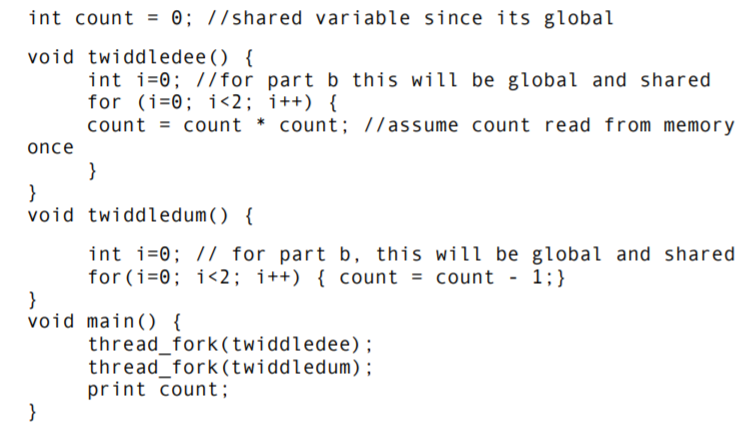
1. Consider a single CPU system with an active process A. Explain what happens in the following circumstances including any interrupts, system calls, etc.. and how they are handled until a process is back to running again
   1. The user presses control-C causing an interrupt to the process
      1. **It first sends signal to the OS, an interrupt happens, stop the current operation and change the OS to kernel mode, move the state from running to waiting state. From the vector table, find the index of the correct interruption. After handling the interruption. Restores the saved state and resume program execution.**
   2. A process executes continuously until it exhausts its scheduler allocated time slice
      1. **When the process uses up its allocated time slice, and interrupt happened. The handler runs other runnable process. Once the time interrupts process other processes. The scheduler will return back to process A.**
   3. A process does a read operation from an open file
      1. **It requests a system call which causes an exception for the kernel handler. Traps OS to kernel mode. The process state goes from running to waiting state. The handler then**
2. Which of the following requires a system call?
   1. Compressing some data to get ready to write it to a file (do not consider the file write)
   2. **The process in (a) writes the data to the file**
   3. **A process experiences a divide by zero**
   4. d) A network packet is received

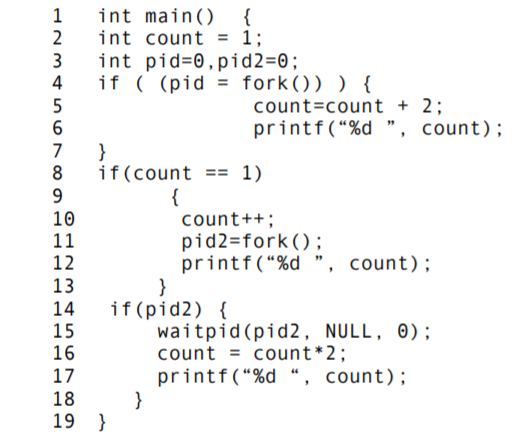
**system calls are used to connect user to the kernel that the user can use the system resource. B does input/output request. C cause an exception and trapped current state.**

1. Consider the following program:



* 1. What are all the values that could be printed in main?
     1. **0,-2, 1, 16**
     2. If twiddledee finishes first then it prints 0. If twiddledum finishes first then -2 prints. However, if twiddledum finishes and then twiddledee uses its count of -2 then we get a count of 16 returned from twiddledee. 1 is also possible because if twiddledum subtract the global count by 1 and then twiddledee multiply count by count we have 1 = -1\*-1. Then it prints 1 before twiddledum can subtract again.
  2. Repeat part 1 considering that i is also a shared variable
     1. **0,-2,1,3, 16 , 81**
     2. If twiddledum go first forloop, it change i = 1 count = -1, and then go to twiddledee, which set i =0 and go back to twidledum forloop, start from i =0 to i =2 count = -3, finish and go to twiddledee for loop, the result can be 81. Twiddledum go first forloop set i = 1 count = -1, then go to twiddledee, set i =0, then go back to twiddledum forloop, start from i=0 to i=1 count = -2, then go to twiddledee forloop, from i= 0 to i =1 count = 4, then go back to twiddledum forloop, from i = 1 to i=2 count =3, then go to twiddledeep from i=2 out of loop, then print 3
  3. Describe a potential schedule of execution that will result in the value printed out being equal to 0. Assuming there is only one CPU core, clearly specify when the transitions between the Ready and Running states occurs for each thread in this execution
     1. **A situation when the value 0 is printed out is when twiddledee is running count = count \* 0 while twiddledum is ready, then twiddledum is switch to running count = count - 1, and back and forth between the two. This gives us a count of 0, -1 , 1 , 0. Zero is last so it gets printed.**

1. Consider the following program:



* 1. How many processes are created during the execution of this program?
     1. **2 processes are created.** The first fokr in line 4 creates a child, then the child will enter the 2nd statement on line 8 which will fork another child.
  2. List all the possible outputs of the program. Full credit if you get 4 correct outputs, bonus if you get all outputs if there are more than 4
     1. parent p goes into first if statement in line 4 and count += 2, giving us count = 3 and then prints the value 3. Parent process is done because it does not meet the condition of the other two if statements. First child cp count = 1 after fork(), so it meets if statement condition on line 8. Count ++ so it is now 2. Then it forks() and gets a pid2 = 1. Then it prints 2. CP has a pid2 = 1 so it goes into if statement on line 14. Count \* 2 so it is now 4, then prints 4. Second child ccp gets a pid2=0 so it does not go into if statement in line 14. Therefore it only prints its current count which is 2.
     2. **3, 2, 2,4**
     3. **3,2,4,2**
     4. **2,4,3,2**
     5. **2,4,2,3**
     6. **4,2,2,3**
     7. **4,2,3,2**
  3. If we delete line 15 (the waitpid) show one output that is possible in the new program that is not possible in the old program.
     1. **4,3,2,2**