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

Fall 2015 John Kubiatowicz

Midterm II
SOLUTIONS
April 22nd, 2015
CS162: Operating Systems and Systems Programming

Your Name:	
SID Number:	
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	18	
2	20	
3	20	
4	24	
5	18	
Total		

[This page left for π]

3.141592653589793238462643383279502884197169399375105820974944

Problem 1: True/False [18 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]: mmap maps memory addresses to portions of files by changing page table entries to map virtual addresses to disk block addresses.

True / [✓ False]

Explain: *mmap() maps virtual memory addresses to pages in the buffer cache which holds the contents of a file.*

Problem 1b[2pts]: In the Fast File System (FFS), an inode includes all metadata about a file, including its size, permissions, name, and pointer to disk blocks.

True / [✓ False]

Explain: *The FFS inode does not include the name of the file.*

Problem 1c[2pts]: Memory mapped I/O devices can be accessed by user-level threads under the right circumstances.

[✓ True] / False

Explain: *If the kernel maps the physical addresses that control the device into the address space of a process, then threads in that process can access that I/O device.*

Problem 1d[2pts]: All physical addresses accessed by the CPU correspond to locations in DRAM.

True / [✓ False]

Explain: *Some physical addresses correspond to memory-mapped I/O devices.*

Problem 1e[2pts]: A transactional system that uses two-phase locking (2PL) guarantees that transactions will be conflict serializable.

[✓ True] / False

Explain: *2PL prevents cyclic dependencies between transactions, since all locks for accessed items must be acquired before any accesses occur and before locks are released. Any potential conflicts with be transformed into deadlocks – which must be resolved by aborting one of the corresponding transactions.*

Problem 1f[2pts]: A fully-associative cache always experiences fewer misses than a direct-mapped cache of the same total size.

True / [✓ False]

Explain: *Consider a program that repeatedly performs a sequential scan through an array that is slightly larger than the cache. Most accesses will fit entirely in the direct-mapped cache, whereas the fully-associative cache will experience a miss on every access.*

Problem 1g[2pts]: An M/M/1 queue has exponential distributions of both arrivals and service times.

[✓ True] / False

Explain: *Here, "M" stands for "Memoryless", which is an exponential distribution. The notation M/M/1 denotes Memoryless arrival and service distributions.*

Problem 1h[2pts]: The optimal policy for handling events is to always utilize interrupts for signaling, since this policy adapts well to the unpredictability of event arrival.

True / [✓ False]

Explain: *Processing of interrupts involves a lot of overhead and is not always desirable when event arrival is regular or predictable. A better policy involves both interrupts and polling, such as typically utilized in network drivers (e.g. NAPI).*

Problem 1i[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: *The use of Network Address Translation (NAT) allows multiple computers to share the same external IP address. As a result, many more than just 2^{32} computers can be connected to the internet.*

Problem 2: Short Answer [20pts]

For the following questions, please provide as short an answer as you can that actually answers the question. In most cases, this means one or two sentences. *We reserve the right to take off points for answers that are not concise.*

Problem 2a[3pts]: What information would an IP router need to allow it to route packets to arbitrary IP addresses? Explain why every router does not need to know about every destination.

The IP router needs enough information to map every IP destination address to an outgoing Port (i.e. to a next hop). Typically, such information is a table of IP prefixes that map to next port. The router only needs to know how to forward a packet closer to its destination (or to another router that knows how to route the given packet).

Problem 2b[2pts]: Under what conditions are file caches effective for speeding up file reads, i.e. how does the effectiveness the cache depend on the file access patterns of user programs?

File caches are effective whenever the working set of the access pattern is smaller than the size of the cache. Or another way to say this is that the access pattern must have enough locality (spatial and/or temporal) to make the cache effective.

Problem 2c[3pts]: What is a precise exception and when are precise exceptions useful?

A precise exception is one where the state of the machine is preserved as if the program executed up to the offending instruction. All previous instructions have completed, and the offending instruction and all following instructions act as if they had not even started. Precise exceptions are useful because they are very easy for the operating system to restart. Consequently, they are particularly useful for exceptions such as page faults or device interrupts which are frequent and must be easily restartable.

Problem 2d[3pts]: Suppose you have a RAID-5 system consisting of 4 disks. What must you do to recover data when one disk fails? What must you do to replace the failed disk? Be explicit.

Data recovery is simple: the remaining 3 disks contain all the necessary information. Either the information that you require is available directly from one of the remaining disks (since said disk was a data disk in that particular stripe), or it can be retrieved by XORing disk blocks from all three disks (if the particular block you must access was on the failed disk). To replace the failed disk, you must first replace it physically, then replace the failed data by XORing together the other 3 disks.

Problem 2e[2pts]: In the context of the ACID transactional semantics, explain how a system can exhibit atomicity, but not consistency.

Atomicity does not prevent race conditions (or prevent transactions from occurring simultaneously). As a result, simultaneous transactions can make inconsistent changes to data: for instance, two simultaneous transactions attempting to both increment a variable might result in only a single increment, as one overwrites the results of another after commit. One solution to this is two-phase locking.

Problem 2f[3pts]: Explain what copy-on-write is (with respect to the virtual memory system) and explain how the virtual memory system can be used to implement copy-on-write.

Copy-on-write is a technique for producing a new address space that is a clone of an old address space without copying all the data in the old address space. Instead, a new page table is constructed that is a clone of the original page table, i.e. that points at all the data pages in the original address space. Further, all pages are marked as read-only in both page tables. Now, when a write occurs in either address space, the resulting page fault will be utilized to make a copy of the data page (i.e. "copying on write").

Problem 2g[2pts]: Explain how the syscall number and the syscall arguments are passed from the user program to the kernel in Pintos.

The syscall number and arguments are passed on the stack using the stack pointer. The arguments are pushed onto the stack in reverse order, and the syscall number is pushed onto the stack last of all.

Problem 2h[2pts]: Explain what DMA is (don't just say what it stands for) and explain how it helps to improve the performance of a file system.

Direct Memory Access (DMA) is a technique by which a DMA controller operates independently of the processor to transfer data between devices and memory. DMA helps to improve the performance of a file system by freeing the processor to do other activities (such as management of the buffer cache) while data is being transferred from the disk. Also, DMA can operate at the full bus bandwidth, which might be difficult for the processor.

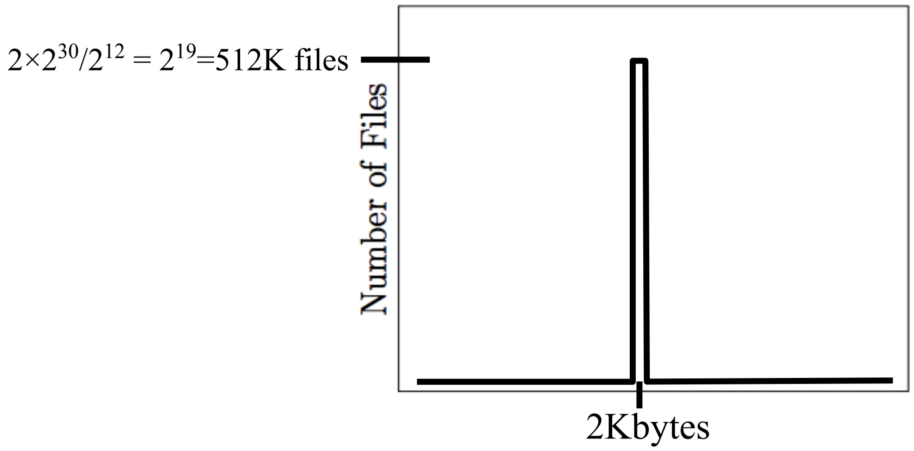
Problem 3: File Systems [20pts]

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

Pintos-inspired file system: Consider the following file system code inspired by Pintos. You may assume all the code is run on Intel x86 32 bit architecture. Instead of inodes, dirents, and data blocks, our file system will be entirely comprised of fnodes. fnodes are constant size structures that will store all the data for the file. The entire disk is composed of fnodes, fnodes are accessed similarly to inodes (via index). The file system only has one directory "/" and all fnodes are under that directory. No more directories will be formed.

```
#define BLOCK_SIZE 4096
/* Block device that contains the file system. */
struct block *fs_device;

/* Reads sector SECTOR from BLOCK into BUFFER */
void block_read(struct block *block, block_sector_t sector,
                void *buffer);

/* Write sector SECTOR to BLOCK from BUFFER, mark block as not free */
void block_write(struct block *block, uint32_t sector,
                const void *buffer) ;

/* Returns one free block */
long get_free_block();

/* Marks given block as free */
void free_block(long bnum);

/* ??? */
int lookup(char* file_name);

/* On-disk fnode. Must be exactly BLOCK_SIZE bytes long. */
struct fnode_disk {
   uint32_t file_name[511]; /* File Name. */
   uint32_t data[512]; /* File Data. */
   unsigned int magic; /* Magic number. */
};

/* In-memory fnode. */
struct fnode {
   struct list_elem elem; /* Element in fnode list. */
   uint32_t sector; /* Sector number of disk location. */
   int open_cnt; /* Number of openers. */
   bool removed; /* True if deleted, false otherwise. */
   int deny_write_cnt; /* 0: writes ok, >0: deny writes. */
   struct fnode_disk *data; /* fnode content. */
};
```

Problem 3a[3pts]: Given a disk with 2 GB (GB = 2^{30} bytes) of usable space (not FS metadata), draw a histogram on the graph below to reflect the optimal usage pattern for this file system. Note that the 2GB also contains file names and magic numbers (complete fnode_disk structures). Remember to label the values on each axis so we can interpret your graph properly. Particularly consider the maximum attainable values on either axis. *Hint: There is exactly one answer*



Problem 3b[3pts]: Given what you know about this particular file system, in order for file system operations to succeed what do you suppose the function lookup will need to do? Describe one possible way the `lookup()` function could be implemented. *Please limit your answer to twenty words or less!*

The lookup() function must map names to fnode numbers (indices). This could be implemented with an in-memory hash table.

Problem 3c[6pts]: Suppose that you want to implement hard links in the above file system, and that you want the links to behave identically to other file systems.

a. What changes do you need to make to the `fnode_disk` structure to make this possible?

b. Describe the necessary modifications to `write()` and `read()` operations, if any.

c. What are two major disadvantages of this file system with respect to hard links?

Please limit each of your answers (a,b,c) to twenty words or less!

There are a couple of options here. One is to make multiple copies of the file that are updated together – so that they look like one file. This option would be described as follows:

a. *Need to add a next_link field*
b. *Writes would follow next_link in a circuarly linked list and update all links*
c. *Each link would essentially be a copy of the file, writes must be repeated*

Another option would be to make special "link" fnodes that point at the "real" file and have no actual data. These fnodes could have a different magic number and would need to be linked to the "primary" fnode with the "next_link" field. So:

a. *Need to add a next_link field*
b. *Writes and reads would follow next_link in a circularly linked list to the primary fnode.*
c. *Writes and reads to hard links would be more expensive; file/link creation and deletion would be more complex.*

Problem 3d[2pts]: You decide to buy storage devices for the file system and have a choice between a 7200 RPM SATA 40 TB (TB = 2^{40} bytes) hard disk or a 20 TB SSD. If you are looking to maximize both read/write throughput and storage capacity which of the choices is a better option. Why? *Please limit your answer to twenty words or less.*

In this case, you should pick the SSD, even though it has less total capacity. The SSD is faster and the fact that sectors are labeled with a 32 bit number means we can't access more than: $2^{32} \times 4K = 2^{32} \times 2^{12} = 2^{44} = 16$ *TB*

Problem 3e[6pts]: Disk requests come in to the driver for cylinders 3, 9, 6, 8, 4, 10, 2, simultaneously, in that order. A seek takes 5µs per cylinder. Calculate the sequence of reads and the total seek time for each of the following policies: First In First Out (FIFO), Shortest Seek Time First (SSTF), Elevator Algorithm (SCAN). In all cases, the disk arm is initially at cylinder 8. In case of a tie, pick the option that will minimize total seek time. For the elevator algorithm, assume the arm is initially moving toward decreasing values.

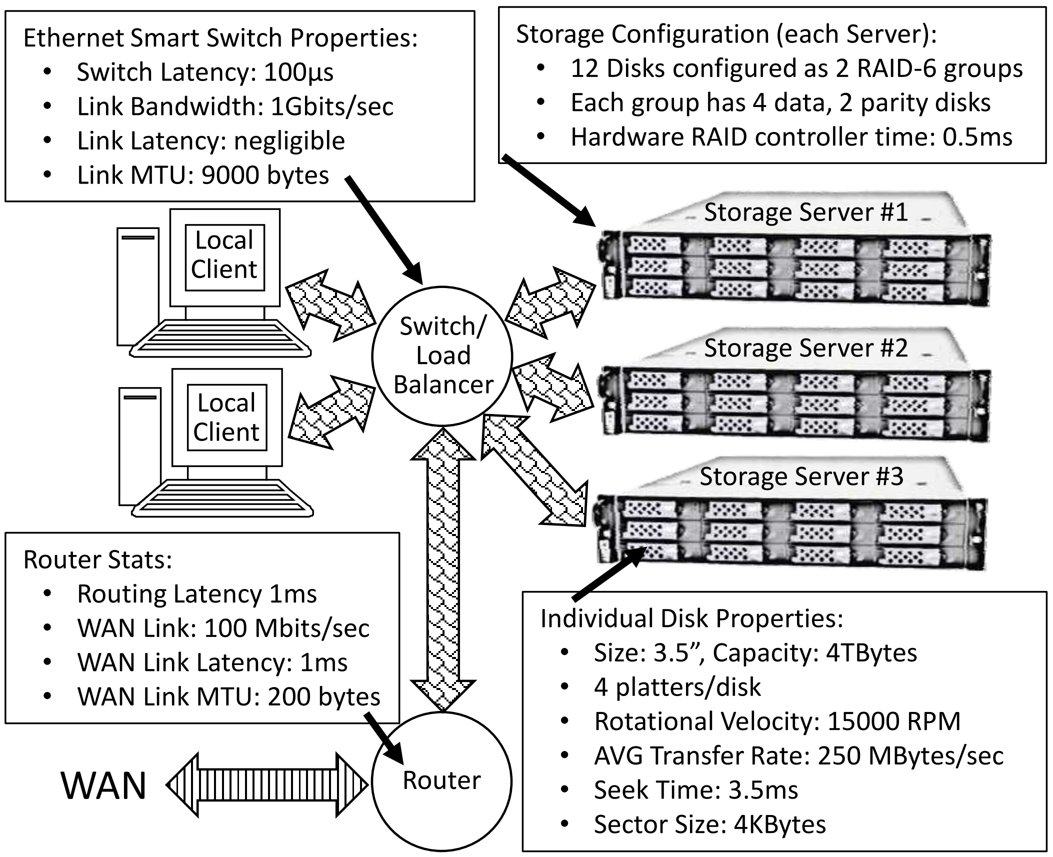
FIFO: *3, 9, 6, 8, 4, 10, 2. Seek time: 5µs × (5 + 6 + 3 + 2 + 4 + 6 + 8) = 170µs*

SSTF: *8, 9, 10, 6, 4, 3, 2. Seek time: 5µs × (0 + 1 + 1 + 4 + 2 + 1 + 1) = 50µs*

SCAN: *8, 6, 4, 3, 2, 9, 10. Seek time: 5µs × (0 + 2 + 2 + 1 + 1 + 7 + 1) = 70µs*

[This page intentionally left blank]

Problem 4: Cloud Video Storage Server [24pts]


Ethernet Smart Switch Properties:
• Switch Latency: 100µs
• Link Bandwidth: 1Gbits/sec
• Link Latency: negligible
• Link MTU: 9000 bytes
Storage Configuration (each Server):
• 12 Disks configured as 2 RAID-6 groups
• Each group has 4 data, 2 parity disks
• Hardware RAID controller time: 0.5ms
Local Client
Local Client
Switch/ Load Balancer
Storage Server #1
Storage Server #2
Storage Server #3
Router Stats:
• Routing Latency 1ms
• WAN Link: 100 Mbits/sec
• WAN Link Latency: 1ms
• WAN Link MTU: 200 bytes
WAN
Router
Individual Disk Properties:
• Size: 3.5", Capacity: 4TBytes
• 4 platters/disk
• Rotational Velocity: 15000 RPM
• AVG Transfer Rate: 250 MBytes/sec
• Seek Time: 3.5ms
• Sector Size: 4KBytes

The above figure illustrates a datacenter network designed to distribute video files locally and over the wide area network (WAN). As shown, multiple local clients interact with local storage servers using TCP/IP. An Ethernet switch connects these two types of elements. Further, a router gives access to the outside world over a WAN connection.

Servers have 12 disks each, divided as 2 separate RAID-6 groups of 6 disks each. Each RAID group has 4 data disks coupled with 2 parity disks. All disks are identical with the same parameters (see the above diagram). Assume that you *do not* have to account for the difference in bit rate between outer and inner tracks; instead, we have quoted an *average* transfer rate.

Assume that the file system *stripes* data across the disks in the RAID group. This means that sequential sectors in the file system are distributed *across* 4 data disks (one on each disk) before continuing with the next sector on a given disk. Further, keep in mind that it is not possible to read from multiple platters or surfaces on a given disk at the same time. You can switch platters with zero delay, however. It is possible to read from more than one disk at the same time.

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 switch and router is highly pipelined and will start forwarding bytes from a packet to the next hop immediately after the routing/switching delay has expired. Assume that the bottleneck link for the WAN is described by the parameters of MTU=200 bytes and BW=100Mbits/sec.

Problem 4a[5pts]: Assume that sectors are *striped* across a RAID stripe (e.g., since the RAID-6 stripe consists of 6 disks, sequential sectors are distributed *across* 4 data disks before touching the next sector in a given track). How long will it take to read 256KB of sequential data from the RAID group (starting at a random starting position on the disk), assuming that you can do so by making a single request to the RAID controller? You can assume that the disks are synchronized with one another.

256KB/4disks = 64KB/disk. Assume all 4 disks are read simultaneously (so that time to read 4 disks is same as time to read one disk):

Controller + Seek + ½ Rotation + Transfer =

$$0.5ms + 3.5ms + \left(\frac{1}{2}rev \times \frac{60000ms/\min}{15000rev/\min}\right) + \left(\frac{65536bytes \times 1000ms/\sec}{250 \times 10^6 bytes/\sec}\right) = 6.262ms$$

Problem 4b[2pts]: What is the maximum *rate* in MB/sec that data can be read randomly from one RAID group, using 256K chunks of data as in (4a)? Assume that each request is independent (and uncorrelated) from the previous request.

Assuming that requests are uncorrelated, then each 256KB requires 6.262 ms.

So, rate = $\left(\frac{256KB \times 1024bytes/KB \times 1000ms/\sec}{6.262ms}\right) = 41.9 \times 10^6 bytes/\sec = 41.9MB/\sec$

Problem 4c[3pts]: Assume that clients are reading H.264-encoded video @1080p quality encoded at a 7.68 Mbit/sec data rate. Further, assume that external clients (over WAN) are using TCP/IP to watch videos. Under *ideal* circumstances, what is the maximum number of simultaneous video streams that could be submitted over the WAN link? Keep in mind that the TCP/IP header is 40 bytes in size.

Assuming no overhead other than TCP/IP header and that packets are exactly MTU bytes in size, then WAN data bandwidth is (200-40)/200 × 100Mbits/sec = 80Mbits/sec

With this bandwidth, we could get $\left\lfloor \frac{80Mbits/\sec}{7.68Mbits/\sec} \right\rfloor = \lfloor 10.41 \rfloor = 10$ *streams. Note that fractional streams make no sense here….*

Problem 4d[2pts]: Can the incoming request rate calculated in (4c) be handled by a single RAID group using a 256K request size? Explain. What is the resulting *utilization* of the single group?

Note that (4b) is in Mbytes/sec, while video streams (4c) are quoted in Mbits/sec. So, 10 streams require 76.8 Mbits/sec, while a single RAID group can provide a total of 41.9 Mbytes/sec × 8 bits/byte = 335.2 Mbits/sec. So – there is plenty of bandwidth in a single RAID group to handle 10 streams.

Utilization = 76.8/335.2 = 0.23

Problem 4e[4pts]: Under the same assumptions as for (4c), what is the maximum number of simultaneous video streams that a local client could submit over the local link? Would a single RAID group be sufficient to handle this traffic? What about all three servers (six RAID groups)? Explain carefully.

We have local link BW of (9000-40)/9000 ×1Gbit/sec =996Mbits/sec.

$$\textit{Max video streams} = \left\lfloor \frac{996 Mbits/\sec}{7.68 Mbits/\sec} \right\rfloor = \lfloor 129.6 \rfloor = 129$$

BW for all these streams = 129 ×7.68Mbits/sec = 990.7Mbits/sec
No: a single RAID group @335.2Mbits/sec would not be sufficient
Yes: six RAID groups @335.2×6=2.011 Gbits/sec would be sufficient

Problem 4f[5pts]: Finally, treat the entire system as an M/M/m queue (that is, a system with m servers rather than one, where each RAID *group* can be treated as a "server"). Assume that the switch contains a smart load-balancing queue that distributes requests to the least loaded RAID group and that every group can be utilized to serve every request. Also assume that requests are 256KBytes in size, as in 4a and 4b. All requests are placed into a single queue in the load balancer. *Assume that a single local client sends a maximum request rate supported by his network* (as in 4e). What is the average number of requests queued at the load balancer? Ignore any latency in the OS of the storage server. The following equations might be useful:

$$\textit{M/M/m queue: } \mathrm{T_{queue}} = \mathrm{T_{server}} \times \left[\frac{\zeta}{m(1-\zeta)} \right] \quad \textit{where: } \zeta = \frac{\lambda}{\frac{1}{\mathrm{T_{server}}/m}} = \lambda \times \frac{\mathrm{T_{server}}}{m}$$

$$\textit{Little's Law: } \mathrm{L_{queue}} = \lambda \times \mathrm{T_{queue}}$$

Answer: The trickiest part about this is computing λ, namely number of 256KB requests/sec. So, this is simply the outgoing bandwidth for the 126 streams/size of a request (256KB).

$$\textit{Careful with units: } \lambda = \frac{129 \times (7.68 \times 10^6 bits/\sec)}{(256KB/req) \times (1024 byte/KB) \times (8 bits/byte)} = 472.4 req/\sec$$

$$\textit{So, we using result from(4a): } \zeta = \lambda \times \frac{\mathrm{T_{server}}}{m} = 472.4 req/s \times \frac{6.262 ms/req}{1000 ms/s} \times \frac{1}{6} = 0.49$$

$$\textit{Thus, } \mathrm{T_{queue}} = \mathrm{T_{server}} \times \left[\frac{\zeta}{m(1-\zeta)} \right] = 6.262 ms \times \left[\frac{0.49}{6 \times (1-0.49)} \right] = 1.00 ms$$

$$\textit{Finally, the ave \# reqs (queue length) is: } \mathrm{L_{queue}} = \lambda \times \mathrm{T_{queue}} = \frac{472.4 req/s \times 1.00 ms}{1000 ms/s} = .47 req$$

Problem 4g[3pts]: Under the same assumptions as (4f), what is the total latency for a request as seen from the standpoint of a local server? Ignore any software latency in the servers.

$$\mathrm{T_{request}} = \mathrm{T_{Route}} + \mathrm{T_{queue}} + \mathrm{T_{server}} + \mathrm{T_{route}} + \mathrm{T_{NetworkTransfer}} \Rightarrow$$

$$\mathrm{T_{request}} = 0.1ms + 1.00ms + 6.262ms + 0.1ms + \left(\frac{256KB \times 1024 bytes/KB \times 8 bits/byte}{(1Gbit/s) \times (10^9 bit/Gbit) \times (10^{-3} s/ms)} \right) = 9.56ms$$

[This page intentionally left blank]

Problem 5: Potpourri [18pts]

Problem 5a[6pts]: 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						
LRU	1	*A*				*E*							*G*						
	2		*B*				*F*										*D*		
	3			*C*											*B*				
	4				*D*				*A*									*F*	
CLOCK	1	*A*				*E*							*G*						
	2		*B*				*F*								*B*				
	3			*C*													*D*		
	4				*D*				*A*									*F*	

Problem 5b[6pts]: Consider a computer system with the following parameters:

Measurement	Value
P_T=prob of TLB miss	0.1
P_F=prob of a page fault when a TLB miss occurs	0.0002
P_D=prob page is dirty when replaced	0.5
T_T= time to access TLB	0 μs
T_M = time to access memory	1 μs
T_D = time to transfer a page to/from disk	10 ms = 10000 μs

The TLB is refilled automatically by the hardware on a miss (the TLB is only accessed once per reference). The page tables are kept in physical memory and are not hierarchical, so looking up a page table entry incurs one memory access. Assume that the costs of the page replacement algorithm and updates to the page table are included in the T_D measurement.

What is the average memory access time (the time for an application program to do one memory reference) on this computer? Assume physical memory is 100% utilized and ignore any software overheads in the kernel. Express your answer symbolically and compute the result to two significant digits. *Show all steps of the computation!*

Answer:

$$T_{access} = T_M + T_T + P_T \times (T_M + P_F \times (T_D \times (1 + P_D))) \Rightarrow$$

$$T_{access} = 1\mu s + 0\mu s + 0.1 \times (1\mu s + 0.0002 \times (10000\mu s \times (1 + 0.5))) = 1.4\mu s$$

Problem 5c[6pts]: Some operating systems have the `park(void *hint)` and `unpark(void *hint)` system calls to support user-level implementations of condition variables. When a thread calls `park(void *hint)`, it should yield the CPU. When a thread calls `unpark(void *hint)`, it should yield the CPU if and only if there are any other threads that had previously called park with the same hint value. Calling `unpark(NULL)` should match any non-null hint value The return value of park is always 0. The return value of `unpark`() is 1 if the thread yielded, otherwise it's 0.

The function `check_pointer(void *ptr)` will check that `ptr` points to valid 4 byte memory location in user memory. Implement the syscalls using the following sketch. You may not need all the blanks (write directly on the following lines):

```
void syscall_park(struct intr_frame *f) {
  uint32_t *args = (uint32_t *) f->esp;
  check_pointer(&args[1]);
  ________________________
  void *hint = args[1]__;
  thread_current()->hint = hint;
  thread_yield();
  f->eax = 0;__
}

void syscall_unpark(struct intr_frame *f) {
  uint32_t *args = (uint32_t *) f->esp;
  check_pointer(&args[1]);
  uint32_t yielded = 0;
  _____________________
  void *hint = args[1]__;
  for each struct thread t in ready_list {
    if (t->hint != NULL && (hint == NULL || hint == t->hint)) {
      t->hint = NULL
      thread_yield();
      yielded = 1;_
      break;_______
    }
  }
  f->eax = yielded;_____
}
```

[Scratch Page (feel free to remove)]