15213/15513/18213 Final Exam Review Recitation
Topics

- Virtual Memory
- I/O Redirection
- Threading
- Processes and Signals
- Deadlock

- Note: other topics may appear on the final exam!
Virtual Memory

Final Exam Fall 2012

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>Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Tag</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
</tr>
<tr>
<td></td>
<td>3A</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
</tr>
<tr>
<td></td>
<td>02</td>
</tr>
<tr>
<td>4</td>
<td>7F</td>
</tr>
<tr>
<td></td>
<td>01</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>03</td>
</tr>
<tr>
<td>6</td>
<td>1B</td>
</tr>
<tr>
<td></td>
<td>00</td>
</tr>
<tr>
<td>7</td>
<td>03</td>
</tr>
<tr>
<td></td>
<td>02</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></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

   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)

<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>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 |
   |----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
   |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |
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 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|
Problem 6. (10 points):

File I/O

The following problems refer to a file called numbers.txt, with contents the ASCII string 0123456789.
You may assume calls to read() are atomic with respect to each other. The following file, read_and_print_one.h, is compiled with each of the following code files.

```c
#ifndef READ_AND_PRINT_ONE
#define READ_AND_PRINT_ONE
#include <stdio.h>
#include <unistd.h>

static inline void read_and_print_one(int fd) {
    char c;
    read(fd, &c, 1);
    printf("%c", c); fflush(stdout);
}
#endif
```
A. List all outputs of the following code.

```c
#include "read_and_print_one.h"
#include <stdlib.h>
#include <fcntl.h>

int main() {
    int file1 = open("numbers.txt", O_RDONLY);
    int file2;
    int file3 = open("numbers.txt", O_RDONLY);
    file2 = dup2(file3, file2);

    read_and_print_one(file1);
    read_and_print_one(file2);
    read_and_print_one(file3);
    read_and_print_one(file2);
    read_and_print_one(file1);
    read_and_print_one(file3);

    return 0;
}
```
B. List all outputs of the following code.

```c
#include "read_and_print_one.h"
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    int file1;
    int file2;
    int file3;
    int pid;

    file1 = open("numbers.txt", O_RDONLY);
    file3 = open("numbers.txt", O_RDONLY);

    file2 = dup2(file3, file2);
    read_and_print_one(file1);
    read_and_print_one(file2);

    pid = fork();
    if (!pid) {
        read_and_print_one(file3);
        close(file3);
        file3 = open("numbers.txt", O_RDONLY);
        read_and_print_one(file3);
    } else {
        wait(NULL);
        read_and_print_one(file3);
        read_and_print_one(file2);
        read_and_print_one(file1);
    }

    read_and_print_one(file3);

    return 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:
A. Does the following program contain a race on the value of myid?  

Yes  No

```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 = Alloc(sizeof(int));
        *ptr = i;
        Pthread_create(&tid[i], 0, foo, ptr);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```
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);
}
```

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

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

t 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);
}
```
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);
}"
```
Processes and Signals

Final Exam Fall 2012

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
Deadlock

Final Exam Spring 2011

Problem 7. (14 points):

Deadlocks and Dreadlocks

Two threads (X and Y) access shared variables A and B protected by mutex_a and mutex_b respectively. Assume all variable are declared and initialized correctly.

Thread X
P(&mutex_a);
A += 10;
P(&mutex_b);
B += 20;
V(&mutex_b);
A += 30;
V(&mutex_a);

Thread Y
P(&mutex_b);
B += 10;
P(&mutex_a);
A += 20;
V(&mutex_a);
B += 30;
V(&mutex_b);

A. Show an execution of the threads resulting in a deadlock. Show the execution steps as follows

<table>
<thead>
<tr>
<th>Thread X</th>
<th>Thread Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(&amp;mutex_a)</td>
<td>P(&amp;mutex_b)</td>
</tr>
<tr>
<td>A+= 10</td>
<td>B+= 10</td>
</tr>
<tr>
<td>P(&amp;mutex_b)</td>
<td>P(&amp;mutex_a)</td>
</tr>
<tr>
<td>B+= 20</td>
<td>A+= 20</td>
</tr>
<tr>
<td>V(&amp;mutex_b)</td>
<td>V(&amp;mutex_a)</td>
</tr>
<tr>
<td>A+= 30</td>
<td>B+= 30</td>
</tr>
</tbody>
</table>

Answer:
B. There are different approaches to solve the deadlock problem. Modify the code above to show two approaches to prevent deadlocks. You can declare new mutex variables if required. Do not change the order or amount of the increments to A and B. Rather, change the locking behavior around them. The final values of A and B must still be guaranteed to be incremented by 60.

Answer:
Questions?