Concurrency

Given a loop-dominated program, insert fork and shared memory commands to break the program into multiple processes during execution.

* Forking
  + This is using processes
  + Each process has a parent process that exists to create the new process
  + pid\_t childpid;  
    int \* status;  
    // print stuff  
    childpid = fork();  
      
    if (childpid == 0)  
    { //send address to thing inc\_n(&x); }  
    else  
    { inc\_n(&y); wait(status); }
  + Easy to modify to a loop.
* Shared memory
  + This is using threads
  + Supposed to be faster
  + Global var: pthread\_t tid[2];
  + Create a function that is much like the main() above:  
    pthread\_t id = pthread\_self();  
      
    if(pthread\_equal(id,tid[0]))  
    { inc\_n(&x);}  
    else  
    { int\_n(&y);}
  + In main:  
    while (i < 2){  
    pthread\_create(&(tid[i]), NULL, &doSomeThing, NULL);  
    i++;

Describe the similarities and differences between concurrency obtained through multiple processes and that generated by multiple threads.

* Multiple threads take place in a single process, which is best used when the threads are dependent on the same data. Harder to debug, not as “safe” as processes (memory can overrun into another thread).
* Multiple processes are best used when you need to work with different data at the same time; these can each have their own threads (I think). Each fork runs in its own virtual address space, so if it crashes, it won’t bring the whole program or system down.

Consider a “broken” program that attempts to use fork and shared memory commands to provide concurrency. Give the code and a description of the “flawed” program behavior, identify what change(s) are necessary to make the program “work”.

* See the first part, and move things around accordingly.

Using words and/or diagrams, describe the difference in the amount of context state maintained by a process compared to that of a thread.

* A thread essentially gets part of the resources that are allocated for a process. Threads get equally fast access to shared data.
* In multiple-processes, however, they have their own memory and maintain a larger amount of context state.
* These are both allocated access through typical process scheduling.