Concordia University
Computer Science & Software Engineering
Faculty of Engineering and Computer Science

COMP 346 – Fall 2018
Theory Assignment 3

Tianlin Yang

40010303

Question # 1

In a 64-bit computer system that uses pure paging with 16KB page size, if each page table entry is 4 bytes long then:

i. What is the maximum size of a page table?
ii. What is the maximum size of user-space main memory that can be supported by this scheme?
iii. If hierarchical paging is used then what is the total level of hierarchies required? Assume that 2-level of hierarchy corresponds to an outer page table and an inner page table. Show all relevant calculations.
iv. If the inverted paging scheme is used instead of paging, with a 16KB page/frame size, then what is the maximum size of the inverted page table considering the same memory size calculated in (ii) above? Assume that each entry of the inverted page table is 4 bytes long.

Answer:

(i) In 64 bits system, the page size is 16KB = $2^{10} + 2^4 = 2^{14} bytes$, the maximum number of entries = $\frac{2^{64}}{2^{14}} = 2^{50}$. Each page table entry is 4 bytes long, so max size of page table is $2^{50} * 4\ bytes = 2^{52} bytes$.

(ii) 4 bytes = 32 bits page table entry, total number of frames can be addressed by 4 bytes which is 2^{32}. Total main memory size can be addressed is $2^{32} * 16KB = 64000GB = 64\ TB$.

(iii) The number of entries in the outer page table = $\frac{2^{52}}{2^{14}} = 2^{38}$ and size is $2^{38} * 4\ bytes = 2^{40}\ bytes = 1000\ GB = 1\ TB$. Page level size = 64 – 38 – 14 = 12 bits. Therefore, 64bits can be divided by : 14 bits offset+12 bits+12 bits +12 bits+12 bits+12 bits+2 bits.
Level should be 5.

(iv) Total number of entries in the inverted page table = total number of frames = 2^{32}
Max size of inverted page table = $2^{38} * 4\ bytes = 16GB$

Question # 2

Answer the following questions:

i. (a) What are relocatable programs? (b) What makes a program relocatable? (c) From the OS memory management context, why programs (processes) need to be relocatable?
ii. What is (are) the advantage(s) and/or disadvantage(s) of small versus big page sizes?
iii. What is (are) the advantage(s) of paging over segmentation?
iv. What is (are) the advantage(s) of segmentation over paging?

Explain your answers.

Answer:

(i) (a)Relocatable programs are the ones that are not bound to specific memory location(s), i.e., they do not use "absolute" memory addresses at compile time. So these programs can be loaded to any location in memory.
(b) In a relocatable program, program address binding is not done at compile time, but rather it is done at load time, i.e., dynamic binding. Dynamic address binding is possible if the program uses only "relative" or "relocatable" memory addressing, e.g., 100 bytes from the start of a module.
(c) From the OS memory management context, (user) programs have to be relocatable because they are loaded dynamically to any arbitrary memory locations. Moreover, programs may be swapped out and then swapped in to a different location; without being relocatable this will not be possible.

(ii) Large page sizes will result in large I/O transfers; it is often more efficient to perform I/O in larger chunks (many DMA interrupts versus single interrupt). With larger page size, the page table size will also be smaller because less number of pages are required. However, the amount of internal fragmentation will be more with larger page sizes. From the page fault perspective, it cannot be positively ascertained which one will be better; it could be either way, e.g., large page size could result in fewer page faults, on the contrary large page size could result in more swapping overhead.

(iii) Paging scheme divides the logical and physical memory to equal sized chunks (pages and frames). Memory management and I/O transfer with equal sized chunks become much more convenient than variable sized segments in the segmentation scheme. Moreover, paging removes the problem of external fragmentation. Though paging can cause some internal fragmentation, due to which about 50% of a page is lost on the average per process, it is not a much of a bigger problem as compared to external fragmentation.

(iv) One main problem with paging is that though it facilitates sharing it is not as convenient: data/code size to be shared must be exact multiples of page size, which is often not the case. Segmentation removes this limitation by allowing variable sized segments that can be shared among processes. Another advantage of segmentation over paging is that segments maintain their logical properties (programmer's view and semantics); however in paging it is lost (whole program is one-byte stream, e.g., code and data are indistinguishable from each other).

Question # 3

Which of the following programming techniques and data structures (in a user-level program) are good for a demand-paged environment, and which are bad? Explain your answer.
i. Sequential search through a linked list
ii. Sequential search through an array
iii. Binary tree search
iv. Hashing with linear probing
v. Queue implemented using a circular array.

Answer:

(i) Not good. The nodes of a linked list can be located non-contiguously in the process address space. It is highly possible that two adjacent nodes reside in different pages. In demand-paged environment, pages always are swapped into memory on demand. So, accessing subsequent nodes can lead to frequent page faults, if the nodes reside on different pages.

(ii) Good. An array is a contiguous memory block. The adjacent entries of the array are more likely to reside on the same page and hence will lead to fewer page faults.

(iii). Not good. Suppose a binary tree is implemented by using linked nodes, then searching through it will lead to the same situation as in part (i) above. If the tree is implemented using an array, then searching through it will require accessing non-contiguous entries of the array. If the array is large and scattered over multiple pages, then accessing non-contiguous entries of the array on a binary tree search could lead to more page faults.

(iv). Good. Hashing is based on an array-based implementation and accessing this array is sequential in the case of linear probing.

(v). Good. Successive enqueue or dequeue operations will lead to contiguous accesses to the array (except at the two boundaries of the array) and hence will lead to fewer page faults.

Question # 4

Consider a demand-paged system where the page table for each process resides in main memory. In addition, there is a fast-associative memory (also known as TLB which stands for Translation Look-aside Buffer) to speed up the translation process. Each single memory access takes 1 microsecond while each TLB access takes 0.2 microseconds. Assume that 2% of the page requests lead to page faults, while 98% are hits. On the average, page fault time is 20 milliseconds (includes everything: TLB/memory/disc access time and transfer, and any context switch overhead). Out of the 98% page hits, 80 % of the accesses are found in the TLB and the rest, 20%, are TLB misses. Calculate the effective memory access time for the system.

Answer:

Effective memory access time = page fault rate * page fault service time + Page hit rate * effective memory access time in page hit (1)

In the above, Page fault rate = 0.02
Page fault service time = 20 milliseconds = 20000 microseconds
Page hit rate = 0.98

Effective memory access time in page hit = TLB hit rate * memory access time in TLB hit + TLB miss rate * memory access time in TLB miss
= 0.8 * (TLB access time + memory access time) + 0.2 * (TLB access time + page table access time +
Memory access time)
= 0.8 * (0.2 + 1) + 0.2 * (0.2 + 1 + 1) microseconds
= 1.4 microseconds

Substituting in equation (1) above:
Effective memory access time = 0.02 * 20000 + 0.98 * 1.4 microseconds = 401.372 microseconds

Question # 5

Consider the two-dimensional integer array A:

```
int A[][] = new int[100][100];
```

Assume that page size of the system = 200 bytes, and each integer occupies 1 byte. Also assume that A[0][0] is located at location 200 (i.e., page 1) in the paged memory system, and the array is stored in memory in row-major order (i.e., row by row). A small process that manipulates the array resides at page 0 (locations 0 to 199). Thus every instruction fetch is from page 0. The process is assigned 3 memory frames: frame 0 is already loaded with page 0 (i.e., process code), and the other two frames are initially empty.

a) Consider the following code fragment for initializing the array:

```
for (int i = 0; i < 100; i++)
        for (int j = 0; j < 100; j++)
            A[i][j] = 0;
```

If the LRU (Least Recently Used) page replacement scheme is used then what is the total number of page faults in executing the previous initialization code?

b) Now, instead, consider the following code fragment for initializing the array:

```
for (int j = 0; j < 100; j++)
        for (int i = 0; i < 100; i++)
            A[i][j] = 0;
```

If the LRU page replacement scheme is used then what is the total number of page faults in executing the above initialization code?

Answer:

a) Since the array is stored in row-major order, the first two rows reside on page 1, the next two rows reside on page 2, and so on. Since there are altogether 100 rows, so 50 pages are required to store the array. When the array is initialized row-by-row, the following is going to be the page reference string in initializing the array:

1 1 1 …(200 times) 2 2 2 …(200 times) …….50 50 50 (200 times)

Reference to every new page will result in a page fault, and hence total number of page faults = 50.

b) When the array is initialized column-by-column, every alternate access is going to reside on a new page and will result in a page fault. The following is the page reference string in traversing one column:1 1 2 2 3 3…..50 50 and there are 50 page faults in the above. There are 100 columns altogether and hence the total number of page faults = 50 * 100 = 5000.

Question #6

Discuss situations in which the least frequently used (LFU) page replacement algorithm can generate fewer page faults than the least recently used (LRU) page replacement algorithm. Also discuss under what situations the opposite holds.

Answer:

In the LFU page replacement algorithm, each page has a counter which is incremented on every access. The page with the lowest counter is replaced each time, i.e., the most frequently used pages are going to stay. This is particularly useful when a page is accessed heavily, but only at periodic intervals. LRU would have replaced the page if it was not accessed in the recent past, however LFU would keep the page due to its high frequency counter. On the contrary, suppose a process heavily calls a function (residing on a page) at the start-up phase and never calls the function in the rest of its execution. LFU would have kept the page in memory, which is not desired. In this situation, LRU could have done better than LFU.

Reference

1. Abraham S, Peter B.G, Gear G, Operating System Concepts 10th , John Wiley & Sons, Inc. ISBN 978-1-118-06333-0, Page 3-4.
2. Abraham S, Peter B.G, Gear G, Operating System Concepts 10th , John Wiley & Sons, Inc. ISBN 978-1-118-06333-0, Page 69. And Wikipedia: https://en.wikipedia.org/wiki/Batch_processing
3. https://www.quora.com/What-is-time-sharing-operating-system-with-example
4. https://en.wikipedia.org/wiki/Real-time_operating_system
5. https://en.wikipedia.org/wiki/Virtualization
6. https://en.wikipedia.org/wiki/Direct_memory_access
7. https://en.wikipedia.org/wiki/Daisy_chain_(electrical_engineering)
8. Abraham S, Peter B.G, Gear G, Operating System Concepts 10th , John Wiley & Sons, Inc. ISBN 978-1-118-06333-0, Page 200.
9. https://www.cs.ubc.ca/~tmm/courses/213-12F/slides/213-2b-4x4.pdf
10. http://bnrg.cs.berkeley.edu/~adj/cs16x/exams/fa05mt1-solutions.pdf
11. https://blog.codinghorror.com/understanding-user-and-kernel-mode/
12. https://www.webopedia.com/DidYouKnow/Internet/virus.asp
13. https://social.microsoft.com/Forums/en-US/fc0cbd0a-4355-4338-ace8-9ce933d64a1d/multiprogrammed-system?forum=whatforum
14. https://cgi.cse.unsw.edu.au/~cs3231/07s1/example_exam_answers_updated.html
15. http://www.padakuu.com/article/181-distributed-system and https://www.quora.com/What-is-the-difference-between-Network-OS-and-Distributed-OS