15-213 Recitation 15: Final Exam Preparation

25 April 2016
Ralf Brown and the 15-213 staff
Agenda

- Reminders
- Final Exam Review
  - Fall 2012 exam
Reminders

- Proxy lab is due **tomorrow**!
  - NO GRACE DAYS
  - Penalty late days are allowed
  - *We will test your proxy manually*
  - *We will read your code*
    - correctness: race conditions, robustness
    - style: write clean, well-documented, modularized code – make it shine!
- Final exam is next week
Final Exam Details

- May 2 through 5
  - sign-ups are open
- Eight to ten problems
  - nominal time is 90-120 minutes, but you get five hours
  - problems cover material from the entire semester
- Notes
  - you are allowed two 8.5x11 double-sided sheets of notes
  - no pre-worked problems allowed
Each thread has its own _______.
- heap
- stack
- global values
- text data

Simply decreasing the size of block headers used internally by malloc:
- decreases internal fragmentation
- increases internal fragmentation
- decreases external fragmentation
- increases external fragmentation
Each thread has its own _______.
- heap
- stack
- global values
- text data

Simply decreasing the size of block headers used internally by malloc:
- decreases internal fragmentation
- increases internal fragmentation
- decreases external fragmentation
- increases external fragmentation
Fall 2012 Final Exam – Multiple Choice (2)

Which of the following sentences about reader-writer locks is **not** true?

- Many readers can hold the same rwlock at the same time
- Two writers cannot hold the same rwlock at the same time
- Many readers and exactly one writer can hold the same rwlock at the same time
- An rwlock can be used as a mutex

Which of the following is the correct ordering (left-to-right) of a file's compilation cycle?

- foo.c -> foo.o -> foo.s -> foo
- foo -> foo.s -> foo.o -> foo.c
- foo.c -> foo.s -> foo -> foo.o
- foo.c -> foo.s -> foo.o -> foo
Which of the following sentences about reader-writer locks is **not** true?

- Many readers can hold the same rwlock at the same time
- Two writers cannot hold the same rwlock at the same time
- Many readers and exactly one writer can hold the same rwlock at the same time
- An rwlock can be used as a mutex

Which of the following is the correct ordering (left-to-right) of a file's compilation cycle?

- foo.c -> foo.o -> foo.s -> foo
- foo -> foo.s -> foo.o -> foo.c
- foo.c -> foo.s -> foo -> foo.o
- foo.c -> foo.s -> foo.o -> foo
Suppose an int A is stored at virtual address 0xff987cf0, while another int B is stored at virtual address 0xff987d98. If the size of a page is 0x1000 bytes, then A's physical address is numerically less than B's physical address.

- always true
- always false
- sometimes true, sometimes false
- not enough information

Assuming no errors, which of the following functions returns exactly once?

- fork()
- execve()
- exit()
- longjmp()
- waitpid()
Suppose an int A is stored at virtual address 0xff987cf0, while another int B is stored at virtual address 0xff987d98. If the size of a page is 0x1000 bytes, then A's physical address is numerically less than B's physical address.

- **always true**
- always false
- sometimes true, sometimes false
- not enough information

Assuming no errors, which of the following functions returns exactly once?

- fork()
- execve()
- exit()
- longjmp()
- waitpid()
On a 64-bit system, which of the following C expressions is equivalent to the C expression $(x[2]+4)[3]$? Assume $x$ is declared as `int **x`.

- `*(((x+16)) + 28)`
- `*(((x + 2)) + 7)`
- `**(x * 28)`
- `*(((x) + 2) + 7)`
- `(**(x + 2) + 7)`

When can short counts occur?

- when an EOF is encountered during a read
- when a `short int` is used as a counter
- when writing to disk files
- when the kernel runs out of kernel memory
On a 64-bit system, which of the following C expressions is equivalent to the C expression \((x[2]+4)[3]\)? Assume \(x\) is declared as int **x.

- \(*((x+16)) + 28\)
- \(*((x + 2)) + 7\)
- \(**(x * 28)\)
- \(*(((\*x) + 2) + 7\)
- \(*((x + 2) + 7)\)

When can short counts occur?
- when an EOF is encountered during a read
- when a short int is used as a counter
- when writing to disk files
- when the kernel runs out of kernel memory
A program blocks SIGCHLD and SIGUSR1. It is then sent a SIGCHLD, a SIGUSR1, and another SIGCHLD, in that order. What signals does the program receive after it unblocks those signals (you may assume it is not sent any further signals afterward)?

- none, signals are discarded while blocked
- just a single SIGCHLD, since all subsequent signals are discarded
- just a single SIGCHLD and single SIGUSR1, since the extra SIGCHLD is discarded
- All three signals, since none are discarded

Which of the following events does not generate a signal?

- division by zero
- a new connection arrives on a listening socket
- a write is attempted on a disconnected socket
- NULL is dereferenced
- a process whose parent has already terminated exits
A program blocks SIGCHLD and SIGUSR1. It is then sent a SIGCHLD, a SIGUSR1, and another SIGCHLD, in that order. What signals does the program receive after it unblocks those signals (you may assume it is not sent any further signals afterward)?

- None, signals are discarded while blocked
- Just a single SIGCHLD, since all subsequent signals are discarded
- Just a single SIGCHLD and single SIGUSR1, since the extra SIGCHLD is discarded
- All three signals, since none are discarded

Which of the following events does not generate a signal?

- Division by zero
- A new connection arrives on a listening socket
- A write is attempted on a disconnected socket
- NULL is dereferenced
- A process whose parent has already terminated exits
In an x86-64 system, how many integers can be stored in a cache line if your cache is 4KB, is 4-way set associative, and contains 4 sets?
- 8
- 16
- 32
- 64
- 128

What types of locality are leveraged by virtual memory?
- spatial locality
- temporal locality
- prime locality
- both spatial and temporal
- both temporal and prime locality
In an x86-64 system, how many integers can be stored in a cache line if your cache is 4KB, is 4-way set associative, and contains 4 sets?
- 8
- 16
- 32
- 64
- 128

What types of locality are leveraged by virtual memory?
- spatial locality
- temporal locality
- prime locality
- both spatial and temporal
- both temporal and prime locality
Fall 2012 Final Exam – Multiple Choice (7)

Which of the following is not a section of an ELF file?

- .text
- .static
- .rodata
- .data
- .bss

Choose the true statement.

- All thread-safe functions are reentrant.
- Some reentrant functions are not thread safe.
- It is never a good idea to use persistent state across multiple function calls.
- It is impossible to have a race condition between two threads with no shared state.
- All functions which call non-thread-safe functions are themselves not thread safe.
Which of the following is not a section of an ELF file?

- .text
- .static
- .rodata
- .data
- .bss

Choose the true statement.

- All thread-safe functions are reentrant.
- Some reentrant functions are not thread safe.
- It is never a good idea to use persistent state across multiple function calls.
- It is impossible to have a race condition between two threads with no shared state.
- All functions which call non-thread-safe functions are themselves not thread safe.
We use dynamic memory because:

- the heap is significantly faster than the stack
- the stack is prone to corruption from buffer overflow
- storing data on the stack requires knowing the size of that data at compile time
- none of the above
We use dynamic memory because:

- the heap is significantly faster than the stack
- the stack is prone to corruption from buffer overflow
- storing data on the stack requires knowing the size of that data at compile time
- none of the above
In the following code, a parent opens a file twice, then the child reads a character:

```c
char c;
int fd1 = open("foo.txt", O_RDONLY);
int fd2 = open("foo.txt", O_RDONLY);
if (!fork()) {
    read(fd1, &c, 1);
}
```

Clearly, in the child, fd1 now points to the second character of foo.txt. Which of the following is now true in the parent?

- fd1 and fd2 both point to the first character
- fd1 and fd2 both point to the second character
- fd1 points to the first character, fd2 points to the second character
- fd2 points to the first character, fd1 points to the second character
In the following code, a parent opens a file twice, then the child reads a character:

```c
char c;
int fd1 = open("foo.txt", O_RDONLY);
int fd2 = open("foo.txt", O_RDONLY);
if (!fork()) {
  read(fd1, &c, 1);
}
```

Clearly, in the child, fd1 now points to the second character of foo.txt. Which of the following is now true in the parent?

- fd1 and fd2 both point to the first character
- fd1 and fd2 both point to the second character
- fd1 points to the first character, fd2 points to the second character
- fd2 points to the first character, fd1 points to the second character
Which of the following is true about races?

- A race occurs when correctness of the program depends on one thread reaching point A before another thread reaches point B.
- Exclusive access to all shared resources eliminates race conditions.
- Race conditions are the same as deadlocks.
- All race conditions occur inside loops, since that is the only way we can interleave processes.
Which of the following is true about races?

- A race occurs when correctness of the program depends on one thread reaching point A before another thread reaches point B.
- Exclusive access to all shared resources eliminates race conditions.
- Race conditions are the same as deadlocks.
- All race conditions occur inside loops, since that is the only way we can interleave processes.
Consider the following two blocks of code, which are contained in separate files.

```c
/* main.c */
int i = 0;
int main() {
    foo();
    return 0;
}

/* foo.c */
int i = 1;
void foo() {
    printf("%d", i);
}
```

What will happen when you attempt to compile, link, and run this code?

- it will fail to compile
- it will fail to link
- it will raise a segmentation fault
- it will print “0”
- it will print “1”
- it will sometimes print “0” and sometimes print “1”
Consider the following two blocks of code, which are contained in separate files.

```c
/* main.c */
int i = 0;
int main() {
    foo();
    return 0;
}

/* foo.c */
int i = 1;
void foo() {
    printf("%d", i);
}
```

What will happen when you attempt to compile, link, and run this code?

- it will fail to compile
- it will fail to link
- it will raise a segmentation fault
- it will print “0”
- it will print “1”
- it will sometimes print “0” and sometimes print “1”
In this problem, you will work with floating point numbers based on the IEEE floating point format. We consider two different 6-bit formats:

**Format A:**
- There is one sign bit \( S \)
- There are \( k=3 \) exponent bits. The bias is \( 2^{k-1} - 1 = 3 \).
- There are \( n=2 \) fraction bits.

**Format B:**
- There is one sign bit \( S \)
- There are \( k=2 \) exponent bits. The bias is \( 2^{k-1} - 1 = 1 \).
- There are \( n=3 \) fraction bits.

Please write down the binary representation for the following (use round-to-even).

Recall that for denormalized numbers, \( E = 1 \)-bias. For normalized numbers, \( E = e \)-bias.

<table>
<thead>
<tr>
<th>Value</th>
<th>Format A Bits</th>
<th>Format B Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>0 011 00</td>
<td>0 01 000</td>
</tr>
<tr>
<td>Three</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7/8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15/8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In this problem, you will work with floating point numbers based on the IEEE floating point format. We consider two different 6-bit formats:

- **Format A:**
  - There is one sign bit $S$
  - There are $k=3$ exponent bits. The bias is $2^{k-1} - 1 = 3$.
  - There are $n=2$ fraction bits.

- **Format B:**
  - There is one sign bit $S$
  - There are $k=2$ exponent bits. The bias is $2^{k-1} - 1 = 1$.
  - There are $n=3$ fraction bits.

Please write down the binary representation for the following (use round-to-even).

Recall that for denormalized numbers, $E = 1$-bias. For normalized numbers, $E = e$-bias.

<table>
<thead>
<tr>
<th>Value</th>
<th>Format A Bits</th>
<th>Format B Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>0 011 00</td>
<td>0 01 000</td>
</tr>
<tr>
<td>Three</td>
<td>0 100 10</td>
<td>0 10 100</td>
</tr>
<tr>
<td>$7/8$</td>
<td>0 010 11</td>
<td>0 00 111</td>
</tr>
<tr>
<td>$15/8$</td>
<td>0 100 00</td>
<td>0 01 111</td>
</tr>
</tbody>
</table>
Problem 3. (6 points):

Arrays. Consider the C code below, where $H$ and $J$ are constants declared with `#define`.

```c
int array1[H][J];
int array2[J][H];

void copy_array(int x, int y) {
    array2[x][y] = array1[y][x];
}
```

Suppose the above C code generates the following x86-64 assembly code:

```assembly
# On entry:
#   %edi = x
#   %esi = y
#
copy_array:
    movslq %esi,%rsi
    movslq %edi,%rdi
    movq %rdi, %rax
    salq $4, %rax
    subq %rdi, %rax
    addq %rsi, %rax
    leaq (%rsi,%rsi,4), %rsi
    leaq (%rdi,%rsi,2), %rsi
    movl array1(%rsi,4), %edx
    movl %edx, array2(%rax,4)
    ret
```

What are the values of $H$ and $J$?

$H = \_\_\_\_\_\_\_\_\_ \hspace{1cm} J = \_\_\_\_\_\_\_\_\_$
Problem 3. (6 points):

Arrays. Consider the C code below, where H and J are constants declared with #define.

```c
int array1[H][J];
int array2[J][H];

void copy_array(int x, int y) {
    array2[x][y] = array1[y][x];
}
```

Suppose the above C code generates the following x86-64 assembly code:

```
# On entry:
#    %edi = x
#    %esi = y

copy_array:
    movslq %esi,%rsi
    movslq %edi,%rdi
    movq %rdi, %rax
    salq $4, %rax
    subq %rdi, %rax
    addq %rsi, %rax
    leaq (%rsi,%rsi,4), %rsi
    leaq (%rdi,%rsi,2), %rsi
    movl array1(,%rsi,4), %edx
    movl %edx, array2(,%rax,4)
    ret
```

What are the values of H and J?

\[ H = 15 \]
\[ J = 10 \]
Problem 4. (8 points):
Loops. Consider the following x86-64 assembly function:

```
loop:
  # on entry: a in %rdi, n in %esi
  movl  $0, %r8d
  movl  $0, %ecx
  testl %esi, %esi
  jle .L3

.L6:
  movl (%rdi, %rcx, 4), %edx
  leal  3(%rdx), %eax
  testl %edx, %edx
  cmovns %edx, %eax
  sarl $2, %eax
  addl %eax, %r8d
  addq $1, %rcx
  cmpl $1, %ecx
  jg .L6

.L3:
  movl %r8d, %eax
  ret
```

Fill in the blanks of the corresponding C code.

- You may only use the C variable names `n`, `a`, `i` and `sum`, not register names.
- Use array notation in showing accesses or updates to elements of `a`. 

```c
int loop(int a[], int n)
{
    int i, sum;
    sum = ____;
    for (i = __________; __________; __________) {
        sum += __________;
    }
    return __________;
}
```
Problem 4. (8 points):
Loops. Consider the following x86-64 assembly function:

```
loop:
    # on entry: a in %rdi, n in %esi
    movl  $0, %r8d
    movl  $0, %ecx
    testl %esi, %esi
    jle .L3
.L6:
    movl (%rdi,%rcx,4), %edx
    leal  3(%rdx), %eax
    testl %edx, %edx
    cmovns %edx, %eax
    sarl  $2, %eax
    addl  %eax, %r8d
    addq  $1, %rcx
    cmpl  %ecx, %esi
    jg    .L6
.L3:
    movl  %r8d, %eax
    ret
```

Fill in the blanks of the corresponding C code.

- You may only use the C variable names `n`, `a`, `i` and `sum`, not register names.
- Use array notation in showing accesses or updates to elements of `a`.

```c
int loop(int a[], int n) {
    int i, sum;
    sum = __________;
    for (i = __________; i < n; i++) {
        sum += __________;
    }
    return __________;
}
```
Problem 5. (10 points):

Stack discipline. Consider the following C code and its corresponding 32-bit x86 machine code. Please complete the stack diagram on the following page.

```c
int fact(int n) {
    if (n == 1)
        return n;
    else
        return n * fact(n-1);
}
```

080483a4 <fact>:

```assembly
080483a4: 55 push %ebp
80483a5: 89 e5 mov %esp,%ebp
80483a7: 53 push %ebx
80483a8: 83 ec 04 sub $0x4,%esp
80483ab: 8b 5d 08 mov 0x8(%ebp),%ebx
80483ae: 83 fb 01 cmp $0x1,%ebx
80483b1: 74 0e je 80483c1 <fact+0x1d>
80483b3: 8d 43 ff lea 0xffffffff(%ebx),%eax
80483b6: 89 04 24 mov %eax,(%esp)
80483be: e8 e6 ff ff ff call 80483a4 <fact>
80483c1: 89 d8 imul %eax,%ebx
80483c3: 83 c4 04 add $0x4,%esp
80483c6: b0 pop %ebx
80483c7: 5d pop %ebp
80483c8: c3 ret
```

32-bit stack convention
A. Draw a detailed picture of the stack, starting with the caller invoking `fact(4)`, and ending immediately **before** the call instruction that invokes `fact(2)`.

- The stack diagram should begin with the argument for `fact` that the caller has placed on the stack. To help you get started, we have given you the first one.
- Use the actual values for function arguments, rather than variable names. For example, use `3` or `2` instead of `n`.
- For callee-saved registers that are pushed to the stack, simply note the register name (e.g., `%ebx`).
- Always label `%ebp` and give its value when it is pushed to the stack, e.g., `%ebp: 0xffff1400`.

Value of `%ebp` when `fact(4)` is called: 0xfffffd848
Return address in function that called `fact(4)`: 0x080483e6

**this problem is obsolete**
B. What is the final value of `%ebp`, immediately before execution of the instruction that calls `fact(2)`?

```
%ebp=0x____________________
```

C. What is the final value of `%esp`, immediately before execution of the instruction that calls `fact(2)`?

```
%esp=0x____________________
```
B. What is the final value of %ebp, immediately before execution of the instruction that calls fact (2)?

%ebp=0x_ffffd818

C. What is the final value of %esp, immediately before execution of the instruction that calls fact (2)?

%esp=0x_ffffd810
Problem 6. (12 points):

Cache memories. Consider the following matrix transpose function

typedef int array[2][2];

void transpose(array dst, array src) {
    int i, j;

    for (j = 0; j < 2; j++) {
        for (i = 0; i < 2; i++) {
            dst[i][j] = src[j][i];
        }
    }
}

running on a hypothetical machine with the following properties:

• sizeof(int) == 4.

• The src array starts at address 0 and the dst array starts at address 16 (decimal).

• There is a single L1 data cache that is direct mapped and write-allocate, with a block size of 8 bytes.

• Accesses to the src and dst arrays are the only sources of read and write accesses to the cache, respectively.
A. Suppose the cache has a total size of 16 data bytes (i.e., the block size times the number of sets is 16 bytes) and that the cache is initially empty. Then for each row and col, indicate whether each access to src[row][col] and dst[row][col] is a hit (h) or a miss (m). For example, reading src[0][0] is a miss and writing dst[0][0] is also a miss.

<table>
<thead>
<tr>
<th>src array</th>
<th>dst array</th>
</tr>
</thead>
<tbody>
<tr>
<td>col 0</td>
<td>col 0</td>
</tr>
<tr>
<td>col 1</td>
<td>col 1</td>
</tr>
<tr>
<td>row 0</td>
<td>row 0</td>
</tr>
<tr>
<td></td>
<td>m</td>
</tr>
<tr>
<td>row 1</td>
<td>row 1</td>
</tr>
</tbody>
</table>

B. Repeat part A for a cache with a total size of 32 data bytes.
A. Suppose the cache has a total size of 16 data bytes (i.e., the block size times the number of sets is 16 bytes) and that the cache is initially empty. Then for each row and col, indicate whether each access to src[row][col] and dst[row][col] is a hit (h) or a miss (m). For example, reading src[0][0] is a miss and writing dst[0][0] is also a miss.

<table>
<thead>
<tr>
<th>src array</th>
<th>dst array</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>row 0</td>
<td>m</td>
</tr>
<tr>
<td>col 0</td>
<td>m</td>
</tr>
<tr>
<td>row 1</td>
<td>m</td>
</tr>
<tr>
<td>col 1</td>
<td>h</td>
</tr>
</tbody>
</table>

B. Repeat part A for a cache with a total size of 32 data bytes.

<table>
<thead>
<tr>
<th>src array</th>
<th>dst array</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>row 0</td>
<td>m</td>
</tr>
<tr>
<td>col 0</td>
<td>h</td>
</tr>
<tr>
<td>row 1</td>
<td>m</td>
</tr>
<tr>
<td>col 1</td>
<td>m</td>
</tr>
</tbody>
</table>
Problem 7. (6 points):

Linking. Consider the executable object file `a.out`, which is compiled and linked using the command

```
unix> gcc -o a.out main.c foo.c
```

and where the files `main.c` and `foo.c` consist of the following code:

```c
/* main.c */
#include <stdio.h>
int a = 1;
static int b = 2;
int c = 3;
int main()
{
    int c = 4;
    foo();
    printf("a=%d b=%d c=%d\n", a, b, c);
    return 0;
}
```

```c
/* foo.c */
int a, b, c;
void foo()
{
    a = 5;
    b = 6;
    c = 7;
}
```

What is the output of `a.out`?

**Answer:**  \( a=____, \ b=_____, \ c=____\)
Problem 7. (6 points):

Linking. Consider the executable object file a.out, which is compiled and linked using the command

unix> gcc -o a.out main.c foo.c

and where the files main.c and foo.c consist of the following code:

/* main.c */
#include <stdio.h>
int a = 1;
static int b = 2;
int c = 3;
int main()
{
    int c = 4;
    foo();
    printf("a=%d b=%d c=%d\n", a, b, c);
    return 0;
}

/* foo.c */
int a, b, c;
void foo()
{
    a = 5;
    b = 6;
    c = 7;
}

What is the output of a.out?

Answer:  a=5, b=2, c=4
Problem 8. (10 points):

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

```c
int main()
{
    int val = 2;

    printf("%d", 0);
    fflush(stdout);

    if (fork() == 0) {
        val++;
        printf("%d", val);
        fflush(stdout);
    }
    else {
        val--;
        printf("%d", val);
        fflush(stdout);
        wait(NULL);
    }

    val++;
    printf("%d", val);
    fflush(stdout);
    exit(0);
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program. You will be graded on each sub-problem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 2 points.
- If you circle the wrong answer, you get $-1$ points (so don’t just guess wildly).

A. 01432   Y   N
B. 01342   Y   N
C. 03142   Y   N
D. 01234   Y   N
E. 03412   Y   N
Problem 8. (10 points):

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

```c
int main()
{
    int val = 2;

    printf("%d", 0);
    fflush(stdout);

    if (fork() == 0) {
        val++;
        printf("%d", val);
        fflush(stdout);
    }
    else {
        val--;
        printf("%d", val);
        fflush(stdout);
        wait(NULL);
    }
    val++;
    printf("%d", val);
    fflush(stdout);
    exit(0);
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program. You will be graded on each sub-problem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 2 points.
- If you circle the wrong answer, you get \(-1\) points (so don’t just guess wildly).

<table>
<thead>
<tr>
<th>Option</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. 01432</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>B. 01342</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>C. 03142</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>D. 01234</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>E. 03412</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>
Problem 9. (12 points):

Address translation. This problem concerns the way virtual addresses are translated into physical addresses. Imagine a system has the following parameters:

- Virtual addresses are 20 bits wide.
- Physical addresses are 18 bits wide.
- The page size is 1024 bytes.
- The TLB is 2-way set associative with 16 total entries.

The contents of the TLB and the first 32 entries of the page table are shown as follows. **All numbers are given in hexadecimal.**

<table>
<thead>
<tr>
<th>TLB</th>
<th>Index</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>C3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>71</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>28</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>35</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>68</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3A</td>
<td>F1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>12</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>02</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7F</td>
<td>05</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>A1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>53</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>03</td>
<td>4E</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1B</td>
<td>34</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>1F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>03</td>
<td>38</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>09</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Table</th>
<th>VPN PPN Valid</th>
<th>VPN PPN Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>71 1</td>
<td>010 60 0</td>
</tr>
<tr>
<td>001</td>
<td>28 1</td>
<td>011 57 0</td>
</tr>
<tr>
<td>002</td>
<td>93 1</td>
<td>012 68 1</td>
</tr>
<tr>
<td>003</td>
<td>AB 0</td>
<td>013 30 1</td>
</tr>
<tr>
<td>004</td>
<td>D6 0</td>
<td>014 0D 0</td>
</tr>
<tr>
<td>005</td>
<td>53 1</td>
<td>015 2B 0</td>
</tr>
<tr>
<td>006</td>
<td>1F 1</td>
<td>016 9F 0</td>
</tr>
<tr>
<td>007</td>
<td>80 1</td>
<td>017 62 0</td>
</tr>
<tr>
<td>008</td>
<td>02 0</td>
<td>018 C3 1</td>
</tr>
<tr>
<td>009</td>
<td>35 1</td>
<td>019 04 0</td>
</tr>
<tr>
<td>00A</td>
<td>41 0</td>
<td>01A F1 1</td>
</tr>
<tr>
<td>00B</td>
<td>86 1</td>
<td>01B 12 1</td>
</tr>
<tr>
<td>00C</td>
<td>A1 1</td>
<td>01C 30 0</td>
</tr>
<tr>
<td>00D</td>
<td>D5 1</td>
<td>01D 4E 1</td>
</tr>
<tr>
<td>00E</td>
<td>8E 0</td>
<td>01E 57 1</td>
</tr>
<tr>
<td>00F</td>
<td>D4 0</td>
<td>01F 38 1</td>
</tr>
</tbody>
</table>
Part 1

1. The diagram below shows the format of a virtual address. Please indicate the following fields by labeling the diagram:

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

   ![Diagram of virtual address format]

2. The diagram below shows the format of a physical address. Please indicate the following fields by labeling the diagram:

   - **PPO**  The physical page offset
   - **PPN**  The physical page number

   ![Diagram of physical address format]
Part 1

1. The diagram below shows the format of a virtual address. Please indicate the following fields by labeling the diagram:

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

   ![Virtual Address Diagram]

2. The diagram below shows the format of a physical address. Please indicate the following fields by labeling the diagram:

   - **PPO** The physical page offset
   - **PPN** The physical page number

   ![Physical Address Diagram]
Part 2

For the given virtual addresses, please indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs. If there is a page fault, enter “-” for “PPN” and leave the physical address blank.

Virtual address: 078E6

1. Virtual address (one bit per box)

   19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0

2. Address translation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
<td>0x</td>
<td>TLB Hit? (Y/N)</td>
<td></td>
</tr>
<tr>
<td>TLB Index</td>
<td>0x</td>
<td>Page Fault? (Y/N)</td>
<td></td>
</tr>
<tr>
<td>TLB Tag</td>
<td>0x</td>
<td>PPN</td>
<td>0x</td>
</tr>
</tbody>
</table>

3. Physical address(one bit per box)

   17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
Part 2

For the given virtual addresses, please indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs. If there is a page fault, enter "-" for “PPN” and leave the physical address blank.

Virtual address: 078E6

1. Virtual address (one bit per box) 0000 0111 1000 1110 0110

2. Address translation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
<td>0x01E</td>
<td>TLB Hit? (Y/N)</td>
<td>N</td>
</tr>
<tr>
<td>TLB Index</td>
<td>0x6</td>
<td>Page Fault? (Y/N)</td>
<td>N</td>
</tr>
<tr>
<td>TLB Tag</td>
<td>0x03</td>
<td>PPN</td>
<td>0x57</td>
</tr>
</tbody>
</table>

3. Physical address(one bit per box) 01 0101 1100 1110 0110
**Virtual address:** 04AA4

1. Virtual address (one bit per box)

```
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

2. Address translation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
<td>0x</td>
<td>TLB Hit? (Y/N)</td>
<td></td>
</tr>
<tr>
<td>TLB Index</td>
<td>0x</td>
<td>Page Fault? (Y/N)</td>
<td></td>
</tr>
<tr>
<td>TLB Tag</td>
<td>0x</td>
<td>PPN</td>
<td>0x</td>
</tr>
</tbody>
</table>

3. Physical address (one bit per box)

```
17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```
1. Virtual address (one bit per box)

Virtual address: 04AA4

<table>
<thead>
<tr>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Address translation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
<td>0x012</td>
<td>TLB Hit? (Y/N)</td>
<td>Y</td>
</tr>
<tr>
<td>TLB Index</td>
<td>0x2</td>
<td>Page Fault? (Y/N)</td>
<td>N</td>
</tr>
<tr>
<td>TLB Tag</td>
<td>0x02</td>
<td>PPN</td>
<td>0x68</td>
</tr>
</tbody>
</table>

3. Physical address (one bit per box)

Physical address: 01100 0010 1010 0100

| 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
Problem 10. (10 points):

Concurrency, races, and synchronization. Consider a simple concurrent program with the following specification: The main thread creates two peer threads, passing each peer thread a unique integer thread ID (either 0 or 1), and then waits for each thread to terminate. Each peer thread prints its thread ID and then terminates.

Each of the following programs attempts to implement this specification. However, some are incorrect because they contain a race on the value of myid that makes it possible for one or more peer threads to print an incorrect thread ID. Except for the race, each program is otherwise correct.

You are to indicate whether or not each of the following programs contains such a race on the value of myid. You will be graded on each subproblem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 2 points.
- If you circle the wrong answer, you get −1 points (so don’t just guess wildly).
A. Does the following program contain a race on the value of `myid`?

```c
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    Free(vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i, *ptr;
    for (i = 0; i < 2; i++) {
        ptr = Malloc(sizeof(int));
        *ptr = i;
        Pthread_create(&tid[i], 0, foo, ptr);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

Yes
No

B. Does the following program contain a race on the value of `myid`?

```c
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;
    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], NULL, foo, &i);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
}
```

Yes
No
A. Does the following program contain a race on the value of `myid`?

```c
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    Free(vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i, *ptr;
    for (i = 0; i < 2; i++) {
        ptr = Malloc(sizeof(int));
        *ptr = i;
        Pthread_create(&tid[i], 0, foo, ptr);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

Yes

B. Does the following program contain a race on the value of `myid`?

```c
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;
    for (i = 0; i < 2; i++) {
        Pthread_create(&tid[i], NULL, foo, &i);
        Pthread_join(tid[0], NULL);
        Pthread_join(tid[1], NULL);
    }
}
```

Yes
C. Does the following program contain a race on the value of `myid`?

```c
void *foo(void *vargp) {
    int myid;
    myid = (int)vargp;
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], 0, foo, i);
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

**Yes**  
**No**

D. Does the following program contain a race on the value of `myid`?

```c
sem_t s; /* semaphore s */

void *foo(void *vargp) {
    int myid;
    P(&s);
    myid = *((int *)vargp);
    V(&s);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;

    sem_init(&s, 0, 1); /* S=1 INITIALLY */

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], 0, foo, &i);
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

**Yes**  
**No**
C. Does the following program contain a race on the value of `myid`?  

```c
void *foo(void *vargp) {
    int myid;
    myid = (int)vargp;
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;
    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], 0, foo, i);
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

Yes  
No

D. Does the following program contain a race on the value of `myid`?  

```c
sem_t s; /* semaphore s */

void *foo(void *vargp) {
    int myid;
    P(&s);
    myid = *((int *)vargp);
    V(&s);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;
    sem_init(&s, 0, 1); /* S=1 INITIALLY */
    for (i = 0; i < 2; i++) {
        Pthread_create(&tid[i], 0, foo, &i);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

Yes
No

The mutex doesn't actually protect `myid` (why?)

`myid` is passed directly on the stack.
E. Does the following program contain a race on the value of `myid`? Yes No

```c
sem_t s; /* semaphore s */

void *foo(void *vargp) {
  int myid;
  myid = *((int *)vargp);
  V(&s);
  printf("Thread %d\n", myid);
}

int main() {
  pthread_t tid[2];
  int i;

  sem_init(&s, 0, 0); /* S=0 INITIALLY */

  for (i = 0; i < 2; i++) {
    Pthread_create(&tid[i], 0, foo, &i);
    P(&s);
  }
  Pthread_join(tid[0], 0);
  Pthread_join(tid[1], 0);
}
```
E. Does the following program contain a race on the value of `myid`? Yes  No

```c
sem_t s; /* semaphore s */

void *foo(void *vargp) {
    int myid;
    myid = *((int*)vargp);
    V(&s);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;

    sem_init(&s, 0, 0); /* S=0 INITIALLY */

    for (i = 0; i < 2; i++) {
        Pthread_create(&tid[i], 0, foo, &i);
        P(&s);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

does protect `myid` (why?)