Developing Soft and Parallel Programming Skills Using Project-Based Learning

Fall-2018

Team Nine30

Noah-Raine De Joya, Angelica Amis, Frankie Sanchez, Sai Pavan Nuthalapati, Phillip King

Planning and Scheduling:

Assignee Name	Email	Task	Duration (Hours)	Dependency	Due Date	Notes
Frankie Sanchez (Coordinator)	fsanchez4@student.gsu.edu	Organize & facilitate team meetings; Create the Work Breakdown Rubric; Assist with drafting Task 4 – Part A: Foundation of Parallel Programming Skills	3.5 hours	None	10/1/18	Meeting #1: 9/27/18 (2.5 hours) -Discussed roles & responsibilities -Set deadlines for tasks & outlined our review period Meeting #2: 10/1/18 (0.5 hours) -Recorded videos, discussed updates & deadlines Review 10/4/18
Phillip King	pking17@student.gsu.edu	Assist with drafting Task 4 – Part A: Foundations of Parallel Programming Skills; Task 4 – Part B: Raspberry Pi Installation & Setup; TA Slack invitation	6.5 hours	Raspberry PI hardware & the tutorials contained in the assignment	9/28/18	Installed Raspberry Pi 3 B+ and system image; Flashed to MicroSD; Verified installation
Angelica Amis	aamis1@student.gsu.edu	Task 3 – Record and upload YouTube video to Team Nine30 channel; Assist with Task 5 – Technical Writing (Appendix)	2 hours	YouTube; Team member involvement for the video recording session; Information needed for report appendix	10/1/18	Recorded videos on 10/1/18 after class
Sai Pavan Nuthalapati	snuthalapati@student.gsu.edu	Create & maintain GitHub as outlined in Task 2: Collaboration; Assist with Task 5 – Technical Writing (Body of Report)	3 hours	GitHub; Rough draft of report	10/1/18 (GitHub); 10/4/18 (Report)	Periodically maintain & update the work progress on GitHub
Noah De Joya	ndejoya@student.gsu.edu	Programming the PI according to Task 3 – Parallel Programming Basics; Assist team with Task 5 – Technical Writing	3 hours	Raspberry PI hardware & the tutorials contained in the assignment	10/4/18	Created detailed lab report to describe observations of parallel programming

Parallel Programming Skills:

a) Foundation:

1. Identify the components on the Raspberry Pi 3 B+.

   The components on the Raspberry Pi 3 B+ consist of:
   - Power input
   - HDMI output
   - Power button
   - 1GB LPDDR2 SDRAM

- Broadcom Videocore-IV
- Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- Dual-band 802.11b/g/n/ac Wi-Fi
- Bluetooth 4.2
- RJ-45 100 base Ethernet port
- 4xUSB 2.0 ports
- Ethernet Controller chip
- CPU/RAM chip
- Display header
- Camera header
- 40-pin GPIO header
- 3.5fmm analogue audio-video jack

2. How many cores does the Rasberry Pi's B+ CPU have?

   The Raspberry Pi's B+ CPU has a total of 4 cores.

3. List four main differences between x86 (CISC) and ARM Raspberry Pi (RISC).

   The four main differences between CISC and RISC are:
   1) RISC processors use simple instructions which only take an average of 1.5 clock cycles to run whereas CISC has more complex instructions which take, on average, anywhere from 2 to 15 clock cycles to run.
   2) RISC has multiple register sets whereas CISC only has one register set.
   3) Decoding of instructions on RISC is simple compared to CISC.
   4) RISC does not have a memory unit, but CISC does.

4. What is the difference between sequential and parallel computation? Identify the practical significance of each.

   In sequential computation a problem is partitioned into a series of discrete instructions that are executed one at a time on a single processor. The instruction execution is comparable to a queue of instructions waiting to be executed by the processor. Sequential computation was prevalent in older generation mainframes, workstations, and single processor PCs bounded by the Von Neumann architecture. In parallel computation a problem is solved with the synchronized utilization of multiple processors (compute resources) with each executing a piece of the instructions required to solve the problem. In essence, the problem is partitioned into discrete parts that can be solved simultaneously. Each discrete part is further split into a series of instructions that can execute concurrently on different processors with the use of a control mechanism. Once the problem is partitioned into differing sets of instructions, these instructions are executed by simultaneously by the various processors. There are many practical applications today where parallel computation offers many benefits. Some examples include: multithreaded applications, multimedia applications, database servers, compilers, and many more.

5. Identify the basic form of data and task parallelism in computational problems.

   Data parallelism in computational problems focuses more on the input data as opposed to the functions to be performed. By focusing on the data, the same computation is applied to multiple data items. From an algorithmic standpoint, this allows for parallelism to be proportional to the input size of the data. This lends itself a solid foundation on which to scale efficiently. Task parallelism focuses more, and is organized around, the functions to be performed on the data rather than the data itself. The functions are decomposed to balance the work and ensure that the results are representative of the work required. Task parallelism is prominent when data must be evaluated through different contextual means. Task parallelism does not scale as well as data parallelism.

6. Explain the differences between processes and threads.

   A process is essentially a program in execution. Processes involve “larger” tasks such as the execution of applications and is stored in main memory. Individual (independent) processes do not share memory with each other whereas threads share the memory of the processes of which they belong. A thread is merely a subset of a process—a lightweight process that allows a single executable/process to be decomposed into smaller, independent parts. A thread executes within the context of a process and shares the same resources.

7. What is OpenMP and what is OpenMP pragmas?

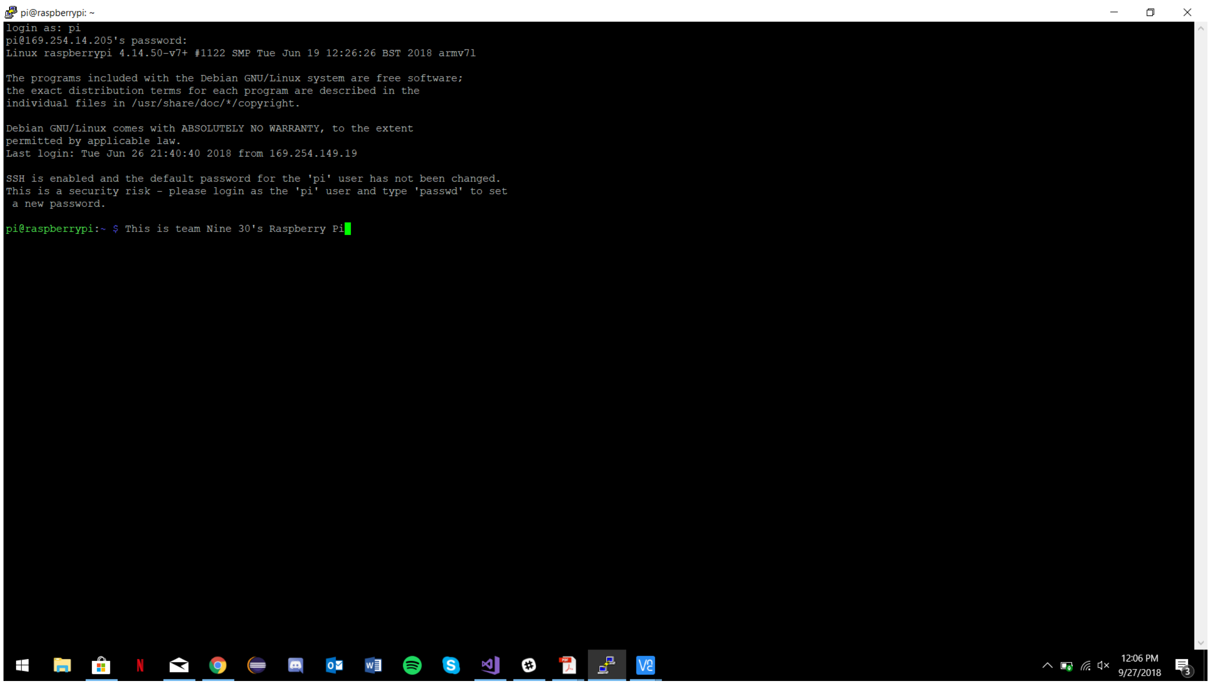
   OpenMP is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, instruction set architectures and OSs, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of computer directives, library routines, and environment variables that influence run-time behavior, and uses an implicit multithreading model (library creation and management). On the other hand, OpenMP pragmas are the compiler directives that enable a compiler to generate threaded code.

8. List four applications that benefit from multi-core.
   1) Multimedia applications
   2) Scientific applications such as CAD and CAM
   3) Database servers
   4) Compilers

9. Why multi-core? (List four)
   1) Multiple instructions/commands can be computed simultaneously.
   2) As we progress into the future, many new applications are multithreaded and there is a trend towards more parallelism in computer architecture.
   3) At the same clock frequency, a multicore processor will process more data than a single core processor. Multicore processors handle more complex tasks utilizing lower energy than a single core processor. It’s difficult to make single-core clock frequencies even higher.

4) Deeply pipelined circuits are difficult to design, verify, implement, and house. Such tendencies conjure expensive production and maintenance costs.

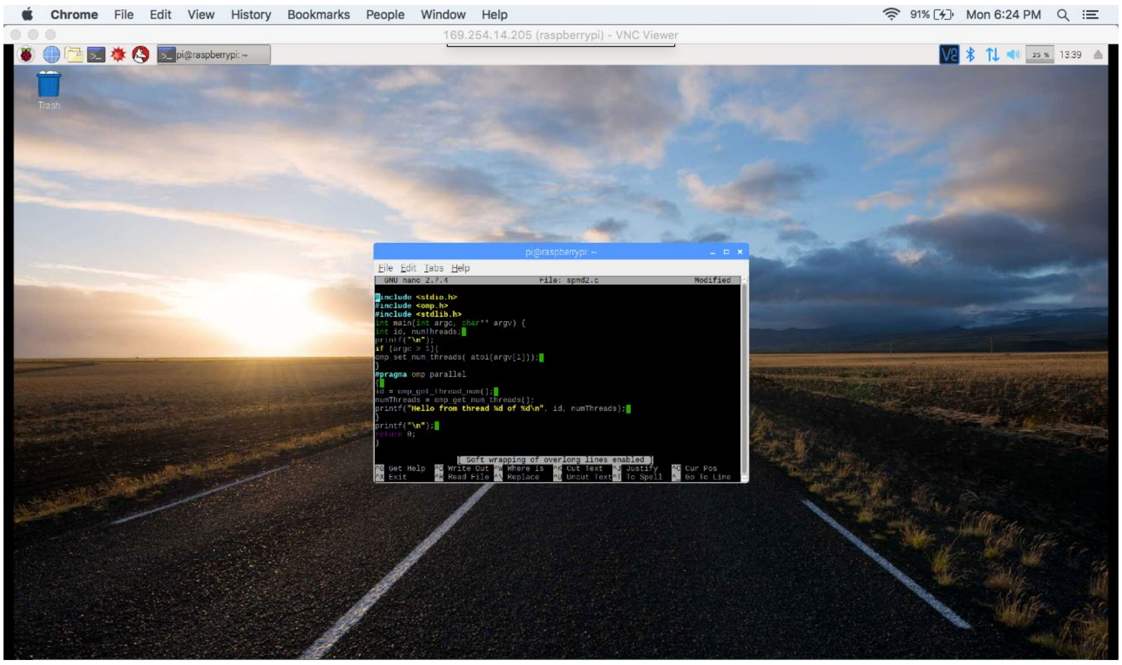
b) PI Installation:



```
login as: pi
pi@169.254.14.205's password:
Linux raspberrypi 4.14.50-v7+ #1122 SMP Tue Jun 19 12:26:26 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jun 26 21:40:40 2018 from 169.254.149.19

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ This is team Nine 30's Raspberry Pi
```

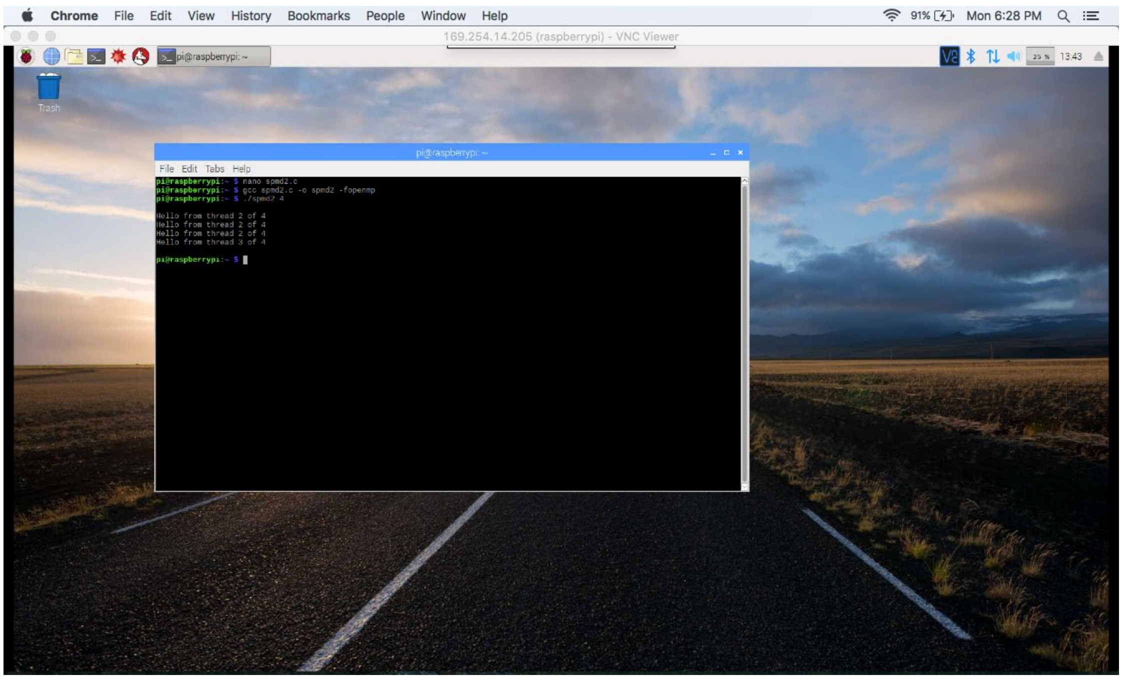
- The revised instructions for Pi set-up were followed. Initially the Pi didn't connect directly via ethernet, so it was necessary to set up remote access by connecting a monitor, mouse, and keyboard directly to the Pi. Then, network access was set manually. Once we were connected using ethernet the above screenshot was taken.

c) Parallel Programming Basics:



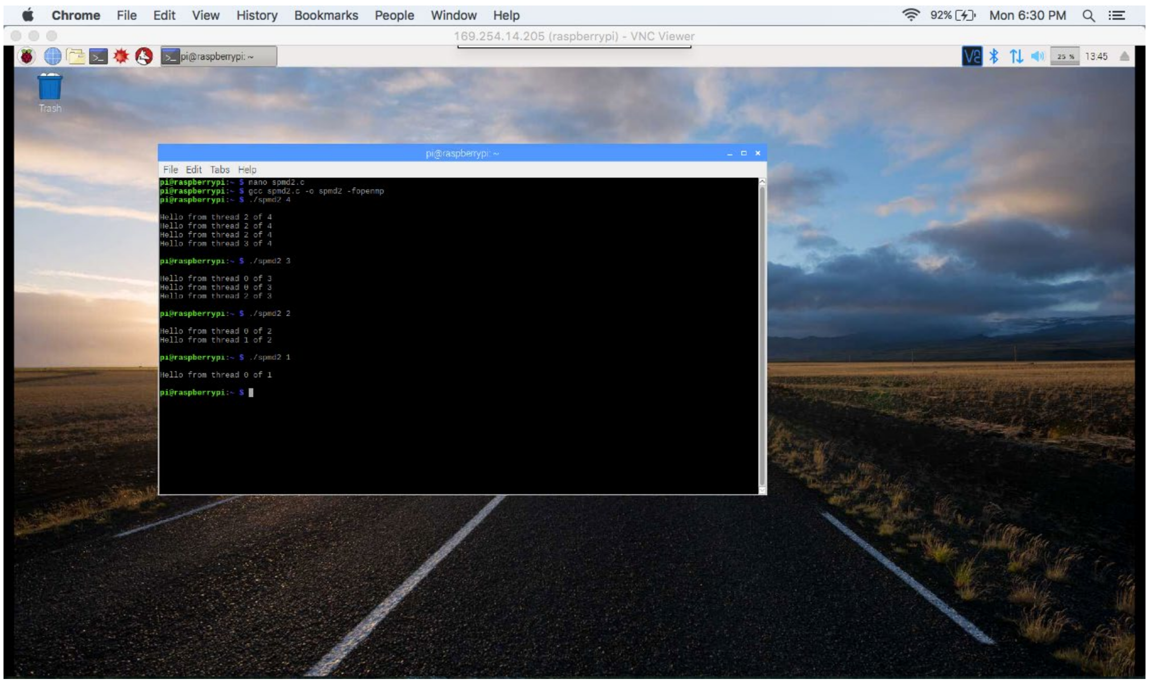
```
  GNU nano 2.7.4                 File: spmd2.c                      Modified

#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv) {
int id, numThreads;
printf("\n");
if (argc > 1){
omp set num threads( atoi(argv[1]));
}
#pragma omp parallel
{
id = omp_get_thread_num();
numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n", id, numThreads);
}
printf("\n");
return 0;
}

                [ Soft wrapping of overlong lines enabled ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

- In this screenshot, the initial code given from the spmd2.c program was input.



```
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

pi@raspberrypi:~ $ 
```

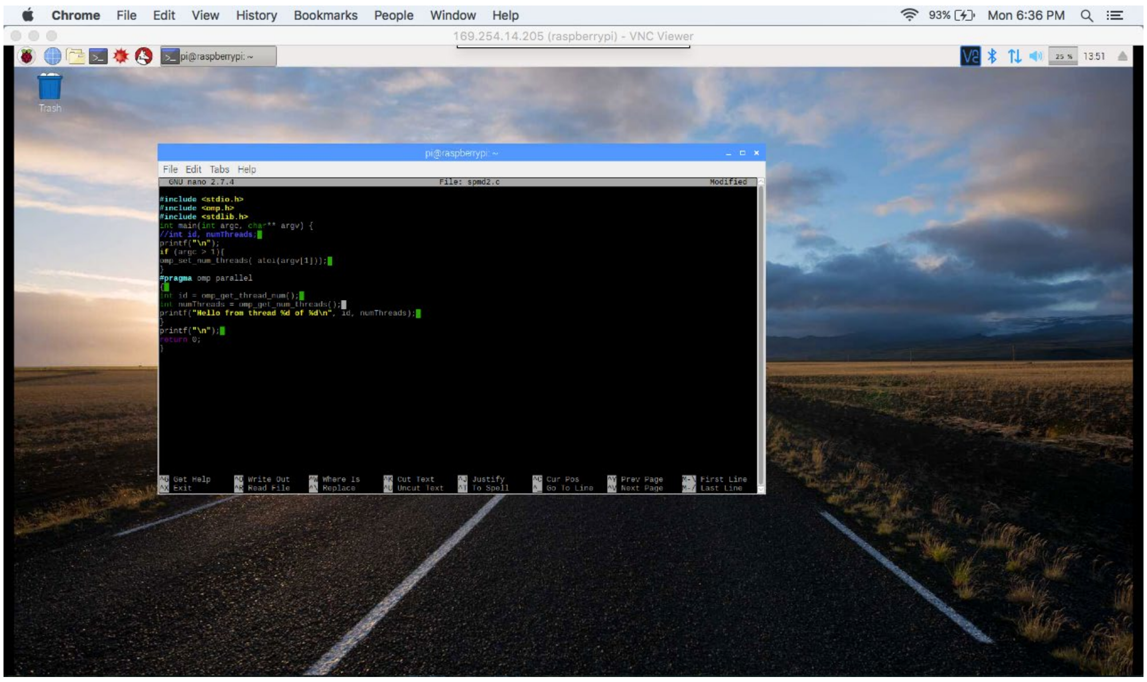
- In this screenshot, 4 threads were forked from the command line-argument ./spmd2 4
- What I found interesting was seeing the thread id number ‘2’ show up more than once for 4 threads.



```
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

pi@raspberrypi:~ $ ./spmd2 3

Hello from thread 0 of 3
Hello from thread 0 of 3
Hello from thread 2 of 3

pi@raspberrypi:~ $ ./spmd2 2

Hello from thread 0 of 2
Hello from thread 1 of 2

pi@raspberrypi:~ $ ./spmd2 1

Hello from thread 0 of 1

pi@raspberrypi:~ $ 
```

- In this screenshot, I tried to run the program again, but this time with 3, 2, and 1 threads being forked.
- What was interesting to me was when 4 threads were forked, the thread id number ‘2’ appeared more than once (3 times), but when 3, 2, or 1 threads were forked, the thread id number ‘0’ appeared for all of them.

- Each thread is supposed to be given its own unique id, so the thread id number can't appear more than once.
- Each thread is supposed to keep track of its id separately, which is why it cannot have the same one in different threads.



```
GNU nano 2.7.4                    File: spmd2.c                    Modified

#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv) {
//int id, numThreads;
printf("\n");
if (argc > 1){
omp_set_num_threads( atoi(argv[1]));
}
#pragma omp parallel
{
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n", id, numThreads);
}
printf("\n");
return 0;
}
```

- In this screenshot, the spmd2.c program is edited to where declaring the variables is changed so that a thread id number does not appear more than once.
- Declaring 'id' and 'numThreads' as int fixes the thread id number issue and now each thread will have their own private copy of the variables names 'id' and 'numThreads'.

Appendix:

Links –

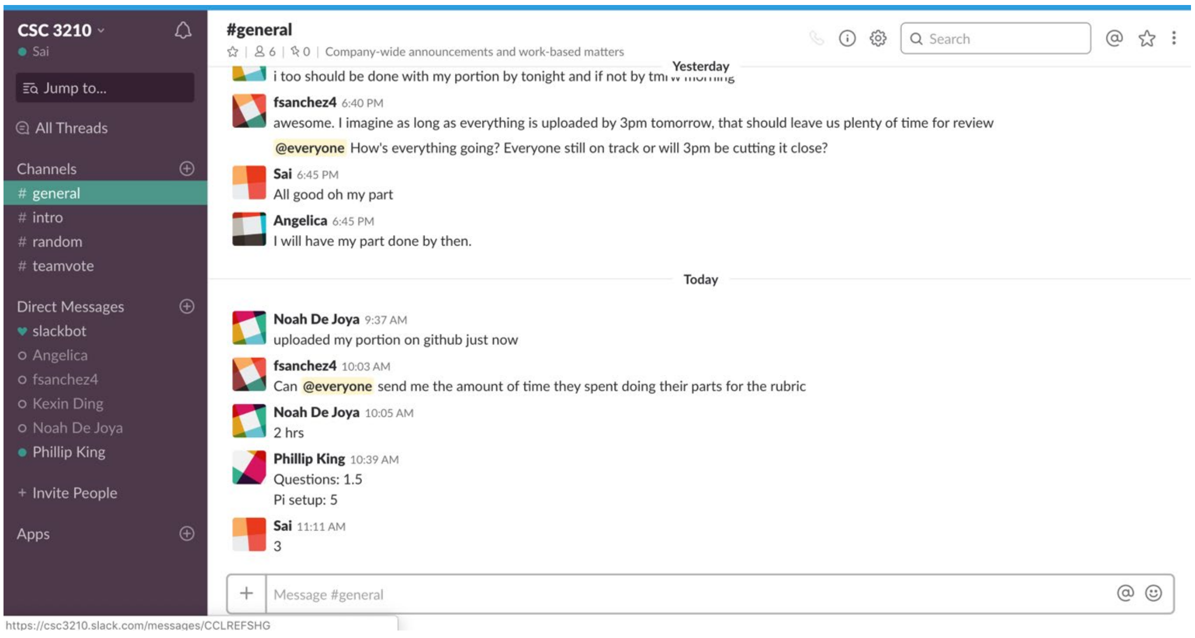
Slack: https://csc3210.slack.com/messages/CCLREFSHG/

GitHub: https://github.com/TeamNine30

YouTube Channel Link: https://www.youtube.com/channel/UCtZwypH0nkXfciumbrXb8HQ

Screen Shots –

Slack:



Yesterday

i too should be done with my portion by tonight and if not by tmrw morning

fsanchez4 6:40 PM
awesome. I imagine as long as everything is uploaded by 3pm tomorrow, that should leave us plenty of time for review
@everyone How's everything going? Everyone still on track or will 3pm be cutting it close?

Sai 6:45 PM
All good oh my part

Angelica 6:45 PM
I will have my part done by then.

Today

Noah De Joya 9:37 AM
uploaded my portion on github just now

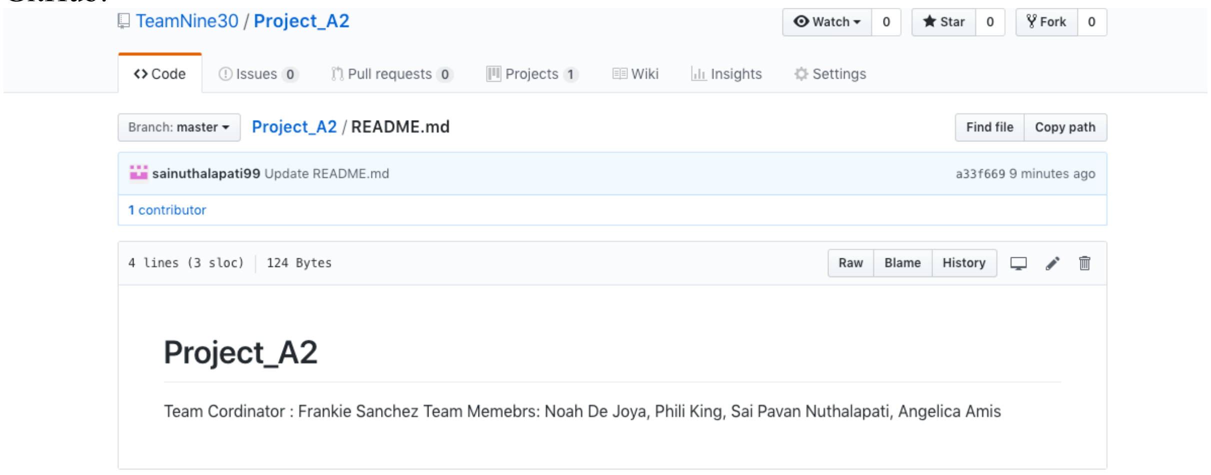
fsanchez4 10:03 AM
Can @everyone send me the amount of time they spent doing their parts for the rubric

Noah De Joya 10:05 AM
2 hrs

Phillip King 10:39 AM
Questions: 1.5
Pi setup: 5

Sai 11:11 AM
3

GitHub:



Project_A2 / README.md

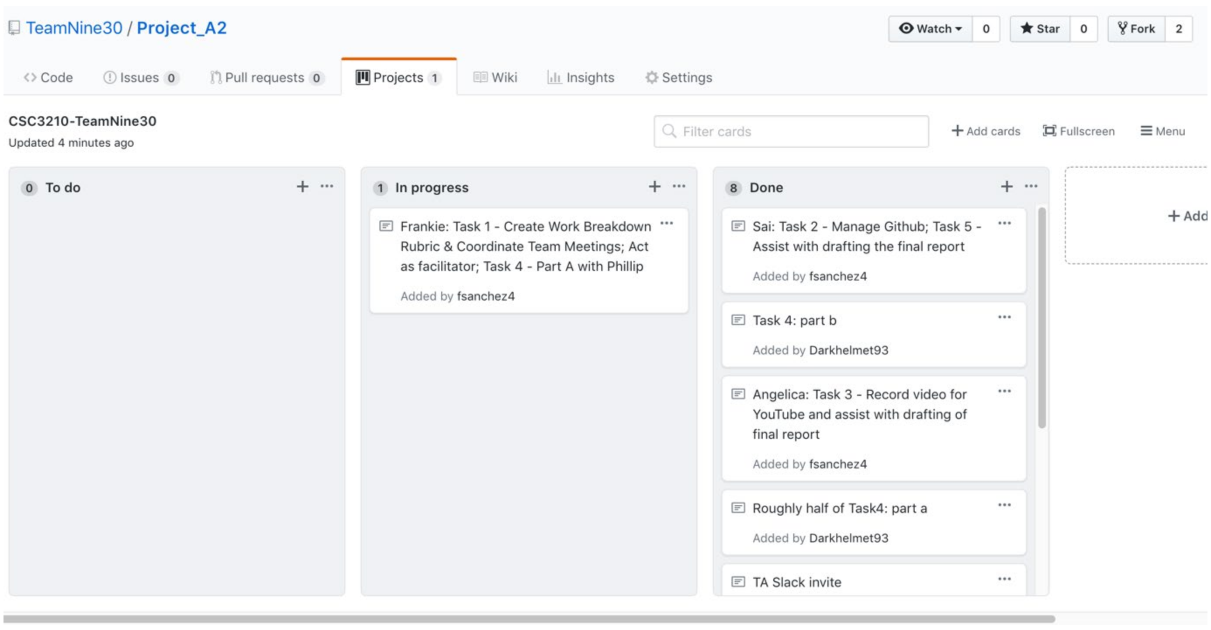
sainuthalapati99 Update README.md a33f669 9 minutes ago

1 contributor

4 lines (3 sloc) 124 Bytes

Project_A2

Team Cordinator : Frankie Sanchez Team Memebrs: Noah De Joya, Phili King, Sai Pavan Nuthalapati, Angelica Amis



CSC3210-TeamNine30
Updated 4 minutes ago

0 To do

1 In progress

Frankie: Task 1 - Create Work Breakdown Rubric & Coordinate Team Meetings; Act as facilitator; Task 4 - Part A with Phillip
Added by fsanchez4

8 Done

Sai: Task 2 - Manage Github; Task 5 - Assist with drafting the final report
Added by fsanchez4

Task 4: part b
Added by Darkhelmet93

Angelica: Task 3 - Record video for YouTube and assist with drafting of final report
Added by fsanchez4

Roughly half of Task4: part a
Added by Darkhelmet93

TA Slack invite