Developing Soft and Parallel Programming Skills Using Project- Based Learning

Fall 2018

Team Blue

Pavan Namani,
Kevin Hartiga,
Harshit Puri,
Keeshai Roberts,
William Lyons

Task Sheet:

Assignee Name	Email	Task	Duration (hours)	Dependency	Due Date	Note
Pavan Namani (Coordinator)	pnamani1@ student.gsu.edu	-Assign tasks - Turn in report - edit and revise report -Video Part -Set up meetings	1-2 hours	None	9/20	Make sure that everyone gets their task done on time.
William Lyons	wlyons2@ student.gsu.edu	-Raspberry Pi & Parallel Programming Tasks -Share & teach teammates of findings -Video Part	2+ hours	Task table, Mr. Mussa's documents, and Raspberry from A1 coordinator	10/1	Fully understands the concepts of the Pi System and teaches it to teammates.
Keeshai Roberts	kroberts34@ student.gsu.edu	-Video Part & Recording -A2 Reading Part: Foundation	2 hours	When we meet up as a group	9/27	Records the Presentation video, and sends it to Harsh to edit, and ensure that everyone is done with reading before answering questions
Kevin Hargita	khargita2@ student.gsu.edu	-Facilitator -Report Part -Video Part	2 hours	Group meeting, Task Table, and help from Keeshai on the reading	9/29	Writes the report alongside Harsh and ensures that meetings go well as facilitator.
Harshit Puri	hpuri1@ student.gsu.edu	-GitHub Account creation -Report Part -Video Part -revise and edit video	2 hours	Group meeting, YouTube Link, Task Table	9/29	Edits video with music, and text to make it look good. Also, works with Kevin to finish report.

Parallel Programming Skills:

1. **Identifying the components on the raspberry PI B+:**
   The main components that can be found on the raspberry PI are the, CPU/RAM, Ethernet Controller, USB, display, power, HDMI, camera, Ethernet

2. **How many cores does the Raspberry Pi's B+ CPU have:**
   The Raspberry Pi B+ has 4 cores.

3. **List four main differences between X86 (CISC) and ARM Raspberry PI (RISC):**
   - X86 (CISC) uses little-endian format and ARM is bi-endian (able to switch between little-endian and big endian)
   - X86 uses a larger set of instructions whereas ARM uses less instructions and more registers
   - X86 is register-register or register-memory and ARM is only register-register
   - ARM decreases execution time but this requires more of programmers to write efficient code. X86 has longer execution time but more operations and does not require code to be as efficient.

4. **What is the difference between sequential and parallel computation and identify the practical significance of each:**
   In sequential or serial computing, a problem is broken down into many, smaller pieces of instruction and executed one at a time through a single processor. In parallel computing the problem is still broken down into many, small pieces, but then the pieces are broken down even further and executed on multiple processors at the same time.

5. **Identify the basic form of data and task parallelism in computational problems.**
   Task parallelism refers to using multiple cores to execute many functions for possibly many datasets. Data parallelism refers to using multiple cores to execute one function across one dataset. Data parallelism

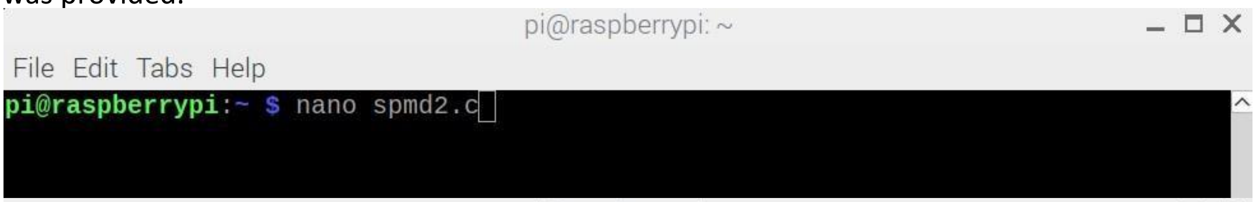
6. **Explain the differences between processes and threads.**
   A process is the abstraction of a running program whereas a thread is a lightweight and allows an executable or process to be broken down into smaller, independent parts.

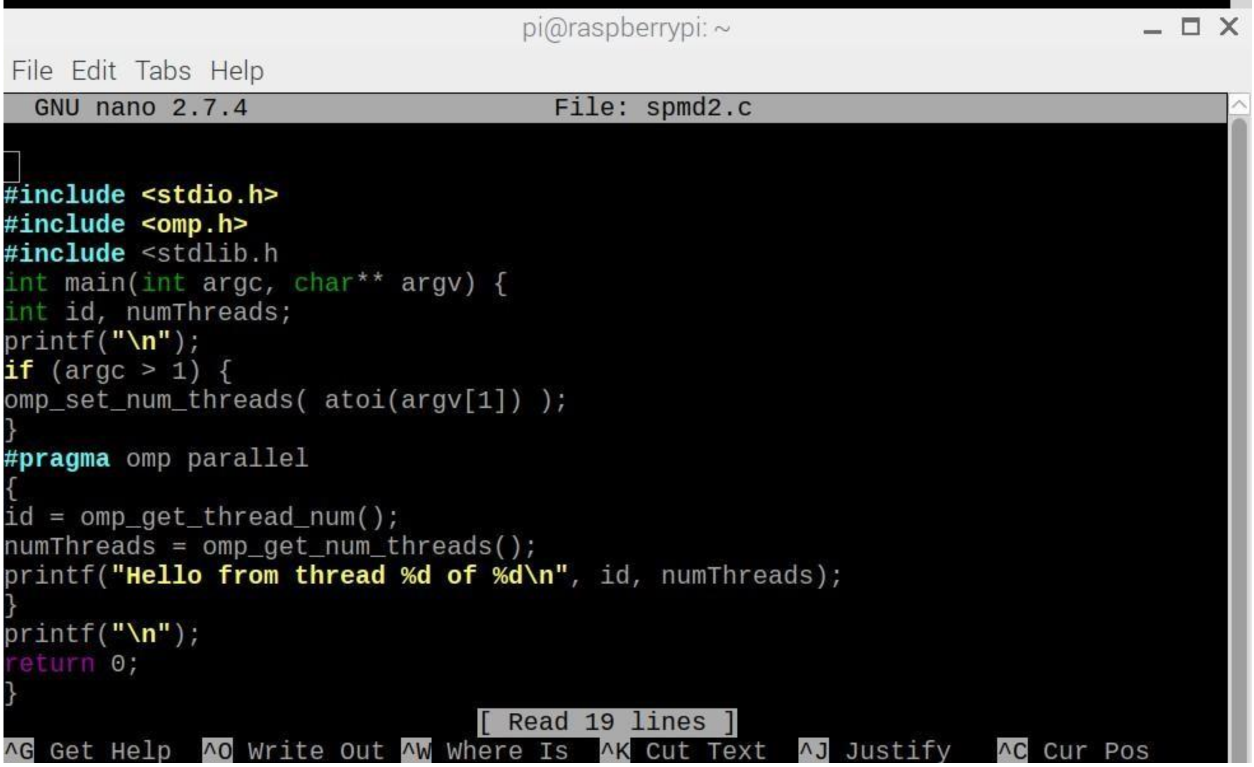
7. **What is OpenMP and what is OpenMP pragmas?**
   OpenMP is a standard that uses an implicit multithreading model in which the library handles thread creation and management. OpenMP pragmas are compiler directives. These directives allow the compiler to create threaded code.

8. **What applications benefit from multi-core? (list four):**
   - Database servers
   - Web Servers
   - Compilers
   - Scientific applications (i.e. CAD/CAM)

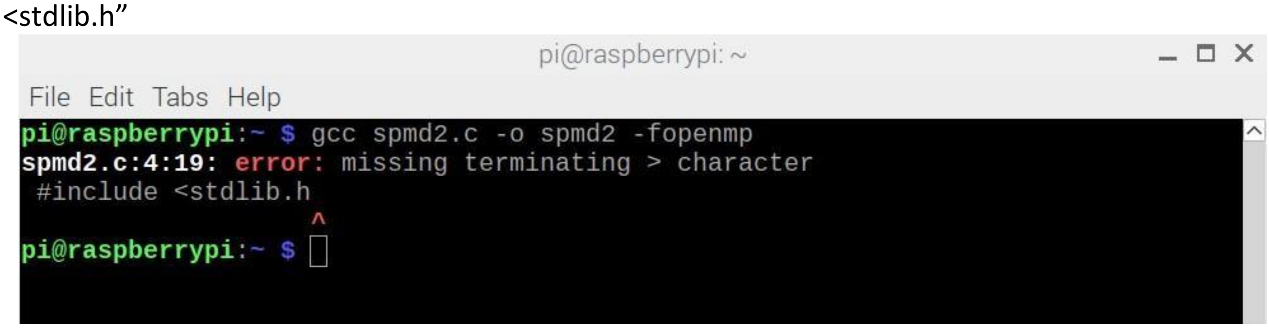
9. **Why Multicore? (why not single core, list four)**
    - Computer Architecture is shifting toward parallelism
    - Trouble making single-core clock frequencies higher
    - Many new applications are multithreaded
    - Deeply pipelined circuits have trouble with heat, speed of light limits, design expense, and cost of cooling.

To begin, I created the document ‘spmd2.c’ by with the built-in text-editor, nano by typing the command ‘nano spmd2.c’ into the terminal. I then used copy and paste to insert the code that was provided.





To create the executable, I typed, ‘gcc spmd2.c -o spmd2 -fopenmp’ into the terminal. However, when this ran I received the error “spmd2.c:3:19: error: missing terminating > character #include <stdlib.h”



I searched the internet to help diagnose the issue as I was unfamiliar with C programs and realized that the error message told me everything I needed to know. I went opened spmd2.c back up in nano and located the missing ‘>’ character and added it to the code on line 3.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
  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;
}
```

This time when I ran the command, 'gcc spmd2.c -o smpd2 -fopenpm' the executable was created as expected.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ 
```

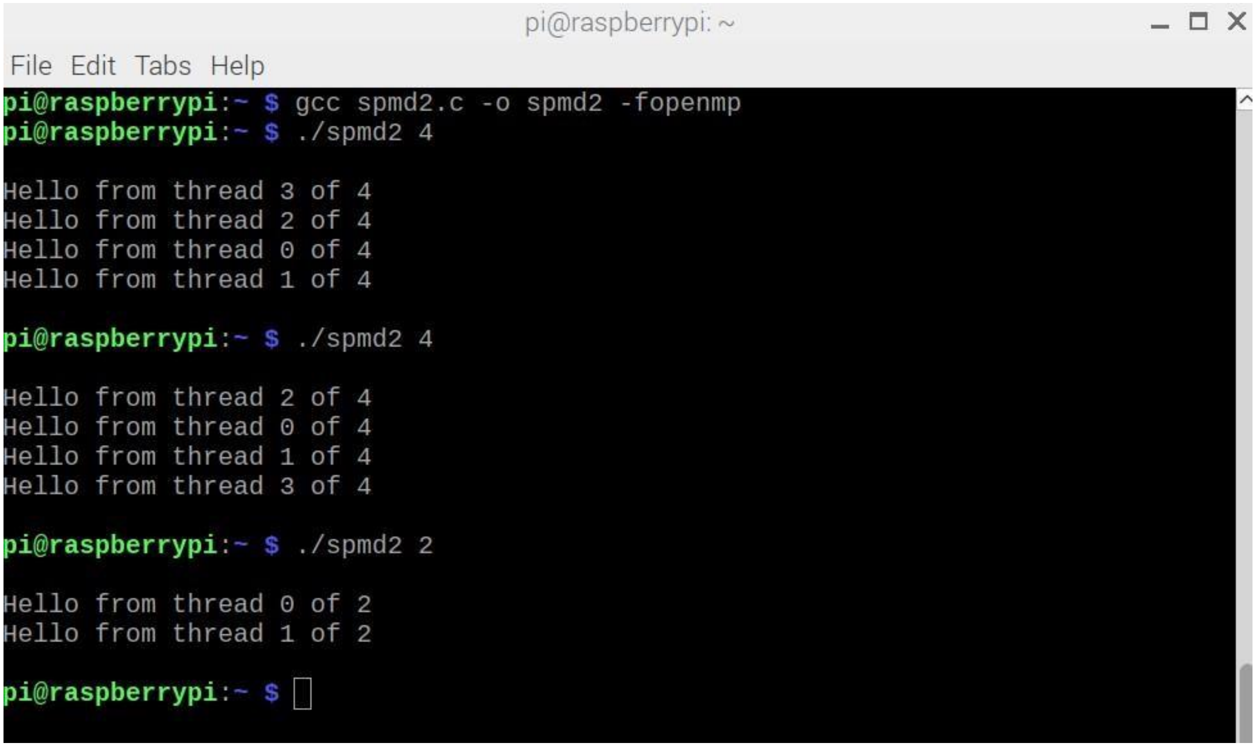
When the new program is executed, there are duplicate thread numbers showing up. This is because the variables were declared outside of the block of code that forks into the different cores.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

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

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:~ $
```

To fix this I went back into the file spmd2.c with nano and commented out the variable declarations for id and numThreads and declared them inside the pragma block of code.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
  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;
}
```

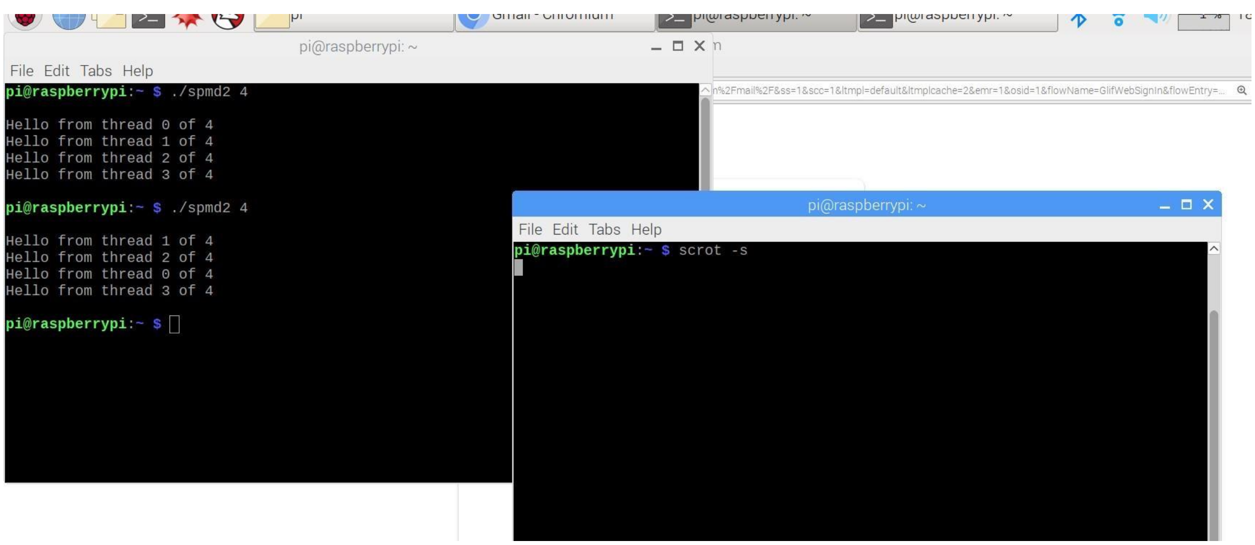
Now when I ran the program spmd2 each thread had its own unique identifier.



Additional Information:

I could not find a built-in screenshot function on Raspbian so after some research I installed one through the terminal with the command ‘sudo apt-get install scrot’

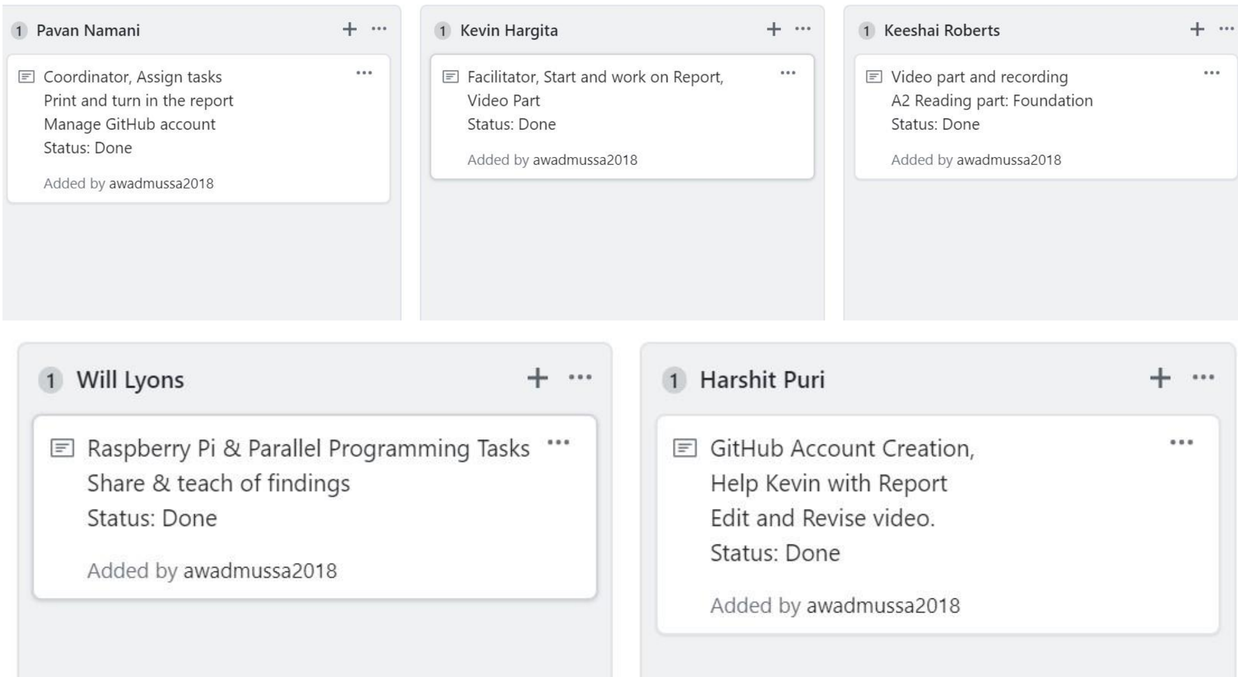
This is a screenshot utility that can be used through terminal and with the option -s I was able to click and drag the desired section of screen I wanted. To accomplish this, I used two terminal windows, one for the smpd2 program and the other for the scrot screenshot function.



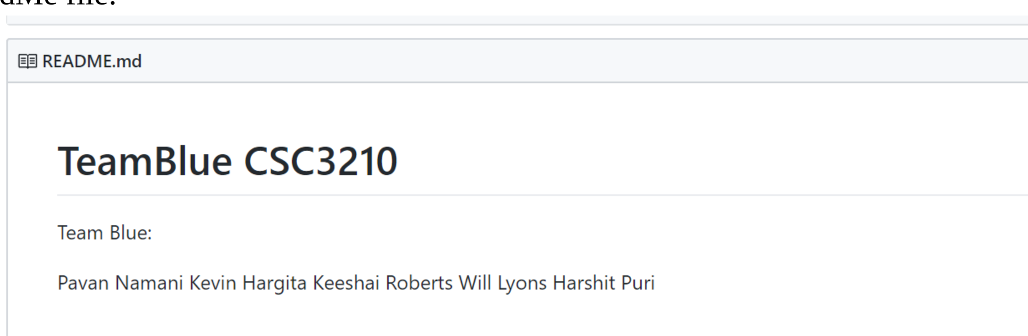
Appendix:

Slack Link: https://csc3210project2018.slack.com/messages/CCMLVETFU/?

GitHub: https://github.com/awadmussa2018/ProjectA2
GitHub username: awadmussa2018
GitHub password: Panther@2018



ReadMe file:


README.md

TeamBlue CSC3210

Team Blue:

Pavan Namani Kevin Hargita Keeshai Roberts Will Lyons Harshit Puri

YouTube Account: https://www.youtube.com/channel/UCMUQUPcmlgmCTFuCvELgo_g