# 15/18-213 Final Review Session 2014

| Time | Topic | Question |
|---|---|---|
| 2:00 | Process Control | S11FinQ4<br>S09E2Q3 |
| 2:30 | Cache | S09FinQ3 |
| 3:00 | File I/O | F07Q4 |
| 3:30 | Signals | F01Q9 |
| 4:00 | Synchronization | F11Q11 |
| 4:30 | Assembly (with function pointers) | S11FinQ3 |
| 5:00 | Stack | F10FinQ6 |
| 5:30 | VM | F12FinQ9 |

Review Problem 1

## Problem 4. (14 points):

*Process control.*

Consider the following C program:

```
int main()
{
    pid_t pid;
    int status, counter = 4;

    while(counter > 0)
    {
        pid = fork();

        if(pid)
        {
            counter /= 2;
        }
        else
        {
            printf("%d", counter);  /* (1) */
            break;
        }
    }

    if(pid)
    {
        waitpid(-1, &status, 0);
        counter += WEXITSTATUS(status);

        waitpid(-1, &status, 0);
        counter += WEXITSTATUS(status);

        printf(";%d", counter);     /* (2) */
    }

    return counter;
}
```

Use the following assumptions to answer the questions:

- All processes run to completion, and no system calls fail.
- `printf` is atomic and calls `fflush(stdout)` after printing its argument(s) but before returning.

For each question, there may be more blanks than necessary.

A. List every individual digit that can be emitted by a call to `printf`. Include any digits that can be printed along with the semicolon by the `printf` annotated with `(2)`. For example, if `1521;3` were a possible output of the program, the solutions would include `1`, `2`, `3`, and `5`.

________ ________ ________ ________

________ ________ ________ ________

B. Notice that the `printf` annotated with `(2)` emits a semicolon in addition to a digit. List all of the digit sequences that can be printed *before* the semicolon is emitted. For example, if `1521;3` were a possible output of the program, `1521` would be one solution.

________ ________ ________ ________

________ ________ ________ ________

________ ________ ________ ________

C. Now list all of the digit sequences that can be printed *after* the semicolon is emitted.

________ ________ ________ ________

________ ________ ________ ________

Review Problem 2

## Problem 3. (15 points):

Suppose we have the following two .c files:

**alarm.c**

```
int counter;

void sigalrm_handler (int num) {
  counter += 1;
}

int main (void) {
  signal(SIGALRM, &sigalrm_handler);
  counter = 2;
  alarm(1);
  sleep(1);
  counter -= 3;
  exit(counter);
  return 0;
}
```

**fork.c**

```
int counter;

void sigchld_handler(int num) {
  int i;
  wait(&i);
  counter += WEXITSTATUS(i);
}

int main (void) {
  signal(SIGCHLD, &sigchld_handler);
  counter = 3;
  if (!fork()) {
    counter++;
    execl("alarm", "alarm", NULL);
  }
  sleep(2);
  counter *= 3;
  printf("%d\n", counter);
  exit(0);
}
```

Assume that all system calls succeed and that all C arithmetic statements are atomic.

The files are compiled as follows:

```
gcc -o alarm alarm.c
gcc -o fork fork.c
```

Suppose we run `./fork` at the terminal. What are the possible outputs to the terminal?

Review Problem 3

## Problem 3. (20 points):

We consider a 128 byte data cache that is 2-way associative and can hold 4 doubles in every cache line. A double is assumed to require 8 bytes.

For the below code we assume a cold cache. Further, we consider an array A of 32 doubles that is cache-aligned (that is, A[0] is loaded into the first slot of a cache line in the first set). All other variables are held in registers. The code is parameterized by positive integers m and n that satisfy m*n = 32 (i.e., if you know one you know the other).

Recall that miss rate is defined as $\frac{\#\text{misses}}{\#\text{accesses}}$.

```
float A[32], t = 0;
for(int i = 0; i < m; i++)
  for(int j = 0; j < n; j++)
    t += A[j*m + i];
```

Answer the following:

1. How many doubles can the cache hold?
2. How many sets does the cache have?
3. For m = 1:
   (a) Determine the miss rate.

   (b) What kind of misses occur?

   (c) Does the code have temporal locality with respect to accesses of A and this cache?

4. For `m = 2`:

   (a) Determine the miss rate.

   (b) What kind of misses occur?

5. For `m = 16`:

   (a) Determine the miss rate.

   (b) What kind of misses occur?

   (c) Does the code have spatial locality with respect to accesses of `A` and this cache?

Review Problem 4

## Problem 4. (9 points):

Consider the C code below:

```
int fdplay() {
    int pid;
    int fd1, fd2;

    fd1 = open("/file1", O_RDWR);
    dup2(fd1, 1);
    printf("A");
    if ((pid = fork()) == 0) {
        printf("B");
        fd2 = open("/file1", O_RDWR);
        dup2(fd2, 1);
        printf("C");
        /* POINT X */
    } else {
        waitpid(pid, NULL, 0);
        printf("D");
        close(fd1);
        printf("E");
    }
    exit(2);
}
```

A. How many processes share the open file structure referred to by `fd1` at "`POINT X`" in the code?

B. How many file descriptors (total among all processes) share the open file structure referred to by `fd1` at "`POINT X`" in the code?

C. Assuming that `/file1` was empty before running this code, what are its contents after the execution is complete?

Review Problem 5

## Problem 9. (16 points):

This problem tests your understanding of exceptional control flow in C programs. Assume we are running code on a Unix machine. The following problems all concern the value of the variable `counter`.

### Part I (6 points)

```
int counter = 0;

int  main()
{
    int i;

    for (i = 0; i < 2; i ++){
        fork();
        counter ++;
        printf("counter = %d\n", counter);
    }

    printf("counter = %d\n", counter);
    return 0;
}
```

A. How many times would the value of `counter` be printed: _______________

B. What is the value of `counter` printed in the first line? _______________

C. What is the value of `counter` printed in the last line? _______________

**Part II (6 points)**

```
pid_t pid;
int counter = 0;

void handler1(int sig)
{
    counter ++;
    printf("counter = %d\n", counter);
    fflush(stdout);   /* Flushes the printed string to stdout */
    kill(pid, SIGUSR1);
}

void handler2(int sig)
{
    counter += 3;
    printf("counter = %d\n", counter);
    exit(0);
}

main() {
    signal(SIGUSR1, handler1);
    if ((pid = fork()) == 0) {
        signal(SIGUSR1, handler2);
        kill(getppid(), SIGUSR1);
        while(1) {};
    }
    else {
        pid_t p; int status;
        if ((p = wait(&status)) > 0) {
            counter += 2;
            printf("counter = %d\n", counter);
        }
    }
}
```

What is the output of this program?

**Part III (4 points)**

```
int counter = 0;

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


int  main()
{
    int i;

    signal(SIGCHLD, handler);

    for (i = 0; i < 5; i ++){
        if (fork() == 0){
            exit(0);
        }
    }

    /* wait for all children to die */
    while (wait(NULL) != -1);

    printf("counter = %d\n", counter);
    return 0;
}
```

A. Does the program output the same value of `counter` every time we run it?  `Yes`  `No`

B. If the answer to A is `Yes`, indicate the value of the `counter` variable. Otherwise, list all possible values of the `counter` variable.

Answer: `counter` = ________________________

Review Problem 6

## Problem 11. (9 points):

*Synchronization.* This problem is about using semaphores to synchronize access to a shared bounded FIFO queue in a producer/consumer system with an arbitrary number of producers and consumers.

- The queue is initially empty and has a capacity of 10 data items.
- Producer threads call the `insert` function to insert an item onto the rear of the queue.
- Consumer threads call the `remove` function to remove an item from the front of the queue.
- The system uses three semaphores: `mutex`, `items`, and `slots`.

Your task is to use P and V semaphore operations to correctly synchronize access to the queue.

A. What is the initial value of each semaphore?

`mutex =` ________

`items =` ________

`slots =` ________

B. Add the appropriate P and V operations to the psuedo-code for the `insert` and `remove` functions:

```
void insert(int item)
{
    /* Insert sem ops here */







    add_item(item);
    /* Insert sem ops here */






}
```

```
int remove()
{
    /* Insert sem ops here */







    item = remove_item();
    /* Insert sem ops here */






    return item;
}
```

Review Problem 7

# Problem 3. (20 points):

*Assembly/C translation.*

Consider the following C code and assembly code for a curiously-named function:

```
typedef struct node
{
    void *data;
    struct node *next;
} node_t;

node_t *lmao(node_t *n, int f(node_t *))
{
    node_t *a, *b;

    if(________________)
    {
        return NULL;
    }

    a = ________________;

    if(________________)
    {
        b = ________________;
        b->data = n->data;
        b->next = ________________;
        return b;
    }

    return ________________;
}
```

```
0x4005d0:  mov    %rbx,-0x18(%rsp)
0x4005d5:  mov    %rbp,-0x10(%rsp)
0x4005da:  xor    %eax,%eax
0x4005dc:  mov    %r12,-0x8(%rsp)
0x4005e1:  sub    $0x18,%rsp
0x4005e5:  test   %rdi,%rdi
0x4005e8:  mov    %rdi,%rbx
0x4005eb:  mov    %rsi,%rbp
0x4005ee:  je     0x40061e <lmao+78>
0x4005f0:  mov    0x8(%rdi),%rdi
0x4005f4:  callq  0x4005d0 <lmao>
0x4005f9:  mov    %rbx,%rdi
0x4005fc:  mov    %rax,%r12
0x4005ff:  callq  *%rbp
0x400601:  mov    %eax,%edx
0x400603:  mov    %r12,%rax
0x400606:  test   %edx,%edx
0x400608:  jle    0x40061e <lmao+78>
0x40060a:  mov    $0x10,%edi
0x40060f:  callq  0x400498 <malloc>
0x400614:  mov    (%rbx),%rdx
0x400617:  mov    %r12,0x8(%rax)
0x40061b:  mov    %rdx,(%rax)
0x40061e:  mov    (%rsp),%rbx
0x400622:  mov    0x8(%rsp),%rbp
0x400627:  mov    0x10(%rsp),%r12
0x40062c:  add    $0x18,%rsp
0x400630:  retq
```

Using your knowledge of C and assembly, fill in the blanks in the C code for `lmao` with the appropriate expressions. (Note: `0x400498` is the address of the C standard library function `malloc`.)

Review Problem 8

## Problem 6. (0xa points):

*The stack discipline.* This problem deals with stack frames in Intel IA-32 machines. Consider the following C function and corresponding assembly code.

```c
struct node_t;
typedef struct node_t{
    void * elem;
    struct node_t *left;
    struct node_t *right;
} node;

void oak(node * tree, void (*printFunc)(node *)){
    /*POINT A*/
    (*printFunc)(tree);
    if (tree->left) {
        /*POINT B*/
        oak(tree->left,printFunc);
    }
    if (tree->right) {
        oak(tree->right,printFunc);
    }
}
```

```
00000000 <oak>:
 0: 55                  push   %ebp
 1: 89 e5               mov    %esp,%ebp
 3: 83 ec 18            sub    $0x18,%esp
 6: 89 5d f8            mov    %ebx,0xfffffff8(%ebp)
 9: 89 75 fc            mov    %esi,0xfffffffc(%ebp)
 c: 8b 5d 08            mov    0x8(%ebp),%ebx
 f: 8b 75 0c            mov    0xc(%ebp),%esi
12: 89 1c 24            mov    %ebx,(%esp)
           /*POINT A*/
15: ff d6               call   *%esi
17: 8b 43 04            mov    0x4(%ebx),%eax
1a: 85 c0               test   %eax,%eax
1c: 74 0c               je     2a <oak+0x2a>
1e: 89 74 24 04         mov    %esi,0x4(%esp)
22: 89 04 24            mov    %eax,(%esp)
           /*POINT B*/
25: e8 fc ff ff ff      call   26 <oak+0x26>
2a: 8b 43 08            mov    0x8(%ebx),%eax
2d: 85 c0               test   %eax,%eax
2f: 74 0c               je     3d <oak+0x3d>
31: 89 74 24 04         mov    %esi,0x4(%esp)
35: 89 04 24            mov    %eax,(%esp)
38: e8 fc ff ff ff      call   39 <oak+0x39>
3d: 8b 5d f8            mov    0xfffffff8(%ebp),%ebx
40: 8b 75 fc            mov    0xfffffffc(%ebp),%esi
43: 89 ec               mov    %ebp,%esp
45: 5d                  pop    %ebp
46: c3                  ret
```

*(over)*

Please draw a picture of the stack frame, starting with any arguments that might be placed on the stack for the `oak` function, showing the stack at each of points `A`, and `B`, as specified in the code above. Your diagram should only include actual values where they are known, if you do not know the value that will be placed on the stack, simply label what it is (i.e., ”old ebp”).

**Stack A:**

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

**Stack B:**

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

Review Problem 9
**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**.

| TLB | | | |
|---|---|---|---|
| Index | Tag | PPN | Valid |
| 0 | 03 | C3 | 1 |
| | 01 | 71 | 0 |
| 1 | 00 | 28 | 1 |
| | 01 | 35 | 1 |
| 2 | 02 | 68 | 1 |
| | 3A | F1 | 0 |
| 3 | 03 | 12 | 1 |
| | 02 | 30 | 1 |
| 4 | 7F | 05 | 0 |
| | 01 | A1 | 0 |
| 5 | 00 | 53 | 1 |
| | 03 | 4E | 1 |
| 6 | 1B | 34 | 0 |
| | 00 | 1F | 1 |
| 7 | 03 | 38 | 1 |
| | 32 | 09 | 0 |

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

## 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 |

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

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 |

| 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)

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

2. Address translation

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

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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | |

**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

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

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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | |