CS 3204 Final Exam

This is a closed-book, closed-internet, closed-cellphone and closed-computer exam. However, you may refer to your sheet of prepared notes. Your exam should have 11 pages with 5 questions totaling 100 points. You have 120 minutes. Please write your answers in the space provided on the exam paper. If you unstaple your exam, please put your initials on all pages. You may use the back of pages if necessary, but please indicate if you do so we know where to look for your solution. You may ask us for additional pages of scratch paper. You must submit all sheets you use with your exam. However, we will not grade what you scribble on your scratch paper unless you indicate you want us to do so. Answers will be graded on correctness and clarity. You may lose points if your solution is more complicated than necessary or if you provide extraneous, but incorrect information along with a correct solution.

Name (printed) _____

I accept the letter and the spirit of the Virginia Tech graduate honor code – I have not given or received aid on this exam.

(signed) _____

#	Problem	Points	Score
1	Virtual Memory	30	
2	Symbolic Links	16	
3	File Systems	18	
4	Networking	16	
5	Short Questions	20	
	Total	100	

1 Virtual Memory (30pts)

a) (6 pts) The following is an excerpt from the NetBSD man page for the sysctl(3) function, which allows retrieval and manipulation of named system parameters:

```
#include <sys/param.h>
#include <sys/sysctl.h>

int
sysctl(int *name, u_int namelen, void *oldp, size_t *oldlenp,
          void *newp, size_t newlen);
```

```
 The sysctl function retrieves system information and allows processes
with appropriate privileges to set system information. […]

 Unless explicitly noted below, sysctl returns a consistent snapshot of
the data requested.  Consistency is obtained by locking the destination
buffer into memory so that the data may be copied out without blocking.
```

On Apr 12, 2006, the NetBSD Team issued an security advisory regarding sysctl(3), part of which is reproduced here:

```
                NetBSD Security Advisory 2006-013
                ================================

Topic:          sysctl(3) local denial of service

Severity:       Any local user can crash the system

Abstract
========

The user supplied buffer where results of the sysctl(3) call are stored
is locked into physical memory without checking its size.  This way, a
malicious user can cause a system lockup by allocating all available
physical memory on most systems.

Technical Details
=================

The system call implementing the sysctl(3) library call tries to lock
the user supplied result buffer into physical memory, to avoid the
interferences of information collection with other system activity.
The size of that buffer is not checked against system limits.

The VM system checks whether the virtual address of the buffer is part
of the user address space, but since the amount of virtual memory a
single user is able to allocate exceeds the available physical memory
on most systems, a user can cause a system lockup by exhaustion of
physical memory. […]
```

Give a short C program that could exploit this vulnerability. You may assume that NetBSD uses demand paging throughout. Ignore those arguments to sysctl(3) that do not play a role in this vulnerability. (*put your answer on the next page)*

(space for answer to 1a)

b) (4 pts) In Project 3, if you implemented eviction, a situation could occur where a page frame that was in the process of being evicted to swap disk would be accessed by the thread that still owned it. Handling this case properly required that you either aborted the eviction in this case, or that the owning thread waited until the eviction was complete, then immediately faulted the same page into a new page frame.

An easier approach would have been to use a lock that is taken on the entry to the page fault handler code (in exception.c). The same lock would also need to be taken whenever a page is being evicted and it would be released afterwards.

Explain the drawback of this approach!

c) (5 pts) Many implementations of Pintos used code similar to this one to check the validity of user pointers:

```
static void* resolve_user_address(void *uaddr, struct thread *t,
                                  uint32_t sz)
{
   if(uaddr == NULL)
    return NULL;

  /* Check to see if uaddr is in the user uaddr space */
  if(!is_user_vaddr(uaddr))
    return NULL;

  /* Get the kernel address for the mapping */
  if((kaddr = pagedir_get_page(t->pagedir, uaddr)) == NULL)
    return NULL;
  … // rest omitted
}
```

i. (2 pts) Explain why the check `if(uaddr == NULL)` is not necessary!

ii. (3 pts) Even though it is not necessary, is it *useful* to have that check? Justify your answer!

d) (3 pts) Assume all allocation requests are multiples of PGSIZE. Under this assumption, does Pintos's palloc() allocator suffer from internal fragmentation? Say why or why not.

e) (3 pts) Under the same assumption, can external fragmentation occur?

f) (5 pts) Consider the following program:

```
001: #include <stdlib.h>
002:
003: int **
004: allocate_matrix(int n)
005: {
006:     int i, ** m = calloc(n, sizeof(int *));
007:     for (i = 0; i < n; i++) {
008:         m[i] = calloc(n, sizeof(int));
009:     }
010:     return m;
011: }
012:
013: int
014: main(int ac, char *av[])
015: {
016:     int i, j, n = 3000;
017:     int **m1 = allocate_matrix(n);
018:     int **m2 = allocate_matrix(n);
019:
020:     for (i = 0; i < n; i++)
021:         for (j = 0; j < n; j++)
022:             m2[j][i] += m1[j][i];
023: }
```

When run on a PowerMac G5 with 6.5 GB of physical memory, this program takes about 5 seconds to finish. If I change line 22 to read

```
022:             m2[i][j] += m1[i][j];
```

then the program finishes in about 0.36 seconds.

Explain a possible reason for this behavior!

g) (4 pts) In lecture we had discussed the similarity of buffer caching to paged virtual memory. Which bit in a hardware page table entry corresponds to the “valid” bit in a buffer cache descriptor?

2 Symbolic Links (16 pts)

Symbolic links in Unix are shortcuts to other files. They can be created with the command *ln –s*. Here are examples to remind you of how they work:

```
% mkdir a
% echo hello > a/b
% ln –s a/b c
% cat c
hello
% pwd
/Users/gback/CS3204/final
% ln –s /Users/gback/CS3204/final/a/b d
% cat d
hello
% ls -l
total 16
drwxr-xr-x   3 gback  gback  102 May  4 23:20 a
lrwxr-xr-x   1 gback  gback    3 May  4 23:20 c -> a/b
lrwxr-xr-x   1 gback  gback   29 May  4 23:20 d -> /Users/gback/CS3204/final/a/b
```

Symbolic links are implemented using the symlink(2) system call, which is defined as follows:

```
#include <unistd.h>
int symlink(const char *oldpath, const char *newpath);
```

Suppose you were to add support for symbolic links to your Project 4 implementation of Pintos (if you did not complete Project 4, answer this problem as if you had completed it.)

a) (6 pts) Name *two* changes you would have to make to the code in the lib/user/* and the userprog/ directory!

b) (5 pts) How would you change the on-disk layout of your filesystem to store symbolic links persistently? Be specific!
Note: the “oldpath” argument can point to a pathname up to PATH_MAX characters in length. filesys/directory.h defines PATH_MAX to be 1024.

c) (5 pts) Explain how your path resolution routine would have to be modified in order to accommodate symlinks. Discuss both relative and absolute paths.

3 File Systems (18 pts)

a) (4 pts) Ignoring directory lookup, in an indexed filesystem, accessing even a short 20-byte file takes at least two disk accesses: one access to retrieve the inode from disk, and a second access to retrieve the file's data. Explain how you could reduce this number to 1 disk access for such short files.

b) (6 pts) Name one performance problem with your filesystem and/or buffer cache design in Project 4. Be specific. For this performance problem, describe a workload/scenario that would trigger the problem. Name a technique that could alleviate the problem.

i. (2 pts) Workload/Scenario:

ii. (2 pts) Performance Problem:

iii. (2 pts) Possible Countermeasure:

c) (8 pts) fsck for Unix implements inode recovery - orphaned inodes after a crash are listed in a special directory /lost+found.

i. (4 pts) What are orphaned inodes and how can they occur?

ii. (4 pts) Say you were to write an fsck program for your version of Pintos. Could you implement inode recovery? Say why or why not. State your assumptions if necessary.

4 Networking & Operating Systems (16 pts)

a) (10 pts) Suppose a host uses a layered network architecture. If a packet is received by layer k-1, it must be processed and eventually passed up to layer k. Write *pseudo code* that could be used by layers k-1 & k for this purpose. Be sure to include all necessary synchronization, but do not worry about memory management.

// Layer k calls “receive()” to receive a packet	// Layer k-1 calls “Deliver()” when a packet is available
// Declare necessary data structures here, if any	

Packet receive() {
 	
}
	Deliver(Packet p) {
	
	}

b) (6 pts) It has been proposed to harness unused RAM to implement remote paging, where a page is evicted not to disk, but sent to another machine on the same network. Name one potential advantage and one potential disadvantage/complication of this idea!

i. (3 pts) Advantage

ii. (3 pts) Disadvantage

5 Short Questions (20 pts)

a) (5 pts) When viewed as a protection system, does the MMU of the 80x86 series of processors use an access-control-list or a capability-based approach? Justify your answer!

b) (5 pts) What problem occurs when implementing stack growth on a system that supports multiple kernel-level threads per process?

c) (5 pts) What is the “small write” problem in a RAID-5 system?

d) (5 pts) Why is LRU usually not the best cache eviction strategy for a buffer cache?