Name:_____

5. *Fork* (**5** points) Consider this code using Linux's `fork`:

```
int x = 7;
if(fork()) {
   x++;
   printf(" %d ", x);
   fork();
   x++;
   printf(" %d ", x);
} else {
  printf(" %d ", x);
}
```

What are *all* the different possible outputs (order of things printed) for this code? (Hint: There are four of them.)

Solution:

```
7 8 9 9
8 7 9 9
8 9 7 9
8 9 9 7
```

Note: If you actually try this out, you may see 5 numbers printed, which is rather surprising. The issue is the implementation of `printf` may *buffer* output and the second `fork` call in the code then copies the not-empty output buffer into the child process. You can fix this in the code by putting `fflush(stdout);` after the first call to `printf`. In terms of the exam, the four outputs above *are* all still possible and of course we didn't expect you to also list the fifth possible output that arises from this buffered-output issue.

SID: _____

Question 4: Caches (11 pts)

We have a 64 KiB address space and two possible data caches. Both are 1 KiB, direct-mapped caches with random replacement and write-back policies. **Cache X** uses 64 B blocks and **Cache Y** uses 256 B blocks.

a) Calculate the TIO address breakdown for **Cache X**: [1.5 pts]

Tag	Index	Offset
6	4	6

b) During some part of a running program, **Cache Y**'s management bits are as shown below. Four options for the next two memory accesses are given (R = read, W = write). Circle the option that results in data from the cache being *written to memory*. [2 pts]

~~Slot~~ Line	Valid	Dirty	Tag
00	0	0	1000 01
01	1	1	0101 01
10	1	0	1110 00
11	0	0	0000 11

(1) R 0x4C00, W 0x5C00
(R then W into ~~slot~~ line 00)

(2) W 0x5500, W 0x7A00
(W into dirty slot 01 – tag matches, W into ~~slot~~ line 10)

(3) W 0x2300, R 0x0F00
(W into ~~slot~~ line 11, then kick dirty block out)

(4) R 0x3000, R 0x3000
(2 reads into non-dirty ~~slot~~ line 00)

c) The code snippet below loops through a character array. Give the value of LEAP that results in a Hit Rate of 15/16 for **Cache Y**. [4 pts]

```
#define ARRAY_SIZE 8192
char string[ARRAY_SIZE];              // &string = 0x8000
for(i = 0; i < ARRAY_SIZE; i += LEAP)
      string[i] |= 0x20;              // to lower
```

32

Access pattern is R then W for each address. To get a hit rate of 15/16, need to access exactly 8 addresses per block (compulsory miss on first R, then followed by all hits). Since block size for Cache Y is 256 B and char size is 1 B (256 array elements per block), we need our LEAP to be 256/8 = 32.

d) For the loop shown in part (c), let LEAP = 64. Circle ONE of the following changes that increases the hit rate of **Cache X**: [2 pts]

Increase Block Size (hit rate ↑) | Increase Cache Size (no change to hit rate) | Add a L2$ (miss penalty ↓) | Increase LEAP (hit rate ↓)

e) For the following cache access parameters, calculate the AMAT. ~~All miss and hit rates are local to that cache level.~~ Please simplify and include units. [1.5 pts]

L1$ Hit Time	L1$ **Miss** Rate	~~L2$ Hit Time~~	~~L2$ **Hit** Rate~~	MEM Hit Time
2 ns	40%	~~20 ns~~	~~95%~~	400 ns

~~AMAT = 2 + 0.4 * (20 + 0.05*400)~~ AMAT = 2 + 0.4 * 400

~~18 ns~~ 162 ns