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:**
```c
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:**
```c
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:**
```c
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.
Problem 8. (6 points):

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

```c
#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 = ____
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.

<table>
<thead>
<tr>
<th>31</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Table Base Address</td>
<td>P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page Directory Entry

<table>
<thead>
<tr>
<th>31</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Address</td>
<td>U</td>
<td>W</td>
<td>P</td>
</tr>
</tbody>
</table>

Page Table Entry

- P = 1 ⇒ Present
- W = 1 ⇒ Writable
- U = 1 ⇒ User-mode

The contents of relevant sections of memory is shown on the next page. All numbers are given in hexadecimal.
<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>001AC021</td>
<td>07693003</td>
</tr>
<tr>
<td>001AC084</td>
<td>00142003</td>
</tr>
<tr>
<td>0021A020</td>
<td>0481C001</td>
</tr>
<tr>
<td>0021A080</td>
<td>04A95001</td>
</tr>
<tr>
<td>0021A2FF</td>
<td>06128001</td>
</tr>
<tr>
<td>0021A300</td>
<td>05711001</td>
</tr>
<tr>
<td>0021ABFC</td>
<td>05176001</td>
</tr>
<tr>
<td>0021AC00</td>
<td>001AC001</td>
</tr>
<tr>
<td>0021B020</td>
<td>01FAC9DA</td>
</tr>
<tr>
<td>0021B080</td>
<td>052DB001</td>
</tr>
<tr>
<td>0021B2C0</td>
<td>0B2B36C2</td>
</tr>
<tr>
<td>0021B2FF</td>
<td>05A11001</td>
</tr>
<tr>
<td>0021B300</td>
<td>01FCF001</td>
</tr>
<tr>
<td>0021BBFC</td>
<td>06213001</td>
</tr>
<tr>
<td>0021BC00</td>
<td>001AC001</td>
</tr>
<tr>
<td>01FCF021</td>
<td>00382003</td>
</tr>
<tr>
<td>0481C048</td>
<td>0523A005</td>
</tr>
<tr>
<td>04A95048</td>
<td>048B8005</td>
</tr>
<tr>
<td>04A95120</td>
<td>07D6A005</td>
</tr>
<tr>
<td>051760F0</td>
<td>0E33F007</td>
</tr>
<tr>
<td>051763C0</td>
<td>08BF1007</td>
</tr>
<tr>
<td>052DB04A</td>
<td>09A62006</td>
</tr>
<tr>
<td>052DB128</td>
<td>0D718006</td>
</tr>
<tr>
<td>05711021</td>
<td>00113003</td>
</tr>
<tr>
<td>05A110F0</td>
<td>01130007</td>
</tr>
<tr>
<td>061280F0</td>
<td>0A114007</td>
</tr>
<tr>
<td>0614504A</td>
<td>0B183006</td>
</tr>
<tr>
<td>062133C0</td>
<td>052F1007</td>
</tr>
</tbody>
</table>

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 pseudo-code for the `insert` and `remove` functions:

```c
void insert(int item) { /* Insert sem ops here */
    add_item(item);
    /* Insert sem ops here */
}
```

```c
int remove() { /* Insert sem ops here */
    item = remove_item();
    /* Insert sem ops here */
    return item;
}
```