CS 31 Homework 7: Processes, Paging, and Virtual Memory

Due Date: Friday, May 3rd, 11:59pm
Names, userids, and lab sections: (please update these on gradescope as well before submission)

Question 1

For the code snippet shown below (assume that all the calls to `fork()` succeed), answer the following questions:

A. Draw a process hierarchy diagram that results from the execution of the code shown below. Your diagram should be similar to Figure 2 in Section 13.2 of the textbook, where you draw a node for every process and arrows from parent to child processes.

- Label each node with a letter to indicate the order in which it was spawned. In cases where the order is not determined, choose a possible order.
- Next to each node, write the output value(s) that the process prints out with `printf()`.

B. After this code executes, are there any zombie processes? Explain your answer in a sentence or two.

```
int i = 0;
pid_t pid;

printf("%d ", i);
for(i = 1; i < 3; i++) {
   pid = fork();
   printf("%d ", i);
}

if(pid != 0) {
  wait(NULL);
} else {
  exit(0);
}
```

Question 2

Consider the code snippet shown below (and assume all calls to `fork()` succeed).

A. Draw the execution timeline corresponding to the code's execution showing a possible ordering of `fork()` and `wait()` calls from the proceses involved. Use Figure 7 in Section 13.2 of the textbook as an example. Note: There are no newlines in the `printf` calls below, so the output should all be on a single line.

```
pid_t pid1, pid2;

printf("1 ");
pid1 = fork();
if (pid1 == 0) {
  pid2 = fork();
  printf("2 ");
  if (pid2 == 0) {
    printf("3 ");
    exit(0);
  } else {
    printf("4 ");
    wait(NULL);
    printf("5 ");
    exit(0);
  }
} else {
  printf("6 ");
  wait(NULL);
  printf("7 ");
}
```

B. Which of the following outputs below are possible from executing the above code? For any that are not, describe in one sentence why not.

i) `1 6 2 3 2 4 7 5`

ii) `1 2 2 4 5 3 6 7`

iii) `1 2 2 3 4 5 6 7`

iv) `1 6 2 4 2 3 5 7`

Question 2

For each of the following x86-64 instructions, indicate whether the instruction **could** cause a page fault, whether it **could** cause a cache miss, and whether it **could** cause the dirty bit in the cache to be set to 1. HINT: Think of whether a particular instruction **could** result in a read or write from main memory, the cache or the registers.

a. `movq $7, %rcx`

```
Page fault? YES or NO   |   Cache miss? YES or NO   |   Dirty bit? YES or NO
```

b. `movq $7, (%rdx)`

```
Page fault? YES or NO   |   Cache miss? YES or NO   |   Dirty bit? YES or NO
```

c. `movq (%rax), %rbx`

```
Page fault? YES or NO   |   Cache miss? YES or NO   |   Dirty bit? YES or NO
```

d. `addq %rbx, -8(%rax)`

```
Page fault? YES or NO   |   Cache miss? YES or NO   |   Dirty bit? YES or NO
```

Question 3

For the rest of the assignment, you will be tracing memory accesses in a system with the following architecture:

- 8-bit virtual addresses
- 16-byte page size
- 4 pages of physical RAM

a. How many bytes of data can a single process store in:

(a) physical memory (b) virtual memory

b. For each the given virtual addresses, divide the address into the `page number` and `page offset`.

`1 0 0 0 1 0 1 0` `1 0 0 0 1 1 1 1` `1 0 1 0 1 0 1 0`

c. On the accompanying Page Table diagram, show the results of the following memory operations on RAM and the Page Table. Assume first-in-first-out replacement. You can also assume that there is only one process accessing memory during these 10 memory operations.

Within each box, time should progress downward, so the first address loaded appears at the top and subsequent changes are written below. To the right of the table, label each change with number of the operation that caused it. Annotate each operation below with *hit* or *page fault* to indicate whether the data was found in physical memory. Don't forget to update the valid bits, especially when a page is kicked out!

(a)	`read`	`0 0 0 1 1 0 1 0`	(f)	`write`	`0 0 0 0 1 0 0 1`
(b)	`write`	`0 0 0 1 1 0 1 1`	(g)	`read`	`0 0 0 0 0 0 0 0`
(c)	`read`	`1 1 1 1 1 0 0 0`	(h)	`read`	`0 1 0 1 0 1 1 1`
(d)	`read`	`1 1 1 1 1 0 1 0`	(i)	`write`	`0 0 0 1 1 0 1 0`
(e)	`read`	`0 1 1 0 1 0 0 0`	(j)	`read`	`0 0 0 1 1 1 0 1`

Use the Page Table (on the next page) and the map of RAM (below) to help keep track of virtual memory.

RAM

frame #	page #
`00`	
`01`	
`10`	
`11`	

Page Table

index	V	frame	notes
0	0		
1	0		
...	...	...	
5	0		
6	0		
...	...	...	
15	0		