**COMPSCI 453/552: Operating Systems**

**Final Examination** (Tuesday, 16th December)

*Time: 110 minutes    Name:__________________    Total Points: 150*

Please read the following carefully.

- You get 3 points for writing your name above (because my cell phone went off in the class around six times!).

- This exam has 9 questions, for a total of 165 points.

- The last problem is extra credit for undergraduates and required for graduate students. The score for graduate students will be scaled down to 150 points.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>165</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. **Bit Byte Virtual Drudge Fudge.** Suppose we have a computer system with a 48-bit virtual address, 32-bit physical address and a page size of 8K.

   (a) (5 points) What is the size of a page frame?

   (b) (5 points) How many page frames are there in the physical address?

   (c) (5 points) How many pages are there in the virtual address?

   (d) (10 points) Show how the 48-bit virtual address gets mapped into the 32-bit physical address by drawing out a virtual address and specifying which bits are used to index the page table and which bits are used to determine the offset into a page?
2. (15 points) Consider the following C fragment that allocates some memory.

```c
struct job {
    struct job *next;
    struct job *prev;
    char *name;
    char **parameters;
    struct status *job_status;
};

struct job_status {
    int state;
    struct event *pending_events;
    struct time_used time;
}

struct job *newjob;

newjob = (struct job *) malloc(sizeof(struct job));
newjob->next = nil;
newjob->prev = nil;
newjob->name = (char *) malloc(sizeof(char)*MAXLENGTH);
newjob->parameters = (char **) malloc(sizeof(char *)*MAX_PARAMETERS);
for (i=0; i<MAX_PARAMETERS; i++)
    newjob->parameters[i] = (char *) malloc(sizeof(char)*MAXLENGTH);
newjob->job_status = (struct job_status *) malloc(sizeof(struct status));
newjob->job_status->state = READY;
newjob->job_status->pending_events =
    (struct event *) malloc(sizeof(struct event)*MAX_NUM_EVENTS);
```

Write a C code fragment that frees the memory properly.
3. (15 points) Memory allocation. The following code shows how to dynamically allocate a two-dimensional $n \times n$ array in C.

```c
int **array;
array = (int **) malloc(n * sizeof(int *));
for (i=0; i<n; i++)
    array[i] = (int *) malloc(n * sizeof(int));
```

The following code also allocates a two-dimensional $n \times n$ array in C. How is the memory layout different from the above version? Why would it be useful?

```c
int **array;
int *ptr;

ptr = (int *) malloc(sizeof(int) * n * n);
array = (int **) malloc(sizeof(int *) * n);
for (i=0; i<n; i++)
    array[i] = ptr + i * n;
```
4. **(25 points) Monitors in C** Consider the following prototype of a hash table in C. We will assume that it is implemented using hashing with chaining, where each bucket is represented as a linked list. Also assume that a thread-safe implementation of linked lists is available. Explain how, using monitor style coding, you would make the hash table code be thread-safe.

```c
#ifndef __HASHTABLE_H
#define __HASHTABLE_H

#include <stdlib.h>
#include <List.h>
#include "HashObject.h"

typedef struct hashtable Hashtable;
typedef struct hashtable * HashtablePtr;

struct hashtable {
    int size;
    unsigned int M;
    unsigned int (*h)(unsigned int M, unsigned int key);
    unsigned int (*getKey) (void * obj);
    ListPtr *table;
    void (*freeHashtable)(HashtablePtr table);
    void (*insert)(HashtablePtr table, HashObject *obj);
    HashObjectPtr (*remove)(HashtablePtr table, unsigned int key);
    HashObjectPtr (*search)(HashtablePtr table, unsigned int key);
};

HashtablePtr createhashTable(unsigned int M,
                               unsigned int (*h)(unsigned int, unsigned int),
                               unsigned int (*getKey)(void * obj));

#endif /* __HASHTABLE_H */
```
5. **(15 points) Ext3 File system.** The following fragment of code is from /usr/include/linux/ext3_fs.h in the Linux kernel source code.

```c
#define EXT3_MIN_BLOCK_SIZE 1024
#define EXT3_MAX_BLOCK_SIZE 4096

/*
 * Constants relative to the data blocks
 */
#define EXT3_NDIR_BLOCKS 12
#define EXT3_IND_BLOCK EXT3_NDIR_BLOCKS
#define EXT3_DIND_BLOCK (EXT3_IND_BLOCK + 1)
#define EXT3_TIND_BLOCK (EXT3_DIND_BLOCK + 1)
#define EXT3_N_BLOCKS (EXT3_TIND_BLOCK + 1)

/*
 * Structure of an inode on the disk
 */
struct ext3_inode {
    ...
    __u32   i_block[EXT3_N_BLOCKS];/* Pointers to blocks */
    ...
}
```

Answer the following questions for the ext3 file system assuming a block size of 4096 bytes.

- What is the maximum size of a file using only direct pointers?

- What is the maximum size of a file using direct and single indirect pointers?

- What is the maximum size of a file using direct, single, double indirect pointers?

- What is the maximum size of a file using direct, single, double, and triple indirect pointers?
6. **(25 points) Mutithreaded Network Server.** Rewrite the following multithreaded network server code to be multi-threaded. Please don’t rewrite the whole code! Just rewrite the parts that will change.

```c
/* appropriate header files */
const char MESSAGE[] = "Hello World\n";
const int BACK_LOG = 5;
int main(int argc, char **argv) {
    int serverSocket = 0, on = 0, port = 0, status = 0, childPid = 0;
    struct hostent *hostPtr = NULL;
    char *machine;
    struct sockaddr_in serverName;
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <hostname> <port>\n", argv[0]);
        exit(1);
    }
    machine = argv[1];
    port = atoi(argv[2]);
    serverSocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (serverSocket == -1) {
        perror("socket()");
        exit(1);
    }
    on = 1;
    status = setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, (char *)&on, sizeof(on));
    if (status == -1) {
        perror("setsockopt(...,SO_REUSEADDR, ...)");
        exit(1);
    }
    /* When connection is closed, linger a bit to ensure all data has arrived. */
    { struct linger linger = { 0 };
        linger.l_onoff = 1;
        linger.l_linger = 30;
        status = setsockopt(serverSocket, SOL_SOCKET, SO_LINGER, &linger, sizeof(linger));
        if (status == -1) {
            perror("setsockopt(...,SO_REUSEADDR, ...)");
            exit(1);
        }
    }
    hostPtr = gethostbyname(machine);
    if (hostPtr == NULL) {
        perror("gethostbyname()" );
        exit(1);
    }
    memset(&serverName, 0, sizeof(serverName));
    memcpy(&serverName.sin_addr, hostPtr->h_addr, hostPtr->h_length);
    ... (remaining code)
}
```
serverName.sin_family = AF_INET;
serverName.sin_port = htons(port);
status = bind(serverSocket, (struct sockaddr *) &serverName, sizeof(serverName));
if (status == -1) {
    perror("bind()" );
    exit(1);
} else {
    fprintf(stderr, "%s: server bound to port %d\n", argv[0], port);
}
status = listen(serverSocket, BACK_LOG);
if (status == -1) {
    perror("listen() ");
    exit(1);
}
while (1) {
    struct sockaddr_in clientName = { 0 };
    int slaveSocket; 
    unsigned int clientLength = sizeof(clientName);
    memset(&clientName, 0,sizeof(clientName));
    slaveSocket = accept(serverSocket, (struct sockaddr *) &clientName, &clientLength);
    if (slaveSocket == -1) err_quit("accept():");
    childPid = fork();
    switch(childPid) {
    case -1: /* fork failed */
        err_quit("fork() ");
    case 0: /* in child */
        close(serverSocket);
        if (-1 == getpeername(slaveSocket, &clientName, &clientLength))
            err_quit("getpeername() ");
        else {
            printf("Connection request from %s\n", inet_ntoa(clientName.sin_addr));
        }
    /* Server application specific code goes here. */
    write(slaveSocket, MESSAGE, strlen(MESSAGE));
    close(slaveSocket);
    exit(0);
    default: /* in parent */
        close(slaveSocket);
    }
    exit(0);
}
7. (15 points) File protection bits, Access Control Lists, Capability Lists, and Encryption are different methods for controlling access to files. Suppose we have a file that a given user has access to currently. For each of the four methods above explain how (if possible at all) to accomplish the following:

- Remove access for a process (owned by the user) that has the file open while the process is running.

- Forbid future access for the file to a specific user out of a group of users.

- Forbid future access to even the system administrator (a.k.a. superuser).
8. **(15 points) Code Insecurity Therapy.** Find all potential security problems in the following piece of code. Examples would be buffer overflows, writing before the start of an array etc.

```c
int main(int argc, char **argv)
{
    char buffer[128];
    char *input1;
    char input2[128];

    scanf("%s", input1);
    strcpy(buffer, input1);
    gets(input2);
    if strcmp(buffer, input2) {
        /* give access to all secrets */
    }
}
```
For Graduate Students (extra credit for undergrads)

9. **Collusion in Hacking.** Suppose you have two processes, say A and B, running on a “secure system.” Suppose process A accesses some strategic information (let’s say the password to the electronic cash account for The Billionaire.) The system doesn’t allow process A to create pipes, named pipes, sockets, send messages, shared memory, open/access any files (except authorized files which the process B cannot open or detect that A has opened/accessed), create any semaphores, fork off a child process, create a new thread, or receive a user defined signal. The hapless process A can, however, send a user-defined signal (*only one kind*). Assume, for the sake of this problem, that the processes are running under a UNIX like system, although the scenario is valid for any operating system.

(a) (5 points) Describe how process A can transmit the password to the process B. (the more concrete your solution, the higher the grade).

(b) (10 points) Suppose now the process A is not allowed to send any signals at all. That is, no direct communication between A and B is allowed. Can process A still transmit the information to process B? Explain your answer.