15-213 Recitation A
Final Exam Review Session

```
int main() {
      if (fork() == 0) {
            printf("a");
      }
      else {
            printf("b");
            waitpid(-1, NULL, 0);
      }
      printf("c");
      exit(0);
}
```

What are the possible outputs of this code? (You may assume that all processes and function calls complete successfully.)

Note in case you talk this over with classmates or look at other slides: this question is subtly different from the one the other sections are doing.

based on Spring 2011, exam 2

Problem 4. (9 points):

Signals. Consider the following three different snippets of C code. Assume that all functions and procedures return correctly and that all variables are declared and initialized properly. Also, assume that an arbitrary number of SIGINT signals, and only SIGINT signals, can be sent to the code snippets randomly from some external source.

For each code snippet, circle the value(s) of `i` that could possibly be printed by the `printf` command at the end of each program. *Careful: There may be more than one correct answer for each question. Circle all the answers that could be correct.*

Code Snippet 1:

```
int i = 0;

void handler(int sig) {
  i = 0;
}

int main() {
  int j;

  signal(SIGINT, handler);
  for (j=0; j < 100; j++) {
    i++;
    sleep(1);
  }
  printf("i = %d\n", i);
  exit(0);
}
```

1. Circle possible values of `i` printed by snippet 1:

1. 0
2. 1
3. 50
4. 100
5. 101
6. Terminates with no output.

Code Snippet 2:

```
int i = 0;

void handler(int sig) {
  i = 0;
}

int main () {
  int j;
  sigset_t s;

  signal(SIGINT, handler);

  /* Assume that s has been
     initialized and declared
     properly for SIGINT */

  sigprocmask(SIG_BLOCK, &s, 0);
  for (j=0; j < 100; j++) {
    i++;
    sleep(1);
  }
  sigprocmask(SIG_UNBLOCK, &s, 0);
  printf("i = %d\n", i);
  exit(0);
}
```

2. Circle possible values of `i` printed by snippet 2:

1. 0
2. 1
3. 50
4. 100
5. 101
6. Terminates with no output.

Code Snippet 3:

```
int i = 0;

void handler(int sig) {
  i = 0;
  sleep(1);
}


int main () {
  int j;
  sigset_t s;

  /* Assume that s has been
     initialized and declared
     properly for SIGINT */

  sigprocmask(SIG_BLOCK, &s, 0);
  signal(SIGINT, handler);
  for (j=0; j < 100; j++) {
    i++;
    sleep(1);
  }
  printf("i = %d\n", i);
  sigprocmask(SIG_UNBLOCK, &s, 0);
  exit(0);
}
```

3. Circle possible values of `i` printed by snippet 3:

1. 0
2. 1
3. 50
4. 100
5. 101
6. Terminates with no output.

from Fall 2010, final exam

Problem 8. (6 points):

Processes vs. threads. This problem tests your understanding of the some of the important differences between processes and threads. Consider the following C program:

```
#include "csapp.h"

/* Global variables */
int cnt;
sem_t mutex;


/* Helper function */
void *incr(void *vargp)
{
    P(&mutex);
    cnt++;
    V(&mutex);
    return NULL;
}
```

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

    sem_init(&mutex, 0, 1); /* mutex=1 */

    /* Processes */
    cnt = 0;
    for (i=0; i<2; i++) {
        incr(NULL);
        if (fork() == 0) {
            incr(NULL);
            exit(0);
        }
    }
    for (i=0; i<2; i++)
        wait(NULL);
    printf("Procs:   cnt = %d\n", cnt);

    /* Threads */
    cnt = 0;
    for (i=0; i<2; i++) {
        incr(NULL);
        pthread_create(&tid[i], NULL, incr, NULL);
    }
    for (i=0; i<2; i++)
        pthread_join(tid[i], NULL);
    printf("Threads: cnt = %d\n", cnt);

    exit(0);
}
```

A. What is the output of this program?

```
Procs:   cnt = ___

Threads: cnt = ___
```

from Spring 2008, exam 2

Problem 4. (12 points):

The following problem concerns virtual memory and the way virtual addresses are translated into physical addresses. Below are the specifications of the system on which the translation occurs.

- The system is a 32-bit machine - words are 4 bytes.
- Memory is byte addressable.
- The maximum size of a virtual address space is 4GB.
- The system is configured with 64MB of physical memory.
- The page size is 4KB.
- The system uses a two-level page tables. Tables at both levels are 4096 bytes (1 page) and entries in both tables are 4 bytes as shown below.

In this problem, you are given parts of a memory dump of this system running 2 processes. In each part of this question, one of the processes will issue a single memory operation (read or write of one byte) to a single virtual address (as indicated in each part). Your job is to figure out which physical addresses are accessed by the process if any, or determine if an error is encountered.

Entries in the first and second level tables have in their low-order bits flags denoting various access permissions.

31		2	1	0
Page Table Base Address				P

Page Directory Entry

31		2	1	0
Page Address		U	W	P

Page Table Entry

- P = 1 $\Rightarrow$ Present
- W = 1 $\Rightarrow$ Writable
- U = 1 $\Rightarrow$ User-mode

The contents of relevant sections of memory is shown on the next page. All numbers are given in **hexadecimal**.

Address	Contents
001AC021	07693003
001AC084	00142003
0021A020	0481C001
0021A080	04A95001
0021A2FF	06128001
0021A300	05711001
0021ABFC	05176001
0021AC00	001AC001
0021B020	01FAC9DA
0021B080	052DB001
0021B2C0	0B2B36C2
0021B2FF	05A11001
0021B300	01FCF001
0021BBFC	06213001
0021BC00	001AC001
01FCF021	00382003
0481C048	0523A005
04A95048	048B8005
04A95120	07D6A005
051760F0	0E33F007
051763C0	08BF1007
052DB04A	09A62006
052DB128	0D718006
05711021	00113003
05A110F0	01133007
061280F0	0A114007
0614504A	0B183006
062133C0	052F1007

For the purposes of this problem, omitted entries have contents = 0.

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;
}
```