University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 2007 John Kubiatowicz

Midterm II

December 3rd, 2007

CS162: Operating Systems and Systems Programming

Your Name:	
SID Number:	
Circle the letters of CS162 Login	First: a b c d e f g h I j k l m n o p q r s t u v w x y z Second: a b c d e f g h I j k l m n o p q r s t u v w x y z
Discussion Section:	

General Information:
This is a **closed book** exam. You are allowed 1 page of **hand-written** notes (both sides). You have 3 hours to complete as much of the exam as possible. Make sure to read all of the questions first, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. *Make your answers as concise as possible.* On programming questions, we will be looking for performance as well as correctness, so think through your answers carefully. If there is something about the questions that you believe is open to interpretation, please ask us about it!

Problem	**Possible**	**Score**
1	20	
2	25	
3	20	
4	35	
Total		

[This page left for π]

3.14159265358979323846264338327950288419716939937510582097494 4

Problem 1: True/False [20 pts]

In the following, it is important that you *EXPLAIN* your answer in TWO SENTENCES OR LESS (Answers longer than this may not get credit!). Also, answers without an explanation *GET NO CREDIT*.

Problem 1a[2pts]: The Clock Algorithm requires hardware support for a "use" bit in the PTE.

True / False
Explain:

Problem 1b[2pts]: The Aloha algorithm for broadcast networking permitted a new transmitter to interrupt a message that was already partially transmitted.

True / False
Explain:

Problem 1c[2pts]: Memory mapped I/O devices cannot be accessed by user-level threads.

True / False
Explain:

Problem 1d[2pts]: It is possible to make Remote Procedure Calls (RPCs) with integer arguments between clients that use big-endian integers and servers that use little-endian integers.

True / False
Explain:

Problem 1e[2pts]: A dictionary attack could be used against a publicly readable password file that contains encrypted passwords.

True / False
Explain:

Problem 1f[2pts]: The rate of page faults in a virtual memory system can always be reduced by adding more memory.

True / False
Explain:

Problem 1g[2pts]: *Compulsory* misses in a cache can be reduced with prefetching.

True / False
Explain:

Problem 1h[2pts]: A "memoryless" probability distribution provides a poor model for any real sources of events.

True / False
Explain:

Problem 1i[2pts]: Nonvolatile Ram (NVRAM) can improve the durability of a file system that uses delayed writes.

True / False
Explain:

Problem 1j[2pts]: Because of the 32-bit IP-V4 address space, it is impossible for more than 2^{32} computers to communicate over the internet.

True / False
Explain:

Problem 2: Virtual Memory and Paging [25pts]

Consider a two-level memory management scheme on 22-bit virtual addresses using the following format for virtual addresses:

Virtual Page # (7 bits)	Virtual Page # (7 bits)	Offset (8 bits)

Virtual addresses are translated into 16-bit physical addresses of the following form:

Physical Page # (8 bits)	Offset (8 bits)

Page table entries are 16 bits in the following format, *stored in big-endian form* in memory (i.e. the MSB is first byte in memory).

Page Table Entry (PTE)

Physical Page # (8 bits)	Kernel Only	Uncacheable	0	0	Dirty	Use	Write	Valid

Note that a virtual-physical translation can fail at any point if an incompatible PTE is encountered. Two types of errors can occur during translation: "invalid page" (page is not mapped at all) or "access violation" (page exists, but access was illegal).

Problem 2a[2pts]: Can you give a logical reason why the designer might have made the virtual page # fields 7 bits each?

Problem 2b[2pts]: What is the maximum amount of physical memory addressable by this system? Can you think of a way to increase the amount of available physical memory without altering the widths of the virtual addresses or PTEs?

Problem 3c[2pts]: How many total bits of storage will each entry of the TLB consume (including the tag and/or other fields)? Explain.

Problem 2d[4pts]: The contents of physical memory are given on the next page. *Assume that the page-table base pointer = 0x2000, and that the CPU is in user-mode*. Translate the following virtual addresses. If there is an error during translation, make sure to say what the error is. Errors can be either "invalid page" or "access violation". Two answers are give. *Hint: be careful to look at the virtual address format!*

Virtual Addr	Physical Addr
0x10123	**0x0023**
0x60423	**Invalid Page**
0x28156	

Virtual Addr	Physical Addr
0x38256	
0x10F00	
0x00278	

Problem 2e[6pts]: Consider the same multi-level memory management scheme. Once again, *assume that the page-table base pointer = 0x2000 and that the CPU is in user-mode.* Please return the results from the following load/store instructions. Addresses are virtual. The return value for load is an 8-bit data value or an error, while the return value for a store is either "**ok**" or an error. For errors, please specify which type of error (either "invalid page" or "access violation"). When the result is not an error, indicate the new value for the PTE after the access is complete. The first two results are given:

Instruction	Return Value	New PTE
Load [0x380FE]	**0xEE**	**0x3005**
Store [0x380FE]	**Access violation**	**------**
Load [0x41015]		
Load [0x00115]		
Store [0x08310]		
Load [0x10102]		
Store [0x10731]		
Load [0x62345]		

Problem 2f[2pts]: Suppose that there are two processes loaded on the ready queue. Process A has page-table base pointer = 0x2000. Process B has page-table base pointer = 0x2F00. Give the *physical address* of a data page that is shared read-write (RW) between processes A and B? Is this data page mapped at the same place in the *virtual* address spaces of the two processes? Explain.

Virtual Address Format

Virtual Page # (7 bits)	Virtual Page # (7 bits)	Offset (8 bits)

Page Table Entry (PTE)

Physical Page # (8 bits)	Kernel	Not Cacheable	0	0	Dirty	Use	Write	Valid

Physical Memory

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x0000	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
0x0010	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D
….																
0x1010	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
0x1020	40	07	41	06	30	06	31	07	00	07	00	00	00	00	00	00
….																
0x2000	21	01	22	01	25	01	22	01	2F	03	28	03	30	03	22	03
0x2010	40	81	41	81	42	81	43	83	00	00	00	00	00	00	00	00
….																
0x2100	30	05	31	01	32	03	33	07	34	00	35	00	36	00	37	00
0x2110	38	00	39	00	3A	00	3B	00	3C	00	3D	00	3E	00	3F	00
….																
0x2200	30	01	31	83	00	01	00	0F	04	00	05	00	06	00	07	00
0x2210	08	00	09	00	0A	00	0B	00	0C	00	0D	00	0E	00	0F	00
….																
0x2500	10	01	00	03	12	85	13	05	14	05	15	05	16	05	17	05
0x2510	18	85	19	85	1A	85	1B	85	1C	85	1D	85	1E	85	00	00
….																
0x2800	50	01	51	03	00	00	00	00	00	00	00	00	00	00	00	00
….																
0x2F00	60	03	28	03	62	00	63	00	64	03	65	00	66	00	67	00
0x2F10	68	00	69	00	00	00	00	00	00	00	00	00	00	00	00	00
0x2F20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
….																
0x30F0	00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF
0x3100	01	12	23	34	45	56	67	78	89	9A	AB	BC	CD	DE	EF	00
0x3110	02	13	24	35	46	57	68	79	8A	9B	AC	BD	CE	DF	F0	01
….																
0x4000	30	00	31	06	32	07	33	07	34	06	35	00	43	38	32	79
0x4010	50	28	84	19	71	69	39	93	75	10	58	20	97	49	44	59
0x4020	23	87	20	07	00	06	62	08	99	86	28	03	48	25	34	21

Problem 2g[4pts]: For the following problem, assume a hypothetical machine with 4 pages of physical memory and 7 pages of virtual memory. Given the access pattern:

A B C D E F C A A F F G A B G D F F

Indicate in the following table which pages are mapped to which physical pages for each of the following policies. Assume that a blank box matches the element to the left. We have given the FIFO policy as an example.

Access→		**A**	B	C	D	E	F	C	A	A	F	F	G	A	B	G	D	F	F
FIFO	1	A				E									B				
	2		B				F										D		
	3			C					A									F	
	4				D								G						
MIN	1																		
	2																		
	3																		
	4																		
LRU	1																		
	2																		
	3																		
	4																		

Problem 2i[3pts]: What is a precise exception and why would we want a software TLB fault to generate a precise exception?

Problem 3: File Systems [20pts]

Please keep your answers short (one or two sentences per question-mark). *We may not give credit for long answers.*

Problem 3a[2pts]: What is spatial locality and which type of file access pattern exploits spatial locality?

Problem 3b[2pts]: Which component of disk access time is the *disk scheduling algorithm* trying to minimize?

Problem 3c[3pts]: How does a *Journaled* file system improve the durability of data on disk? Give two reasons why a journaled file system would have higher performance than a file system that forced every block to the disk immediately, while at the same time giving the same or better durability.

Problem 3d[2pts]: Name at least two ways in which the *buffer cache* is used to improve performance for file systems.

Problem 3e[2pts]: The Fast File System (FFS) of Berkeley 4.2 Unix utilized "Skip Sector Positioning" to improve performance. Explain what "Skip Sector Positioning" is and why this optimization may no longer be important.

Problem 3f[3pts]: The Network File System (NFS) uses a "stateless" protocol between clients and servers. What does this mean? Name one advantage and one disadvantage of stateless filesystem protocols.

Problem 3j[6pts]: Suppose that a new disk technology provided access times that are of the same order of magnitude as memory access times. What, if anything, must be changed in the following three OS components to take advantage of the quicker access time? If something doesn't change, be very specific why it doesn't change. If it will change, contrast these changes with current implementation and be as specific as possible in your answers (i.e. identify what would change and why).

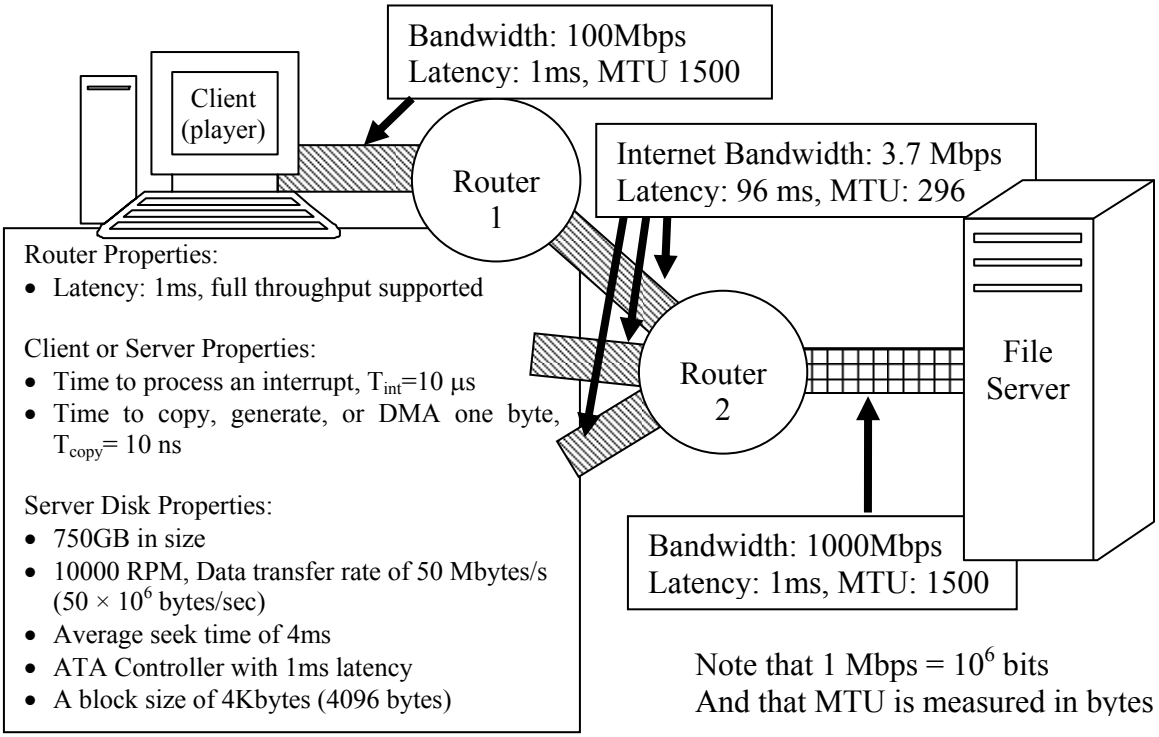
1. Process Scheduler

2. Memory Management

3. Device Driver for the new disk

[This page intentionally left blank]

Problem 4: Network File Server [35 pts]



The above figure illustrates a network in which one client interacts with a server to display video files. Each link is characterized by its Bandwidth, one-way Latency (for the first bit to arrive), and Maximum Transfer Unit (MTU) in *bytes*. All links are full-duplex (can handle traffic in both directions at full bandwidth). Each router is highly pipelined and will start forwarding a packet to an output port 1ms after it receives the first bit of the packet. Assume that no packets will be lost.

Both the video player (on the client) and file server run at user level. To display a video, the client opens a TCP connection to the server, sends multiple requests, then waits for the response. Assume that the time to setup the connection is negligible relative to the overall viewing time.

To send data over the network, a user-level process issues a write system call on a socket. The OS will copy the data to a kernel buffer (inside the socket). Then, the TCP routines generate headers (by copying them from templates) and use DMA to copy the completed packet to the network controller. Immediately after receiving all of the data for a packet, the network controller sends it. Afterwards, it generates an interrupt (one per packet).

At the destination, the network controller DMAs the packet to memory, then interrupts the CPU. Subsequently, the TCP routines transmit an acknowledgement (ACK) back to the sender. For simplicity, you may treat ACKs as if they are zero bytes long. After sending an ACK, the OS copies the received data into user-level buffer of the receiving user-level process.

A user-level process reads data from the disk by executing a *read()* system call with a user-level buffer for the results. The OS splits read requests into chunks that consist of sequential blocks on the same track and that are 16 blocks or less in size. It queues requests in the *device driver request queue (DDRQ).* Whenever the disk controller finishes reading a chunk of data into kernel memory, the controller generates an interrupt, causing the OS to dequeue the next request from the DDQ. The OS then copies data from kernel memory into the user's buffer and returns from *read().*

Problem 4a[3pts]: What is the maximum amount of *data* that the client or server can send in each packet while avoiding fragmentation along the way? Remember that the TCP+IP header is 40 bytes. Explain.

Problem 4b[2pts]: Explain how the client or server could automatically discover the answer to 4a:

Problem 4c[2pts]: Under ideal circumstances (and ignoring interrupt and copying overheads and window sizes), what is the maximum *data* bandwidth that could be sent from the server to the client without dropping packets, assuming that the server utilizes the mechanism in 4b? Explain.

Problem 4d[3pts]: Assume that a maximal sized packet (as in 4a) is sent from the TCP send buffer of the server to the TCP receive buffer of the client. Compute the total roundtrip latency from the point at which the server invokes the device driver to send the packet until it receive the ACK via an interrupt. *Hint: don't forget the interrupt at the receiving side and the DMA mechanism into and out of the network.*

Problem 4e[3pts]: What TCP window size is necessary to achieve the bandwidth of 4c without dropping packets in the network? Use simple constants and values that you computed for 4a-4d. Explain. Make sure that you correct for units.

Problem 4f[6pts]: Suppose that video files are laid out in 64K (65536 bytes) chunks on the disk (i.e. 64K in successive sectors on a track). Compute the overhead for reading such a 64K chunk from a random place on the disk. Assume that the disk controller automatically DMAs the data to kernel memory in a fashion that is overlapped with reading it from the disk (so that you do not have to worry about DMA for this operation). After finishing, the controller generates an interrupt; the interrupt routine may submit another request to the controller (if one is queued on the DDRQ). Assume the disk parameters given above (repeated here):

- 750GB in size
- 10000 RPM, Data transfer rate of 50 Mbytes/s (50×10^6 bytes/sec)
- Average seek time of 4ms
- ATA Controller with 1ms controller initiation time
- A block size of 4Kbytes (4096 bytes)

What is the total time to read 64K chunk from a random place on the disk into memory including the interrupt? *Hint: there are 5 terms here including the interrupt!*

Problem 4g[3pts]: Now, assume that the video player works by sending requests for 64KB (=65536 bytes) at a time to the video server. Assuming that these requests are pipelined for maximum bandwidth, at what rate must it send these requests to achieve the bandwidth of 4c?

Problem 4h[4pts]: Compute the total *processor* overhead for satisfying a 64K request at the server. Processor overhead includes (1) interrupts, (2) header generation overhead and (3) copying overhead. *Note that generating a byte of header is as expensive as copying*. Processor overhead does not include DMA or disk controller actions since these are overlapped with the CPU. The data portion of a request message is 16 bytes.

Problem 4i[4pts]: Suppose that we wanted to handle 300 clients each at a request rate as given in (4g). Assume that multiple clients connect through network resources that are independent up to router #2. Each client would have its own user-level process working on its behalf. How many disks would we need to handle this rate? What else would need to be upgraded to handle this rate?

Problem 4j[5pts]: Returning to our single-disk server, assume that 15 clients connect through network resources that are independent up to router #2. Each client has its own user-level process working on its behalf. Assume that all 15 clients send requests at a rate of (4g). If the disk response distribution can be described with C=1.5 and the aggregate network request rate can be described as exponential, what is the average length of the disk device driver request queue (DDRQ)? *Hint: the service time for the disk is the time between requests to the disk controller when the DDRQ is full. Possibly useful formulas include:*

Mean Service: $T_{server} = \frac{1}{n}\sum_{i=1}^{n} T_i$ *Variance:* $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(T_i - T_{server})^2$

M/G/1 queue: $T_q = T_{server} \times \left(\frac{1+C}{2}\right) \times \left(\frac{u}{1-u}\right)$ *Little's law:* $L_q = \lambda \times T_q$

[Scratch Page (feel free to remove)]