ALL: CODE ANALYSIS: 251-260 (Not T/F questions)

251 When understanding concurrency, it's good to *draw* the lattice of the code execution (to see the happen-before relationship). From the lattice, I can know which actions are concurrent

252 When drawing the lattice, the basic building blocks are:
- fork() implies a branch: a→b, a→c
- a→b and a→c imply that "b" and "c" are concurrent.
- Concurrent means, a-then-c or c-then-a could happen.
- Essentially, if there is no chain of single bidirectional edges connecting two actions, then the two actions are concurrent
- pthread_create() also implies a branch: a→b, a→c
- waitpid() implies a join: x→z, y→z
- pthread_join() also implies a join: x→z, y→z
- pthread_exit() only kills the thread calling the function (i.e. end of flow for this thread).
- exit() implies that *all* threads will be killed of the process (i.e. the end of the lattice).
- Any thread can call exit().
- When a thread executes a-b-c and another thread executes d-e-f, the first three instructions and the second three instructions can interleave in all possible ways.
- Given these building blocks, you can build the complete lattice of a program execution.

253 sleep(N) means sleep for N seconds. In the lattice, it's good to draw a long N-inch of vertical line to represent sleep(N).

254 In this class, adding sleep() is enough to remove non-determinism. (Although in reality, please use locks or wait calls for proper synchronization).

255 I have practiced the "Which data is shared?" slides. To answer this type of question, the basic building blocks are:
- Draw, draw, draw, i.e. draw the memory layout, e.g. an integer is a box, a pointer is a box that has an edge pointing to another box.
- Understand that a local variable can have multiple instances (more than one boxes), e.g. when the function is called recursively or the function is called by multiple threads.
- When asked if "ptr" is shared, don't just see who owns "ptr", but rather go to the data being pointed and ask whether the data's memory line can be accessed by which threads.
- When asked if "x" (data) is shared, don't just see who owns "x", but rather checks all the pointers pointing to the x's memory line and check who can access those pointers.

256 I am happy that I *DO NOT* have to memorize:
- precise definitions (e.g. critical section, atomicity), as long as I understand general concept,
- lists of system call numbers/functions, signal numbers, exception numbers/functions, etc.,
- every line of the echo server code, the rioBuffer code, the semaphore code , (as long as I understand what the code is trying to achieve),
- signal number and names,
- how paging and page table works,
- every instruction in rio_read()

257 Do you know the output of this code below? ..

```
int main(...)
{
    ...
    int a=1, b=2, c=3;
    printf(" A \n");
    pthread_create(&tid1, NULL, tfunc, &a);
    pthread_create(&tid2, NULL, tfunc, &b);
    pthread_create(&tid3, NULL, tfunc, &c);
    printf(" C \n");
    pthread_join(tid1, NULL);
    printf(" D \n");
    sleep(100);
}
void *tfunc (void *argp) {
    int x =  *((int*)argp);
    sleep(x);
    printf(" S-%d \n", x);
    if (x == 2) {
      exit(0);
    }
}
```

258 How about this one? ...

```
int main(...)
{
  ...
  printf(" A \n");
  ret = fork();
  if (ret == 0) {
    pthread_create(&tid, NULL, tfunc, NULL);
    sleep(2);
    printf(" B \n");
  }
  else {
    printf(" C \n");
    waitpid(ret, NULL, 0);
    printf(" D \n");
  }
}
void *tfunc (void *argp) {
  sleep(1);
  printf(" E \n");
  exit(0);
}
```

259 Have you done all the (a) "fun with file descriptors" examples again including the ones with APPEND flag and lseek() call, (b) all the address tranlation examples again ("mem3" lectures), and (c) all the memory-sharing and deadlock examples again ("sy" lectures)?

260 Okay, I think I'm ready for the exam now!!