Full Name:________________________________

EECS 213 Fall 2011
Final Exam

1. (10 points):

Assume a 32-bit address machine with a 1024 byte cache. For each block size (B) and lines per set (E) specified below, fill in the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b). Give answers in decimal.

B	E	S	t	s	b
8	4	32	24	5	3
8	64	2	28	1	3
16	1	64	22	6	4
16	32	2	27	1	4
32	1	32	22	5	5
32	8	4	25	2	5

2. (5 points)

What would be the average time to access a sector on a disk with

- a rotational speed of 15,000 RPM,
- a $T_{avg\ seek}$ of 4ms
- an average of 800 sectors / track

$T_{rotation} = \frac{1}{2} \times 60/15000 \times 1000 = 30/15 = 2$

$T_{transfer} = 60/15000 \times 1/800 \times 1000 = 60/15 \times 1/800 = 4/800 = 1/400 = 0.005$

$T_{access} = 4 + 2 + 0.005 = 6.005$ms

Most common mistake: forgetting the $\frac{1}{2}$ in $T_{rotation}$

3. (10 points):

Say how many ``hello'' lines `main()` will print, for each definition of `doit()`.

```
int main() {
  doit();
  printf("hello\n");
  exit(0);
}
```

`void doit() {`
`  fork();`
`  fork();`
`  printf("hello\n");`
`  return;`
`}`
	`void doit() {`
	`  if (fork() == 0) {`
	`    fork();`
	`    printf("hello\n");`
	`    exit(0);`
	`  }`
	`  return;`
	`}`
		`void doit() {`
		`  if (fork() == 0) {`
		`    fork();`
		`    printf("hello\n");`
		`    return;`
		`  }`
		`  return;`
		`}`

Number of lines: 8	Number of lines: 3	Number of lines: 5

4. (5 points):

Study the following two files.

file1.c	**file2.c**

`void p2(void);`
 	
`int main()`
`{`
`  p2();`
`  return 0;`
`}`
	`#include <stdio.h>`
	
	`char main;`
	
	`void p2()`
	`{`
	`  printf("0x%x\n", main);`
	`}`

Will there be any failures in compiling or linking or executing? If so, be specific about what the failure will be. If there are no failures, what will the output represent?

There will be no problems compiling or linking. There is a strong reference to main in file1.c and a strong reference to p2 in file2.c. Because main is a function in file1.c, the char main reference in file2.c will pick up the first byte of main.

Since every function starts with a push operation, the output will be 0x55 but all I was looking for was being the first byte of main's code.

5. (10 points):

Consider an allocator that maintains double-word (8 byte) alignment, using an implicit free list, where the layout of each memory block is

- 32 bit header, with block size and least significant bit set to 1 if allocated
- the payload
- any padding needed

For the following memory requests, what block size would be allocated and what would be in the block header?

Request	Block size (decimal)	Block header (hex)
malloc(3)	8	0x9
malloc(11)	16	0x11
malloc(20)	24	0x19
malloc(21)	32	0x21

6. (10 points):

What is the output or possible outputs of this code?

```
int counter = 0;

void handler(int sig)
{
  counter++;
  sleep(1);
}

int main()
{
  int i;
  signal(SIGUSR2, handler);
  if (fork()==0) {
    for (i = 0; i < 5; i++) {
      // get parent process id
      kill(getppid(), SIGUSR2);
      printf("send SIGUR2 to parent\n");
    }
    exit(0);
  }

  wait(NULL);
  printf("counter=%d\n", counter);
  exit(0);
}
```

Because signals are not queued, and because the handler sleeps for a second, it will most likely miss most of the signals. So the output will be

counter = n

where n is most likely 1 or 2.

7. (10 points):

What is in the file `foo.txt` after this code executes?

```
int main()
{
  int fd1, fd2, fd3;
  char *fname = "foo.txt";
  fd1 = open(fname, O_CREAT| O_TRUNC | O_RDWR);
  write(fd1, "pqrs", 4);
  fd3 = open(fname, O_APPEND | O_WRONLY, 0);
  write(fd3, "jklmn", 5);
  fd2 = dup(fd1);
  write(fd2, "wxyz", 4);
  exit(0);
}
```

The first write will put "pqrs" in the file.
The second write will append "jklmn" to the end, so the file will be "pqrsjklmn."
The third write will use the same file position fd1 has, which is 4, so the final file output will be

pqrswxyzn

8. (10 points):

Assume:

- byte addressable memory accesses to 1-byte words (not 4-byte words)
- 16-bit virtual addresses
- 13-bit physical addresses
- 512 byte page size
- 8-way set associative TLB with 16 total entries
- 2-way set associative cache, 4 byte line size, 16 total lines.

The contents of the TLB, the page table for the first 32 pages, and the cache are as follows. All numbers are in hexadecimal.:

TLB			
Index	Tag	PPN	Valid
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	0
	08	1	0
	06	7	0
	03	1	0
	07	5	0
	02	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	7	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

2-way Set Associative Cache													
			Byte						Byte				
Index	Tag	Valid	0	1	2	3	Tag	Valid	0	1	2	3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

Part A. Enter the bits for the **virtual** address 1DDE in the boxes below. ABOVE the boxes, label:

VPO The virtual page offset

VPN The virtual page number

BELOW the boxes label:

TLBI The TLB index

TLBT The TLB tag

|------------VPN-------------|-------------VPO----------------------|

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	1	1	1	0	1	1	1	1	0

|------TLBT-----------------|TLBI|

Address translation	
Parameter	Value
VPN	0x0E
TLB Index	0x0
TLB Tag	0x07
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x1

Fill in the table on the left. Use hexadecimal numbers where indicated. If this is a page fault, put a question mark in PPN.

Part B. ABOVE the boxes below, label

PPO The physical page offset

PPN The physical page number

BELOW the boxes label

CO The block offset within the cache line

CI The cache index

CT The cache tag

|------PPN------|--------------PPO----------------------|

12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	0	1	1	1	1	0

|----------------CT-------------------|------CI-------|---CO---|

If Part A was NOT a page fault, then enter the bits for the **physical** address found for the virtual address 1DDE in the boxes above. Then fill in the table below. Use **hexadecimal** where indicted. If there's a cache miss, put a question mark for "Cache Byte returned".

Parameter	Value
Byte offset	0x2
Cache Index	0x7
Cache Tag	0x1E
Cache Hit? (Y/N)	Y
Cache Byte returned	0x0F