15-213 Introduction to Computer Systems

Exam 2

April 11, 2006

Name: ____________________

Andrew User ID: ____________________

Recitation Section: ____________________

- This is an open-book exam.
- Notes and calculators are permitted, but not computers.
- Write your answer legibly in the space provided.
- You have 80 minutes for this exam.
- We will drop your lowest score among questions 1–6.

Problem	**Max**	**Score**
1	**15**	
2	**15**	
3	**15**	
4	**15**	
5	**15**	
6	**15**	
Total	**75**	

1. Symbols and Linking (15 points)

Consider the following three files, `main.c`, `fib.c`, and `bignat.c`:

```
/* main.c */
void fib (int n);
int main (int argc, char** argv) {
  int n = 0;
  sscanf(argv[1], "%d", &n);
  fib(n);
}

/* fib.c */
#define N 16

static unsigned int ring[3][N];

static void print_bignat(unsigned int* a) {
  int i;
  for (i = N-1; i >= 0; i--)
    printf("%u ", a[i]);        /* print a[i] as unsigned int */
  printf("\n");
}

void fib (int n) {
  int i, carry;
  from_int(N, 0, ring[0]);      /* fib(0) = 0 */
  from_int(N, 1, ring[1]);      /* fib(1) = 1 */
  for (i = 0; i <= n-2; i++) {
    carry = plus(N, ring[i%3], ring[(i+1)%3], ring[(i+2)%3]);
    if (carry) { printf("Overflow at fib(%d)\n", i+2); exit(0); }
  }
  print_bignat(ring[n%3]);
}
```

Furthermore assume that a file `bignat.c` defines functions `plus` and `from_int` of the form

```
int plus (int n, unsigned int* a, unsigned int* b, unsigned int* c);
void from_int (int n, unsigned int k, unsigned int* a);
```

A possible definition of these functions is given in Problem 6, although this is not relevant here.

1. (9 points) Fill in the following tables by stating for each name whether it is local or global, and whether it is strong or weak. Cross out any box in the table that does not apply. For example, cross out the first box in a line if the symbol is not in the symbol table, or cross out the second box in a line if the symbol is not global (and therefore neither weak nor strong). Recall that in C, external functions do not need to be declared.

   `main.c`

	Local or Global?	Strong or Weak?
`fib`		
`main`		

   `fib.c`

	Local or Global?	Strong or Weak?
`ring`		
`print_bignat`		
`fib`		
`plus`		

2. (3 points) Now assume that the file `bignat.c` is compiled to a static library in archive format, `bignat.a` exporting the symbols `plus` and `from_int`.

   For each of the following calls to `gcc`, state if it

   (A) compiles and links correctly, or
   (B) linking fails due to undefined references, or
   (C) linking fails due to multiple definitions .

Command	Result (A, B, or C)
`gcc -o fib main.c fib.c bignat.a`	
`gcc -o fib bignat.a main.c fib.c`	
`gcc -o fib fib.c main.c bignat.a`	

3. (3 points) Consider the case where the programmer accidentally declared the variable `ring` in the file `fib.c` with

   ```
   static int ring[3][N];
   ```

   instead of `static unsigned int ring[3][N]`. Mark each of the following statements as true or false.

   - The files all still compile correctly. True False
   - The files can all still be linked correctly. True False
   - The resulting executable will still run correctly. True False

2. Virtual Address Translation (15 points)

We consider a virtual address system with the following parameters.

- The memory is byte addressable.
- Virtual addresses are 16 bits wide.
- Physical addresses are 16 bits wide.
- The page size is 1024 bytes.
- The TLB is fully associative with 16 total entries.

Recall that a fully associative cache has just one set of entries. In the following tables, all numbers are given in hexadecimal. The contents of the TLB and the page table for the first 16 virtual pages are as follows. If a VPN is not listed in the page table, assume it generates a page fault.

TLB

Tag	PPN	Valid
03	1B	1
06	06	0
28	23	1
01	18	0
31	01	1
12	00	0
07	3D	1
0B	11	1
2A	2C	0
11	1C	0
1F	03	1
08	14	1
09	2A	1
3F	30	0
10	0D	0
32	11	0

Page Table

VPN	PPN	Valid
00	27	1
01	0F	1
02	19	1
03	1B	1
04	06	0
05	03	0
06	06	0
07	3D	0
08	14	1
09	2A	1
0A	21	1
0B	11	1
0C	1C	1
0D	2D	0
0E	0E	0
0F	04	1

1. (5 points) In the four rows below, mark the bits that constitute the indicated part of the virtual address with an **X**. Leave the remaining bits of each row blank.

Virtual Page Number

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

Virtual Page Offset

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

TLB Tag

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

2. (5 points) For the virtual address `0xC7A4`, indicate the physical address and various results of the translation. If there is a page fault, enter "—" for the PPN and Physical Address. All answers should be given in hexadecimal.

   Virtual Address (one bit per box)

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

Parameter	Value
VPN	
TLB Tag	
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	
Physical Address	

3. (5 points) For the virtual address `0x05DD`, indicate the physical address and various results of the translation. If there is a page fault, enter "—" for the PPN and Physical Address. All answers should be given in hexadecimal.

   Virtual Address (one bit per box)

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

Parameter	Value
VPN	
TLB Tag	
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	
Physical Address	

3. Process Control (15 points)

Consider the following C program. For space reasons, we do not check return codes, so assume that all functions return normally. Also assume that `printf` is unbuffered.

```
int main() {
  pid_t pid1, pid2;

  pid1 = fork();
  pid2 = fork();
  if (pid1 && pid2) printf("A\n");
  if (pid1 || pid2) printf("B\n");
  exit(0);
}
```

Mark each column that represents a valid possible output of this program with 'Yes' and each column which is impossible with 'No'.

A	B	B	B	B
B	A	B	B	B
B	B	A	B	B
B	B	B	A	B

4. Signals (15 points)

Consider the following code.

```
        int i = 1;

        void handler (int sig) {
          i++;
        }

        int main() {
          pid_t pid;
          sigset_t s;
          sigemptyset(&s);
          sigaddset(&s, SIGUSR1);
          signal(SIGUSR1, handler);
          sigprocmask(SIG_BLOCK, &s, 0);
          pid = fork();
<LINE A>
          if (pid != 0) {
            i = 2;
<LINE B>
          } else {
            i = 3;
<LINE C>
          }
          sigprocmask(SIG_UNBLOCK, &s, 0);
          pause();    /* pause to allow all signals to arrive */
          printf("%d\n", i);
          exit(0);
        }
```

We assume that `pause();` pauses long enough that all signals in a process arrive before the following `printf` command is executed and that concurrently running processes proceed to their `pause();` command if they are not already there. We also assume that `fork();` is successful and that all processes run to successful completion.

Now consider the effect of adding the command

```
kill(pid, SIGUSR1);
```

either at `<LINE A>`, `<LINE B>`, or `<LINE C>`. Recall that if the first argument to `kill` is 0, it sends the signal to all processes in the current process group. For each resulting program, list the possible values that may be printed for any given run. You may assume that no other process sends a `SIGUSR1` signal.

1. (5 pts) `<LINE A>`

2. (5 pts) `<LINE B>`

3. (5 pts) `<LINE C>`

5. Garbage Collection (15 points)

In this problem we consider a tiny list processing machine in which each memory word consists of two bytes: the first byte is a pointer to the tail of the list and the second byte is a data element. The end of a list is marked by a pointer of `0x00`. We assume that the data element is never a pointer.

1. (8 points) In the first question we consider a copying collector.

   We start with the memory state on the left, where the range `0x10`–`0x1F` is the from-space and the range `0x20`–`0x2F` is the to-space. All addresses and values in the diagram are in hexadecimal.

   Write in the state of memory after a copying collector is called with root pointers `0x12` and `0x1A` and answer the subsequent question. You may leave cells that remain unchanged blank.

Before GC

Addr	Ptr	Data
10	12	2C
12	18	FF
14	12	0E
16	1C	AB
18	16	10
1A	00	00
1C	12	1D
1E	1A	00

After GC

Addr	Ptr	Data	Addr	Ptr	Data
10			20		
12			22		
14			24		
16			26		
18			28		
1A			2A		
1C			2C		
1E			2E		

After garbage collection, free space starts as address ______

2. (7 points) In the second question we consider a mark and sweep collector.

We use the lowest bit of the pointer as the mark bit, because it is normally always zero since pointers must be word-aligned.

Assume the garbage collector is once again called with root pointers `0x12` and `0x1A`. Write in the state of memory after the mark phase, and then again after the sweep phase and answer the subsequent question. You may leave cells that remain unchanged blank.

Before GC

Addr	Ptr	Data
10	12	2C
12	18	FF
14	12	0E
16	1C	AB
18	16	10
1A	00	00
1C	12	1D
1E	1A	00

After Marking Phase

Addr	Ptr	Data
10		
12		
14		
16		
18		
1A		
1C		
1E		

After Sweep Phase

Addr	Ptr	Data
10		
12		
14		
16		
18		
1A		
1C		
1E		

The free list now starts at address ______.

6. Cyclone (15 points)

Now consider the file `bignat.c` that implements addition and conversion of an unsigned integer to a bignat representation as we assumed in Problem 1.

```
typedef unsigned int uint;

int plus (int n, uint* a, uint* b, uint* c) {
  int i;
  int carry = 0;
  for (i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
    if (carry) {
      c[i]++;
      carry = (c[i] <= a[i]);
    } else {
      carry = (c[i] < a[i]);
    }
  }
  return carry;
}

void from_int (int n, uint k, uint* a) {
  int i;
  a[0] = k;
  for (i = 1; i < n; i++)
    a[i] = 0;
  return;
}
```

1. (3 points) Is it legal for `a`, `b`, and `c` to be pointers to different regions of memory? Circle one

   Yes   No

2. (4 points) Assume `a`, `b`, and `c` are declared as fat pointers. Use the notation $\texttt{curr}(p)$, $\texttt{lower}(p)$, and $\texttt{upper}(p)$ to denote the current value of a fat pointer p and its lower and upper bounds. The bounds are *inclusive*: $\texttt{lower}(p) \leq \texttt{curr}(p) \leq \texttt{upper}(p)$, and you may assume all fat pointers will always satisfy this invariant. Under which conditions on `n`, `a`, `b`, and `c` will this code execute without an illegal pointer access exception?

In order to avoid the still possible overflow, we would like to implement arbitrarily large unsigned integers as linked lists of unsigned integers. In order to avoid always using linked lists, we represent bignats as a tagged union, either consisting just of an integer or a pointer to a linked list representation.

```
struct List {
  unsigned int head;
  struct List* tail;
};

@tagged union Bignat {
  unsigned int x;
  struct List* l;
};
```

4. (5 points) Recall that Cyclone translates its source into C code. Give a representation of the tagged union construct above that would be a plausible result of a translation from Cyclone to C.

5. (3 points) Give a plausible translation of the following Cyclone code fragment using your translation of the tagged union.

   ```
   union Bignat a;
   a.x = 15213;
   ```