Developing Soft and Parallel Programming Skills Using Project-Based Learning

Spring-2019 **JupitorInk**

Nhi Ong, Alejandro Herbener, Alec Buchinski, Johnathan Moore

Table of Contents

I. Project Description

1.1 Purpose of the Project

The purpose of this project is to 1) become familiar with unified, team-based project management tools and communication applications in order to develop soft skills, 2) identify Raspberry Pi 3+ components and architecture; explain basic multicore architectures and programming using OpenMP and C language 3) gain more practice with ARM assembly programming (Mussa, Assignment 2 Worksheet).

II. Introduction to the Team

2.1 Members' Roles and Assignments

Coordinator: Alec Buchinski

For this assignment, he was the coordinator; his responsibilities contained tasks one, two and six. Task one involves group planning and scheduling and creating a table for task breakdowns. A new GitHub repository was created for assignment 3, sent out invites to the slack and access to upload to the GitHub as well. The final duties involved editing and uploading the video for part 6 to the YouTube channel.

Team Member: Alejandro Herbener

This team member is responsible for task five. I created this report after the other members completed their sections and compiled them into this document.

Team member: Nhi Ong

This team member is responsible for task three. Nhi completed the Parallel Programming portion of this project, as well as taking screenshots and reporting their findings.

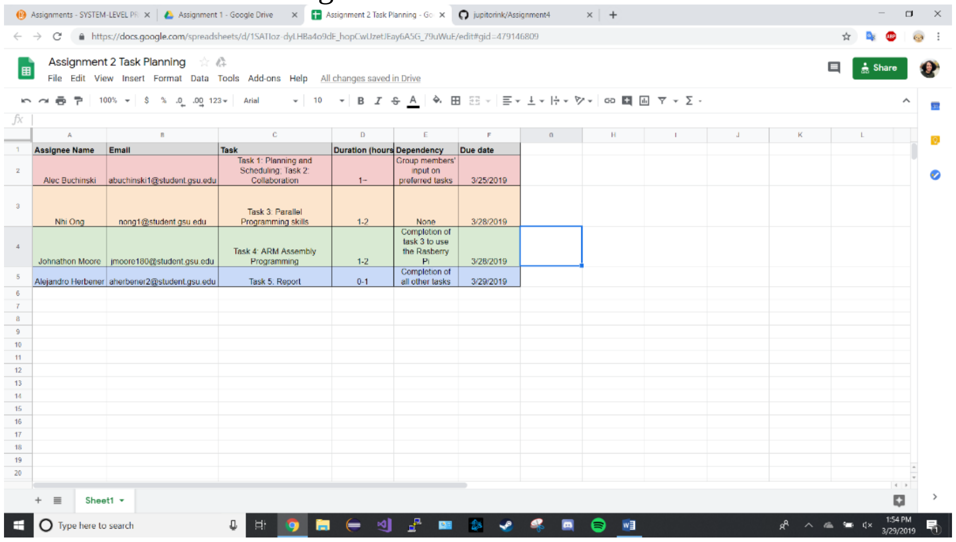
Team Member: Johnathon Moore

This team member is responsible for task four. Johnathon completed the ARM Assembly Programming section of this project. This includes taking screenshots, completing the assigned work, and creating a write-up to send to the GitHub for further editing.

III. Planning and Scheduling

3.1 Role as Coordinator

As the coordinator, Alec completed the new assignment table and created a new repository for the assignment. He contacted us to discuss our preferred roles over Slack, and we selected our preferred roles and quickly started to work on each of our sections as seen in figure 1.1.

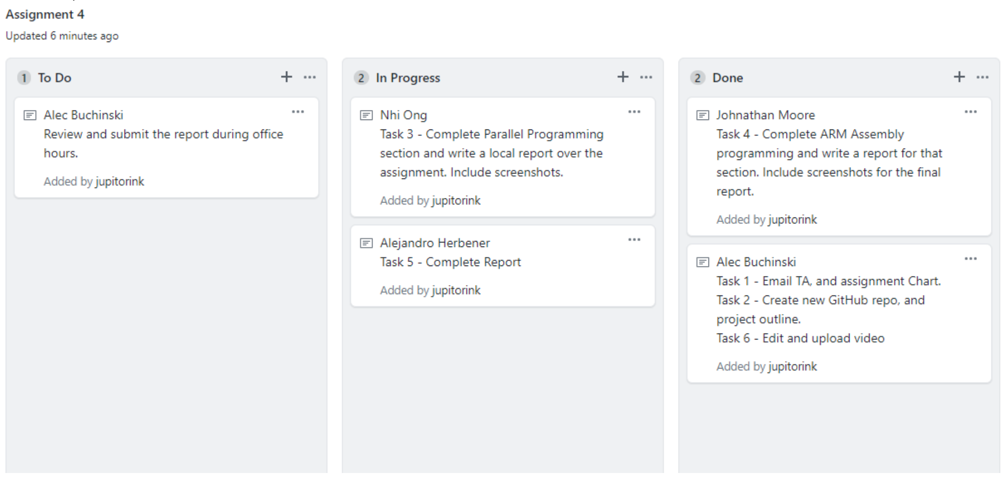

Assignment 2 Task Planning
File Edit View Insert Format Data Tools Add-ons Help
All changes saved in Drive
Share
Sheet1
Type here to search

Assignee Name	Email	Task	Duration (hours	Dependency	Due date
Alec Buchinski	abuchinski1@student.gsu.edu	Task 1: Planning and Scheduling; Task 2: Collaboration	1~	Group members' input on preferred tasks	3/25/2019
Nhi Ong	nong1@student.gsu.edu	Task 3: Parallel Programming skills	1-2	None	3/28/2019
Johnathon Moore	jmoore180@student.gsu.edu	Task 4: ARM Assembly Programming	1-2	Completion of task 3 to use the Rasberry Pi	3/28/2019
Alejandro Herbener	aherbener2@student.gsu.edu	Task 5: Report	0-1	Completion of all other tasks	3/29/2019

Figure 1.1 – Task One, Assignment Chart

3.2 Additional Tasks

Alec was responsible for creating a new repository for this assignment with a readme page. He added a projects page which details our tasks and allows for any member to update their status on their role. (see Figure 1.2) In addition to this, he also edited our video and uploaded it to YouTube. (see Appendix A, for link to video)


Assignment 4
Updated 6 minutes ago

1 To Do

Alec Buchinski
Review and submit the report during office hours.
Added by jupitorink

2 In Progress

Nhi Ong
Task 3 - Complete Parallel Programming section and write a local report over the assignment. Include screenshots.
Added by jupitorink

Alejandro Herbener
Task 5 - Complete Report
Added by jupitorink

2 Done

Johnathan Moore
Task 4 - Complete ARM Assembly programming and write a report for that section. Include screenshots for the final report.
Added by jupitorink

Alec Buchinski
Task 1 - Email TA, and assignment Chart.
Task 2 - Create new GitHub repo, and project outline.
Task 6 - Edit and upload video
Added by jupitorink

Figure 1.2 – Task Two, GitHub Projects Page Screenshot

IV. Parallel Programming Skills

Task 3a.

What is a race condition?

A race condition occurs within software or systems when there is a dependency on the sequence of other events beyond the programmer's control that affects the output.

Why race condition is difficult to reproduce and debug?

Race conditions are hard to reproduce because often, the ending results or output will vary despite using the same input and conditions, since it depends on the timing of the threads.

How can it be fixed? Provide an example from your project_A3.

You can fix this issue by having good software architecture to prevent the issue from occurring in the first place. An example where the issue is fixed beforehand comes from Assignment 3's Parallel Programming task. The parallel for pragma without the reduction clause resulted in an incorrect outcome, but the solution to this problem simply required declaring the variables as private so that each thread had its own clone. This fixed the issue because the sum variable was dependent on each of the threads' processes during computation.

Summarize the Parallel Programming Patterns section in the "Introduction to Parallel Computing_3.pdf" in your own words (one paragraph, no more than 150 words)

Parallel programs have patterns which are used often by programmers due to their success in previous programs. These patterns are documented for new programmers to use. There are two major categories for the patterns that developers use: strategies and concurrent execution mechanism. Strategies simply refer to the programmer's choice on which "algorithmic strategies" to use as well has how to implement those strategies. Concurrent execution mechanisms have two major categories as well: process/thread control patterns and coordination patterns. Coordination patterns can further be separated into message passing and mutual exclusion. Process/thread control patterns involve the processing units and how they are executed at runtime whereas coordination patterns involve determining how processing units coordinate to complete tasks.

Compare collective synchronization (barrier) with Collective communication (reduction) and Master-worker with fork join.

Barriers are used to enforce collective synchronization by adding wait cycles into a program where a process cannot continue until the other processes have reached the same point, where they will be synchronized. Reductions are used to enforce collective communication by combining multiple vectors of processes or processes into one. Both patterns are concurrent execution mechanisms.

A Master-Worker pattern uses a master process/thread to hold a group of worker processes with a certain number of tasks. The "workers" complete each task simultaneously while removing them from the tasks held by the master process. The Fork-Join pattern describes the process of forking parent tasks to create new tasks and completing them concurrently with other forked tasks, then waiting until they finish executing before joining them further continue the parent task. Both patterns are strategies.

Where can we find parallelism in programming?

You can find them in loops!

What is dependency and what are its types?

A dependency occurs when the execution of one operation is reliant on the outcome of a previous operation and its execution. There are two types of dependencies—independent and dependent.

Independent:

Statement 1 – a = 3;
Statement 2 – b = 3;

Dependent:

Statement 1 – a = 3;
Statement 2 – b = a;

When a statement is dependent and when it is independent?

A dependency is independent if changing the order of execution does not affect the outcome. Otherwise, if changing the order causes an error or results in an undesirable outcome, then the dependency is dependent.

Independent:

Statement 1 – a = 33;
Statement 2 – b = 33;

Dependent:

Statement 1 – a = b;
Statement 2 – b = 33;

When can two statements be executed in parallel?

Two statements can be executed in parallel when the order of the operations do not affect the outcome and the statements themselves, do not have any dependencies.

How can dependency be removed?

Dependencies can be removed by changing the order of the statements or removing some of them completely.

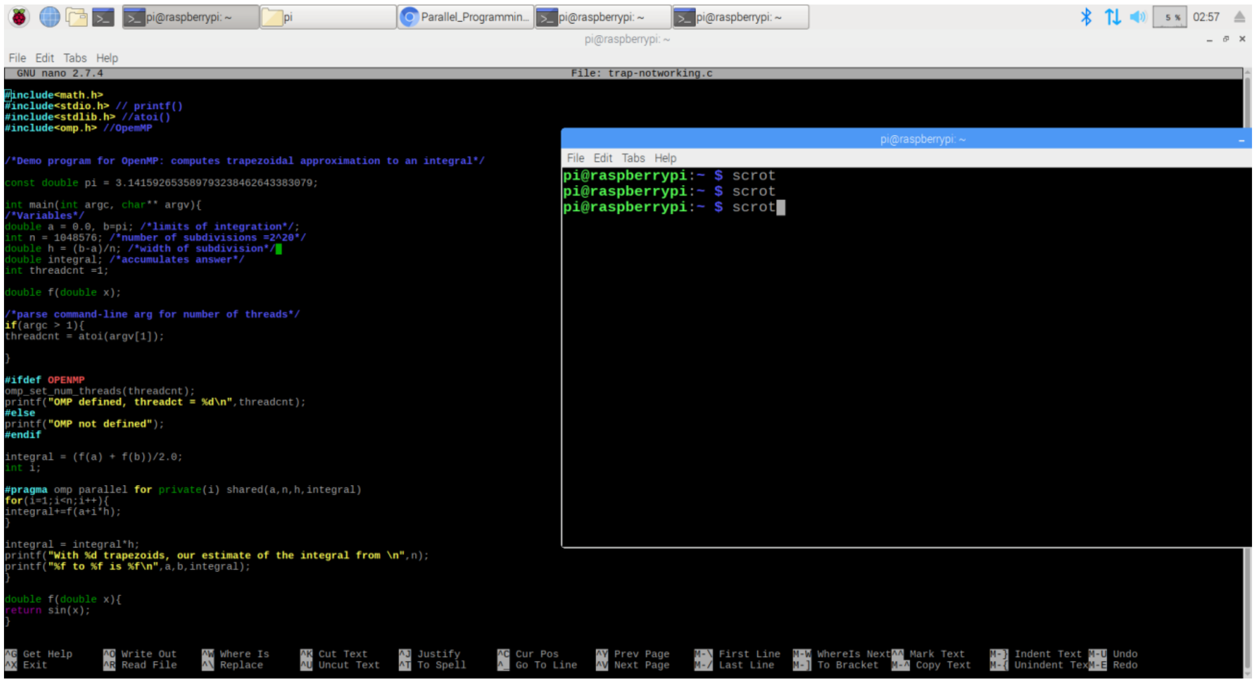
How do we compute dependency for the following two loops and what type/s of dependency?

The two loops are dependent on the value of i in order to find the correct index of a and set the correct value to it.

5.2 Task 3b.

Part One: Integration Using the Trapezoidal Rule

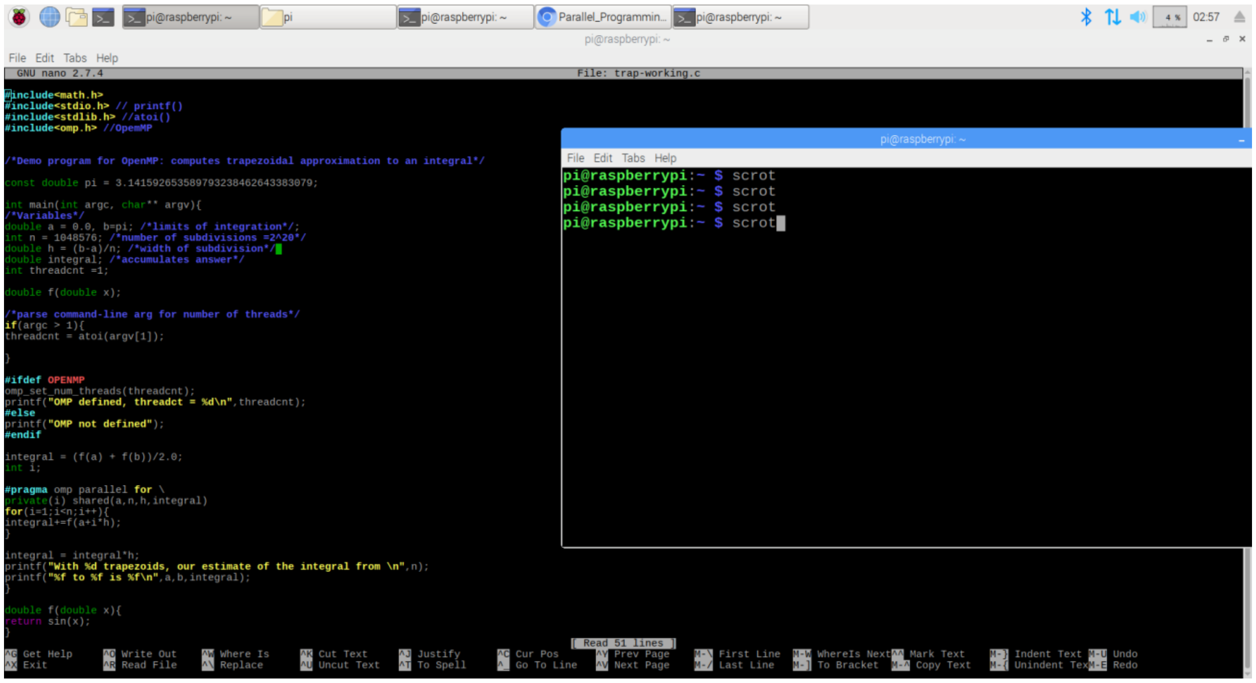
For the first part of this section, I created a file called trap-notworking.c using nano and copied he code provided on the pdf file. I then created an executable as instructed.



```
  GNU nano 2.7.4                                   File: trap-notworking.c

#include<math.h>
#include<stdio.h> // printf()
#include<stdlib.h> //atoi()
#include<omp.h> //OpemMP


/*Demo program for OpenMP: computes trapezoidal approximation to an integral*/

const double pi = 3.141592653589793238462643383079;

int main(int argc, char** argv){
/*Variables*/
double a = 0.0, b=pi; /*limits of integration*/;
int n = 1048576; /*number of subdivisions =2^20*/
double h = (b-a)/n; /*width of subdivision*/
double integral; /*accumulates answer*/
int threadcnt =1;

double f(double x);

/*parse command-line arg for number of threads*/
if(argc > 1){
threadcnt = atoi(argv[1]);

}

#ifdef OPENMP
omp_set_num_threads(threadcnt);
printf("OMP defined, threadct = %d\n",threadcnt);
#else
printf("OMP not defined");
#endif

integral = (f(a) + f(b))/2.0;
int i;

#pragma omp parallel for private(i) shared(a,n,h,integral)
for(i=1;i<n;i++){
integral+=f(a+i*h);
}

integral = integral*h;
printf("With %d trapezoids, our estimate of the integral from \n",n);
printf("%f to %f is %f\n",a,b,integral);
}

double f(double x){
return sin(x);
}
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

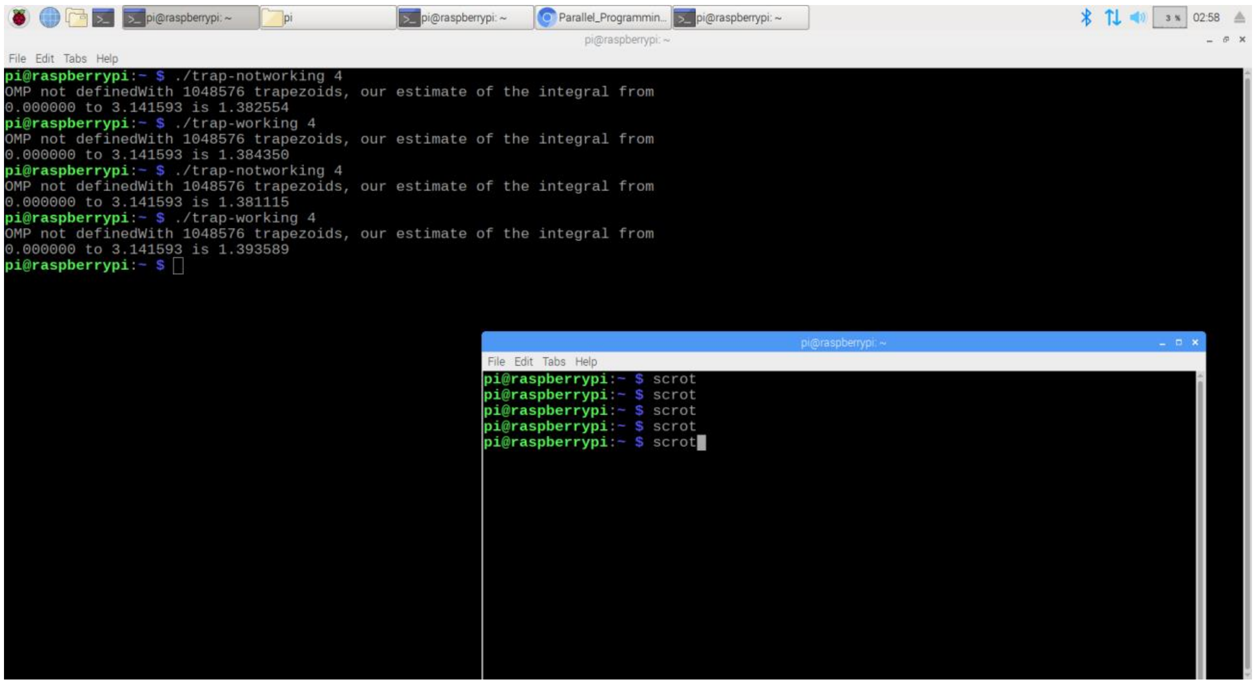
After that, I created a new file called trap-working.c and corrected the error by adding a backslash after “#pragma omp parallel for” and moving the line with “private” to the next line. I then created another executable.



```
  GNU nano 2.7.4                                   File: trap-working.c

#include<math.h>
#include<stdio.h> // printf()
#include<stdlib.h> //atoi()
#include<omp.h> //OpemMP


/*Demo program for OpenMP: computes trapezoidal approximation to an integral*/

const double pi = 3.141592653589793238462643383079;

int main(int argc, char** argv){
/*Variables*/
double a = 0.0, b=pi; /*limits of integration*/;
int n = 1048576; /*number of subdivisions =2^20*/
double h = (b-a)/n; /*width of subdivision*/
double integral; /*accumulates answer*/
int threadcnt =1;

double f(double x);

/*parse command-line arg for number of threads*/
if(argc > 1){
threadcnt = atoi(argv[1]);

}

#ifdef OPENMP
omp_set_num_threads(threadcnt);
printf("OMP defined, threadct = %d\n",threadcnt);
#else
printf("OMP not defined");
#endif

integral = (f(a) + f(b))/2.0;
int i;

#pragma omp parallel for \
private(i) shared(a,n,h,integral)
for(i=1;i<n;i++){
integral+=f(a+i*h);
}

integral = integral*h;
printf("With %d trapezoids, our estimate of the integral from \n",n);
printf("%f to %f is %f\n",a,b,integral);
}

double f(double x){
return sin(x);
}
                                    [ Read 51 lines ]
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

I compared the results between two of the programs multiple times to check if the intended result was reached.

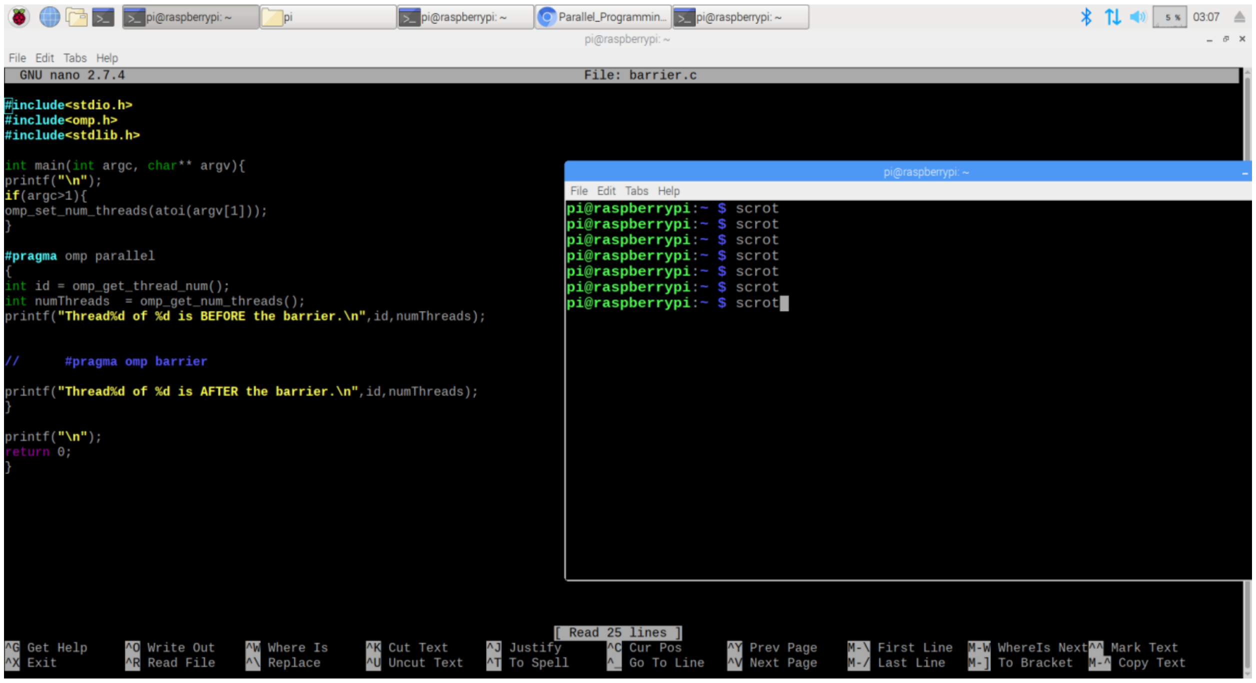


```
pi@raspberrypi:~ $ ./trap-notworking 4
OMP not definedWith 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.382554
pi@raspberrypi:~ $ ./trap-working 4
OMP not definedWith 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.384350
pi@raspberrypi:~ $ ./trap-notworking 4
OMP not definedWith 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.381115
pi@raspberrypi:~ $ ./trap-working 4
OMP not definedWith 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.393589
pi@raspberrypi:~ $
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

Part Two: Coordination: Synchronization with a Barrier

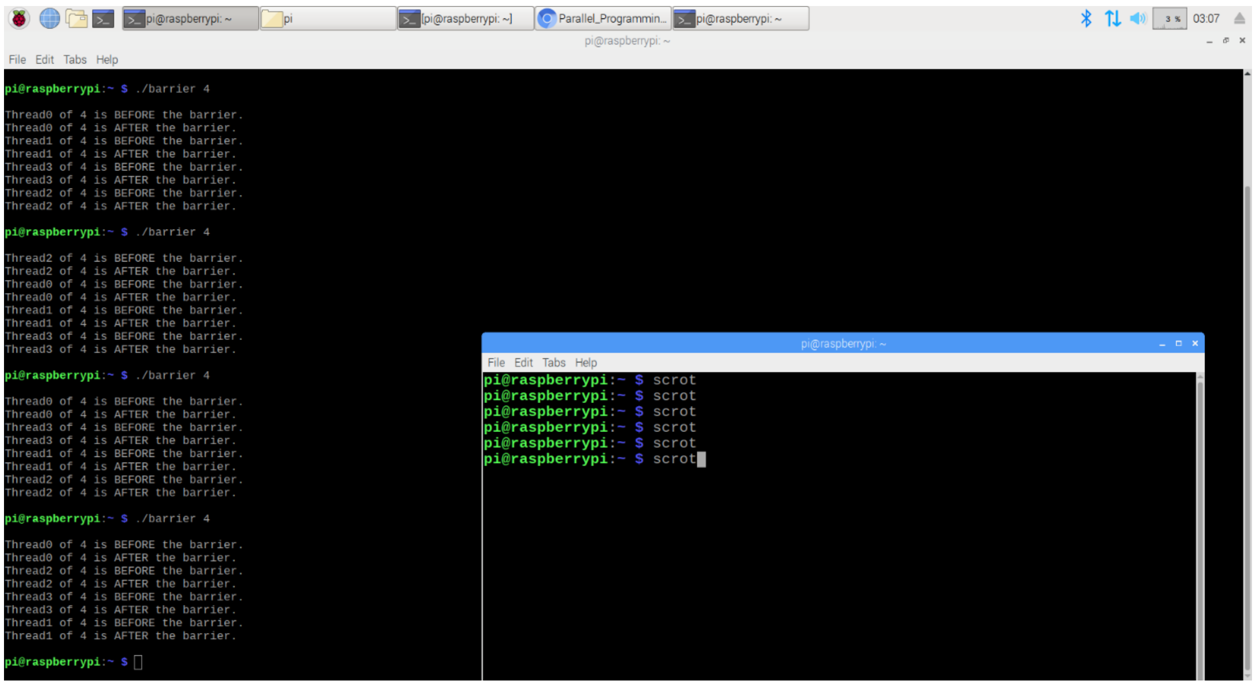
For this section of the task, I read the short excerpt on barriers and then proceeded to complete the programming portion. I created a barrier.c file and copied the code from the pdf.



```
  GNU nano 2.7.4                                              File: barrier.c

#include<stdio.h>
#include<omp.h>
#include<stdlib.h>

int main(int argc, char** argv){
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("Thread%d of %d is BEFORE the barrier.\n",id,numThreads);

//      #pragma omp barrier

printf("Thread%d of %d is AFTER the barrier.\n",id,numThreads);
}

printf("\n");
return 0;
}

                                          [ Read 25 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     ^Y Prev Page   M-\ First Line  M-W WhereIs Next^^ Mark Text
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  ^V Next Page   M-/ Last Line   M-] To Bracket  M-^ Copy Text
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

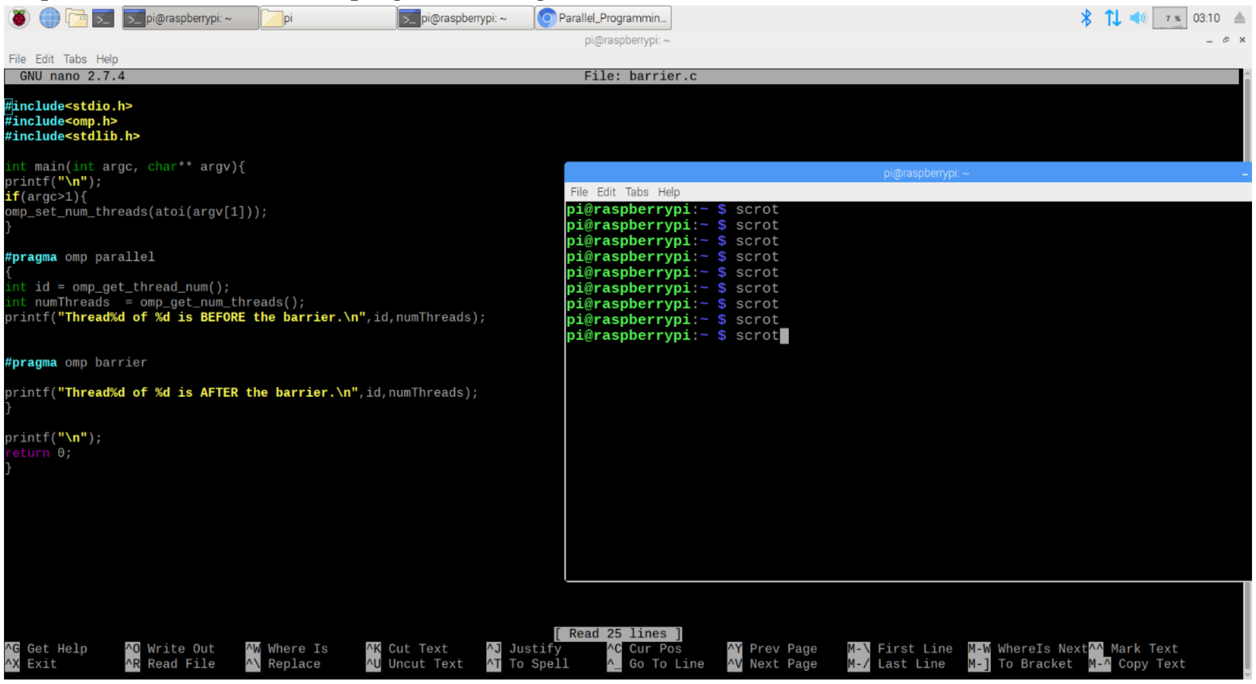
I created an executable for this program and ran it with the “#pragma omp barrier” line still commented out.



```
pi@raspberrypi:~ $ ./barrier 4

Thread0 of 4 is BEFORE the barrier.
Thread0 of 4 is AFTER the barrier.
Thread1 of 4 is BEFORE the barrier.
Thread1 of 4 is AFTER the barrier.
Thread3 of 4 is BEFORE the barrier.
Thread3 of 4 is AFTER the barrier.
Thread2 of 4 is BEFORE the barrier.
Thread2 of 4 is AFTER the barrier.

pi@raspberrypi:~ $ ./barrier 4

Thread2 of 4 is BEFORE the barrier.
Thread2 of 4 is AFTER the barrier.
Thread0 of 4 is BEFORE the barrier.
Thread0 of 4 is AFTER the barrier.
Thread1 of 4 is BEFORE the barrier.
Thread1 of 4 is AFTER the barrier.
Thread3 of 4 is BEFORE the barrier.
Thread3 of 4 is AFTER the barrier.

pi@raspberrypi:~ $ ./barrier 4

Thread0 of 4 is BEFORE the barrier.
Thread0 of 4 is AFTER the barrier.
Thread3 of 4 is BEFORE the barrier.
Thread3 of 4 is AFTER the barrier.
Thread1 of 4 is BEFORE the barrier.
Thread1 of 4 is AFTER the barrier.
Thread2 of 4 is BEFORE the barrier.
Thread2 of 4 is AFTER the barrier.

pi@raspberrypi:~ $ ./barrier 4

Thread0 of 4 is BEFORE the barrier.
Thread0 of 4 is AFTER the barrier.
Thread2 of 4 is BEFORE the barrier.
Thread2 of 4 is AFTER the barrier.
Thread3 of 4 is BEFORE the barrier.
Thread3 of 4 is AFTER the barrier.
Thread1 of 4 is BEFORE the barrier.
Thread1 of 4 is AFTER the barrier.

pi@raspberrypi:~ $
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

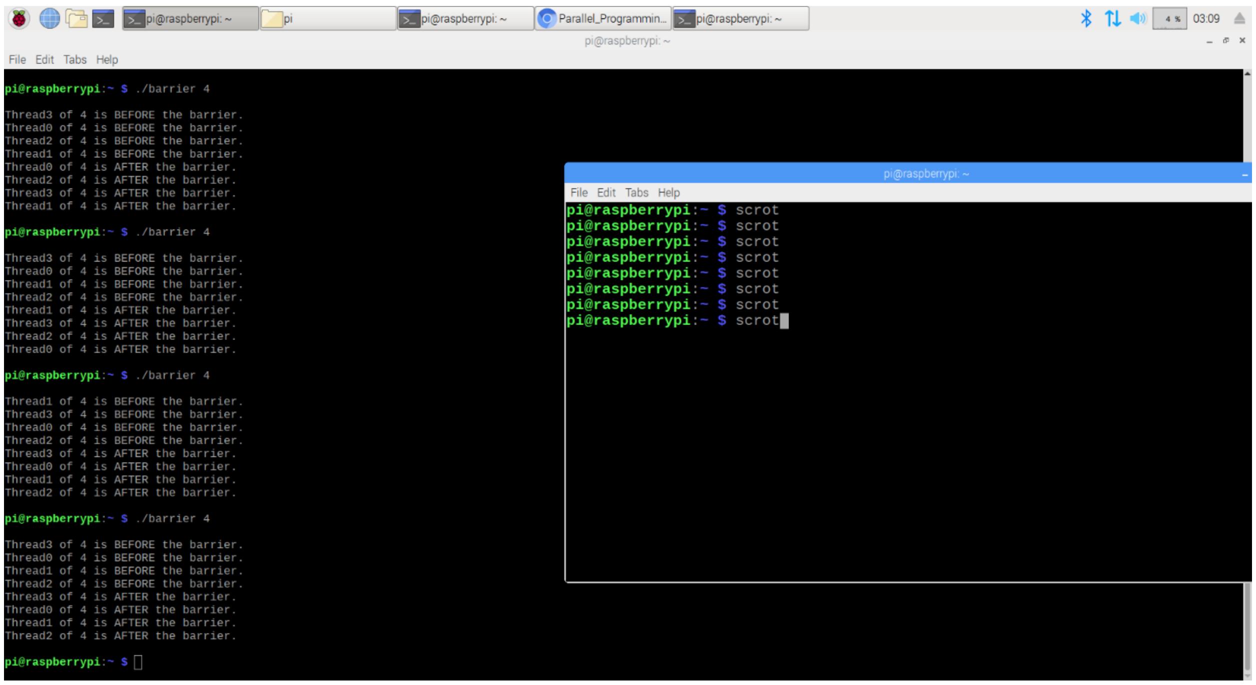
As you can see, with the barrier not effect, the program simply runs the operations in random, but sequential order, one after another without stopping. Continuing the task, I uncommented the “#pragma omp barrier” line and ran the program once again.



```
  GNU nano 2.7.4                                    File: barrier.c

#include<stdio.h>
#include<omp.h>
#include<stdlib.h>

int main(int argc, char** argv){
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("Thread%d of %d is BEFORE the barrier.\n",id,numThreads);


#pragma omp barrier

printf("Thread%d of %d is AFTER the barrier.\n",id,numThreads);
}

printf("\n");
return 0;
}

                                            [ Read 25 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page  M-\ First Line M-W WhereIs Next^^ Mark Text
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line ^V Next Page  M-/ Last Line  M-] To Bracket M-^ Copy Text
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

With the barrier in effect this time, you can see that the threads stop before the barrier and continue executing together after the barrier, in contrast to what we observed earlier without the barrier.

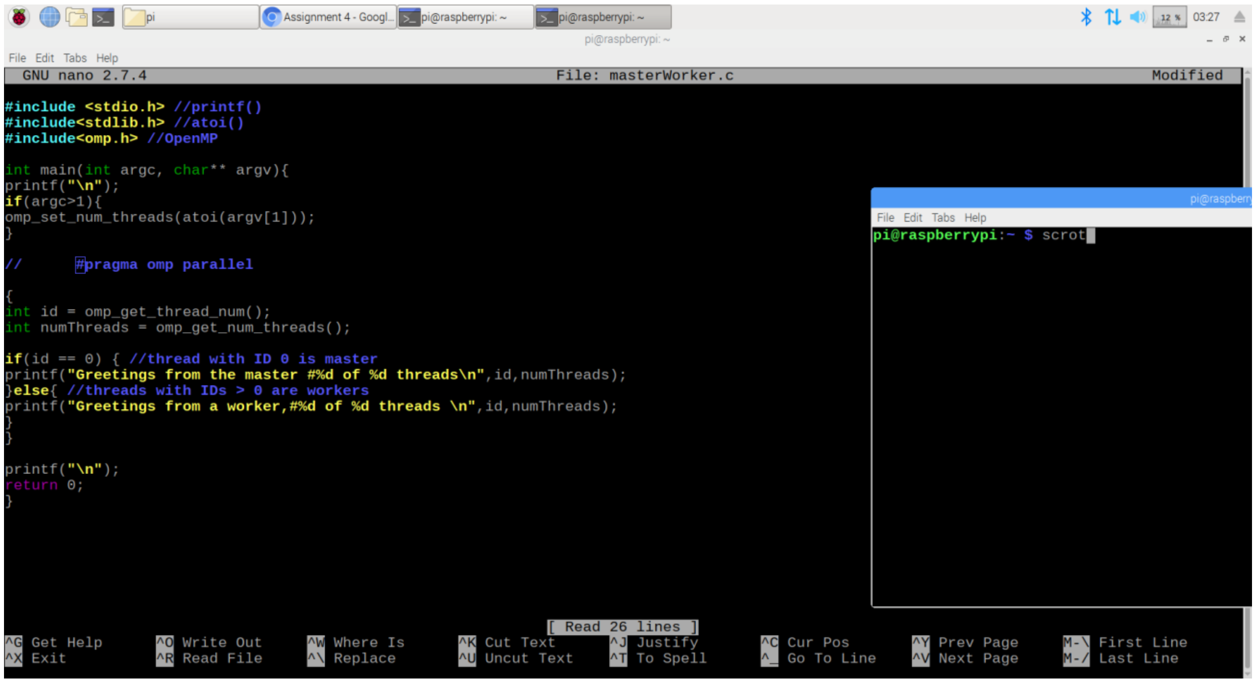


```
pi@raspberrypi:~ $ ./barrier 4

Thread3 of 4 is BEFORE the barrier.
Thread0 of 4 is BEFORE the barrier.
Thread2 of 4 is BEFORE the barrier.
Thread1 of 4 is BEFORE the barrier.
Thread0 of 4 is AFTER the barrier.
Thread2 of 4 is AFTER the barrier.
Thread3 of 4 is AFTER the barrier.
Thread1 of 4 is AFTER the barrier.

pi@raspberrypi:~ $ ./barrier 4

Thread3 of 4 is BEFORE the barrier.
Thread0 of 4 is BEFORE the barrier.
Thread1 of 4 is BEFORE the barrier.
Thread2 of 4 is BEFORE the barrier.
Thread1 of 4 is AFTER the barrier.
Thread3 of 4 is AFTER the barrier.
Thread2 of 4 is AFTER the barrier.
Thread0 of 4 is AFTER the barrier.

pi@raspberrypi:~ $ ./barrier 4

Thread1 of 4 is BEFORE the barrier.
Thread3 of 4 is BEFORE the barrier.
Thread0 of 4 is BEFORE the barrier.
Thread2 of 4 is BEFORE the barrier.
Thread3 of 4 is AFTER the barrier.
Thread0 of 4 is AFTER the barrier.
Thread1 of 4 is AFTER the barrier.
Thread2 of 4 is AFTER the barrier.

pi@raspberrypi:~ $ ./barrier 4

Thread3 of 4 is BEFORE the barrier.
Thread0 of 4 is BEFORE the barrier.
Thread1 of 4 is BEFORE the barrier.
Thread2 of 4 is BEFORE the barrier.
Thread3 of 4 is AFTER the barrier.
Thread0 of 4 is AFTER the barrier.
Thread1 of 4 is AFTER the barrier.
Thread2 of 4 is AFTER the barrier.

pi@raspberrypi:~ $
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

Part Three: Program Structure: The Master-Worker Implementation Strategy

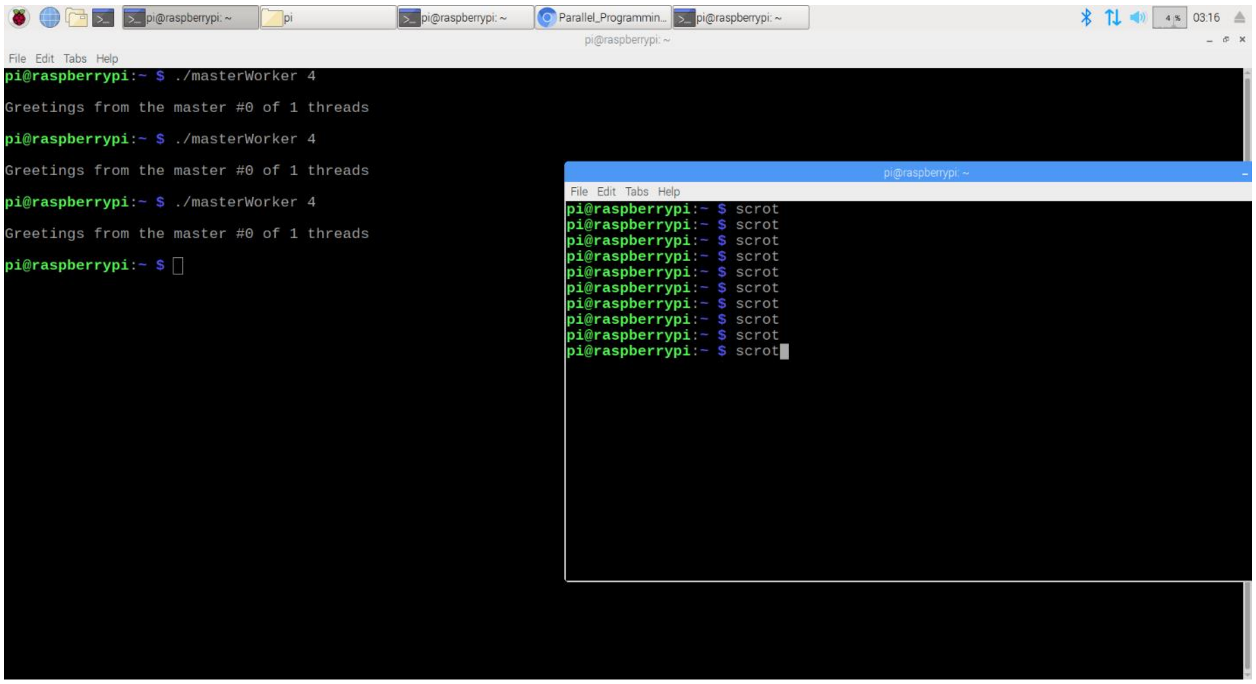
For this portion of the task, I created a file called masterWorker.c and once again copied the code from the provided pdf.



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

#include <stdio.h> //printf()
#include<stdlib.h> //atoi()
#include<omp.h> //OpenMP

int main(int argc, char** argv){
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();

if(id == 0) { //thread with ID 0 is master
printf("Greetings from the master #%d of %d threads\n",id,numThreads);
}else{ //threads with IDs > 0 are workers
printf("Greetings from a worker,#%d of %d threads \n",id,numThreads);
}
}

printf("\n");
return 0;
}

                                        [ Read 26 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     ^Y Prev Page   M-\ First Line
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  ^V Next Page   M-/ Last Line
```

```
pi@raspberrypi:~ $ scrot
```

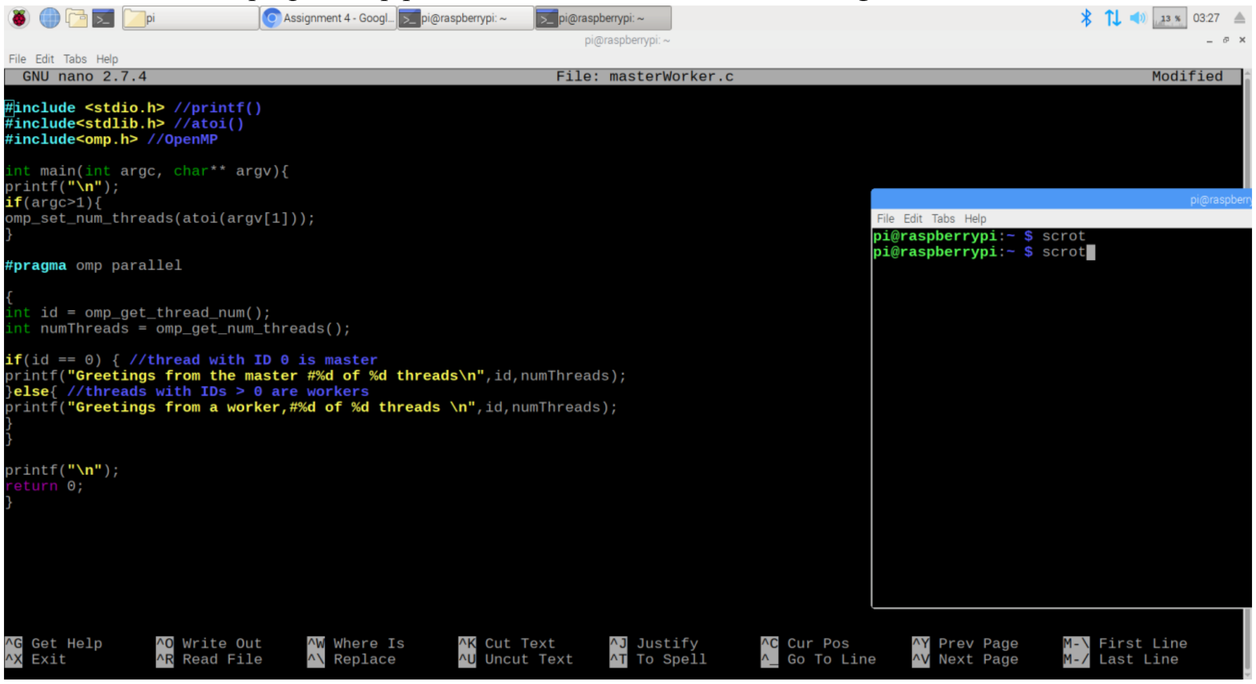
I created an executable to check the results of the program with the “#pragma omp parallel” line commented out.



```
pi@raspberrypi:~ $ ./masterWorker 4

Greetings from the master #0 of 1 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from the master #0 of 1 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from the master #0 of 1 threads

pi@raspberrypi:~ $
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

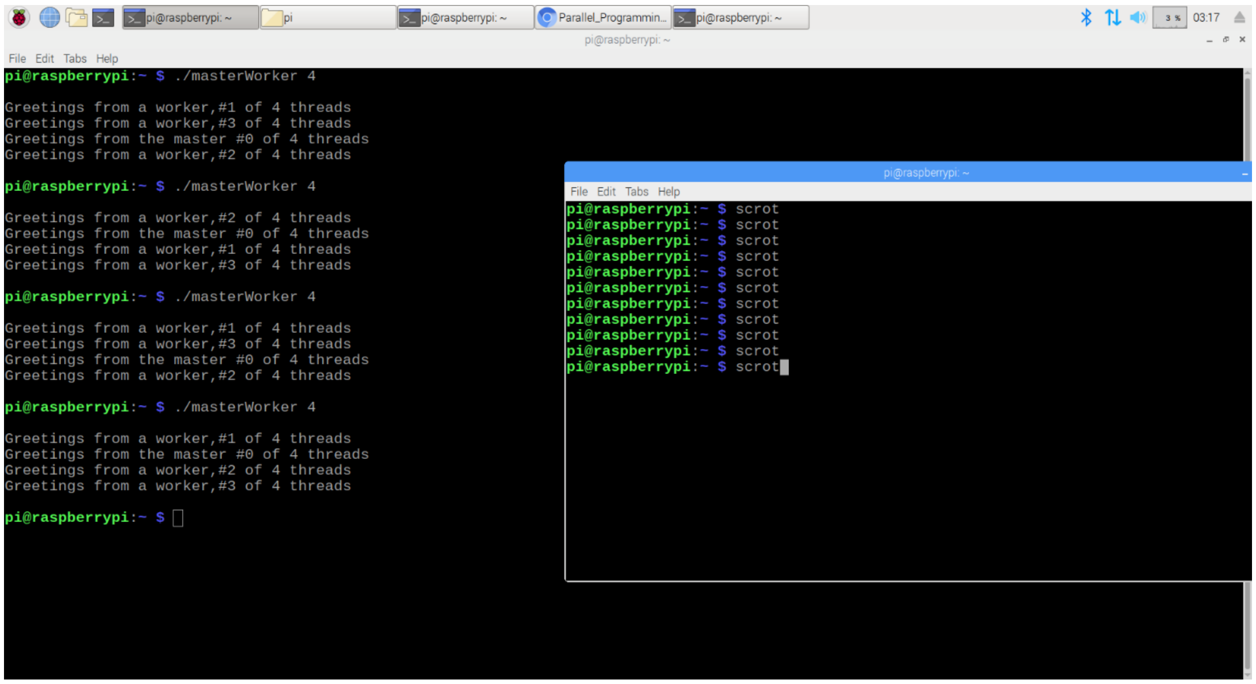
As expected, only the first condition of the if statement was executed since it was not run parallel. I uncommented the "#pragma omp parallel" line and checked the results again.



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

#include <stdio.h> //printf()
#include<stdlib.h> //atoi()
#include<omp.h> //OpenMP

int main(int argc, char** argv){
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();

if(id == 0) { //thread with ID 0 is master
printf("Greetings from the master #%d of %d threads\n",id,numThreads);
}else{ //threads with IDs > 0 are workers
printf("Greetings from a worker,#%d of %d threads \n",id,numThreads);
}
}

printf("\n");
return 0;
}

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page  M-\ First Line
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line ^V Next Page  M-/ Last Line
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

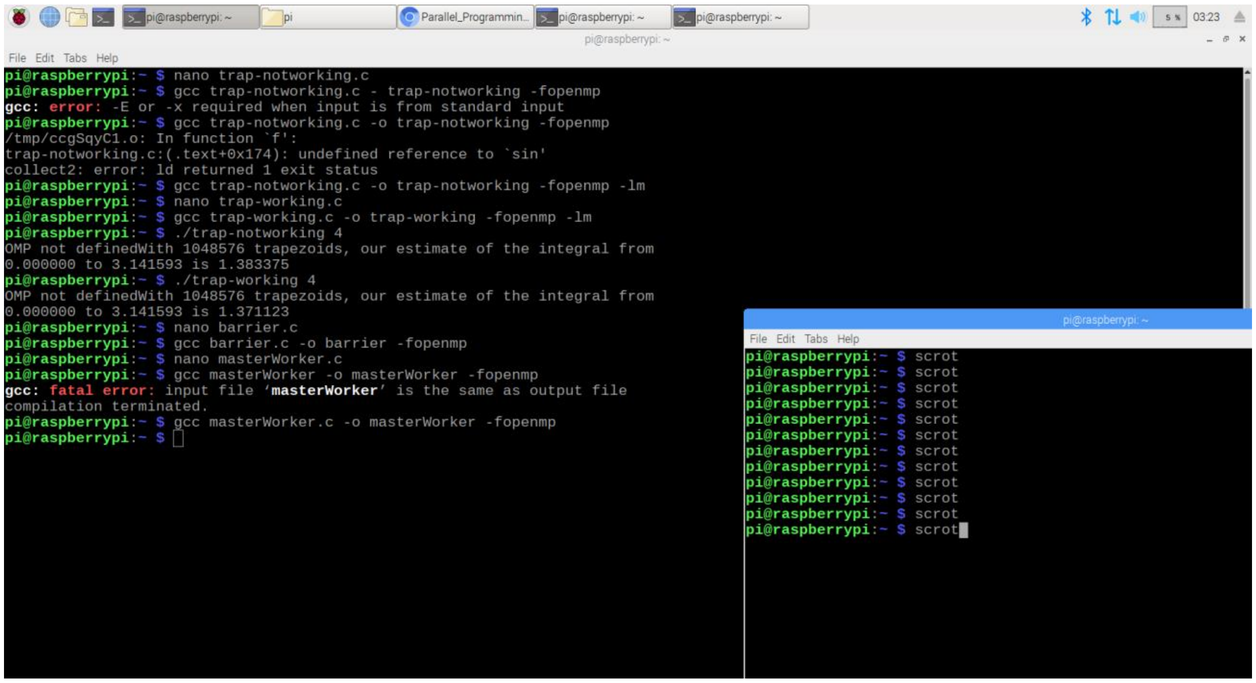
The line is now uncommented. The results shown are below.



```
pi@raspberrypi:~ $ ./masterWorker 4

Greetings from a worker,#1 of 4 threads
Greetings from a worker,#3 of 4 threads
Greetings from the master #0 of 4 threads
Greetings from a worker,#2 of 4 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from a worker,#2 of 4 threads
Greetings from the master #0 of 4 threads
Greetings from a worker,#1 of 4 threads
Greetings from a worker,#3 of 4 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from a worker,#1 of 4 threads
Greetings from a worker,#3 of 4 threads
Greetings from the master #0 of 4 threads
Greetings from a worker,#2 of 4 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from a worker,#1 of 4 threads
Greetings from the master #0 of 4 threads
Greetings from a worker,#2 of 4 threads
Greetings from a worker,#3 of 4 threads

pi@raspberrypi:~ $
```

```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

Now we are able to see the workers and their corresponding thread number. There are only four threads because it is 4 core. Everything is as expected!
Below is an image of the creation of each of the files and their corresponding gcc line which created the executable for each file.



```
pi@raspberrypi:~ $ nano trap-notworking.c
pi@raspberrypi:~ $ gcc trap-notworking.c - trap-notworking -fopenmp
gcc: error: -E or -x required when input is from standard input
pi@raspberrypi:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp
/tmp/ccgSqyC1.o: In function `f':
trap-notworking.c:(.text+0x174): undefined reference to `sin'
collect2: error: ld returned 1 exit status
pi@raspberrypi:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp -lm
pi@raspberrypi:~ $ nano trap-working.c
pi@raspberrypi:~ $ gcc trap-working.c -o trap-working -fopenmp -lm
pi@raspberrypi:~ $ ./trap-notworking 4
OMP not definedWith 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.383375
pi@raspberrypi:~ $ ./trap-working 4
OMP not definedWith 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.371123
pi@raspberrypi:~ $ nano barrier.c
pi@raspberrypi:~ $ gcc barrier.c -o barrier -fopenmp
pi@raspberrypi:~ $ nano masterWorker.c
pi@raspberrypi:~ $ gcc masterWorker -o masterWorker -fopenmp
gcc: fatal error: input file 'masterWorker' is the same as output file
compilation terminated.
pi@raspberrypi:~ $ gcc masterWorker.c -o masterWorker -fopenmp
pi@raspberrypi:~ $
```

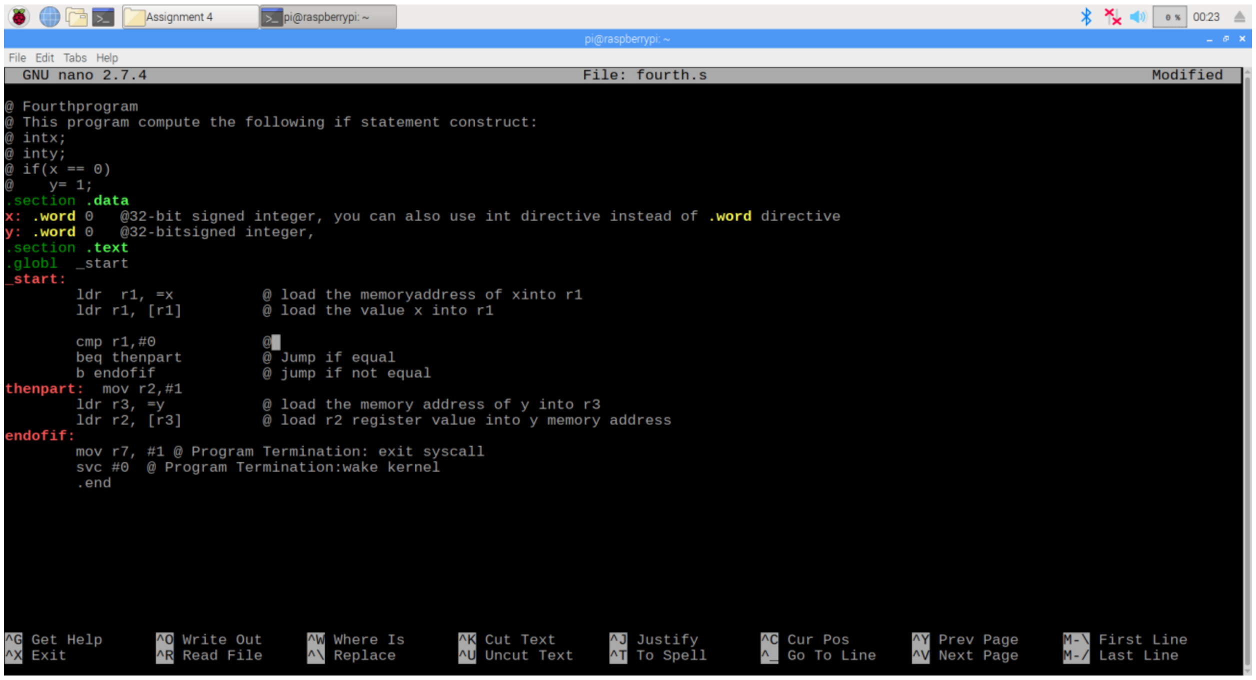
```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

I had to add “-lm” at the end of the gcc lines for the trapezoidal program because it threw an error stating “undefined reference to sin”. That was the only major issue I ran into during the completion of this task.

V. ARM Assembly Programming

3/13 Received the PI
3/23 Because of the extended time of the break I had extra time till my assignment was due But I finally got around to it. Programming begins.

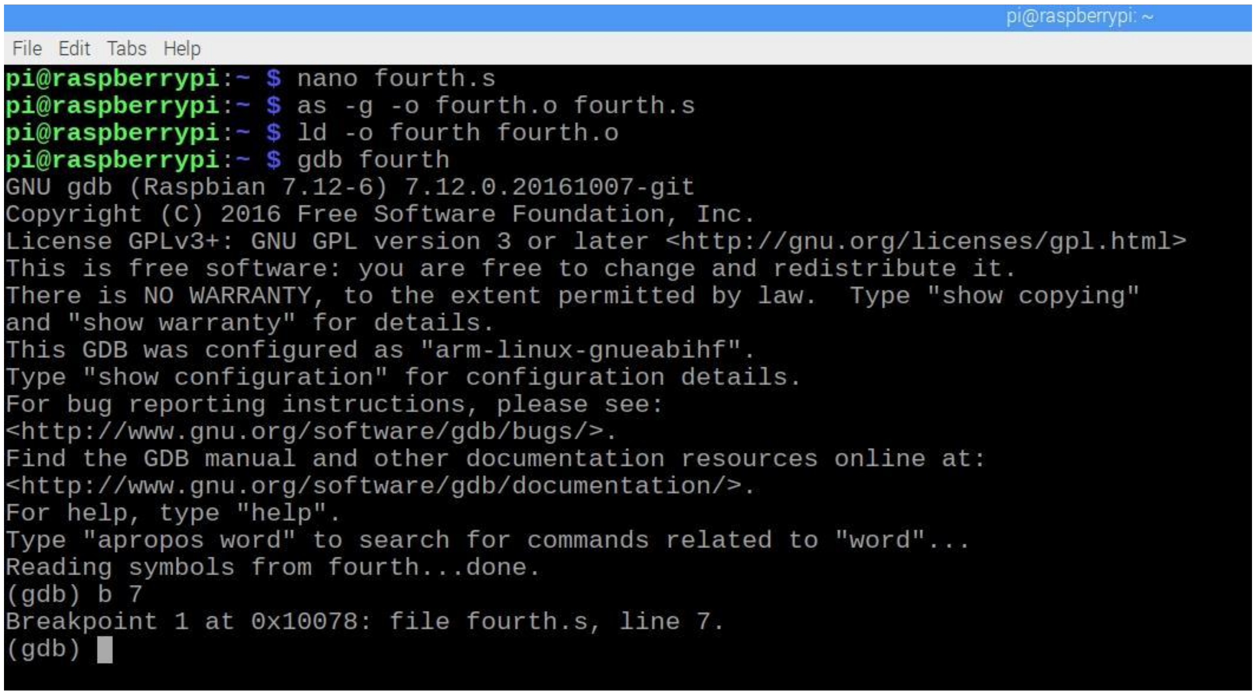
Question 1) I began by copying the given code into a program file named fourth.s



```
  GNU nano 2.7.4                                   File: fourth.s                                    Modified

@ Fourthprogram
@ This program compute the following if statement construct:
@ intx;
@ inty;
@ if(x == 0)
@    y= 1;
.section .data
x: .word 0   @32-bit signed integer, you can also use int directive instead of .word directive
y: .word 0   @32-bitsigned integer,
.section .text
.globl  _start
_start:
        ldr  r1, =x          @ load the memoryaddress of xinto r1
        ldr r1, [r1]         @ load the value x into r1

        cmp r1,#0            @
        beq thenpart         @ Jump if equal
        b endofif            @ jump if not equal
thenpart:  mov r2,#1
        ldr r3, =y           @ load the memory address of y into r3
        ldr r2, [r3]         @ load r2 register value into y memory address
endofif:
        mov r7, #1 @ Program Termination: exit syscall
        svc #0  @ Program Termination:wake kernel
        .end
```

I noticed some issues which would not allow my program to be assembled correctly so I corrected them in the code like adding a “#” before 1 in the line that begins with “thenpart:”

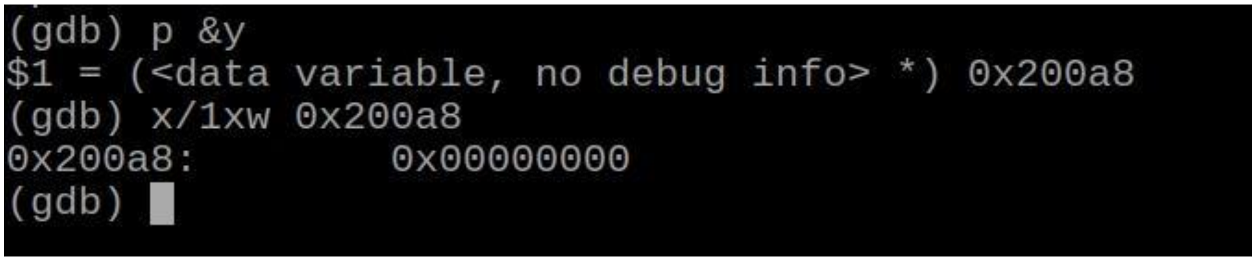
After correcting the mistakes I assembled the program and began the debugging section



```
pi@raspberrypi:~ $ nano fourth.s
pi@raspberrypi:~ $ as -g -o fourth.o fourth.s
pi@raspberrypi:~ $ ld -o fourth fourth.o
pi@raspberrypi:~ $ gdb fourth
GNU gdb (Raspbian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from fourth...done.
(gdb) b 7
Breakpoint 1 at 0x10078: file fourth.s, line 7.
(gdb) 
```

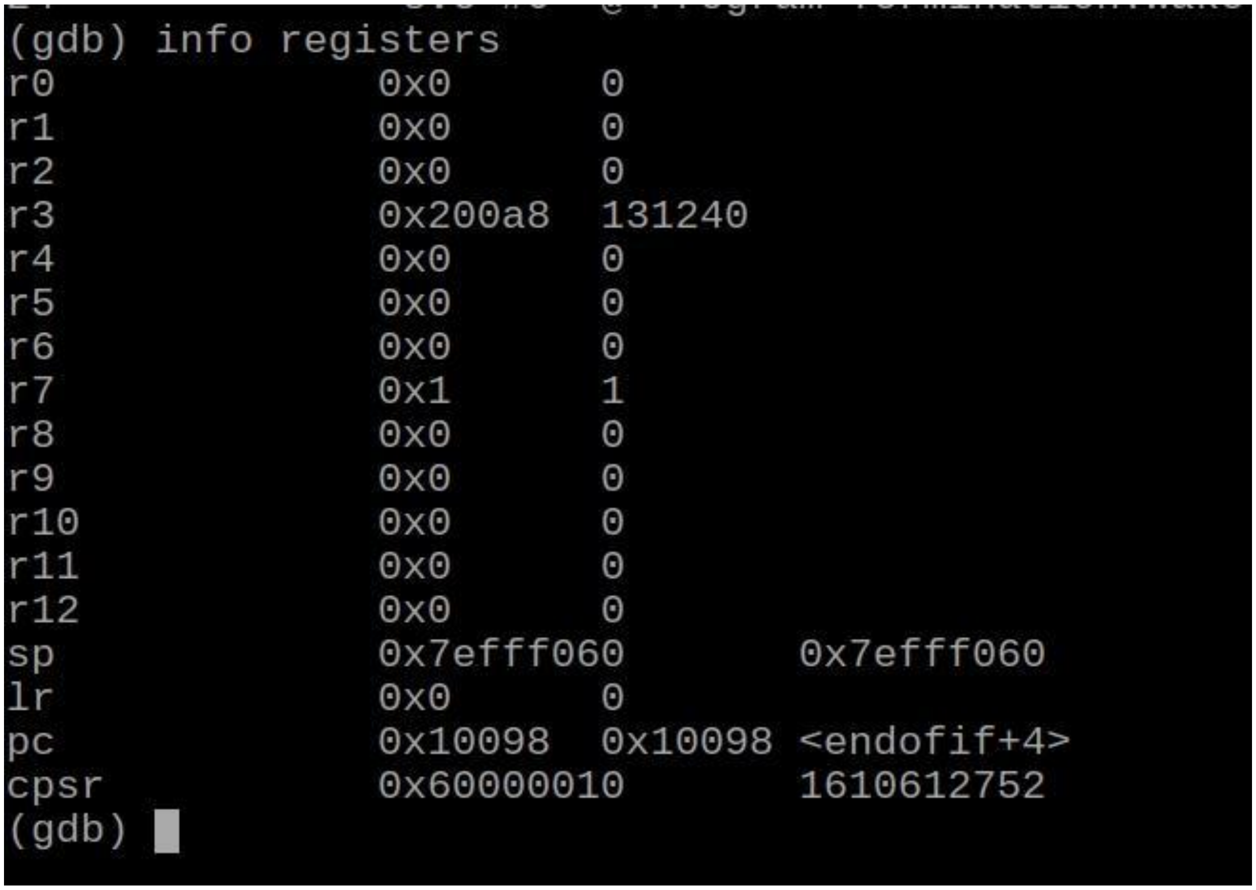
After seeing up the breakpoint and running the program, I stepped through the program far enough to see the affected values.

First the Y (using &y to find the address and then x/1xw address to retrieve the value)



```
(gdb) p &y
$1 = (<data variable, no debug info> *) 0x200a8
(gdb) x/1xw 0x200a8
0x200a8:        0x00000000
(gdb) 
```

Then I locate the value of the Z-flag by using "info registers" and determining it through the cpsr register

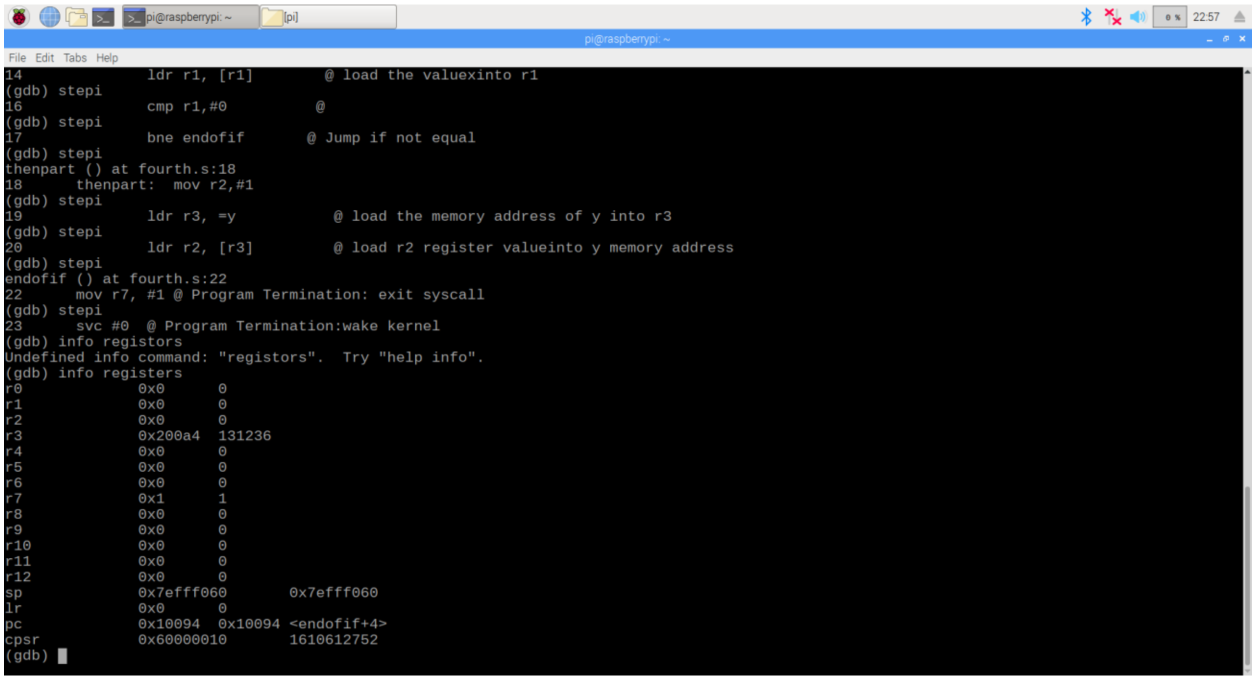


```
(gdb) info registers
r0             0x0      0
r1             0x0      0
r2             0x0      0
r3             0x200a8  131240
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x1      1
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff060       0x7efff060
lr             0x0      0
pc             0x10098  0x10098 <endofif+4>
cpsr           0x60000010       1610612752
(gdb)
```

This process was a bit confusing but I learned that cpsr is the register that holds flag values. To find the Z-flag we convert the "60000010" from hex to binary getting 0110 0000 0000… (The rest does not matter for this assignment). We can use prior knowledge to tell that the second binary number from the right represents the status of the Z-Flag. The Z-flag is 1 which is what we are looking for.

2) Question 2 **-** Again I am assuming there was an error in the instructions as it tells us to replace beq with "bnq" however it appears to be referencing "bne" so I used that instead.
Other that that I just removed the the line b endofit and replace the thenpart to endpart.

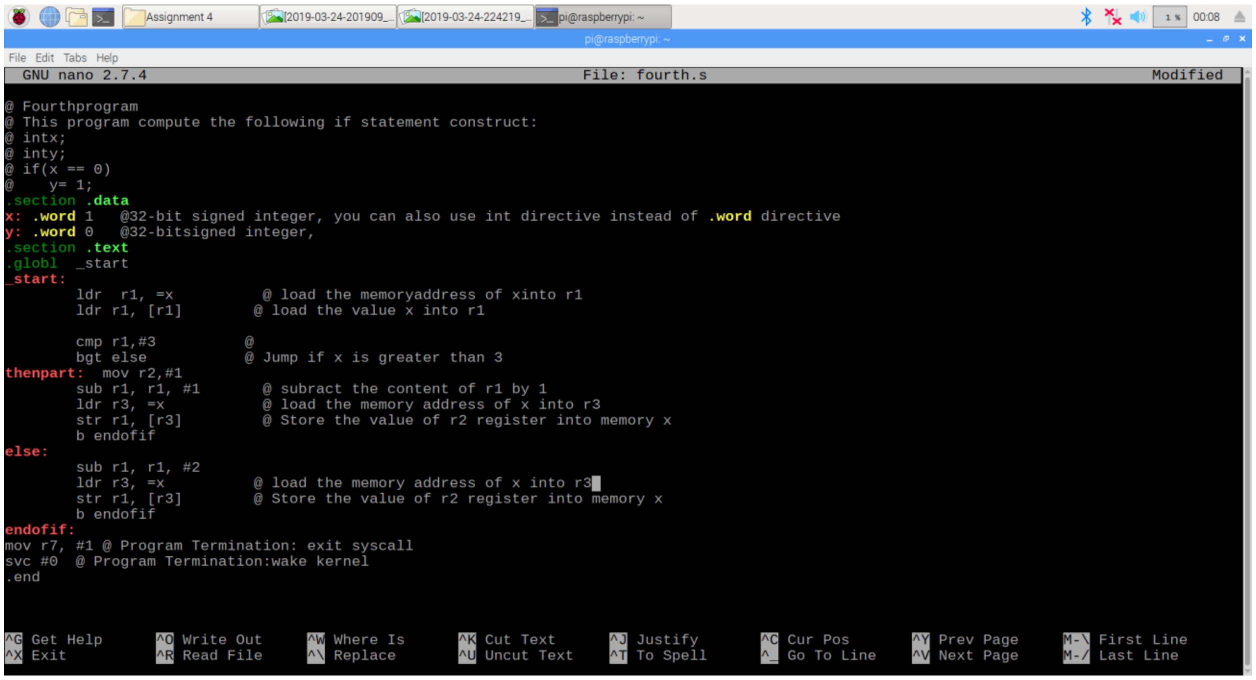
I then assembled the code, ran it through debugging and found the Z-flag through the cpsr just like before.



```
14              ldr r1, [r1]        @ load the valuexinto r1
(gdb) stepi
16              cmp r1,#0          @
(gdb) stepi
17              bne endofif       @ Jump if not equal
(gdb) stepi
thenpart () at fourth.s:18
18        thenpart:  mov r2,#1
(gdb) stepi
19              ldr r3, =y           @ load the memory address of y into r3
(gdb) stepi
20              ldr r2, [r3]         @ load r2 register valueinto y memory address
(gdb) stepi
endofif () at fourth.s:22
22        mov r7, #1 @ Program Termination: exit syscall
(gdb) stepi
23        svc #0  @ Program Termination:wake kernel
(gdb) info registors
Undefined info command: "registors".  Try "help info".
(gdb) info registers
r0             0x0      0
r1             0x0      0
r2             0x0      0
r3             0x200a4  131236
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x1      1
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff060       0x7efff060
lr             0x0      0
pc             0x10094  0x10094 <endofif+4>
cpsr           0x60000010       1610612752
(gdb)
```

Just as before, I found cpsr and translated the value given (60...) to binary (01100000…)
Finding that the Z-flag is set just as expected.

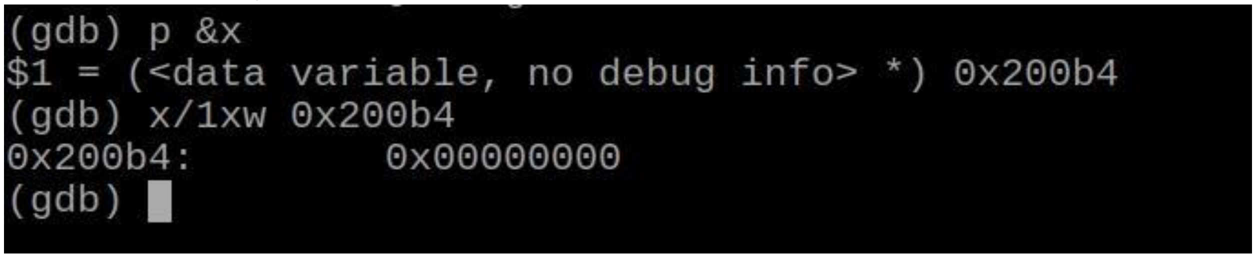
Question 3) - Next I had to edit fourth program to make a specific conditional statement that was specified, the issue here was deciding what conditional branch jump commands that I needed for the particular problem. Here is the code:



```
  GNU nano 2.7.4                          File: fourth.s                                Modified

@ Fourthprogram
@ This program compute the following if statement construct:
@ intx;
@ inty;
@ if(x == 0)
@    y= 1;
.section .data
x: .word 1   @32-bit signed integer, you can also use int directive instead of .word directive
y: .word 0   @32-bitsigned integer,
.section .text
.globl  _start
_start:
        ldr  r1, =x          @ load the memoryaddress of xinto r1
        ldr r1, [r1]        @ load the value x into r1

        cmp r1,#3          @
        bgt else           @ Jump if x is greater than 3
thenpart:  mov r2,#1
        sub r1, r1, #1       @ subract the content of r1 by 1
        ldr r3, =x           @ load the memory address of x into r3
        str r1, [r3]         @ Store the value of r2 register into memory x
        b endofif
else:
        sub r1, r1, #2
        ldr r3, =x          @ load the memory address of x into r3
        str r1, [r3]        @ Store the value of r2 register into memory x
        b endofif
endofif:
mov r7, #1 @ Program Termination: exit syscall
svc #0  @ Program Termination:wake kernel
.end

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page  M-\ First Line
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line ^V Next Page  M-/ Last Line
```

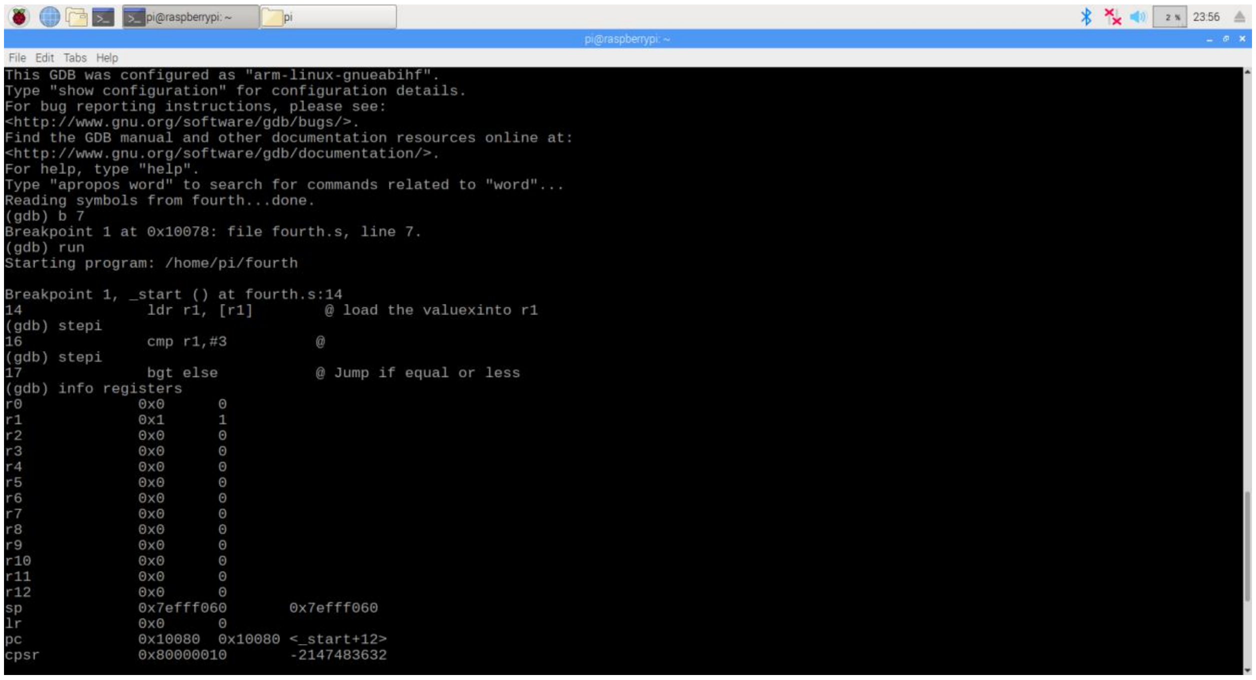
In this code I replace bne with bgt (which jumps to ____ when X is greater than 3) and had it jump to else, thus skipping thenpart. We were asked to find “if x<=3” but it was simpler to use the inverse statement (x>3) to skip to the else statement. I also added a line “b endofif” which skips the else statement and goes straight to the end of the code thus making it impossible to have both if and else happen at the same time.

I found the X value by:



```
(gdb) p &x
$1 = (<data variable, no debug info> *) 0x200b4
(gdb) x/1xw 0x200b4
0x200b4:        0x00000000
(gdb)
```

And the Z-flag, the same as before by:



```
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from fourth...done.
(gdb) b 7
Breakpoint 1 at 0x10078: file fourth.s, line 7.
(gdb) run
Starting program: /home/pi/fourth

Breakpoint 1, _start () at fourth.s:14
14              ldr r1, [r1]        @ load the valuexinto r1
(gdb) stepi
16              cmp r1,#3          @
(gdb) stepi
17              bgt else           @ Jump if equal or less
(gdb) info registers
r0             0x0      0
r1             0x1      1
r2             0x0      0
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff060       0x7efff060
lr             0x0      0
pc             0x10080  0x10080 <_start+12>
cpsr           0x80000010       -2147483632
```

These values matched with what we were looking for using the given x=1. The Z-flag is set to 0 which means that 1-3!=0 which is true. The x value is 0 as 1-3=-2 which is Less than 3 so we would subtract x-1 which is 1-1=0 which is what we got!

VI. Appendix

A. Link to Slack, GitHub, YouTube Channel, YouTube Video

Slack:
https://csc-3210.slack.com

GitHub:
https://github.com/jupitorink

GitHub Assignment One Repository:
https://github.com/jupitorink/Assignment4

YouTube Channel:
https://www.youtube.com/channel/UChTgBqHzG9G-Mvv-q7uK96g

YouTube Video:
https://www.youtube.com/watch?v=ZP6kCrWliCQ

B. Task One Screenshot

Task Assignment Table:
https://github.com/jupitorink/Assignment-3/blob/master/Task%20Table.png

C. Task Two Screenshot

GitHub Projects Page:
https://github.com/jupitorink/Assignment4/projects/1

D. Task Three Parallel Programming Report, Screenshots

Parallel Programming Report
https://github.com/jupitorink/Assignment4/blob/master/parallelprogramming%20report%20assign4.docx

E. Task Four ARM Assembly Programming Report, Screenshots

ARM Assembly Programming Report
https://github.com/jupitorink/Assignment4/blob/master/Task%204-Write%20up.docx