Andrew login ID:______________________________________

Full Name:______________________________________

CS 15-213, Spring 2003

MAKEUP - Final Exam

May 7, 2003

Instructions:

- Make sure that your exam has 25 pages and 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 114 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 not use a calculator, laptop or any other electronic or wireless device. Good luck!

1 (9):	
2 (11):	
3 (10):	
4 (5):	
5 (12):	
6 (8):	
7 (25):	
8 (12):	
9 (15):	
10 (4):	
11 (6):	
12 (9):	
TOTAL (114):	

Problem 1. (9 points):

Assume we are running code on an 8-bit machine using two's complement arithmetic for signed integers. Short integers are encoded using 4 bits. Sign extension is performed whenever a `short` is cast to an `int`. For this problem, assume that all shift operations are arithmetic. All constants are `int`s. Fill in the empty boxes in the table below.

```
int i = -6;
unsigned ui = i;
short s = -7;
unsigned short us = s;
```

Note: TMax denotes the largest positive two's complement `int` and TMin denotes the minimum negative two's complement `int`. Please use only hex notation (i.e. not binary) for 'Hex Representation'.

Expression	Decimal Representation	Hex Representation
`27`	27	
`-16`	-16	
`i`		
`i >> 2`		
`ui`		
`(int) s`		
`(int)(6 ^ s)`		
`(int) us`		
TMax		
TMin		

Problem 2. (11 points):

Consider the following 7-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next 3 bits encode the exponent. The exponent bias is 3.
- The last 3 bits encode the significand.
- The representation encodes numbers of the form: $V = (-1)^s \times M \times 2^E$, where M is the significand and E the integer value of the exponent.
- The rounding mode is round-to-even.

Please fill in the table below. You do not have to fill in boxes with ”——” in them. **If a number is NAN or infinity, you may disregard the M, E, and V fields below.** However, fill the Description and Binary fields with valid data.
Here are some guidelines for each field:

- **Description -** A verbal description if the number has a special meaning
- **Binary -** Binary representation of the number
- M **-** Significand (same as the M in the formula above)
- E - Exponent (same as the E in 2^E)
- V - Fractional Value represented

Please fill the M, E, and V fields below with rational numbers (fractions) rather than decimals or binary decimals

Description	Binary	M	E	V
——	0 101 010			
1 $\frac{7}{8}$				
	1 111 111			
1 $\frac{5}{8}$ + 1				
Farthest from zero Negative **Normalized**				
Closest to zero Negative **Denormalized**				

Problem 3. (10 points):

This problem tests your understanding of assembly code, control flow and multidimensional array layout. Consider the following assembly code for a procedure `loopy`:

```
loopy:
    pushl   %ebp
    movl    %esp, %ebp
    pushl   %esi
    pushl   %ebx
    movl    8(%ebp), %ebx
    movl    $1, %edx
    movl    $0, %ecx
    cmpl    %ebx, %ecx
    jge     .L33
    movl    $arr, %esi
.L35:
    leal    0(,%ecx,8), %eax
    addl    %ecx, %eax
    imull   (%esi,%eax,4), %edx
    incl    %ecx
    cmpl    %ebx, %ecx
    jl      .L35
.L33:
    movl    %edx, %eax
    popl    %ebx
    popl    %esi
    popl    %ebp
    ret
```

A. For each register listed, indicate which C variable(s) from the C source on the next page (`i, tmp, arr, n` it can hold during the lifetime of the function. If the register is not used to hold a variable on the next page, then write in "none." Note that there is not necessarily a one-to-one correspondence between variables and registers.

`%eax` ____________

`%ebx` ____________

`%ecx` ____________

`%edx` ____________

`%esi` ____________

`%edi` ____________

`%ebp` ____________

(Question 3 cont'd)

B. Based on the assembly code, fill in the blanks below in `loopy`'s C source code. (Note: you may only use symbolic variables from the source code in your expressions below. Do *not* use register names.)

```
int arr[5][____];

int loopy(int n)
{
    int i;
    int tmp = ____;

    for(i = 0; __________; ______)
    {

        tmp = ___________________arr[____][0];

    }

    return _______;
}
```

Problem 4. (5 points):

Consider the following C declarations:

```
typedef struct
{
    union {
       char d;
       short s;
    } u;
    char c;
    double *dptr;
    char buf[2];
} final;
```

Using the templates below (allowing a maximum of 24 bytes), indicate the allocation of data for structs of type `final`. Mark off and label the areas for each individual element (arrays and unions may be labeled as a single element). **Cross hatch the parts that are allocated, but not used, and be sure to clearly indicate the end of the structure. Assume the Linux alignment rules discussed in class.**

```
final:

  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                                                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

What would the following program print out?

```
int main()
{
    printf("%d\n",sizeof(final));
}
```

Output:

Problem 5. (12 points):

In this problem, we will compare the best-case and worst-case scenarios for a memory access in a virtual memory subsystem. You will want to pay careful attention to the following information about the available features of the subsystem:

- No hardware caches (besides a Translation Lookaside Buffer) are present.
- The system uses a **3-Level Page Table**. The page directory (1st level page table) address is stored as a physical address in a special register on the processor. The page directory is always in main memory. All page tables store physical addresses.
- The **Translation Lookaside Buffer (TLB)** takes **1ns** to query.
- An **access to main memory** given a physical address costs **20ns**.
- A **page fault** costs **6ms**. After a page fault, the page's address translation is inserted into the TLB.
- Assume that the processor **restarts the address translation** after a page fault.

1. What is the **best case** data lookup time? **NOTE:** This is the time that elapses between when the processor presents a virtual address to the memory subsystem and when the word at that address has been loaded/stored to/from a register.

   Answer:

   ______________ ms ____________ ns

2. If the second level page-table is in main memory, what is the **worst case** data lookup time? Assume no individual page faults more than once (for example, other processes will not interfere with the memory access).

   Answer:

   ______________ ms ____________ ns

Problem 6. (8 points):

The following problem deals with memory allocation schemes. You have seen several ways to manage heap blocks. This problem deals with the buddy allocation scheme. In the buddy scheme, the heap starts out as one large free block. During allocation, blocks are split in half until a minimum-sized block that can satisfy the request is created. Each pair created by a split are called ”buddies.” A free block can *only* be coalesced with its buddy. For this problem, you may **ignore block headers and footers**.

In this problem, the valid block sizes are 32, 16, 8, and 4 bytes. As an example, imagine the heap starts as an empty 32-byte block. Suppose a request is made for 7 bytes. The heap will be split into two 16-byte blocks. The rightmost of these two 16-byte blocks will then be split into two 8-byte blocks. The rightmost of these two 8-byte blocks will be used to store the 7 bytes requested.

Your job:

- Draw the state of the heap by **marking the boundaries of allocated and free blocks** existing in the heap after the following trace of function calls.
- If a block is allocated, mark which variable below stores a pointer to that block.
- If a block is free, mark it with the letter ’X’.
- If you have a choice about which block to allocate a request to, use the rightmost block.

Note: you must have a vertical bar to mark each block boundary, as well as a letter in each byte indicating its allocated state.

```
a = malloc(12);
b = malloc(12);
free(a);
c = malloc(5);
d = malloc(3);
e = malloc(4);
free(c);
f = malloc(4);
free(d);
```

Workspace (not graded):

0				4				8				12				16				20				24				28			

0				4				8				12				16				20				24				28			

0				4				8				12				16				20				24				28			

Graded Solution:

0				4				8				12				16				20				24				28			

Problem 7. (25 points):

You have been assigned the task of optimizing (the assembly code for) the doubly recursive fibonacci function. For some unknown reason, the code must remain doubly recursive, but otherwise you are free to optimize it however you wish. Among the many possibilities, you should do at least do things. First, don't construct the stack frame unless absolutely necessary. Second, pass arguments in registers (e.g., `%eax`), not on the stack. The resulting code should run much faster and be much smaller than gcc's code (even when -O4 is turned on).

Implementing the second suggestion requires that the calling convention be changed. So, in order to encapsulate this change, `fib` should obey the calling convention to callers and needs to be changed to call `myfib` with the new calling convention.

To help you get started we include the assembly output of gcc for the following C program:

```
#include <stdio.h>

int fib(int n)
{
   return myfib(n);
}

int myfib(int n)
{
   if (n < 3) {
      return 1;
   }
   return myfib(n-2)+myfib(n-1);
}

main(int argc, char** argv)
{
   int x = atoi(argv[1]);
   printf("fib(%d) = %d\n", x, fib(x));
}
```

The important pieces of the assembly code are:

```
_fib:
        pushl %ebp
        movl %esp,%ebp
        subl $8,%esp
        movl 8(%ebp),%eax
        addl $-12,%esp
        pushl %eax
        call _myfib
L6:
        movl %ebp,%esp
        popl %ebp
        ret
        .align 4

_myfib:
        pushl %ebp
        movl %esp,%ebp
        subl $16,%esp
        pushl %esi
        pushl %ebx
        movl 8(%ebp),%ebx
        cmpl $2,%ebx
        jle L12
        addl $-12,%esp
        leal -2(%ebx),%eax
        pushl %eax
        call _myfib
L7:
        movl %eax,%esi
        addl $-12,%esp
        leal -1(%ebx),%eax
        pushl %eax
        call _myfib
L8:
        addl %esi,%eax
        jmp L16
        .align 4
L12:
        movl $1,%eax
L16:
        leal -24(%ebp),%esp
        popl %ebx
        popl %esi
        movl %ebp,%esp
        popl %ebp
        ret
```

(Question 7 cont’d)

A. Show the state of the stack when the function `fib` on the previous page has been invoked with `4` as the argument and `myfib(2)` has been entered for the second time. If they are known, use actual values in your answer, otherwise use register names.

4
L7

B. Implement a new version of `myfib` according to the suggestions above. The opcodes for a suggested final solution are listed below, you can fill in the operands (or cross these out and enter your own solution). *Hint: we strongly suggest that you follow the framework below.*

```
_myfib:
        cmpl


        jle


        pushl


        leal


        call


L7:
        popl


        pushl


        leal


        call


L8:
        addl


        addl


        ret

L1:
        movl   $1,%eax
        ret
```

(Question 7 cont'd)

C. Change the site in `fib` where function `myfib` is called to reflect the new calling convention. Indicate any and all changes here.

```
_fib:
        pushl %ebp
        movl %esp,%ebp
        subl $8,%esp
        movl 8(%ebp),%eax
        addl $-12,%esp
        pushl %eax
        call _myfib
L6:
        movl %ebp,%esp
        popl %ebp
        ret
```

D. Show the state of the stack when **your optimized** version of the program is invoked with 4 as the argument and myfib(2) has been entered for the second time. If they are known, use actual values in your answer, otherwise register names.

L7

Problem 8. (12 points):

This problem tests your understanding of concurrent programming and exceptional control flow. For each of the following programs, list **all** possible outputs in a comma separated list. Assume that syscalls execute without error.

Program 1

```
int i = 0;

int main()
{
    if(fork() == 0)
      i++;
    else
      i+=2;

    printf("%d",i);
}
```

All possible outputs:

Program 2

```
int i = 0;

int main()
{
    if(fork() == 0)
      i++;
    else
    {
      i+=3;
      wait(NULL);
    }
    printf("%d",i);
}
```

All possible outputs:

Program 3

```
int i = 0;

void *doit(void *vargp)
{
    pthread_detach(pthread_self());
    i++;
}

int main()
{
    pthread_t tid;
    pthread_create(&tid, NULL, doit, NULL);
    i++;
    printf("%d",i);
}
```

All possible outputs:

Program 4

```
int i = 0;

void *doit(void *vargp)
{
    i = i + 5;
}

int main()
{
    pthread_t tid;
    ptr = &i;
    pthread_create(&tid, NULL, doit, NULL);
    i = i + 3;
    pthread_join(tid, NULL);
    printf("%d",i);
}
```

All possible outputs:

(Question 8 cont'd)

Program 5

```
void *doit(void *vargp)
{
    int i = 3;
    int *ptr = (int*)vargp;
    (*ptr)++;
}

int main()
{
    int i = 0;
    pthread_t tid;
    pthread_create(&tid, NULL, doit, (void*)&i);
    pthread_join(tid,NULL);
    i = i + 4;
    printf("%d",i);
}
```

All possible outputs:

Problem 9. (15 points):

This question tests your understanding about basic POSIX standard Berkeley sockets interface. There's an **IRC server** running on 128.193.0.40, port 6667. (IRC stands for Internet Relay Chat, one of the most prevalent forms of online chat forum) Assume that the capitalized sys-calls below check for errors returned by their corresponding actual sys-calls. You might find these definitions helpful.

```
int Sem_init(sem_t *sem, int pshared, unsigned int value)
int Open(const char *pathname, int flags)
int Close(int fd);
ssize_t Read(int fd, void *buf, size_t count)
ssize_t Write(int fd, const void *buf, size_t count)

int Socket(int domain, int type, int protocol)
int Bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen)
int Listen(int s, int backlog)
int Connect(int sockfd, const struct sockaddr *serv_addr,
            socklen_t addrlen)
int Accept(int s, struct sockaddr *addr, socklen_t *addrlen)

int Select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout)
int Pthread_create(pthread_t th, pthread_attr_t *attr,
                   void *(*start_routine)(void *), void *arg)
int Pthread_detach(pthread_t th);

int Inet_aton(const char *cp, struct in_addr *inp)
FD_CLR(int fd, fd_set *set)
FD_ISSET(int fd, fd_set *set)
FD_SET(int fd, fd_set *set)
FD_ZERO(fd_set *set)

unsigned short htons(unsigned short hostshort)
unsigned long htonl(unsigned long hostlong)
unsigned short ntohs(unsigned short netshort)
unsigned long ntohl(unsigned long netlong)
```

Part 1

You're given a **client** that connects to the server on the next page. Fill in the blanks with the appropriate code.

```
#define MAXLINE      1024
#define IRCLINE      512   /* IRC messages are at most 512 bytes */
#define STDIN_FILENO 0
typedef SA struct sockaddr;
```

(Question 9 cont’d)

```
int main(int argc, char **argv) /* Client Main */
{
    int fd, bytes;
    struct sockaddr_in srv_addr;
    char buf[MAXLINE];
    fd_set read_set, ready_set;

    fd = Socket(AF_INET, SOCK_STREAM, 0);
    bzero((char *) &srv_addr, sizeof(srv_addr));
    srv_addr.sin_family = AF_INET;

    srv_addr.sin_port = _______________;
    Inet_aton("128.193.0.40", &srv_addr.sin_addr);

    ______________________________________________________________;

    __________________________________________;

    __________________________________________;

    __________________________________________;
    while (1) {
        ready_set = read_set;

        Select(_______________, _______________, NULL, NULL, NULL);

        if (FD_ISSET(_______________, &ready_set)) {

            bytes = Read(_______________, (void *)buf, IRCLINE);
            if (!bytes) {
                Close(fd);
                exit(0);    /* EOF - quits client */
            }
            buf[bytes] = ’\r’;   /* IRC protocol mandates each line */
            buf[bytes+1] = ’\n’; /* being terminated by \r\n */
            buf[bytes+2] = ’\0’;

            Write(_______________, (const void *)buf, strlen(buf));
        }
        if (FD_ISSET(_______________, &ready_set)) {

            bytes = Read(_______________, (void *)buf, MAXLINE-1);
            buf[bytes] = ’\0’;
            printf ("%s", buf);
        }
    }
}
```

Part 2

This simplified IRC **server** accepts connections from the client, logs the connection request, and proceeds to process the request. It keeps track of all the currently connected file descriptors in a global array.

The following code comes from the IRC server. You may assume that the helper functions work as expected, no buffer overflows occur, all signals are handled appropriately, and MAXBUDDY is never exceeded. Again, fill in the blanks below.

```
int buddy_fd[MAXBUDDY];
sem_t lock;

int main(int argc, char **argv) /* Server Main */
{
    int listenfd, n;
    int cli_len = sizeof(struct sockaddr_in);
    struct sockaddr_in srv_addr, cli_addr;
    pthread_t tid;

    init_slot(); /* Initialize each element in buddy_fd to -1 */

    _____________________________________________________________;
    listenfd = Socket(AF_INET, SOCK_STREAM, 0);
    bzero((char *) &srv_addr, sizeof(srv_addr));
    srv_addr.sin_family = AF_INET;

    srv_addr.sin_port = _______________;
    Inet_aton("128.193.0.40", &srv_addr.sin_addr);

    _____________________________________________________________;

    _____________________________________________________________;
    while(1)
    {
        n = avail_slot(); /* Returns a slot in buddy_fd which is -1 */

        buddy_fd[n] = Accept(listenfd, (struct sockaddr *)&cli_addr,
                             &cli_len);
        Pthread_create(&tid, NULL, irc_thread, &(buddy_fd[n]));
    }
}
```

(Question 9 cont’d)

```
void *irc_thread(void *connfd_p)
{
    char buf[MAXLINE];
    int fd, connfd = *((int *)connfd_p);

    ______________________________________________________________;

    format_log_entry(buf, connfd);  /* Fills buf with null */
                                    /* terminated string */

    fd = Open("connection.log", O_APPEND);

    P(&lock);
    Write(fd, buf, _________________________);
    V(&lock);

    Close(fd);

    process_request(connfd); /* Sends message to users in the */
                             /* same chatroom (buddy_fd) */

    Close(connfd);

    release_slot(connfd);    /* Set connfd’s slot in buddy_fd to -1 */

    return NULL;
}
```

Problem 10. (4 points):

After successfully writing your own *malloc* and *free* in Lab 6, you want to make it work in a multi-threaded environment. Consider the following extract of an over-simplified version of *malloc* and *free*.

```
#define PRED(p)   *((char **)(p + 4))
#define SUCC(p)   *((char **)p)
#define SIZE(p)   *((int *)(p - 4))

char *free_head;

void insert_free_block(char *bp) {

    PRED(bp) = NULL;

    SUCC(bp) = free_head;

    free_head = bp;
}

char *remove_free_block(int size) {
    char *free_tmp = free_head;

    while (free_tmp != NULL && SIZE(free_tmp) < size) {

        free_tmp = SUCC(free_tmp);
    }

    if (free_tmp == NULL)
        return NULL;

    if (PRED(free_tmp) != NULL) {
        SUCC(PRED(free_tmp)) = SUCC(free_tmp);
    } else {
        free_head = SUCC(free_tmp);
    }

    if (SUCC(free_tmp) != NULL) {
        PRED(SUCC(free_tmp)) = PRED(free_tmp);
    }

    return free_tmp;
}
```

(Question 10 cont'd)

```
char *malloc(size s) {

    char *bp;

    while ((bp = remove_free_block(s)) == NULL) {
        extend_heap();
    }

    set_alloc_bit(bp);

    return bp;
}

void free(char *bp) {

    unset_alloc_bit(bp);

    coalease(bp);

    insert_free_block(bp);

}
```

Incorporating semaphores only in the *malloc* and *free* functions, make the entire package work even when multiple threads call malloc and free. You may assume the omitted functions work in a single threaded environment. Also, **P()** and **V()** utilize a global mutex `lock` initialized to 1. Points will be deducted for unnecessary code.

Problem 11. (6 points):

The following table gives the parameters for a number of different caches, where *m* is the number of physical address bits, *C* is the cache size (number of data bytes), *B* is the block size in bytes, *E* is the number of lines per set, *S* is the number of cache sets, *t* is the number of tag bits, *s* is the number of set index bits, and *b* is the number of block offset bits.
Your task is to fill in the missing fields in the table.

Cache	*m*	*C*	*B*	*E*	*S*	*t*	*s*	*b*
1	32		8	1		21	8	3
2	32	2048			128	23	7	2
3	32	1024	2	8	64			1
4	32	1024		2	16	23	4	

Problem 12. (9 points):

Consider the following code for the next problem:

```
#define N 1000
#define TRUE 1
#define FALSE 0

int graph[N][N];

int isEdge(int x, int y) {
  if(graph[x][y] > 0)
    return TRUE;
  else
    return FALSE;
}

int numNodes() {
  return N;
}

int numEdges() {
  int cnt = 0;
  int i, j;

  for(i = 0;i < numNodes(); i++) {
    for(j = 0;j < numNodes(); j++) {
      if(graph[j][i] > 0)
        cnt += 1;
    }
  }
  return cnt>>1;
}
```

This problem asks you to find ways to increase the speed of the above piece of code, which counts the number of edges in an undirected, simple graph. A simple graph is a graph in which there are no self loops, and in which there is at most one edge between any two nodes. We will be representing graphs in adjacency matrix form. In this form, if there is an edge between nodes i and j, then `graph[i][j] = 1`. For example the adjacency matrix of a graph with two nodes and an edge between them is:

$$\begin{pmatrix} 01 \\ 10 \end{pmatrix}$$

Describe three ways in which you could **significantly** improve the running time of this code and explain why they are applicable. Your code must correctly count the number of edges in a simple, undirected graph in adjacency matrix form. Other than that, you may propose changes to any aspect of the code you please.

Technique 1:

Why it is applicable:

Technique 2:

Why it is applicable:

Technique 3:

Why it is applicable: