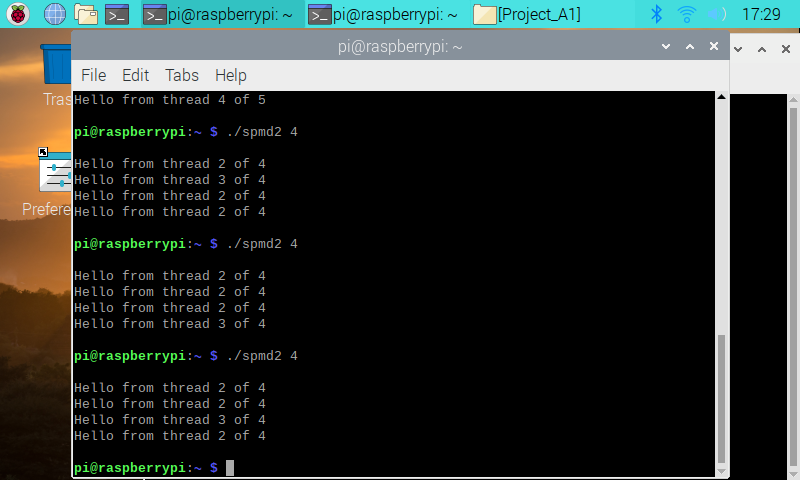
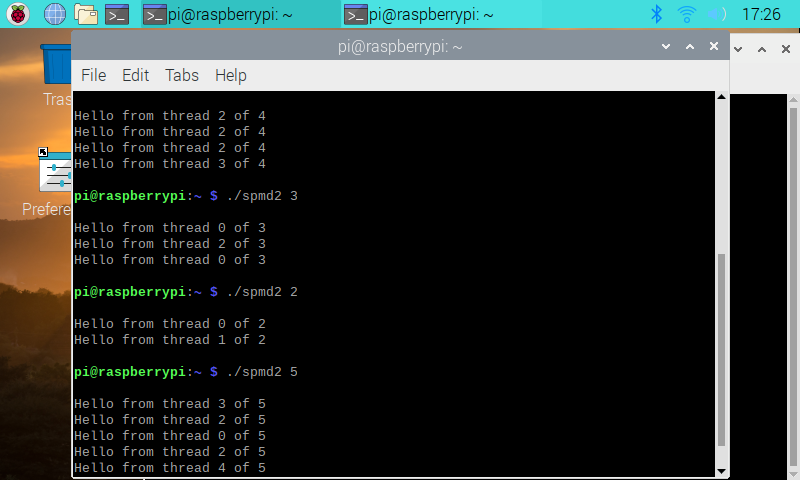
Bryanna Hardy

TASK 3: PARALLEL PROGRAMMING SKILLS

**PART A: FOUNDATION**

1. Identify the components on the Raspberry Pi B+.
   1. The components of a Raspberry Pi are the ethernet, USB, ethernet controller, display, HDMI, camera, the CPU/RAM, and the power.
2. How many cores does the Raspberry Pi’s B+ CPU have?
   1. It has 4 cores.
3. List the three main differences between the X866 (CISC) and ARM Raspberry PI (RISC).
   1. The X86 is a complex instruction set computing (CISC) which allows the processor to have a bigger feature instruction set that allows multiple ways to access the memory. The ARM Raspberry PI is a reduced instruction set computing (RISC) which is a more simplified instruction set and have more general registers than the X86 which load/store to access the memory. The instructions used in ARM can be used for conditional execution. On the other hand, X86’s processors use the Little-Endian format.
4. What is the difference between sequential and parallel computation? Identify the practical significance of each.
   1. Sequential computations processes are executed in a consecutive and ordered process. Parallel computation processes are executed concurrently.
   2. Sequential can be used unchanged and if different modules use the same data distribution, no communication is really required at the module interfaces. Parallel can be converted to a sequential composition and mainly it can enhance scalability and locality.
5. Identify the basic form of data and task parallelism in computational problems.
   1. Data parallelism uses multiple processing elements to solve problems by breaking the problem into parts. Task is a form of parallelization of computer code across multiple processors.
6. Explain the differences between processes and threads.
   1. Processes is the abstraction of a running program and it does not share memory. On the other hand, a thread is a lightweight process that allows a single executable/process to be decomposed to smaller parts. Each thread shares common memories of the processes they belong to.
7. What is OpenMP and what is OpenMP pragmas?
   1. OpenMP is a library/language and OpenMP pragmas are compiler directives that enable the compiler to generate threaded code.

**PART B: PARALLEL PROGRAMMING BASICS**

These two images are before the code was corrected. The reason why each thread shares the same fork/thread is because of the code not being correct. On the other hand, as you can see in images they are not in order even if they share the same fork/thread. It is because it does not matter the order of when a thread is finished and printed. When the code is corrected, the images will be inserted down below, the threads will still not be in order, but it will have its own independent fork/thread. 