CSE 410 - Spring 2006
Final Exam (r1)

Closed book, no notes, no electronic devices

The appendix includes various tables and figures that you will find useful in working the problems.

One of the appendix tables gives you all the powers of 2. Do not try to calculate 2^n by hand, read it from the table!

Name: Solution DWJ

UWNetid : _____

This page is for use by the graders in recording the scores.

Page 1 _____ of 10 points

Page 2 _____ of 7 points

Page 3 _____ of 5 points

Page 4 _____ of 10 points

Page 5 _____ of 8 points

Page 6 _____ of 6 points

Total _____ of 46 points possible

1. (4pt) Fill in the blanks.

a. 111_2 (binary) is equal to ___7___$_{10}$ (decimal).

b. 10_{10} (decimal) is equal to ___A___$_{16}$ (hexadecimal).

c. A 2-bit field in a binary number can hold ___4___ different values.

d. A hexadecimal number like $FEDC_{16}$ can be represented exactly as a binary number. How many binary digits are needed to represent each of the hexadecimal digits? ___4___

2. (4pt) Answer True or False for each question.

a. (T) F A system with two CPUs might have four threads in the ready state at the same time.

b. T (F) A disk file that is allocated sequentially must fit entirely within one track on the disk surface.

c. (T) F A process is a dynamic entity with an associated address space.

d. (T) F In an operating system with pre-emptive scheduling, there must be some sort of external interrupt that can take control away from a running thread and give it to the scheduler.

3. (2pt) A thread control block is the generic term for the table that holds information about a running thread. Of the many entries that are held in a thread control block, name two of them and briefly describe their purpose.

program counter - points to next instruction to execute when this thread regains control

stack pointer - points to top of stack for this thread

registers - all registers at current point of interruption

Thread ID

This system is running Windows 2000 on 1 CPU w/ 1GB of installed memory

4. Consider the screenshot of the Windows Task Manager shown in Figure A2 in the appendix. During the period of time covered by the history shown, I was running three user application programs: Word (word processor), Firefox (web browser), and PaintShop (image editor). Each of these programs was run as a separate process.

a. (1pt) In addition to these three processes, how many other processes were running on my machine at the time the snapshot was taken? 45

b. (2pt) Notice that the kernel is using more than 130 MB of memory, of which more than 25 MB is "nonpaged". Give an example of an operating system task that might require some amount of nonpaged memory. Describe why you think this might require nonpaged memory.

Top level of page tables - access speed
page fault handling code - avoid page fault loop
scheduling code + good argument (scheduling needs to be fast)

During the entire period of time shown in the screenshot, Firefox was downloading a very large file. You can see the CPU usage bouncing along between 0 and 5% when the download is the only activity. The first major burst of CPU activity is when I started PaintShop. I then loaded an image file, did two CPU-intensive image processing operations, and eventually exited PaintShop.

c. (2pt) Do you think that PaintShop was CPU-bound at any point during this time? If yes, describe when this was true and ~~why~~ what evidence you see for this; if no, describe why not.

Yes, cpu-bound during the image processing. The evidence is that CPU usage goes to 100% and stays there for a short period of time twice.

d. (2pt) Do you think that Firefox was IO-bound at any point during this time? If yes, describe when this was true and ~~why~~ what evidence you see for this; if no, describe why not.

Yes i/o-bound during the entire period. The evidence is that even though the download was continuous, CPU load was only 0-5%. It was waiting on I/O the rest of the time.

5. Again consider the Windows Task Manager, figure A2.

One thread in the Firefox process performed the file download (which continued throughout the entire time shown). One thread in PaintShop read and initialized all the various image filters and program extensions when the program was first loaded and then continued running to process user commands.

a. (1pt) In addition to the two threads described above, how many other threads were running on my machine at the time the snapshot was taken? 391

b. (2pt) The three primary thread states that we discussed are *running*, *waiting*, and *ready*. Consider the period while PaintShop was first loading into memory. Describe a situation in which both the Firefox and PaintShop threads might have been in the same state (ie, both threads are running, waiting, or ready) during this period. Explain why each thread is in the chosen state in your example.

running - no - there is only one CPU

waiting - possible - both threads are waiting for I/O on disk, net, console

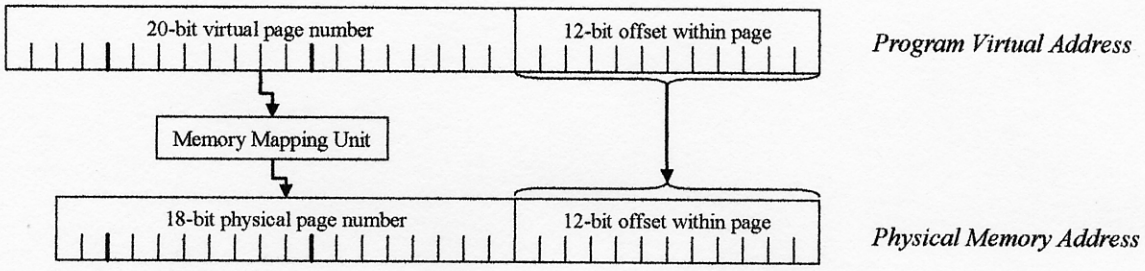
ready - possible - both threads have work and are ready to run but some other process (like Word) is running at the moment and so the other two cannot run right now.

c. (2pt) During most of the time shown here most of the other threads on the machine were all in the same thread state. What state is that most likely to be? waiting
Explain why you think this is the likely choice.

not running, only 1 CPU

not ready, CPU usage would be 100% because another thread would be running when Firefox and PShopPro were waiting

6. One of the key abstractions that have made modern systems possible is the separation of virtual addresses used by the program from the physical addresses in memory. In a paged system, a memory mapping unit translates virtual page numbers to physical page numbers. For example, consider this architecture for mapping virtual byte addresses to physical byte addresses:



a. (2pt) In the scheme shown above, is the virtual address space of each process larger or smaller than the physical address space of the system?

(Larger) Smaller

b. (2pt) By what factor is the virtual address space larger or smaller than the physical address space?

$2^2 = 4$

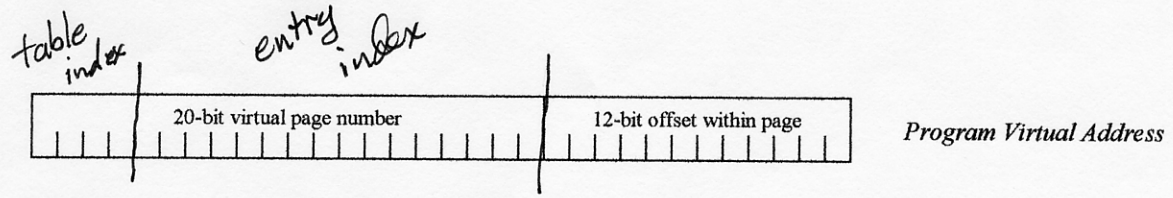
c. (2pt) How many bytes are included in each page of memory in this system?

$2^{12} = 4K = 4096$

d. (2pt) How many bytes of physical memory can be addressed using this system?

$2^{30} = 1GB = 1,073,741,824$

e. (2pt) One way to implement this memory mapping scheme would be with a 2-level page table for each process address space. Indicate on this drawing where you would divide the virtual page number so that the system could pick one of 16 possible top level tables, each of which contains 65,536 entries. Label the resulting "table index" and the "entry index" fields on the drawing.



7. For this question, refer to figure A3, Memory Mapping, in the appendix.

One of the capabilities enabled by the use of paging is demand paging. For this problem, assume that we are using a 16-bit virtual addressing scheme with 4-bit virtual page numbers, as shown in figure A3.

The memory management unit in this problem keeps three pieces of information about each virtual page: valid?, in?, and physical page number (PPN). A page is *valid* if the given virtual address is valid for this address space. (An address is valid if there can be code or data at that location, it is invalid if nothing can be put there.) A page is *in* if it is loaded in physical memory, it is *not in* if it is mapped to the page file. The *physical page number* points to a virtual page's location in memory or on disk as appropriate.

a. (2pt) How many pages of the given program are actually in memory at the time shown in the figure? 6

b. (2pt) Give an example of an address that will cause a page fault and cause the operating system to read from the page file. Give your answer as a 4-digit hexadecimal number. E000

c. (2pt) Given the configuration of memory shown here, is it likely that this system is suffering from external fragmentation of memory, meaning that there is not enough contiguous free memory to load another program? Explain why external fragmentation is or is not a problem for this system.

No. This system is paged and does not require contiguous memory for anything.

d. (2pt) Describe a circumstance in which there will be internal fragmentation on one of the pages in memory. This does not have to be a long involved answer; just describe how internal fragmentation can happen in this system.

Internal fragmentation occurs when the needed space is smaller than the allocation block. This would happen if the code stopped short of a page boundary, if the stack is ~~smaller~~ not exactly a page multiple in size, if a small set of global variables were allocated a complete page but did not fill it, etc

8. For this question, refer to figure A4, Data Acquisition System, in the appendix.

a. (2pts) The network connection is a BufferedOutputStream. Should the getData process call the flush() method after it writes each data packet? Briefly explain your answer.

Yes. otherwise the data packets may linger in the buffer until the buffer fills. The display process will not see the data until it is flushed from the buffer, either explicitly or due to buffer filling.

b. (2pts) Which synchronization construct is more likely to be useful for managing access to the common data area in item Ⓑ: a plain lock (ie, a mutex, a synchronized object) or a monitor (a lock and associated wait / notifyAll logic)? Explain your choice and say why it is more appropriate than the alternative.

Monitor. That way the display threads can wait for data to arrive and the receiver can notify them when it does. With a plain lock the displays would have to loop continuously.

c. (2pts) Which synchronization construct is more likely to be useful for managing access to the control variables in item Ⓒ: a plain lock (ie, a mutex, a synchronized object) or a monitor (a lock and associated wait / notifyAll logic)? Explain your choice and say why it is more appropriate than the alternative.

Lock. The goal is to avoid corruption, but no waiting is necessary. The receiver locks the variables, uses them, and releases the lock. The control thread locks the variables, updates them, and releases the lock.