MEMORY MANAGEMENT WORKED EXAMPLES

This is a closed-book examination. You are not allowed to use any notes or other references during the exam. You will have 50 minutes to answer all the questions. Answer the true/false and short-answer questions on this paper. For other questions, write as much of your answer as will fit in the space provided below the question. If your answer does not fit entirely in the space below the question, use the blank paper provided to you to complete your answer. The exam will be graded on a scale of 100, but it is possible to achieve a score of up to 120. The first 10 numbered divisions of the exam are each worth 10 points. The last question requires you to explain your solution to Programming Assignment #3, and is worth 20 points. You will probably find some questions are easier for you than others. Therefore, it is important to budget your time. Do not miss the chance to answer some easier questions by puzzling too long over a question that is giving you difficulty. Remember to write your name on every page, and turn in all pages of the exam.

1. Answer each question by marking a "T" or "F" next to the statement, inside the parentheses on the left.
    a. (F)External fragmentation is a problem with paged memory systems.

        There can be no external fragmentation, because the memory is allocated in equal-sized blocks.

    b. (T)Internal fragmentation is a problem with paged memory systems.
    c. (F)Memory compaction is employed to counteract internal fragmentation.

        Compaction is only used to counteract external fragmentation.

    d. (T)Memory compaction is employed to counteract external fragmentation.
    e. (F)Dynamic relocation requires memory compaction.

        Paging systems do dynamic relocation.without any need for compaction.

    f. (T)Memory compaction requires dynamic relocation.
    g. (F)First-Fit is a replacement algorithm.

        It is a placement algorithm.

    h. (F)Best-Fit generally causes less fragmentation than First-Fit.

        Best-Fit is well known to be suffer badly from fragmentation, because it makes the fragment sizes very small.

i. (T)A function of a linker is to combine several object modules into a single load module.
j. (F)A function of a linker is to replace absolute references in an object module by symbolic references to locations other modules.

The reverse is true.

2. Answer each question by marking a "T" or "F" next to the statement, inside the parentheses on the left.
   a. (F)Demand paging is placement policy.

   It is a *fetch* policy.

   b. (F)Hashing is a fetch policy.

   Hashing is a table-lookup technique, and also an information hiding technique.

   c. (F)The last process activated is most likely to have its entire working set resident.

   On the contrary, when a process is first activated it will encounter a cluster of page faults until its full working set has become resident.

   d. (T)The process that caused the most recent page fault is a good choice to suspend (swap the entire process out) when suspension is called for.
   e. (F)The 50% criterion is a rule used to decide which page to replace.

   This rule is applied for *load control* to decide whether we can admit another process, and whether we need need to suspend a process.

   f. (T)Fixed allocation with global replacement is not a workable combination for a resident set management scheme.

   Global replacement implies a process can take a page frame from another process. That would increase the allocation of the process, so it must be a variable allocation scheme.

   g. (T)The term TLB stands for Translation Lookaside Buffer.
   h. (T)The TLB is a cache for page table entries.
   i. (T)Without paging, a segmented memory system suffers external fragmentation.
   j. (T)Memory segmentation can be used for protection, allowing different privileges to a process for different memory segments.

3. Explain the differences between the meanings of the terms logical address, relative address, and physical address.

   logical
   reference to a memory location that is independent of the current location of the process code and data in memory; translation must be made to the current physical address
   relative
   a special case of logical address, expressed as a location relative to some known "zero" point, such as the start of the memory occupied by the process
   physical
   the absolute address or actual location in main memory

4. Consider the Buddy System applied to a range of memory starting with address 0.
   a. If a block of size 16 has address 0000111100100000 what is the address of its buddy?

      0000111100110000

   b. In general, how is the address of the buddy of the block of size 2^k whose address is *x*. computed?

      The simplest way to say this would be "Flip (or toggle, or invert, or complement) the *(k+1)*st bit of *x*".

      The answer is *not* "x + 2^k". That would get the right value in the *(k+1)*st bit, but if the bit was previously 1, there would be a carry to the *(k+2)*nd bit, which is incorrect.

   c. Cite an example of an application of the Buddy System in an operating system.

      The example given in your text is the allocation of kernel memory in the Unix family (including Linux) of operating systems. The key here is that the Buddy system allows us to allocated*contiguous* ranges of addresses of *different sizes*.The kernel needs to allocate memory in blocks of smaller than a full page in size, so it needs such a scheme. It cannot rely on the page allocation mechanism alone.

5. What is thrashing? How might on OS detect that thrashing is occurring? How would it show up in the page fault frequency? How would it show up in the average time between page faults? What can an operating system do to deal with thrashing when it is detected? (Use separate paper.)

Thrashing is a condition in which the system is getting very little useful work done because of an exessive rate of page faults. That is, most of the time the CPU is idle because all the processes are waiting for pages to be swapped into main memory. The system might detect this by looking at the paging disk activity and the CPU activity. If the paging disk is busy more than 50% of the time, that would be a sign that we have a thrashing problem. If the CPU is idle much of the time, but there are several processes waiting for page fault service that would be another indication. We can also use the page fault frequency or the average time between page faults. With thrashing, the page fault frequency will be higher than normal, and the average time between page faults will be less than the time it takes to serve a page fault. When thrashing is detected, the system must reduce the level of multiprogramming, by swapping out one or more processes, and then allow the resident sets of the resident processes to grow.

6. Define and contrast the terms *resident set* and *working set*.

   Resident set
   The set of pages (of a process) that are resident in main memory.
   Working set
   Intuitively, this is the set of pages that a process needs to have in main memory to run without page faults, for some time interval. The formal definition is the set of pages that a process references during the time window *(t-D, t)*, for some time *t* and some window size *D*.

   The process will have no page faults so long as the working set is the same as the resident set.

7. Consider a paged virtual memory system. Suppose the page table for the process currently executing on the processor looks like the following. All numbers are decimal, everything is numbered starting from zero, and all addreses are memory byte addresses. The page size is 1024 bytes.

Virtual page number	**Valid bit**	**Reference bit**	**Modify bit**	**Page frame number**
0	1	1	0	3
1	0	0	0	-
2	1	0	1	7
3	1	1	1	0
4	1	1	0	2
5	0	0	0	-

What physical address, if any, would each of the following virtual addresses correspond to? (If there would be a page fault, just indicate that one would take place. Do not try to handle it.)

a. 1052

1052 = 1024 + 28, so the page number is 1 and the offset is 28. The page table shows that page 1 is not currently resident, so there would be a page fault.

b. 2221 2221 = 2 * 1024 + 173 , so the page number is 2 and the offset is 173. The page table shows that page 2 is resident in frame 7, so the physical address would be 7 * 1024 + 173 = 7341.

8. A process has four page frames allocated to it. (All the following numbers are decimal, and everything is numbered starting from zero). The time of the last loading of a page into each page frame, the time of last access to the page in each page frame, the virtual page number in each page frame, and the referenced (R) and modified (M) bits for each page frame are as shown (the times are in clock ticks from the process start at time zero to the event -- not the number of ticks since the event to the present).

Virtual page number	Page frame	Time loaded	Time referenced	R bit	M bit
2	0	60	164	1	1
1	1	30	166	1	0
0	2	150	162	0	1
3	3	20	163	1	1

A page fault to virtual page 4 has occurred. Which page frame will have its contents replaced for each of the following memory management policies? Explain why in each case.

a. FIFO (first-in-first-out)

3. The first frame loaded was frame 3, loaded at time 20.

b. LRU (least recently used)

2. The least recently referenced frame is frame 0, referenced at time 162. (Note that the question asked for the page frame, not the virtual page number. For part (a), the page frame number and virtual page number happened to be the same, but for this part we have virtual page number 0 in page frame 2.)

9. Continue the previous question.
    a. Clock. (Suppose the order of the frames in the circular buffer is the same as the order of the page frame numbers.)

        2. The only frame with a zero Referenced bit is frame number 2. The algorithm would run through frames 3, 0, and 1, setting their Referenenced bits to zero, before detecting the zero Referenced bit on frame 2.

    b. Optimal (Use the reference string: 4, 0, 0, 0, 2, 4, 2, 1, 0, 3, 2.)

        3. Pages 4, 0, 2, and 1 are referenced before page 3, so the optimal choice is to put page 4 into the frame of virtual page 3. (Note that it happens that page 3 is in frame 3, so the answer is three, even though the question asks for the frame number.)

10. Assuming a page size of 2K bytes and that a page table entry takes 4 bytes, how many levels of page tables would be required to map a 32-bit address space, if the top level page table fits into a single page? Explain why.

    Each page is of size $2048 = 2^{11}$, so 11 bits are used for the offset. This leaves 21 bits to specify the page, so we have 2^{21} pages. Each page holds $2048/4 = 2^{11}/2^2 = 2^9$ entries, so the top level can address 2^9 pages. Going to two levels allows us to address only $2^9*2^9 = 2^{18}$ pages. We need to go to three levels, which allows us to address $2^9*2^9*2^9 = 2^{27} > 2^{21}$ pages.

11. **ASAS**

Q1) (Tannenbaum Ch3 Q12)

A computer with a 32-bit address uses a two level page table. Virtual addresses are split into a 9-bit top-level page table field, an 11-bit second-level page table field, and an offset. How large are the pages and how many are there in the virtual address space ?

Ans)

Twenty bits are used for the virtual page numbers, leaving 12 over for the offset. This yields a 4K page. Twenty bits for the virtual page implies 2^20 pages.

Q2) (Tannenbaum Ch3 Q18)

A machine has 48-bit virtual addresses and 32-bit physical addresses. Pages are 8K.

How many entries are needed for a conventional page table ? For an inverted page table ?

Ans)

With 8K pages and a 48-bit virtual address space, the number of virtual pages is (2^48)/(2^13), which is 2^35 (about 34 billion). An inverted page table needs as many entries as there are page frames in memory, in this case, 524,288. Clearly, this is a lot more manageable.

Q3) (Tannenbaum Ch3 Q29)

Explain the difference between internal fragmentation and external fragmentation. Which one occurs in paging systems? Which one occurs in systems using pure segmentation ?

Ans)

Internal fragmentation occurs when the last allocation unit is not full. External fragmentation occurs when space is wasted between two allocation units. In a paging system, the wasted space in the last page is lost to internal fragmentation. In a pure segmentation system, some space is invariably lost between the segments. This is due to external fragmentation.

Q4) (Silberschatz 10.6, p.367)

Ans)

If arbitrarily long names can be used then it is possible to simulate a multilevel directory structure. This can be done, for example, by using the character "." to indicate the end of a subdirectory. Thus, for example, the name *jim.pascal.F1* specifies that *F1* is a file in subdirectory *pascal* which in turn is in the root directory *jim*.
If file names were limited to seven characters, then the above scheme could not be utilized and thus, in general, the answer is no. The next best approach in this situation would be to use a specific file as a symbol table (directory) to map arbitrarily long names (such as *jim.pascal.F1*) into shorter arbitrary names (such as *XX00743*), which are then used for actual file access.

Q5) (Silberschatz 11.2, p. 392)

Ans)

a. In order to reconstruct the free list, it would be necessary to perform "garbage collection". This would entail searching the entire directory structure to determine which pages are already allocated to jobs. Those remaining unallocated pages could be relinked as the free-space list.
b. The free-space list pointer could be stored on the disk, perhaps in several places.

12. ASAS
13. A virtual memory system has a page size of 1024 words, eight virtual pages, and four physical page frames. The page table is as follows:

Virtual page Number	Page Frame Number
0	1
1	0
2	3
3	-
4	-
5	2
6	0
7	-

a. What is the size of the virtual address space? (How many bits in a virtual address?)

```
13 bits
```

b. What is the size of the physical address space? (How many bits in a physical address?)

```
12 bits
```

c. What are the physical addresses corresponding to the following decimal virtual addresses (yes, you have to convert from decimal to binary): 0, 3728, 1023, 1024, 1025, 7800, 4096?

```
0       000 00 0000 0000        01 00 0000 0000
3728    011 10 1001 0000        Page Fault
1023    000 11 1111 1111        01 11 1111 1111
1024    001 00 0000 0000        00 00 0000 0000
1025    001 00 0000 0001        00 00 0000 0001
7800    111 10 0111 1000        Page Fault
4096    100 00 0000 0000        Page Fault
```

14. ASA
15. Give reasons why the page size in a virtual memory system should be neither too large or too small.

```
As the page size grows, more and more bits are used in the offset field,
meaning the size of a page table can shrink.  Also, there are cache size
ramifications.  However, as the page size grows, there is more and more
fragmentation (since there are going to be more and more pages brought in
that
are not fully utilized).  It also takes a longer amount of time to bring in a
very large page, which can be bad if you are not using much data on that
page.
```

16. Assume a task is divided into 4 equal-sized segments, and that the system builds an 8-entry page descriptor table for each segment. Thus, the system has a combination of segmentation and paging. Assume also that the page size is 2K bytes.

    a. What is the maximum size of each segment?

    ```
    16K
    ```

    b. What is the maximum logical address space for the task?

    ```
    64K
    ```

    c. Assume that an element in physical location 0x1ABC is accessed by this task. What is the format of the logical address that the task generates for it?

    ```
    0x1ABC = 0001 1010 1011 1100 =  00      011    010 1011 1100
                                  Seg #   Page #      Offset
    ```

17. Consider a paged logical address space (composed of 32 pages of 4K bytes each) mapped into a 1-MByte physical memory space.

    a. What is the format of the processor's logical address?

    ```
    5 bit Page number field, followed by 12 bit Offset field.
    ```

    b. What is the length and width of the page table (ignoring any access control bits)?

Length is 32 entries, width is 8 bits

c. What is the effect on the page table if the physical memory space is reduced by half?

Width of page table entries becomes 7 bits.

18. Why are the page size, the number of pages in the virtual address space, and the number of page frames in the physical address space all a power of 2 in binary machines?

SOL

In binary machines, you are dealing with bits. Each bit can have one of two values. It is either set or unset. It is either zero or one. At the hardware level, this may be represented as the presence of voltage or the absence of voltage. The voltage is on or off. If we have one bit, we have 2^1 or 2 possible values that can be represented. If we have two bits, we have 2^2 or 4 possible values that can be represented. If we have three bits, we have 2^3 or 8 possible values. And this continues as we add bits.

As mentioned above, at the hardware level we are dealing with bits that are represented by the presence or absence of voltage. By keeping the page size, the number of pages in the virtual address space and the number of page frames in the physical address space all a power of 2, we can use the entire address space without having gaps or wasted address space. For example, a page size of 1K is 1024 bytes (bytes 0 to 1023). Log_2 (1024) = 10 bits. If we make the page size 1000 bytes (bytes 0 to 999), Log_2 (1000) = 9.96578 bits. However, you cannot have a partial bit; therefore, you need 10 bits to represent a page size of 1000 bytes. Note that because this is binary, we still have the capability of representing 1024 bytes even if the page size is only 1000 bytes. In this case, the bit sequences that represent bytes 1000 to 1023 are invalid. This is wasted address space. The binary machine has the capability of referencing them, but because we only go to 1000 bytes instead of 1024 bytes it creates gaps in the address space that point to invalid byte numbers.

Therefore, because of the nature of binary machines, it makes the most sense to have the page size and the different address spaces be a power of two.

(YH) It is also more efficient to map the virtual address to the physical address.

19. Suppose the page size in a computing environment is 1KB. What is the page number and the offset for the following:

.a. 899 (a decimal number)

page # = 899 / 1024 = page 0

offset = 899 mod 1024 = offset 899

.b. 23456 (a decimal number)

page # = 23456 / 1024 = page 22

offset = 23456 mod 1024 = offset 928

.c. 0x3F244 (a hexadecimal number)

1024 decimal = 400 hexadecimal

page # = 0x3F244 / 0x400 = page 0xFC

offset = 0x3F244 mod 0x400 = offset 0x244

.d. 0x0017C (a hexadecimal number)

1024 decimal = 400 hexadecimal

page # = 0x0017C / 0x400 = page 0

offset = 0x0017C mod 0x400 = offset 0x17C

20. Contemporary computers often have more than 100MB of physical memory. Suppose the page size is 2KB. How many entries would an associative memory need in order to implement a page table for the memory?

100 x 1024 x 1024 / (2 x 1024) = 51200 pages = 50 K pages

We need 50 K entries in the table (1 for each page).

21. What factors could influence the size of the virtual address space in a modern computer system? In your answer consider the memory mapping unit, compiler technology, and instruction format.

SOL

To answer this question, we must consider the following information. A program address is relative to the program and start at address 0. The CPU relocation register contains the starting virtual address of where the program is located in virtual memory. The program address is added to the relocation register contents to get the virtual address of the instruction being executed. This is sent to the Memory Management Unit (MMU). The MMU divides the virtual address up into two parts. The higher bits represent the memory page. The lower bits represent the offset on the page. The higher bits of the virtual address are looked up in the page translation table in the MMU to determine the physical memory page. The offset is then added to the physical memory address to get the physical memory address of the instruction being executed.

Having given the above short analysis of what is taking place; I believe the most important factor in determining the virtual address space of a modern computer system is the hardware architecture of the computer. This includes considering the size in bits of the CPU registers, the communication channel between the CPU and the MMU, the CPU instruction addresses, etc. The hardware architecture determines the maximum address that the computer is capable of handling. For example, if the computer is a 32 bit computer it can handle a virtual address space of 2^32 (or 4GB). If the computer architecture is 64 bits, it can handle a virtual address space of 2^64 (or 17,179,869,184 GB). The MMU then maps the virtual address to the physical address.

As far as compiler technology is concerned, it will assume that the program addresses start at zero and that programs have the entire virtual address space available to them. I believe the compiler would mainly be concerned about not hitting the upper limit of the address space.

(YH) Instruction format works with instruction registers. Instructions such as load and save may have different format for different virtual address space.

Other factors that determine the virtual address space include cost and complexity of the architecture.

22. What factors could influence the size of the physical address space in a modern computer system (consider various parts of the hardware).

SOL

The computer architecture determines the maximum physical address space. For example, a 32 bit architecture can have up to 2^32 bytes (or 4GB) of physical address space. This is assuming the MMU is capable of handling a physical address space equal to the virtual address space. (Note that when the physical address space is equal to the virtual address space, there is no need for the virtual address space to be different than the physical address space.)

The computer chip technology determines how many bits a single chip can hold. The physical size of the memory card determines the number of chips a single card can hold. The number of memory slots available determines the number of memory cards the computer can hold. All of these things influence the size of the physical address space in a modern computer system.

Other considerations include cost and complexity.

23. Suppose w = 2 3 4 3 2 4 3 2 4 5 6 7 5 6 7 4 5 6 7 2 1 is a page reference stream.

.b. Given a page frame allocation of 3 and assuming the primary memory is initially unloaded, how many page faults will the given reference stream incur under LRU (leased recently used)?

Frame	2	3	4	3	2	4	3	2	4	5	6	7	5	6	7	4	5	6	7	2	1
0	*2	2	2	2	2	2	2	2	2	2	*6	6	6	6	6	6	*5	5	5	*2	2
1		*3	3	3	3	3	3	3	3	*5	5	5	5	5	5	*4	4	4	*7	7	7
2			*4	4	4	4	4	4	4	4	4	*7	7	7	7	7	7	*6	6	6	*1

This will have 12 page faults.

.c. Given a page frame allocation of 3 and assuming the primary memory is initially unloaded, how many page faults will the given reference stream incur under FIFO (first-in-first-out)?

Frame	2	3	4	3	2	4	3	2	4	5	6	7	5	6	7	4	5	6	7	2	1
0	*2	2	2	2	2	2	2	2	2	*5	5	5	5	5	5	*4	4	4	*7	7	7
1		*3	3	3	3	3	3	3	3	3	*6	6	6	6	6	6	*5	5	5	*2	2
2			*4	4	4	4	4	4	4	4	4	*7	7	7	7	7	7	*6	6	6	*1

This will have 12 page faults.

24. In a paging system, page boundaries are transparent to the programmer. Explain how a loop might cause thrashing in a static allocation paging system when the memory allocation is too small.

SOL

A static paging algorithm allocates a fixed number of page frames to a process when it is created. It is possible for a loop to contain instructions that require the program to get a new page of instructions multiple times in a loop. For example, there may be function calls in the loop that are located in different memory pages. To execute these functions, the required text pages must be moved into primary memory. If the memory allocation is too small, it is conceivable that each function call could require moving a different page into primary memory from virtual memory. As the loop starts again, the page with the first function called could have been paged out. It must be paged in again. This could continue with each iteration of the loop. This is one example of how a loop might result in thrashing if the memory allocation is too small.

25. **(counts double = 20 pts)** Explain in words how you implemented deadlock detection for Programming Assignment #3. Your answer should cover at least: what is the deadlock criterion you check for (*i.e.* how do you tell if there is a deadlock?); what components (if any) you added to the *pthread_t* and *pthread_mutex_t* structs; in what function(s) those components are initialized; in what function(s) they are updated; how an avoided deadlock is reported back to the application program.

A person could have earned most of the points on this question without having actually completed the assignment, if they had read and understood the statement of the assignment, which says:

1. Arrange to record which mutex a thread is waiting for, when it is waiting for a mutex. This will involve adding fields to the thread and/or mutex structures, and adding code to initialize and update these new fields.
2. When a thread attempts to lock a mutex, chain forward, using the owner field of locked mutexes and the information you have recorded about which mutex each thread is waiting for, until you either find the end of the chain, or cycle back to the current thread. If this detects a cycle return EDEADLK; otherwise, go ahead with normal mutex lock operation processing.

It also told you which files and functions to modify:

- *mutex.h* 2 new lines of code, in *mac_mutex_lock* and *mac_mutex_unlock*
- *mutex.c* 45 new lines of code, in *pthread_mutex_lock*, *pthread_mutex_unlock*, *pthread_mutex_init*, and a new subprogram
- *pthread.h* 1 new line of code in the declaration of *struct pthread*
- *pthread.c* 2 new lines of code, for data structure initialization, in *pthread_init* and *pthread_create*

In grading, I looked for the following:

1. Deadlock criterion: As mentioned above, you should be checking for a cycle in the alternating chain of wait-for and hold relationships.
2. Components added to structs: You need to add a component to the pthread_t structure to record which mutex (if any) the thread is waiting for. You do not need to add anything to the mutex_t structure, since it already has a field to record the current holder (i.e., owner) of the mutex.
3. Functions where new data components are initialized: As mentioned in above, this is in *pthread_init* and *pthread_create*.
4. Functions where new data components are updated: *pthread_mutex_lock* and *pthread_mutex_unlock*.
5. How deadlock is reported back: return EDEACLCK, as mentioned above.