Andrew login ID: _____

Full Name: _____

CS 15-213, Spring 2002

Final Exam

May 9, 2002

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and Andrew login ID on the front.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of ___ points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. You may use a calculator, but no laptops or other wireless devices. Good luck!

1 (7):	
2 (9):	
3 (8):	
4 (8):	
5 (15):	
6 (10):	
7 (9):	
8 (5):	
9 (8):	
10 (9):	
TOTAL (86):	

Problem 1. (7 points):

In this problem, you will complete a function that converts `float` to `int` without explicit use of a conversion operator.
The following information may prove useful.

- The IEEE `float` type uses 1 bit for sign, 8 bits for the exponent (with a bias of 127), and 23 bits for the fraction.
- Your function should truncate floating pointing numbers (*i.e.*, round toward zero). For example: $1.8 \rightarrow 1$, $-3.1 \rightarrow -3$.
- Since you are writing the conversion function, you may not use the built-in type conversion facilities of C. You may not use relational operators (<, ==, and so on) taking floating point arguments. Keep in mind that C does not allow the use of bitwise operators on floating point types.
- In the event of overflow or infinity, you should return the largest (`INT_MAX`) or smallest (`INT_MIN`) representable integer, as appropriate.
- NaN (not a number) should be converted to 0.

The following is the framework for the conversion function. Fill in the blank lines with an appropriate C expression.

```
union conv {
  float f;
  unsigned long u;
};

int float2int(float f)
{
  union conv conv;
  unsigned long u;
  int sign,exp,frac,shift;

  conv.f = f;
  u = conv.u;

  if ( ____________________ )
    sign = -1;
  else
    sign = 1;
  exp = ____________________________ ;
  frac = ____________________________ ;

  if ( ____________________ ) /* zero or denormalized */
    return 0;
  if ( ____________________ ) {
    if ( ____________________ ) /* NaN */
      return 0;
    else if (sign > 0) /* +Inf */
      return INT_MAX;
    else
      return INT_MIN;
  }
```

```
  /* Add implicit 1.x in normalized representation */
  frac |= 1 << 23;

  /* compute decimal point position, i.e., total right shift needed */
  shift = _____________________ ;

  if (shift > 0) {
    if (shift > 32)
      return 0;
    else
      return sign * (frac >> shift);
  } else {
    if (-shift > 32) {
      if (sign > 0)
        return INT_MAX;
      else
        return INT_MIN;
    }
    return sign * (frac << -shift);
  }
}
```

Problem 2. (9 points):

Part 1

Given the assembly for the function **mystery1**, fill in the corresponding function in C.

```
<mystery1>:
  push   %ebp
  mov    %esp,%ebp
  push   %ebx
  sub    $0x18,%esp
  movl   $0x0,%ecx
  movl   $0x0,%ebx
.L1
  cmp    0xc(%ebp),%ebx
  jl     .L2
  jmp    .L3
.L2
  mov    0x8(%ebp),%eax
  mov    (%eax,%ebx,4),%edx
  add    %edx, %ecx
  incl   %ebx
  jmp    .L1
.L3
  mov    %ecx,%eax
  pop    %ebx
  mov    %ebp,%esp
  pop    %ebp
  ret

int mystery1(int A[], int n) {
  int i;

  int mystery = _______; % answer: 0

  for (_____________________________________) % answer: i = 0; i < n; i++
  {
       _____________________________________; % answer: mystery += A[i];
  }

  return(_________________); % answer: mystery
}
```

Part 2

```
<mystery2>:
  push   %ebp
  mov    %esp,%ebp
  sub    $0x18,%esp
  mov    0x8(%ebp),%edx
  mov    0xc(%ebp), %ecx
  cmp    %ecx, %edx
  jge    .L1
  add    $0xfffffff8,%esp
  sub    %edx,%ecx
  push   %ecx
  push   %edx
  call   mystery2
  add    $0x10,%esp
  jmp    .L3
.L1
  cmp    %ecx, %edx
  jle    .L2
  add    $0xfffffff8,%esp
  push   %ecx
  sub    %ecx,%edx
  push   %edx
  call   mystery2
  add    $0x10,%esp
  jmp    .L3
.L2
  mov    %edx,%eax
.L3
  mov    %ebp,%esp
  pop    %ebp
  ret
```

A. What would the following function call return?

```
x = mystery2(6, 4);
x = _____________________________________  % answer: 2
```

B. What is the mystery2 function computing?

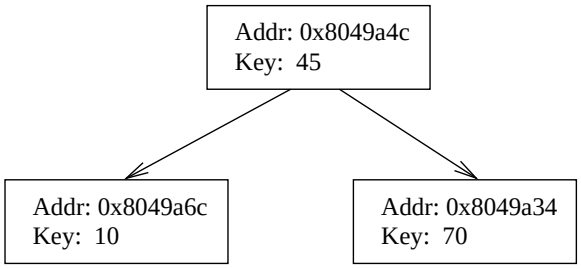
Problem 3. (8 points):

This question is testing your understanding of the stack frame structure.

Part I The following memory image embeds a binary tree with root node at `0x804961c`. Please draw the logical organization of the tree in the same format as the shown example. Please indicate the address and key value (in hexidecimal) of all the tree nodes and the pointers from parent nodes to child nodes. The declaration of the tree node structure is as follows.

```
struct tree_node {
    int   key;
    struct tree_node * left;
    struct tree_Node * right;
};

/* address of the root node */
tree_node * root;
```



Example binary tree

```
Memory image:
<address>   <value>
0x80495f8:  0x0000000c
0x80495fc:  0x00000000
0x8049600:  0x00000000
0x8049604:  0x0000001f
0x8049608:  0x080495f8
0x804960c:  0x08049610
0x8049610:  0x00000022
0x8049614:  0x00000000
0x8049618:  0x00000000
0x804961c:  0x00000037
0x8049620:  0x08049604
0x8049624:  0x08049628
0x8049628:  0x0000003c
0x804962c:  0x00000000
0x8049630:  0x08049634
0x8049634:  0x0000004e
0x8049638:  0x00000000
0x804963c:  0x00000000
...
```

Part II The following function traverses the binary tree to locate the node with a given key value.

```
 1: struct tree_node * search(struct tree_node * node,
 2:                           int value)
 3: {
 4:     if (node->key == value)
 5:         return node;
 6:     else if (node->key > value) {
 7:         if (node->left == NULL)
 8:             return NULL;
 9:         else return search(node->left, value);
10:     }
11:     else {
12:         if (node->right == NULL)
13:             return NULL;
14:         else return search(node->right, value);
15:     }
16: }
```

Suppose we call `search(root, 0x4e)`. Fill in the blanks the value of these memory location so that it shows the stack when the execution is at line 5. (More space than needed is provided.) You can assume that the stack stores only arguments, return address, and the `ebp` register value. The value of `ebp` is `0xbffff880` when the program calls the function. Write "rtn_addr" for return addresses.

Address	Value
0xbffff800	0x4e
0xbffff7fc	0x804961c
0xbffff7f8	rtn_addr
0xbffff7f4	
0xbffff7f0	
0xbffff7ec	
0xbffff7e8	
0xbffff7e4	
0xbffff7e0	
0xbffff7dc	
0xbffff7d8	
0xbffff7d4	
0xbffff7d0	
0xbffff7cc	
0xbffff7c8	
0xbffff7c4	

Problem 4. (8 points):

In this problem, you will compare the performance of direct mapped and 4-way associative caches for the initialization of 2-dimensional array of data structures. Both caches have a size of 2048 bytes. The direct mapped cache has 128-byte lines while the 4-way associative cache has 32-byte lines.

You are given the following definitions

```
typedef struct{
    float* position;
    float  velocity[3];
    float  forces[3];
    particle_t *adjacent;
} particle_t;

particle_t  cloth[32][32];
register int i, j, k;
```

Also assume that

- `sizeof(* float) and sizeof(* particle_t) = 4`
- `sizeof(float) = 4`
- `surface` begins at memory address `0`
- Both caches are initially empty
- The array is stored in row-major order
- Variables `i`,`j`,`k` are stored in registers and any access to these variables does not cause a cache miss

A. What fraction of the writes in the following code will result in a miss in the direct mapped cache?

```
for (i = 0; i < 32; i ++)
{
    for (j = 0; j < 32; j ++)
    {
        for(k = 0; k < 3; k ++)
        {
            cloth[i][j].forces[k] = 0.;
        }
    }
}
```

Miss rate for writes to surface:_____%

B. Using code in part A, what fraction of the writes will result in a miss in the 4-way associative cache?

Miss rate for writes to surface: _____%

C. What fraction of the writes in the following code will result in a miss in the direct mapped cache?

```
for (i = 0; i < 32; i ++)
{
    for (j = 0; j < 32; j ++)
    {
        for (k = 0; k < 16; k ++)
        {
            cloth[j][i].forces[k] = 0;
        }
    }
}
```

Miss rate for writes to surface:_____%

D. Using code in part C, what fraction of the writes will result in a miss in the 4-way associative cache?

Miss rate for writes to surface:_____%

Problem 5. (15 points):

The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable, and memory accesses are to 1-byte {**not 4-byte**} words.
- Virtual addresses are 18 bits wide.
- Physical addresses are 12 bits wide.
- The page size is 512 bytes.
- The TLB is 8-way set associative with 16 total entries.
- The cache is 2-way set associative, with a 4-byte line size and 32 total entries.

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB and the page table for the first 32 pages, and the cache are as follows:

TLB			
Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1

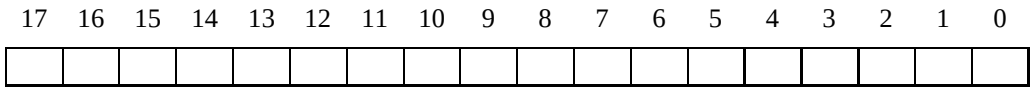
Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
000	7	0	010	1	0
001	5	0	011	3	0
002	1	1	012	3	0
003	5	0	013	0	0
004	0	0	014	6	1
005	5	0	015	5	0
006	2	0	016	7	0
007	4	1	017	2	1
008	7	0	018	0	0
009	2	0	019	2	0
00A	3	0	01A	1	0
00B	0	0	01B	3	0
00C	0	0	01C	2	0
00D	3	0	01D	7	0
00E	4	0	01E	5	1
00F	7	1	01F	0	0

2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	7A	1	09	EE	12	64	00	0	99	04	03	48
1	02	0	60	17	18	19	7F	1	FF	BC	0B	37
2	55	1	30	EB	C2	0D	0B	0	8F	E2	05	BD
3	07	1	03	04	05	06	5D	1	7A	08	03	22

Part 1

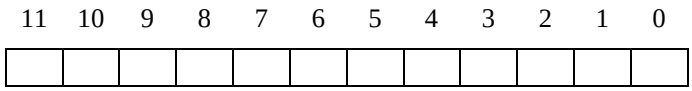
1. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

   VPO The virtual page offset
   VPN The virtual page number
   TLBI The TLB index
   TLBT The TLB tag



2. The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

   PPO The physical page offset
   PPN The physical page number
   CO The Cache Block Offset
   CI The Cache Index
   CT The Cache Tag



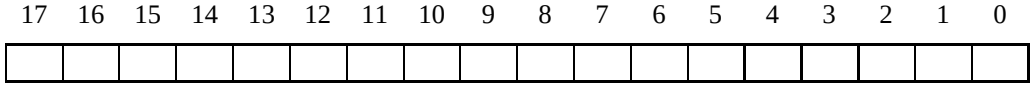
Part 2

For the given virtual addresses, indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs.

If there is a cache miss, enter "-" for "Cache Byte Returned." If there is a page fault, enter "-" for "PPN" and leave part C blank.

Virtual address: `0x1A9F4`

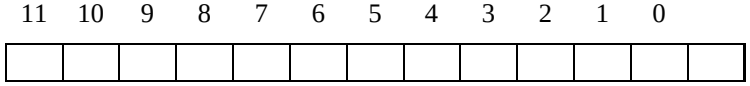
1. Virtual address format (one bit per box)



2. Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

3. Physical address format (one bit per box)



4. Physical memory reference

Parameter	Value
Block Offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Value of Cache Byte Returned	0x

Problem 6. (10 points):

This question deals with various aspects of the implementation of dynamic memory allocators.

Memory utilization

Suppose your memory allocator uses a simple implicit list, and each block in the heap has a 1 word header and footer. Assume that the heap has a unused word at the start of the heap to enforce double-word alignment, followed by a prologue block consisting of only a header and footer. At the end of the heap is a 1 word epilogue header. The allocator's minimum allocation unit is 8 bytes. The total size of the heap is 2048 bytes.

A. If the heap is full (this means a request of `malloc(1)` will fail), what is worst case memory utilization of the heap?

B. Briefly (no more than 2 sentences), describe the difference between internal and external fragmentation.

Free list strategies

Consider a heap with N blocks, where $M \leq N$ are allocated. For the following questions, express your answer in O-notation:

A. What is the worst case running time of first fit allocation with explicit free lists?

B. What is the worst case running time of first fit allocation with implicit free lists?

C. What is the worst case running time of best fit allocation with explicit free lists?

Coalescing strategies

Immediate coalescing is the strategy we have seen where a block is coalesced with its neighbor(s) immediately after it is freed. Another possible strategy is *lazy* or deferred coalescing, where free blocks are not coalesced immediately. Usually, coalescing is done when a allocation request cannot be fulfilled.

For the following two questions, express your answer in O-notation. Borrowing from the previous section, there are N blocks in the heap, where $M \leq N$ are allocated. Assume the allocator uses some form of explicit free lists.

A. What is the worst case running time of coalescing using an immediate strategy?

B. What is the worst case running time of coalescing using a lazy strategy?

For the following situations, describe whether an allocator using immediate or lazy coalesing (coalescing done when an allocation request fails) would be more appropriate. If the choice of coalescing strategy has negligible performance impact, write "does not matter".

D. The allocator allocates only two types of structures that have fixed sizes, and expects heavy reuse patterns.

E. The allocator must operate in real time. This means the allocator makes hard performance guarantees that its operations will not take more than some fixed specified amount of time.

F. The allocator will alternately allocate and free blocks where the next allocation request is 1/2 the size of the last request.

Problem 7. (9 points):

Part 1

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
#include <stdio.h>
#include <pthread.h>

#define NTHREADS 20

void *thread(void *vargp)
{
  static int cnt = 0;

  cnt++;
  printf("%d\n", cnt);
}

int main ()
{
  int i;
  pthread_t tid;

  for(i = 0; i < NTHREADS; i++)
  {
    pthread_create(&tid, NULL, thread, NULL);
  }

  pthread_exit(NULL);

}
```

What are the maximal guarantees you can make about the output of the above program? Check all that apply. Note: Checking all of the boxes implies that the output is

```
1
2
3
...
20
```

☐ 20 numbers will be printed.
☐ The numbers lie in the range 1 through 20, inclusive.
☐ There are no duplicate numbers printed.
☐ The numbers will be printed in ascending order.

Part 2

Using mutexes, modify the code to guarantee that the output of the program is

```
1
2
3
...
20
```

The numbers 1 through 20 printed sequentially in ascending order.
Here are the mutex operations:

```
int pthread_mutex_init(pthread_mutex *mutex, const pthread_mutexattr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex));
int pthread_mutex_unlock(pthread_mutex_t *mutex));
int pthread_mutex_destroy(pthread_mutex_t *mutex));
```

Here is the code from the previous page. Make the appropriate additions.

```
#include <stdio.h>
#include <pthread.h>

#define NTHREADS 20

% answer: pthread_mutex mutex;

void *thread(void *vargp)
{

  static int cnt = 0;

  % answer: pthread_mutex_lock(&mutex);

  cnt++;

  printf("%d\n", cnt);

  % answer: pthread_mutex_unlock(&mutex);

}

int main ()
{
  int i;

  pthread_t tid;

  % pthread_mutex_init(&mutex, NULL);

  for(i = 0; i < NTHREADS; i++)
  {

    pthread_create(&tid, NULL, thread, NULL);

  }
```

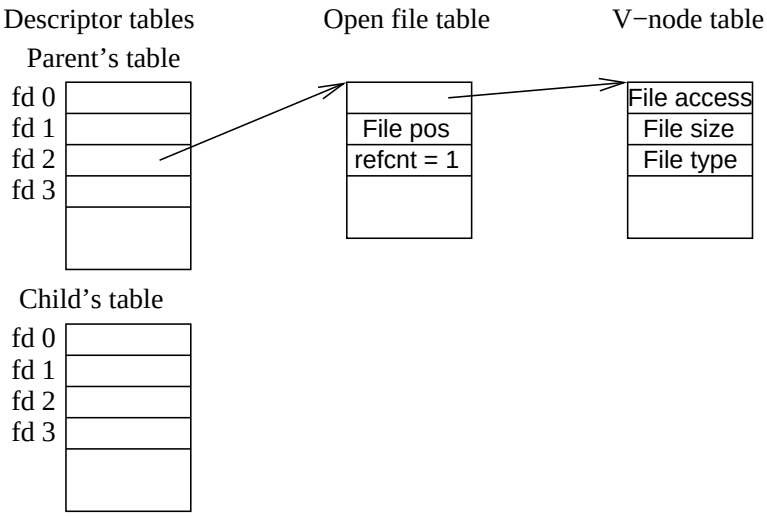
Problem 8. (5 points):

This problem tests your understanding of the file sharing between processes. Consider the following program.

```
 1: int  main()
 2: {
 3:    int fd_x, fd_y, fd_z;
 4:
 5:    fd_x = Open("file1", O_RDONLY);
 6:    if (Fork() == 0) {
 7:        fd_y = Open("file1", O_RDONLY);
 8:        ...
 9:    }
10:    else {
11:        fd_z = Open("file2", O_RDONLY);
12:        Dup2(fd_z, fd_x);
13:     }
14:    ...
15: }
```

The following graph shows the kernel data structure after line 5 is executed. Please draw the kernel data structure when the child process execution stops at line 8 and the parent stops at line 13. Add additional entries to the tables if needed. Remember to update the refcnt field in the open file table. You can assume that

1. there is no error in opening the files, and
2. the value of `fd_x`, `fd_y`, and `fd_z` are 2, 3, 3 respectively.



Problem 9. (8 points):

Part 1

You have just been hired by a software company to complete some database software. You've been assigned the task of writing the networking code that will allow server and client software to communicate. The database software is designed to handle many simultaneous connections.
Both the server and client's networking code need to be written. The server will be hosted on `DEFAULT_PORT` and it does not care which IP address on which it's to be hosted. The connection backlog should be 10.
At the moment, you will only be testing the client's code. Therefore, the client will only connect to the local host (`127.0.0.1`) while testing.
The following function declarations may prove useful.

- `int bind(int sockfd, struct sockaddr *addr, socklen_t addrlen)`
- `int connect(int sockfd, struct sockaddr *addr, socklen_t addrlen)`
- `int socket(int domain, int type, int protocol)`
- `int listen(int s, int backlog)`
- `int accept(int s, struct sockaddr *addr, socklen_t addrlen)`
- `unsigned long int htonl(unsigned long int hostlong)`
- `unsigned short int htons(unsigned short int hostshort)`

A

The following is the framework for the server code you are required to write. Note that the function returns a socket descriptor. Assume there are no errors and no error handling is needed.

```
int setup_server()
{
  int fd;
  struct sockaddr_in addr;
  int len = sizeof(struct sockaddr);


  fd = ____________________________________________;

  bzero(&addr, len);
  addr.sin_family = AF_INET;


  addr.sin_port   = ______________________________;


  addr.sin_addr.s_addr = _________________________;


  ________________________________________________;


  ________________________________________________;


  ________________________________________________;
}
```

B

The following is your framework for the client code. Note that it also returns a socket descriptor. Again, no error handling is needed.

```
int client_setup()
{
  struct sockaddr addr;
  int fd;

  fd = ____________________________________________;

  addr.sin_family = AF_INET;


  addr.sin_port   = ______________________________;


  addr.sin_addr.s_addr = 0x_____________________;


  ____________________________________________________;


  ____________________________________________________;
}
```

Part 2

Fill in a valid time ordering for the networking functions called by both the client and the server. Use the functions listed below. Some may be used more than once. **Only enter one per numbered line. No numbered line is to be left blank**

- bind
- socket
- connect
- listen
- accept

	Client	Server
1	_____	_____
2	_____	_____
3	_____	_____
4	_____	_____
5	_____	_____
6	_____	_____

Problem 10. (9 points):

You now need to demonstrate your understanding of concurrency across multiple ways of implementing it. Given a concurrent program using threads, fill in the key blanks of a concurrent program using `select()` so that it accomplishes the same goal.

The following simple program echos the text entered onto 10 different terminals to the screen one character at a time. The code for you to complete is on the following page.

```
int main ( int argc, char *argv[] )
{
    int j;
    for ( j = 0; j < 10; j++ )
    {
        pthread_t pthread;
        int* fd = malloc(sizeof(int));

        /* Get a file descriptor for a terminal.
         * Don't worry about how.
         */
        *fd = getNextTerminalFD();

        pthread_create ( &pthread, NULL, thread, fd );
    }

    while (1) {}
}

void thread ( void *fd )
{
    int file_d = *(int*)fd;
    while (1)
    {
        printf ( ''%c '', getc(file_d) );
    }
}
```

```
int main ( int argc, char *argv[] )
{
    int j;
    int fd_list[10];
    fd_set read_set, ready_set;

    for ( j = 0; j < 10; j++)
    {
        fd_list[j] = getNextTerminalFD();


        ____________________________________________________________;
    }

    while (1)
    {


        ____________________________________________________________;

        for ( j = 0; j < 10; j++ )
        {

            if ( ____________________________________________________ )
            {
                printf ( ‘‘%c ‘‘, getc(fd_list[j]) );
            }
        }
    }
}
```