **Best-fit**:

h. 115 KB is put in 125 KB partition, leaving (300 KB, 600 KB, 350 KB,

200 KB, 750 KB, 10 KB)

i. 500 KB is put in 600 KB partition, leaving (300 KB, 100 KB, 350 KB,

200 KB, 750 KB, 10 KB)

j. 358 KB is put in 750 KB partition, leaving (300 KB, 100 KB, 350 KB,

200 KB, 392 KB, 10 KB)

k. 200 KB is put in 200 KB partition, leaving (300 KB, 100 KB, 350 KB, 0

KB, 392 KB, 10 KB)

l. 375 KB is put in 392 KB partition, leaving (300 KB, 100 KB, 350 KB, 0

KB, 17 KB, 10 KB)

m. **Worst-fit**:

n. 115 KB is put in 750 KB partition, leaving (300 KB, 600 KB, 350 KB,

200 KB, 635 KB, 125 KB)

o. 500 KB is put in 635 KB partition, leaving (300 KB, 600 KB, 350 KB,

200 KB, 135 KB, 125 KB)

p. 358 KB is put in 600 KB partition, leaving (300 KB, 242 KB, 350 KB,

200 KB, 135 KB, 125 KB)

q. 200 KB is put in 350 KB partition, leaving (300 KB, 242 KB, 150 KB,

200 KB, 135 KB, 125 KB)

r. 375 KB must wait

In this example, only worst-fit does not allow a request to be satisfied. An argument could be made that best-fit is most efficient as it leaves the largest holes after allocation. However, best-fit runs at time *O*(*n*) and first-fit runs in constant time *O*(1).

**8.12** Most systems allow a program to allocate more memory to its address space during execution. Allocation of data in the heap segments of programs is an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes?

a. Contiguous memory allocation b. Pure segmentation

c. Pure paging

**Answer:**

a. contiguous-memory allocation: might require relocation of the entire program since there is not enough space for the program to grow its allocated memory space.

b. pure segmentation: might also require relocation of the segment that needs to be extended since there is not enough space for the segment to grow its allocated memory space.

c. pure paging: incremental allocation of new pages is possible in this scheme without requiring relocation of the program’s address space.

**8.13** Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:

a. External fragmentation b. Internal fragmentation

c. Ability to share code across processes

**Answer:**

The contiguous memory allocation scheme suffers from external frag-

mentation as address spaces are allocated contiguously and holes

develop as old processes die and new processes are initiated. It also

does not allow processes to share code, since a process’s virtual memory

segment is not broken into noncontiguous finegrained segments. Pure

segmentation also suffers from external fragmentation as a segment of a

process is laid out contiguously in physical memory and fragmentation

would occur as segments of dead processes are replaced by segments of

new processes. Segmentation, however, enables processes to share code;

for instance, two different processes could share a code segment but have

distinct data segments. Pure paging does not suffer from external frag-

mentation, but instead suffers from internal fragmentation. Processes are

allocated in page granularity and if a page is not completely utilized, it

results in internal fragmentation and a corresponding wastage of space.

Paging also enables processes to share code at the granularity of pages.

**8.14** On a system with paging, a process cannot access memory that it does not own. Why? How could the operating system allow access to other memory? Why should it or should it not?

**Answer:**

An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table. To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process’s page table. This is useful when two or more processes need to exchange data — they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.

**8.15** Explain why mobile operating systems such as iOS and Android do not support swapping.

**Answer:** There are three reasons: First is that these mobile devices typically use flash memory with limited capacity and swapping is avoided because of this space constraint. Second, flash memory can support a limited number of write operations before it becomes less reliable. Lastly, there is typically poor throughput between main memory and flash memory.

**8.16** Although Android does not support swapping on its boot disk, it is possible to set up a swap space using a separate SD nonvolatile memory card. Why would Android disallow swapping on its boot disk yet allow it on a secondary disk?

**Answer:** Primarily because Android does not wish for its boot disk to

be used as swap space for the reasons outlined in the previous question

– the boot disk has limited storage capacity. However, Android does

support swapping, it is just that users must provide their own separate

SD card for swap space.

**8.17** Compare paging with segmentation with respect to how much memory the address translation structures require to convert virtual addresses to physical addresses.

**Answer:**

Paging requires more memory overhead to maintain the translation

structures. Segmentation requires just two registers per segment: one

to maintain the base of the segment and the other to maintain the extent

of the segment. Paging on the other hand requires one entry per page,

and this entry provides the physical address in which the page is located.

**8.18** Explain why address space identifiers (ASIDs) are used.

**Answer:** ASIDs provide address space protection in the TLB as well as

supporting TLB entries for several different processes at the same time.

**8.19** Program binaries in many systems are typically structured as follows.

Code is stored starting with a small, fixed virtual address, such as 0. The

code segment is followed by the data segment that is used for storing

the program variables. When the program starts executing, the stack is

allocated at the other end of the virtual address space and is allowed

to grow toward lower virtual addresses. What is the significance of this structure for the following schemes?

a. Contiguous memory allocation b. Pure segmentation

c. Pure paging

**Answer:**

1) Contiguous-memory allocation requires the operating system to

allocate the entire extent of the virtual address space to the program

when it starts executing. This could be much larger than the actual

memory requirements of the process. 2) Pure segmentation gives the

operating system flexibility to assign a small extent to each segment at

program startup time and extend the segment if required. 3) Pure paging

does not require the operating system to allocate the maximum extent of

the virtual address space to a process at startup time, but it still requires

the operating system to allocate a large page table spanning all of the

program’s virtual address space. When a program needs to extend the

stack or the heap, it needs to allocate a new page but the corresponding

page table entry is preallocated.

**8.20** Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

|  |  |
| --- | --- |
| a. | 3085 |
| b. | 42095 |
| c. | 215201 |
| d. | 650000 |
| e. | 2000001 |

**Answer:**

a. page = 3; offset = 13

b. page = 41; offset = 111

c. page = 210; offset = 161 d. page = 634; offset = 784 e. page = 1953; offset = 129

**8.21** The BTV operating system has a 21-bit virtual address, yet on certain embedded devices, it has only a 16-bit physical address. It also has a

2-KB page size. How many entries are there in each of the following?

a. A conventional, single-level page table b. An inverted page table

**Answer:** Conventional, single-level page table will have 210 = 1024 entries. Inverted page table will have 25 = 32 entries.

**8.22** What is the maximum amount of physical memory in the BTV operating system?

**Answer:** 216 = 65536 (or 64-KB.)

**8.23** Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

a. How many bits are required in the logical address?

b. How many bits are required in the physical address?

**Answer:**

a. 12 + 8 = 20 bits. b. 12 + 6 = 18 bits.

**8.24** Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512 MB of physical memory. How many entries are there in each of the following?

a. A conventional single-level page table b. An inverted page table

**Answer:**

a. 220 entries.

b. 512 K K/4K = 128K entries.

**8.25** Consider a paging system with the page table stored in memory.

a. If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?

b. If we add TLBs, and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

**Answer:**

a. 400 nanoseconds: 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.

b. Effective access time = 0.75 × (200 nanoseconds) + 0.25 × (400 nanoseconds) = 250 nanoseconds.

**8.26** Why are segmentation and paging sometimes combined into one scheme?

**Answer:**

Segmentation and paging are often combined in order to improve upon

each other. Segmented paging is helpful when the page table becomes

very large. A large contiguous section of the page table that is unused

can be collapsed into a single-segment table entry with a page-table

address of zero. Paged segmentation handles the case of having very

long segments that require a lot of time for allocation. By paging the

segments, we reduce wasted memory due to external fragmentation as well as simplify the allocation.

**8.27** Explain why sharing a reentrant module is easier when segmentation is used than when pure paging is used.

**Answer:**

Since segmentation is based on a logical division of memory rather than

a physical one, segments of any size can be shared with only one entry

in the segment tables of each user. With paging there must be a common

entry in the page tables for each page that is shared.

**8.28** Consider the following segment table:

|  |  |  |
| --- | --- | --- |
| Segment    0 | Base    219 | Length    600 |
| 1 | 2300 | 14 |
| 2 | 90 | 100 |
| 3 | 1327 | 580 |
| 4 | 1952 | 96 |

What are the physical addresses for the following logical addresses?

|  |  |
| --- | --- |
| a. | 0,430 |
| b. | 1,10 |
| c. | 2,500 |
| d. | 3,400 |
| e. | 4,112 |

**Answer:**

a. 219 + 430 = 649 b. 2300 + 10 = 2310

c. illegal reference, trap to operating system d. 1327 + 400 = 1727

e. illegal reference, trap to operating system

**8.29** What is the purpose of paging the page tables?

**Answer:**

In certain situations the page tables could become large enough that

by paging the page tables, one could simplify the memory allocation

problem (by ensuring that everything is allocated as fixed-size pages

as opposed to variable-sized chunks) and also enable the swapping of

portions of page table that are not currently used.

**8.30** Consider the hierarchical paging scheme used by the VAX architecture.

How many memory operations are performed when a user program

executes a memory-load operation?

**Answer:**

When a memory load operation is performed, there are three memory operations that might be performed. One is to translate the position where the page table entry for the page could be found (since page tables themselves are paged). The second access is to access the page table entry itself, while the third access is the actual memory load operation.

**8.31** Compare the segmented paging scheme with the hashed page table scheme for handling large address spaces. Under what circumstances is one scheme preferable to the other?

**Answer:**

When a program occupies only a small portion of its large virtual address

space, a hashed page table might be preferred due to its smaller size.

The disadvantage with hashed page tables however is the problem that

arises due to conflicts in mapping multiple pages onto the same hashed

page table entry. If many pages map to the same entry, then traversing

the list corresponding to that hash table entry could incur a significant

overhead; such overheads are minimal in the segmented paging scheme

where each page table entry maintains information regarding only one

page.

**8.32** Consider the Intel address-translation scheme shown in Figure 8.22.

a. Describe all the steps taken by the Intel Pentium in translating a logical address into a physical address.

b. What are the advantages to the operating system of hardware that provides such complicated memory translation?

c. Are there any disadvantages to this address-translation system? If so, what are they? If not, why is this scheme not used by every manufacturer?

**Answer:**

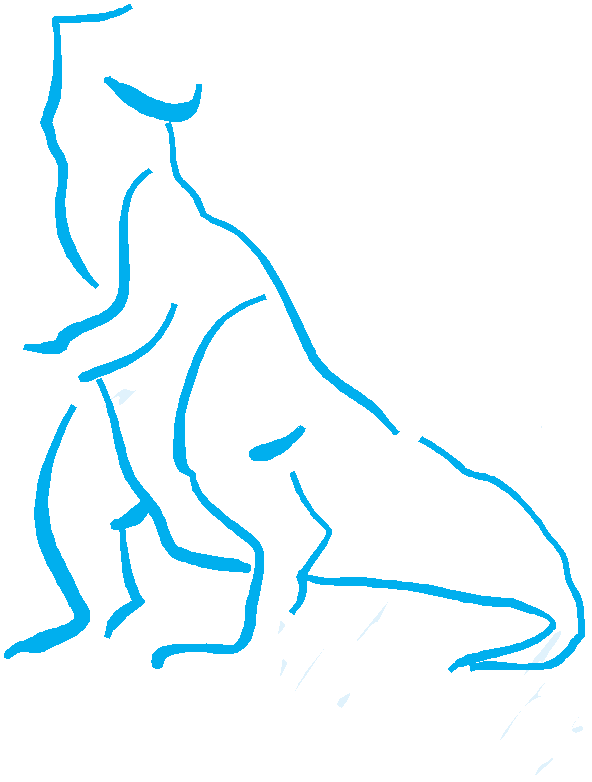
a. The selector is an index into the segment descriptor table. The segment descriptor result plus the original offset is used to produce a linear address with a dir, page, and offset. The dir is an index into a page directory. The entry from the page directory selects the page table, and the page field is an index into the page table. The entry from the page table, plus the offset, is the physical address.

b. Such a page-translation mechanism offers the flexibility to allow most operating systems to implement their memory scheme in hardware, instead of having to implement some parts in hardware and some in software. Because it can be done in hardware, it is more efficient (and the kernel is simpler).

c. Address translation can take longer due to the multiple table lookups it can invoke. Caches help, but there will still be cache misses.

Virtual

Memory



C H9APTER

Virtual memory can be a very interesting subject since it has so many different aspects: page faults, managing the backing store, page replacement, frame allocation, thrashing, page size. The objectives of this chapter are to explain these concepts and show how paging works.

A simulation is probably the easiest way to allow the students to program

several of the page-replacement algorithms and see how they really work.

If an interactive graphics display can be used to display the simulation as it

works, the students may be better able to understand how paging works. We

also present an exercise that asks the student to develop a Java program that

implements the FIFO and LRU page-replacement algorithms.

Exercises

**9.14** Assume a program has just referenced an address in virtual memory.

Describe a scenario how each of the following can occur: (If a scenario

cannot occur, explain why.)

• TLB miss with no page fault

• TLB miss and page fault

• TLB hit and no page fault

• TLB hit and page fault

**Answer:**

• TLB miss with no page fault page has been brought into memory, but has been removed from the TLB

• TLB miss and page fault page fault has occurred

• TLB hit and no page fault page is in memory and in the TLB. Most likely a recent reference

• TLB hit and page fault cannot occur. The TLB is a cache of the page table. If an entry is not in the page table, it will not be in the TLB.

**61**

**9.15** A simplified view of thread states is **Ready**, **Running**, and **Blocked**, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O). This is illustrated in Figure 9.31. Assuming a thread is in the Running state, answer the following questions, and explain your answer:

a. Will the thread change state if it incurs a page fault? If so, to what new state?

b. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what new state?

c. Will the thread change state if an address reference is resolved in the page table? If so, to what new state?

**Answer:**

• On a page fault the thread state is set to blocked as an I/O operation is required to bring the new page into memory.

• On a TLB-miss, the thread continues running if the address is resolved in the page table.

• The thread will continue running if the address is resolved in the page table.

**9.16** Consider a system that uses pure demand paging:

a. When a process first start execution, how would you characterize the page fault rate?

b. Once the working set for a process is loaded into memory, how would you characterize the page fault rate?

c. Assume a process changes its locality and the size of the new working set is too large to be stored into available free memory. Identify some options system designers could choose from to handle this situation?

**Answer:**

a. Initially quite high as needed pages are not yet loaded into memory. b. It should be quite low as all necessary pages are loaded into

memory.

c. (1) Ignore it; (2) get more physical memory; (3) reclaim pages more aggressively due to the high page fault rate.

**9.17** What is the copy-on-write feature, and under what circumstances is it beneficial? What hardware support is required to implement this feature?

**Answer:**

When two processes are accessing the same set of program values (for

instance, the code segment of the source binary), then it is useful to

map the corresponding pages into the virtual address spaces of the

two programs in a write-protected manner. When a write does indeed

take place, then a copy must be made to allow the two programs to

individually access the different copies without interfering with each other. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write protected. If it is indeed write protected, a trap would occur and the operating system could resolve the issue.

**9.18** A certain computer provides its users with a virtual-memory space of

232 bytes. The computer has 218 bytes of physical memory. The virtual

memory is implemented by paging, and the page size is 4096 bytes.

A user process generates the virtual address 11123456. Explain how

the system establishes the corresponding physical location. Distinguish

between software and hardware operations.

**Answer:**

The virtual address in binary form is

0001 0001 0001 0010 0011 0100 0101 0110

Since the page size is 212 , the page table size is 220 . Therefore the low- order 12 bits “0100 0101 0110” are used as the displacement into the page, while the remaining 20 bits “0001 0001 0001 0010 0011” are used as the displacement in the page table.

**9.19** Assume we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds.

Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

**Answer:**

0.2 sec = (1 − *P*) × 0.1 sec + (0.3*P*) × 8 millisec + (0.7*P*) × 20 millisec

0.1 = −0*.*1*P* + 2400 *P* + 14000 *P*

0.1 16,400 *P*

*P*  0.000006

**9.20** When a page fault occurs, the process requesting the page must block while waiting for the page to be brought from disk into physical memory. Assume that there exists a process with five user-level threads and that the mapping of user threads to kernel threads is many to one. If one user thread incurs a page fault while accessing its stack, will the other user threads belonging to the same process also be affected by the page fault

— that is, will they also have to wait for the faulting page to be brought

into memory? Explain.

**Answer:**

Yes, because there is only one kernel thread for all user threads, that

kernel thread blocks while waiting for the page fault to be resolved.

Since there are no other kernel threads for available user threads, all other user threads in the process are thus affected by the page fault.

**9.21** Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

• LRU replacement

• FIFO replacement

• Optimal replacement

**Answer:**

• 18

• 17

• 13

**9.22** The following page table is for a system with 16-bit virtual and physical addresses and with 4,096-byte pages. The reference bit is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit. A dash for a page frame indicates the page is not in memory. The page-replacement algorithm is localized LRU, and all numbers are provided in decimal.

Page Page Frame Reference Bit

|  |  |  |
| --- | --- | --- |
| 0 | 9 | 0 |
| 1 | 1 | 0 |
| 2 | 14 | 0 |
| 3 | 10 | 0 |
| 4 | – | 0 |
| 5 | 13 | 0 |
| 6 | 8 | 0 |
| 7 | 15 | 0 |
| 8 | – | 0 |
| 9 | 0 | 0 |
| 10 | 5 | 0 |
| 11 | 4 | 0 |
| 12 | – | 0 |
| 13 | – | 0 |
| 14 | 3 | 0 |
| 15 | 2 | 0 |

a. Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses. You may provide answers in either

hexadecimal or decimal. Also set the reference bit for the appro- priate entry in the page table.

• 0xE12C

• 0x3A9D

• 0xA9D9

• 0x7001

• 0xACA1

b. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.

c. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

**Answer:**

• ◦ 0xE12C → 0x312C

◦ 0x3A9D → 0xAA9D

◦ 0xA9D9 → 0x59D9

◦ 0x7001 → 0xF001

◦ 0xACA1 → 0x5CA1

• The only choices are pages 4, 8, 12, and 13. Thus, example addresses include anything that begins with the hexadecimal sequence

0x4..., 0x8..., 0xC..., and 0xD....

• Any page table entries that have a reference bit of zero. This includes the following frames {9*,* 1*,* 14*,* 13*,* 8*,* 0*,* 4}

**9.23** Assume that you are monitoring the rate at which the pointer in the clock algorithm (which indicates the candidate page for replacement) moves. What can you say about the system if you notice the following behavior:

a. pointer is moving fast b. pointer is moving slow **Answer:**

If the pointer is moving fast, then the program is accessing a large

number of pages simultaneously. It is most likely that during the period

between the point at which the bit corresponding to a page is cleared

and it is checked again, the page is accessed again and therefore cannot

be replaced. This results in more scanning of the pages before a victim

page is found. If the pointer is moving slow, then the virtual memory

system is finding candidate pages for replacement extremely efficiently,

indicating that many of the resident pages are not being accessed.

**9.24** Discuss situations in which the LFU page-replacement algorithm gener- ates fewer page faults than the LRU page-replacement algorithm. Also discuss under what circumstances the opposite holds.

**Answer:**

Consider the following sequence of memory accesses in a system that

can hold four pages in memory: 1 1 2 3 4 5 1. When page 5 is accessed,

the least frequently used page-replacement algorithm would replace a

page other than 1, and therefore would not incur a page fault when page

1 is accessed again. On the other hand, for the sequence “12 34 5 2,” the

least recently used algorithm performs better.

**9.25** Discuss situations in which the MFU page-replacement algorithm gen- erates fewer page faults than the LRU page-replacement algorithm. Also discuss under what circumstances the opposite holds.

**Answer:**

Consider the sequence in a system that holds four pages in memory: 1 2

3 4445 1. The most frequently used page replacement algorithm evicts

page 4 while fetching page 5, while the LRU algorithm evicts page 1. This

is unlikely to happen much in practice. For the sequence “1 2 3 4 4 4 5

1,” the LRU algorithm makes the right decision.

**9.26** The VAX/VMS system uses a FIFO replacement algorithm for resident pages and a free-frame pool of recently used pages. Assume that the free-frame pool is managed using the least recently used replacement policy. Answer the following questions:

a. If a page fault occurs and if the page does not exist in the free-frame pool, how is free space generated for the newly requested page?

b. If a page fault occurs and if the page exists in the free-frame pool, how is the resident page set and the free-frame pool managed to make space for the requested page?

c. What does the system degenerate to if the number of resident pages is set to one?

d. What does the system degenerate to if the number of pages in the free-frame pool is zero?

**Answer:**

a. When a page fault occurs and if the page does not exist in the free-frame pool, then one of the pages in the free-frame pool is evicted to disk, creating space for one of the resident pages to be moved to the free-frame pool. The accessed page is then moved to the resident set.

b. When a page fault occurs and if the page exists in the free-frame pool, then it is moved into the set of resident pages, while one of the resident pages is moved to the free-frame pool.

c. When the number of resident pages is set to one, then the system degenerates into the page replacement algorithm used in the free-frame pool, which is typically managed in a LRU fashion.

d. When the number of pages in the free-frame pool is zero, then the system degenerates into a FIFO page-replacement algorithm.

**9.27** Consider a demand-paging system with the following time-measured utilizations:

|  |  |
| --- | --- |
| CPU utilization | 20% |
| Paging disk | 97.7% |
| Other I/O devices | 5% |

For each of the following, say whether it will (or is likely to) improve

CPU utilization. Explain your answers.

a. Install a faster CPU.

b. Install a bigger paging disk.

c. Increase the degree of multiprogramming. d. Decrease the degree of multiprogramming. e. Install more main memory.

f. Install a faster hard disk or multiple controllers with multiple hard disks.

g. Add prepaging to the page fetch algorithms. h. Increase the page size.

**Answer:**

The system obviously is spending most of its time paging, indicating

over-allocation of memory. If the level of multiprogramming is reduced

resident processes would page fault less frequently and the CPU utiliza-

tion would improve. Another way to improve performance would be to

get more physical memory or a faster paging drum.

a. Install a faster CPU —No.

b. Install a bigger paging disk — No.

c. Increase the degree of multiprogramming — No. d. Decrease the degree of multiprogramming — Yes.

e. Install more main memory — Likely to improve CPU utilization as more pages can remain resident and not require paging to or from the disks.

f. Install a faster hard disk or multiple controllers with multiple hard disks — Also an improvement, for as the disk bottleneck is removed by faster response and more throughput to the disks, the CPU will get more data more quickly.

g. Add prepaging to the page fetch algorithms — Again, the CPU will get more data faster, so it will be more in use. This is only the case if the paging action is amenable to prefetching (i.e., some of the access is sequential).

h. Increase the page size — Increasing the page size will result in fewer page faults if data is being accessed sequentially. If data access is

more or less random, more paging action could ensue because fewer pages can be kept in memory and more data is transferred per page fault. So this change is as likely to decrease utilization as it is to increase it.

**9.28** Suppose that a machine provides instructions that can access memory locations using the one-level indirect addressing scheme. What is the sequence of page faults incurred when all of the pages of a program are currently non resident and the first instruction of the program is an indirect memory load operation? What happens when the operating system is using a per-process frame allocation technique and only two pages are allocated to this process?

**Answer:**

The following page faults take place: page fault to access the instruction,

a page fault to access the memory location that contains a pointer to

the target memory location, and a page fault when the target memory

location is accessed. The operating system will generate three page

faults with the third page replacing the page containing the instruction.

If the instruction needs to be fetched again to repeat the trapped

instruction, then the sequence of page faults will continue indefinitely.

If the instruction is cached in a register, then it will be able to execute

completely after the third page fault.

**9.29** Suppose that your replacement policy (in a paged system) is to examine each page regularly and to discard that page if it has not been used since the last examination. What would you gain and what would you lose by using this policy rather than LRU or second-chance replacement? **Answer:**

Such an algorithm could be implemented with the use of a reference bit. After every examination, the bit is set to zero; set back to one if the page is referenced. The algorithm would then select an arbitrary page for replacement from the set of unused pages since the last examination.

The advantage of this algorithm is its simplicity — nothing other than a reference bit need be maintained. The disadvantage of this algorithm is that it ignores locality by using only a short time frame for determining whether to evict a page or not. For example, a page may be part of the working set of a process, but may be evicted because it was not referenced since the last examination (that is, not all pages in the working set may be referenced between examinations).

**9.30** A page-replacement algorithm should minimize the number of page faults. We can achieve this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages associated with that frame. Then, to replace a page, we can search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specif- ically address these problems:

i. What the initial value of the counters is

ii. When counters are increased iii. When counters are decreased

iv. How the page to be replaced is selected

b. How many page faults occur for your algorithm for the following reference string, for four page frames?

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

c. What is the minimum number of page faults for an optimal page- replacement strategy for the reference string in part b with four page frames?

**Answer:**

a. Define a page-replacement algorithm addressing the problems of:

i. Initial value of the counters — 0.

ii. Counters are increased — whenever a new page is associated with that frame.

iii. Counters are decreased — whenever one of the pages associated with that frame is no longer required.

iv. How the page to be replaced is selected — find a frame with the smallest counter. Use FIFO for breaking ties.

b. 14 page faults c. 11 page faults

**9.31** Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory.

Assume that 80 percent of the accesses are in the associative memory

and that, of those remaining, 10 percent (or 2 percent of the total) cause

page faults. What is the effective memory access time?

**Answer:**

effective access time = (0.8) × (1 sec)

+ (0.1) × (2 sec) + (0.1) × (5002 sec)

= 501.2 sec

= 0.5 millisec

**9.32** What is the cause of thrashing? How does the system detect thrashing?

Once it detects thrashing, what can the system do to eliminate this

problem?

**Answer:**

Thrashing is caused by underallocation of the minimum number of

pages required by a process, forcing it to continuously page fault. The

system can detect thrashing by evaluating the level of CPU utilization

as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

**9.33** Is it possible for a process to have two working sets, one representing data and another representing code? Explain.

**Answer:**

Yes, in fact many processors provide two TLBs for this very reason.

As an example, the code being accessed by a process may retain the

same working set for a long period of time. However, the data the code

accesses may change, thus reflecting a change in the working set for data

accesses.

**9.34** Consider the parameter used to define the working-set window in the working-set model. What is the effect of setting to a small value on the page fault frequency and the number of active (non-suspended) processes currently executing in the system? What is the effect when is set to a very high value?

**Answer:**

When is set to a small value, then the set of resident pages for a

process might be underestimated, allowing a process to be scheduled

even though all of its required pages are not resident. This could result

in a large number of page faults. When is set to a large value, then

a process’s resident set is overestimated and this might prevent many

processes from being scheduled even though their required pages are

resident. However, once a process is scheduled, it is unlikely to generate

page faults since its resident set has been overestimated.

**9.35** Assume there is an initial 1024 KB segment where memory is allocated using the Buddy system. Using Figure Figure 9.26 as a guide, draw the tree illustrating how the following memory requests are allocated:

• request 240 bytes

• request 120 bytes

• request 60 bytes

• request 130 bytes

Next, modify the tree for the following releases of memory. Perform coalescing whenever possible:

• release 240 bytes

• release 60 bytes

• release 120 bytes

**Answer:**

The following allocation is made by the Buddy system: The 240-byte

request is assigned a 256-byte segment. The 120-byte request is assigned a

128-byte segement, the 60-byte request is assigned a 64-byte segment and

the 130-byte request is assigned a 256-byte segment. After the allocation,

the following segment sizes are available: 64-bytes, 256-bytes, 1K, 2K,

4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

After the releases of memory, the only segment in use would be a

256-byte segment containing 130 bytes of data. The following segments

will be free: 256 bytes, 512 bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K,

256K, and 512K.

**9.36** A system provides support for user-level and kernel-level threads. The mapping in this system is one to one (there is a corresponding kernel thread for each user thread). Does a multithreaded process consist of (a) a working set for the entire process or (b) a working set for each thread? Explain.

**Answer:**

A working set for each thread. This is because each kernel thread has

its own execution sequence, thus generating its unique sequence of

addresses.

**9.37** The slab allocation algorithm uses a separate cache for each different object type. Assuming there is one cache per object type, explain why this doesn’t scale well with multiple CPUs. What could be done to address this scalability issue?

**Answer:**

This has long been a problem with the slab allocator — poor scalability

with multiple CPUs. The issue comes from having to lock the global cache

when it is being access. This has the effect of serializing cache accesses

on multiprocessor systems. Solaris has addressed this by introducing a

per-CPU cache, rather than a single global cache.

**9.38** Consider a system that allocates pages of different sizes to its processes.

What are the advantages of such a paging scheme? What modifications

to the virtual memory system provide this functionality?

**Answer:**

The program could havea large code segment or use large-sized arrays

as data. These portions of the program could be allocated to larger pages,

thereby decreasing the memory overheads associated with a page table.

The virtual memory system would then have to maintain multiple free

lists of pages for the different sizes and also needs to have more complex

1. code for address translation to take into account different