Project 4 Questions

* A race condition happens when two processes or threads are trying to update data from within a critical section/area of the .data section of a piece of code.
* A race condition is difficult to reproduce and debug because there is no set amount of time that some calculations are required to complete their work. Because of this, in a multiprogramming scenario we can have many processes or threads altering communal pieces of code in the .data section at seemingly random points in time. This is difficult to track and debug.
* This can be fixed by allowing each thread the ability to have their own memory location that does not conflict with other memory locations. Or we can implement Mutex or Semaphore locks.
* Parallel Programming patterns allow for the programmer to use pre-bottled and ready to use templates, in essence, to write their code. Following the common patterns seen in parallel computing allows us the ability to better code with parallelism in mind. We can use strategies in the pure algorithms that we write or we can use strategies that allow for the composition of the overall program to be in line with the patterns we observe. We can implement strategies that allow for concurrent data access as well as being able to use multiple processing cores to implement concurrency. We use two types of patternlets in parallelism. We use Process Thread control as a built in “methods”. We can use coordination patterns as well which make use of a Message Passing Interface or OpenMP for multithreaded shared memory applications. Message passing allows for two processes to coordinate while we have one more coordination pattern which is mutual exclusion.
* Asdf master worker, barrier, reduction q
* We can find parallelism in the code that we are attempting to write in the Program, Data, and Resource viewpoint. Essentially, we can see that we can work towards programs that have non-conflicting statements that can execute concurrently. We can also find parallelism in repetitive tasks. Finally we can find it in how we implement parallelizable code on the hardware.
* Dependency happens when the input of one operation is the output of another or vice versa. There are three types of dependencies. True dependence, output dependence, and anti-dependence. True (flow) dependence happens when a second statement is dependent on the value of a first statement. Ex. D = 3: a = D. Output dependence happens when the output of the first statement determines the value of a second statement. Ex. A = g(y): a = b. Finally anti-dependence happens when a statement that has already tried to be executed can’t fully execute because its value is dependent on the value of the next statement. D = c: c = 42.
* A statement is dependent when it is dependent on the value of another statement. Both of these statements cannot run independent of each other. A = f’(x) = 8x^2 + 2x + 2: b = a. Here we can see that be cannot run till is computed.
  + - Independence occurs when two statement can be run concurrently with no input or binding behavior of another process. B = a; d = c. There is no relation between these two statements. They are independent and can run concurrently.
* Two statement can be executed in parallel if and only if both statement are independent of each other and have no dependences.
* Dependency can be removed by adding or removing statements that cause dependencies between to statements in a piece of code.
* Computing dependence occurs when one compares the In and out sets for statements. For both the first and the second loops we can see that neither of them has any dependencies between statements. What we do see is dependency on the main loop counter. However, statements that are in the second loop can be split and given in any amount deemed appropriate by the OS scheduler. Thus we can have 25 iteration of the incrementing value of “i” go to core 1, 25 to core 2, 25 to 3 and so on. Or any other combination thereof.