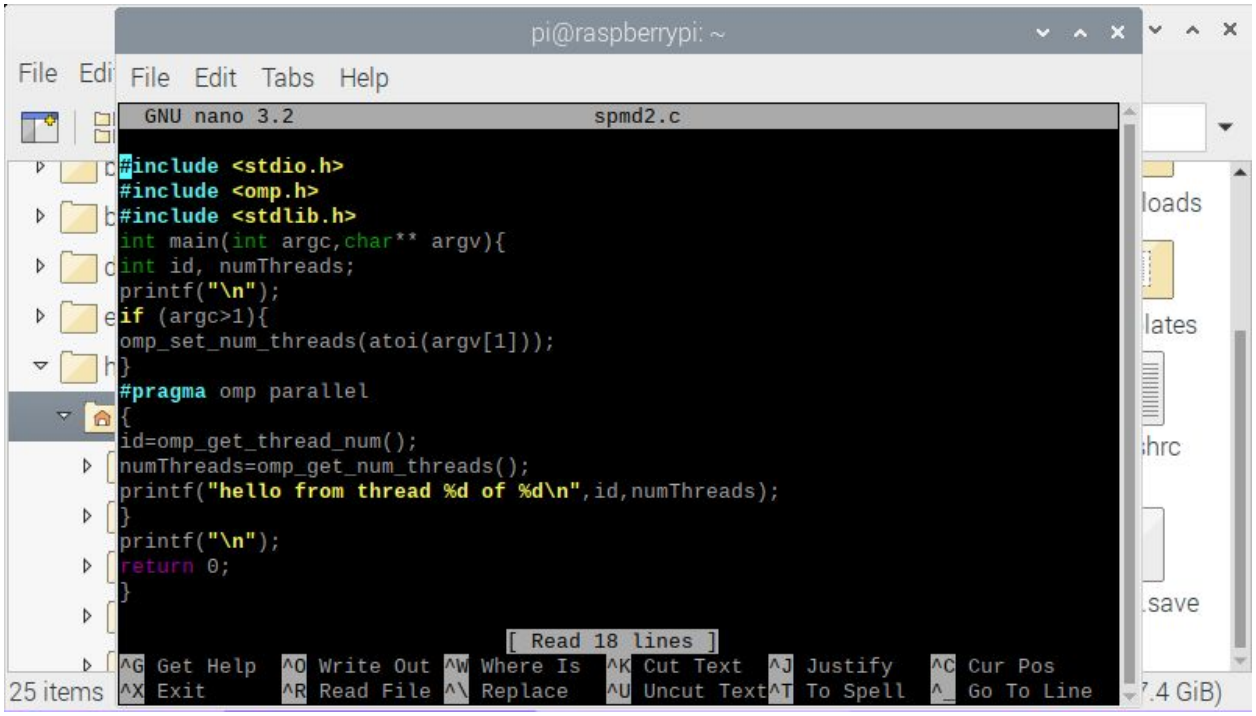
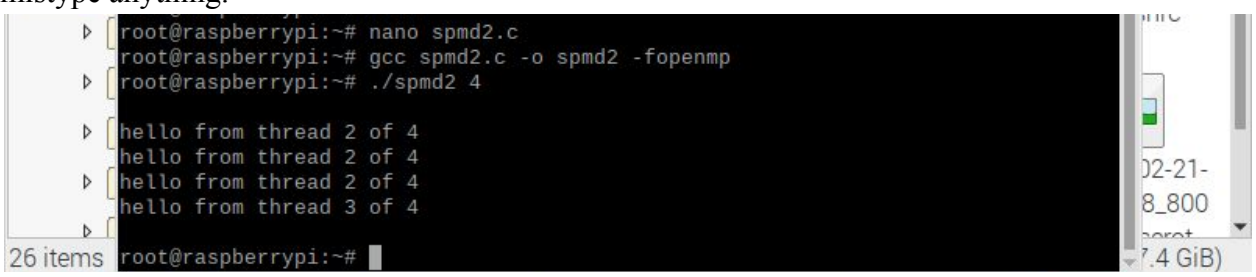
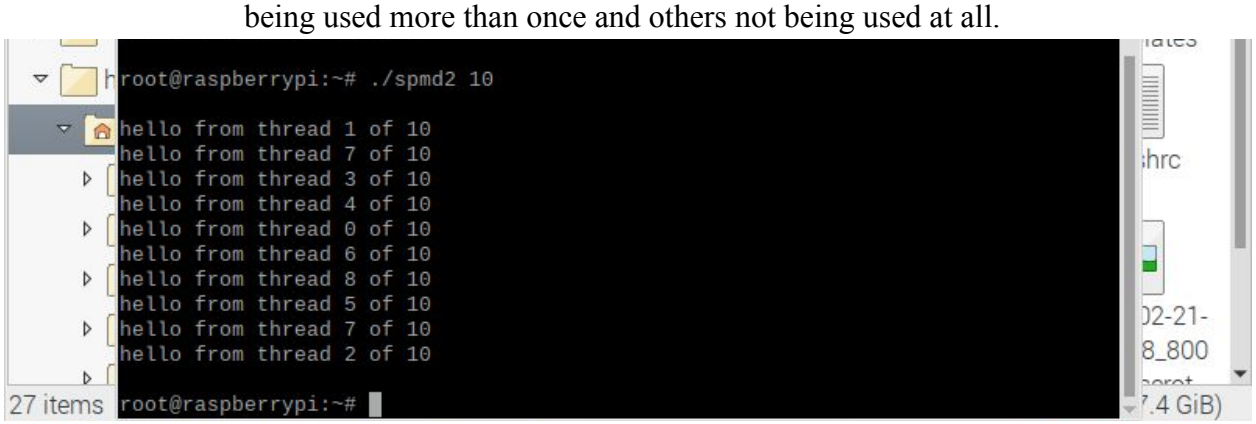
Parallel Programming Task



I began by typing out the program as listed in the instructions, checking over it to be sure I didn't mistype anything.



Next, I ran the program using the dot slash command and tested the output with 4 threads. I tested again with 10 and 2 threads. I noticed that not all threads were being used, with some being used more than once and others not being used at all.



```
root@raspberrypi:~# ./spmd2 2

hello from thread 0 of 2
hello from thread 1 of 2

root@raspberrypi:~#
```

Next, I altered the code to the specifications in the instructions. I decided to rename this altered version spmd3.c, so I wouldn't confuse it with the prior program.

```
GNU nano 3.2                    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;
}
```

I linked the file and ran it, and then it began using each thread once and only once. I again tested it with the inputs 4, 10, and 2, just to be sure my outputs were consistent across all initial inputs.

```
root@raspberrypi:~# gcc spmd3.c -o spmd3 -fopenmp
root@raspberrypi:~# ./spmd3 4

hello from thread 0 of 4
hello from thread 1 of 4
hello from thread 3 of 4
hello from thread 2 of 4

root@raspberrypi:~#
```

```
root@raspberrypi:~# ./spmd3 10

hello from thread 5 of 10
hello from thread 8 of 10
hello from thread 2 of 10
hello from thread 9 of 10
hello from thread 6 of 10
hello from thread 0 of 10
hello from thread 4 of 10
hello from thread 1 of 10
hello from thread 7 of 10
hello from thread 3 of 10

root@raspberrypi:~#
```

They were all consistent, confirming the changes fixed the initial problem.

ARM Assembly Programming Task

Using second.s as a template, I wrote up a version of arithmetic2 that loaded the memory address and then values of var1, var2, and var3 into r1, r2, and r3 respectively. Then I used r0 to store all the arithmetic and finished values as they were being done.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
GNU nano 3.2                    arithmetic2.s

@reg = val2+9+val3-val1
.section .data
val1: .word 6
val2: .word 11
val3: .word 16
.section .text
.globl _start
_start:
        ldr r2,=val2
        ldr r2,[r2]
        ldr r1,=val1
        ldr r1,[r1]
        ldr r3,=val3
        ldr r3,[r3]
        add r0, r2, #9
        add r0, r0, r3
        sub r0, r0, r1

        mov r7,#1
                                [ Wrote 21 lines ]
^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
```

```
pi@raspberrypi:~ $ nano arithmetic2.s
pi@raspberrypi:~ $ as -o arithmetic2.o arithmetic2.s
pi@raspberrypi:~ $ ld -o arithmetic2 arithmetic2.o
pi@raspberrypi:~ $ ./arithmetic2
pi@raspberrypi:~ $
```

After linking arithmetic2, I ran it with the dot slash instruction, and nothing happened because no output is specified. Next, I ran arithmetic2 with the debugger, adding breakpoints at lines 11 and 14.

```
(gdb) list
11              ldr r1,=val1
12              ldr r1,[r1]
13              ldr r3,=val3
14              ldr r3,[r3]
15              add r0, r2, #9
16              add r0, r0, r3
17              sub r0, r0, r1
18
19              mov r7,#1
20              svc #0
(gdb) b 14
Breakpoint 1 at 0x10088: file arithmetic2.s, line 14.
(gdb) b 11
Breakpoint 2 at 0x1007c: file arithmetic2.s, line 11.
(gdb) run
Starting program: /home/pi/arithmetic2

Breakpoint 2, _start () at arithmetic2.s:11
11              ldr r1,=val1
(gdb) x/3xw 0x1007c
0x1007c <_start+8>:     0xe59f1020      0xe5911000      0xe59f301c
(gdb)
```

I checked the memory for three hexadecimal words at the address specified at line 11, and above is what I got. After checking the memory, I used the stepi command to walk through lines 12 to 19, and checked the registers to make sure the program executed correctly.
11 + 9 + 16 - 6 = 30, so this is correct.