تکلیف شماره 3

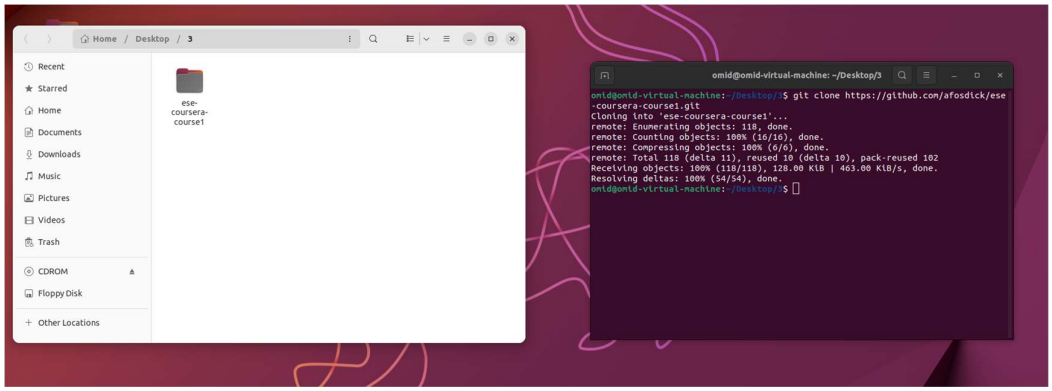
هوش مصنوعی در سیستم های نهفته

ارائه‌دهنده: امید عسکری حداد شماره دانشجویی:4011301110 تاریخ ارائه تمرین: 1402/1/30

باسمه تعالی

این گزارش مربوط به قسمت های الف و ب تمرین 3 است

قسمت الف)



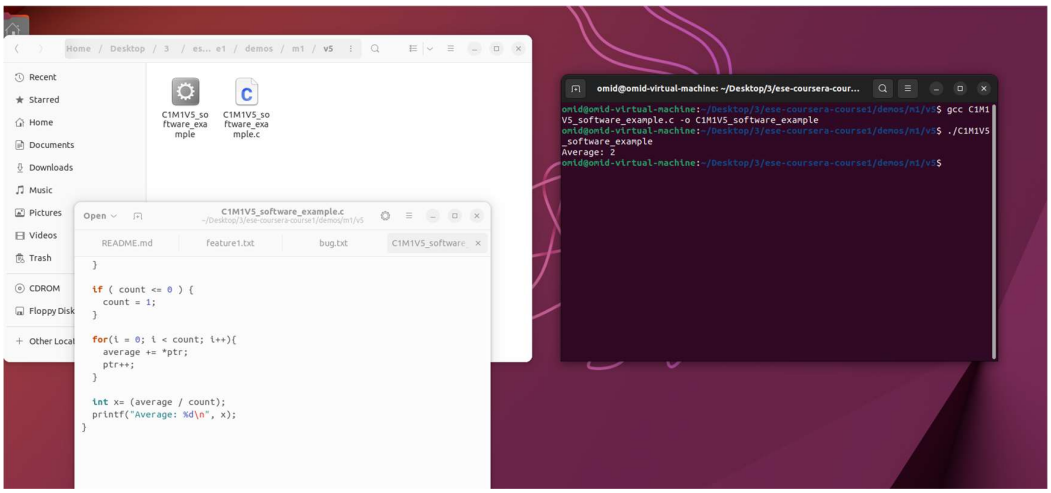
DEMO RESULTS

M1 V5

این کد به زبان C نوشته شده است و میانگین، حداکثر، حداقل و هیستوگرام عددی مجموعه ای از اعداد را پیدا می کند.

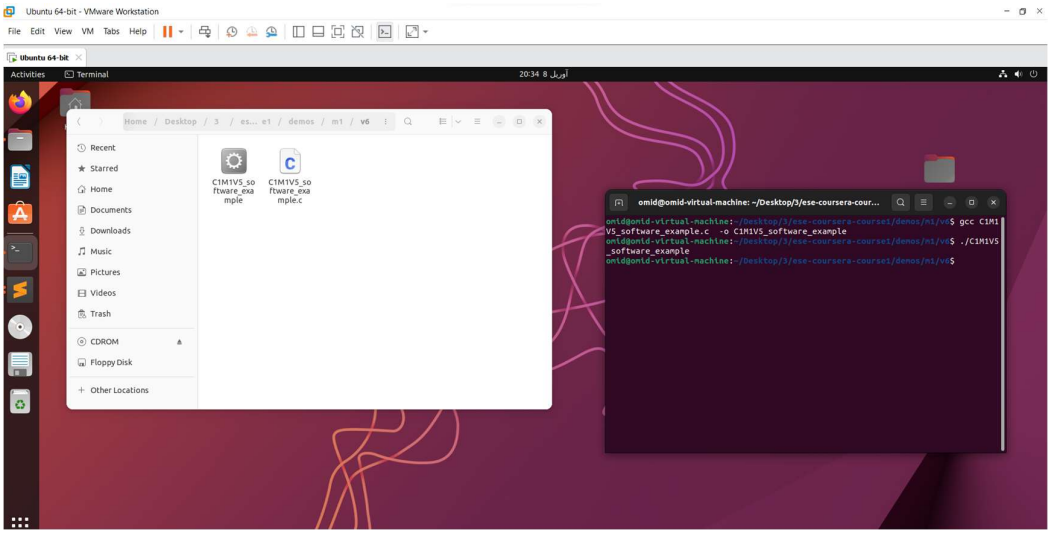
خب برای این قسمت یک قسمت از کد را عوض کردم تا خروجی مشاهده شود زیرا فقط جواب را return میکرد.

من پرینت هم اضافه کردم



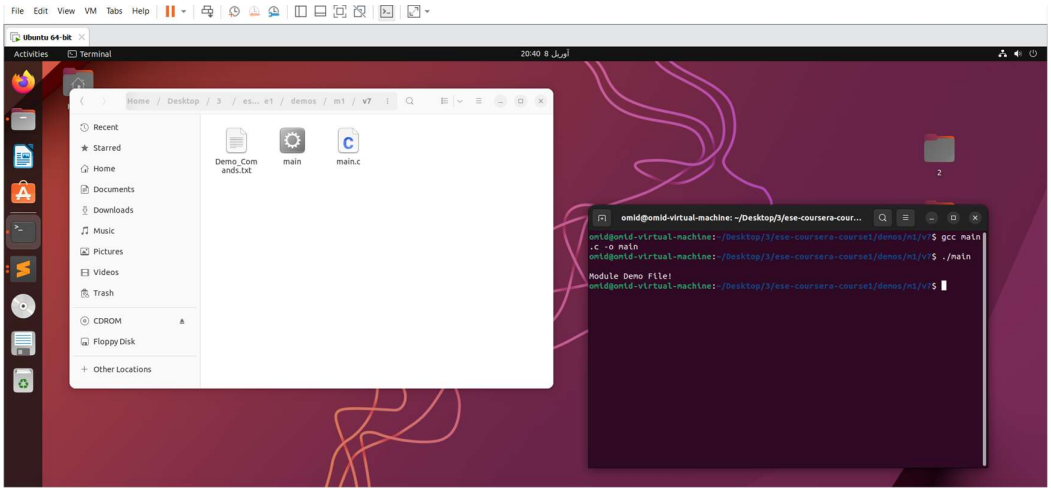
M1 V6

این کد کپی کد قسمت قبل است و هیچ فرقی ندارد



M1 V7

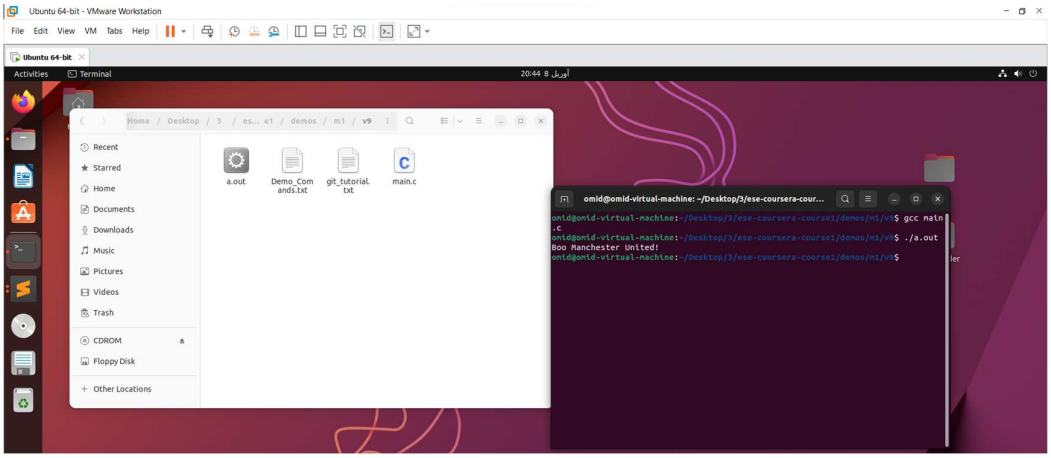
این دمو برای اموزش اجرا کردن کد های C در ترمینال است.



فایل main.c یک برنامه C ساده است که جمله "Module Demo File" را چاپ میکند.

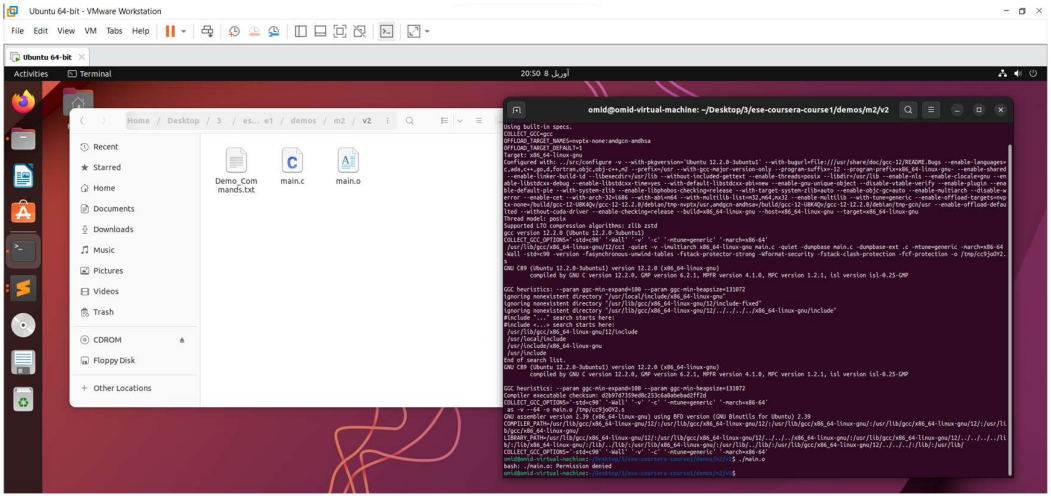
M1 V9

این دمو برای اموزش کار با گیت هاب است.



M2 V2

از این دمو برای اموزش cross compile کردن استفاده میشود.



M2 V3

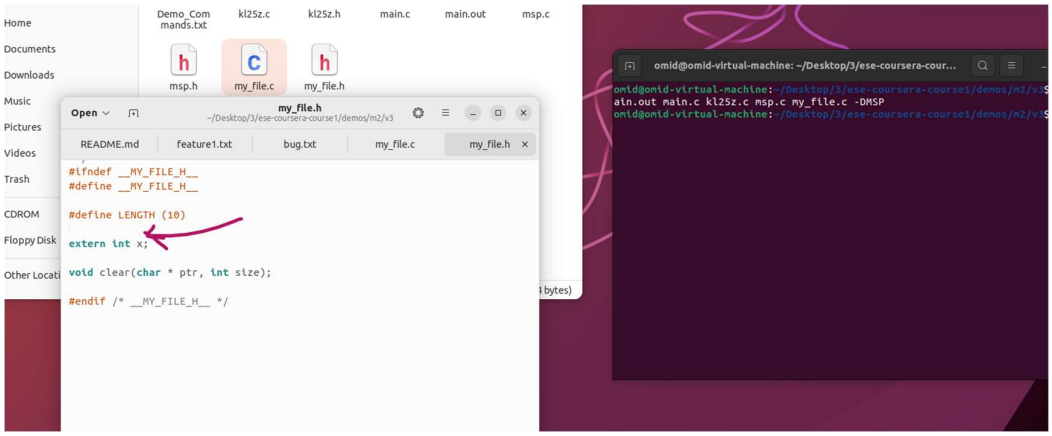
این کد برای اجرا با انتخاب پلتفرم می باشد.هنگامی که برنامه کامپایل می شود، ماکروهای پیش پردازنده DKL25Z یا DMSP را می توان به عنوان پلتفرم در زمان اجرا فراخوانی و استفاده کرد. اگر هر دو ماکرو تعریف شده باشند، برنامه با پیغام خطا خارج می شود.

البته برای اجرای برنامه یک مشکل وجود داشت که کد را درست کردم:

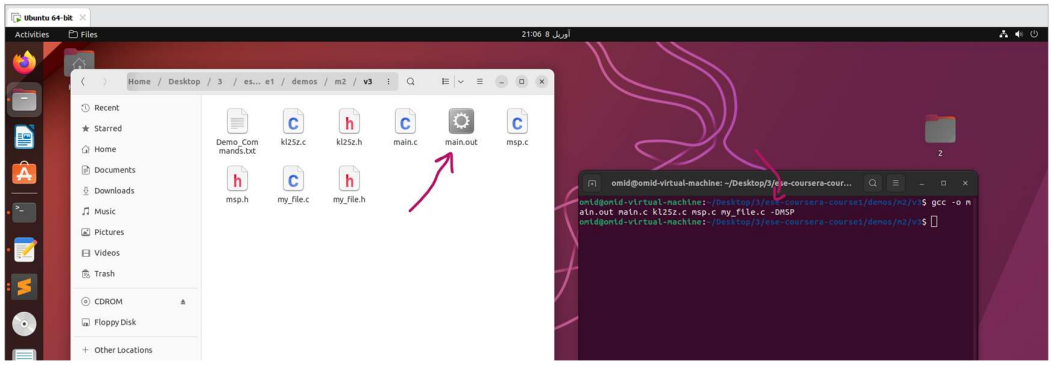
```
omid@omid-virtual-machine:~/Desktop/3/ese-coursera-course1/demos/m2/v3$ gcc -o main.out main.c kl25z.c msp.c my_file.c -DMSP
/usr/bin/ld: /tmp/ccYNBpUo.o:(.bss+0x0): multiple definition of `x';
/tmp/ccA7xtcr.o:(.bss+0x0): first defined here collect2: error: ld returned 1 exit status
```

پیام خطا نشان می دهد که در فایل های main.o و my_file.o چندین تعریف از متغیر x وجود دارد. این اتفاق می افتد زیرا x در فایل هدر my_file.h اعلان شده و در main.c و my_file.c تعریف شده است.

برای رفع این خطا می توان از کلمه کلیدی extern استفاده کرد تا x را در فایل هدر بدون تعریف آن اعلام کرد و سپس آن را در یکی از فایل های پیاده سازی تعریف کرد:




```
any
 * misuse of this material.
 *

************************************************
**/
#include "my_file.h"

int x;

void clear(char * ptr, int size){
  int i;
  for(i = 0; i < size; i++) {
    ptr[i] = 0;
  }
}
```



این دمو ، مفهوم محافظها و نحوه استفاده از آنها برای جلوگیری از خطاهای تعریف مجدد در برنامه‌های C را نشان می‌دهد.

فایل main.c شامل هدرهای "my_file.h" و "my_set.h" است و از توابع تعریف شده در هر دو فایل استفاده می کند. فایل همچنین ماکرو "LENGTH" را تعریف می کند که در "my_file.h"و "my_set.h" استفاده می شود.

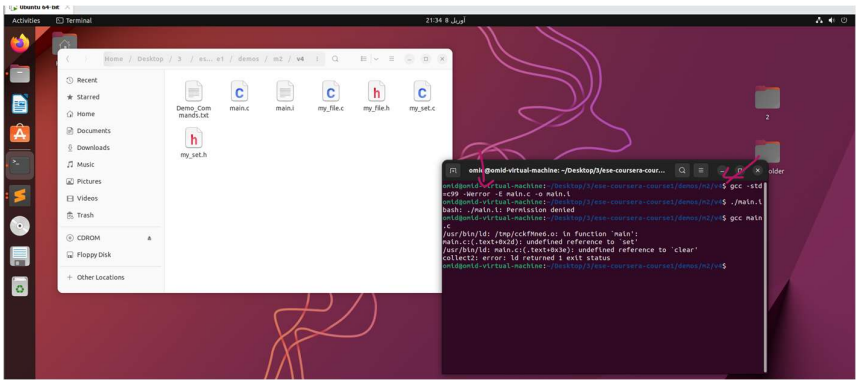
فایل my_file.h حاوی اعلان تابع "clear" و تعریف ماکرو "LENGTH" است. فایل همچنین متغیر جهانی "x" را اعلان می کند.

فایل my_file.c تابع "clear" را تعریف می کند که یک بافر با اندازه مشخص را پاک می کند. این فایل همچنین متغیر جهانی "arr" را که برای ذخیره بافر استفاده می شود، تعریف می کند.

فایل my_set.h حاوی اعلان تابع "set" و تعریف ماکرو "LENGTH" است. فایل همچنین متغیر جهانی "x" را اعلان می کند.

فایل my_set.c تابع "set" را تعریف می کند که یک بافر با اندازه مشخص را به مقدار مشخصی تنظیم می کند. این فایل همچنین متغیر جهانی "arr" را که برای ذخیره بافر استفاده می شود، تعریف می کند.

باید توجه داشت که هر دو "my_file.h" و "my_set.h" یک متغیر جهانی "x" را تعریف می کنند. همچنین، "my_set.h" از محافظ استفاده نمیکند و محافظت نمی شود، بنابراین می تواند چندین بار اضافه شود، که این قضیه می تواند باعث خطاهای تعریف مجدد شود. از طرف دیگر، "my_file.h" با یک محافظ ، محافظت می شود، بنابراین می توان آن را چندین بار بدون مشکل اضافه کرد.



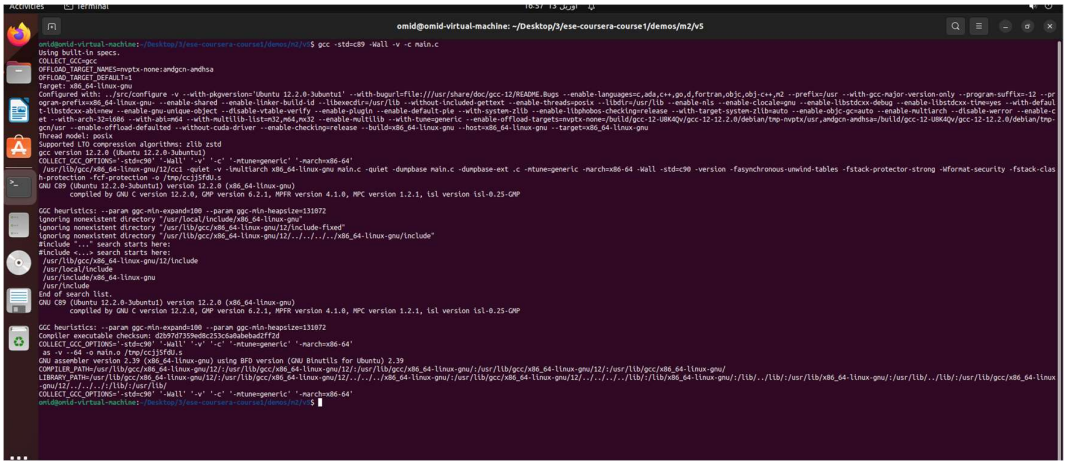
کد ارائه شده از سه فایل main.c، msp432p401r.lds و my_file.c و همچنین فایل هدر my_file.h تشکیل شده است.

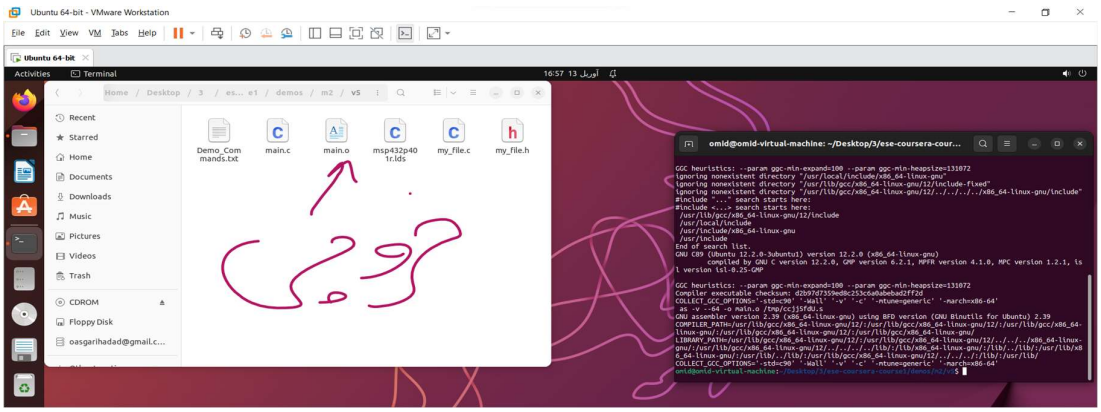
فایل main.c یک آرایه char به نام "arr" با اندازه LENGTH را مقداردهی اولیه می کند و سپس تابع clear را از my_file.c فراخوانی می کند که محتویات آرایه را پاک می کند. سپس 0 را برمی گرداند.

فایل msp432p401r.lds یک فایل اسکریپت پیوند دهنده است که ساختار یک حافظه را نشان میدهد.

فایل my_file.c شامل دو تابع است. اولی یک اشاره گر و اندازه را به عنوان ورودی می گیرد و هر عنصر آرایه را که نشانگر به آن اشاره می کند، 0 می کند. تابع دوم، foo، اولین عنصر آرایه را بر روی 1 قرار می دهد.

فایل my_file.h حاوی اعلان های تابع برای توابع clear و foo و همچنین یک دستورالعمل پیش پردازنده برای تعریف یک مقدار ثابت برای LENGTH است.



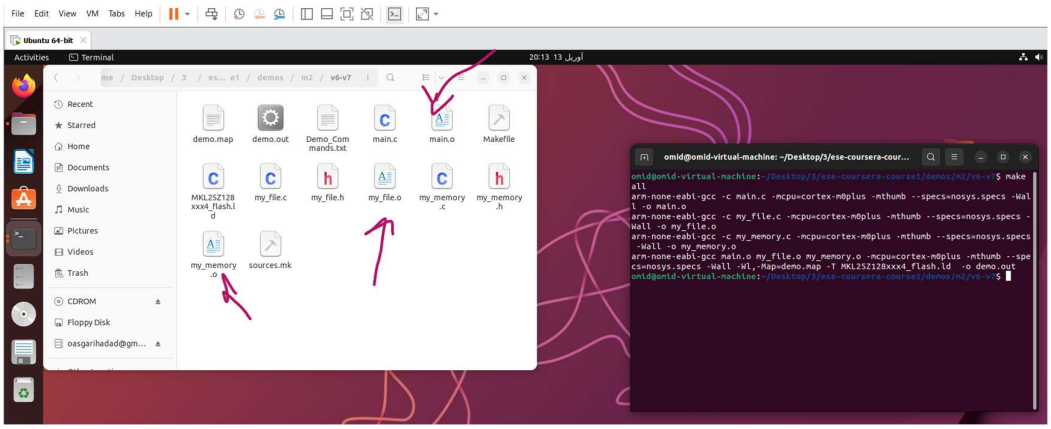


کد داده شده یک اسکریپت لینکر است،و برای میکروکنترلر MKL25Z128 طراحی شده است. چیدمان حافظه میکروکنترلر، از جمله مناطق مختلف حافظه، مثل بخش های خروجی و ورودی را مشخص می کند.

بخش اول کد، میکروکنترلر را به عنوان تابع main تعریف می کند. همچنین اندازه حافظه پشته، بافر را مشخص می کند.

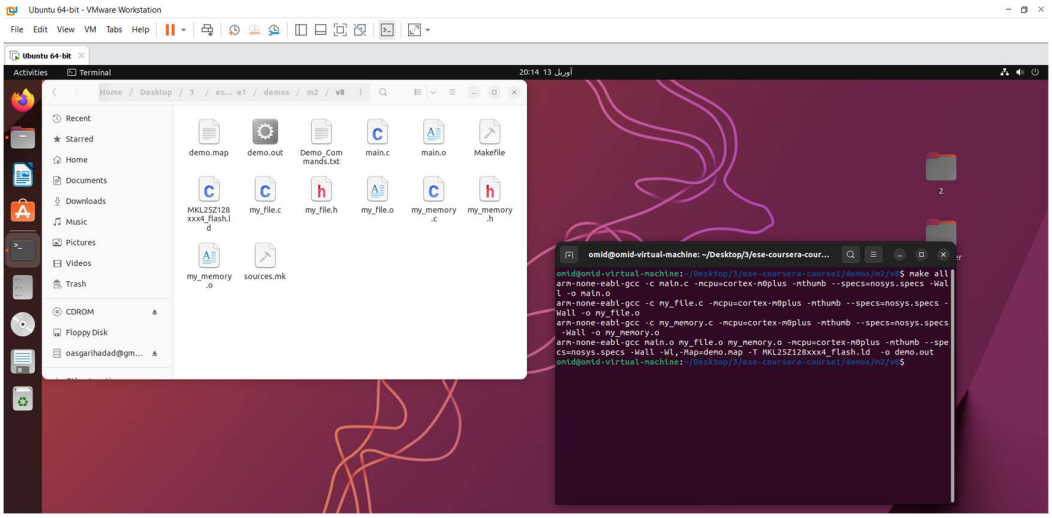
قسمت MEMORY چهار ناحیه حافظه را برای میکروکنترلر تعریف می کند. m_interrupts یک منطقه حافظه فقط خواندنی برای جدول بردار وقفه است، m_flash_config یک منطقه حافظه فقط خواندنی برای فیلد پیکربندی فلش است، m_text یک منطقه حافظه خواندنی برای کد برنامه و سایر داده‌ها است، و m_data یک منطقه خواندن-نوشتن است که در واقع منطقه ای از حافظه برای ذخیره سازی داده هاست.

بخش SECTIONS به عنوان قسمت خروجی برای لینکر تعریف میشود. بخش وقفه حاوی کد راه اندازی میکروکنترلر است که شامل جدول بردار وقفه است. بخش flash_config شامل فیلد پیکربندی فلش است. بخش متن حاوی کد برنامه، مقادیر ثابت و رشته ها است. بخش های .ARM.extab و .ARM حاوی اطلاعات رسیدگی به استثنا هستند. بخش‌های .ctors، .dtors، .preinit_array، .init_array و fini_array حاوی کدهایی برای سازنده‌ها و مقداردهی اولیه و تعریف کردن اشیاء گلوبال و استاتیک هست. در نهایت، بخش .mtb حافظه را برای بافر micro trace ذخیره می کند که توسط برنامه استفاده نمی شود.



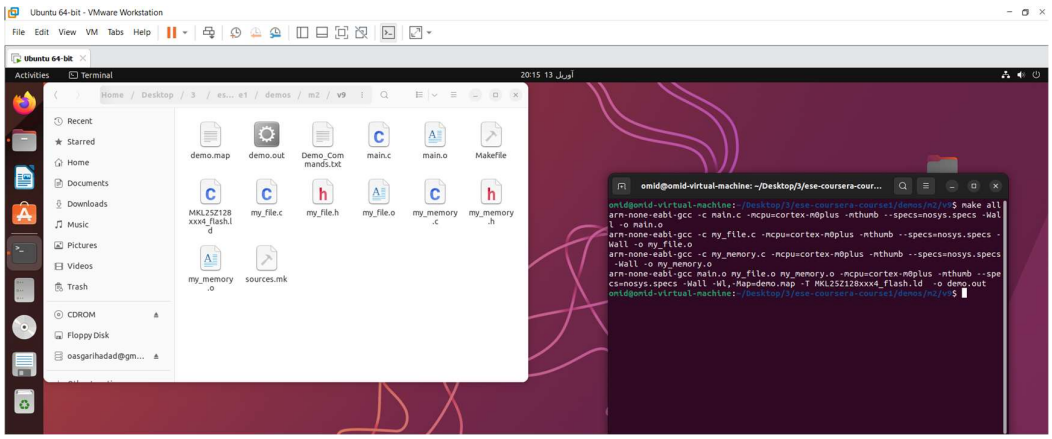
M2 v8

این کد دقیقا کد V6-V7 است فقط یک سری باگ ها را حل کرده است. این کد برای رفع وابستگی printf در platform.h میباشد.



M2 V9

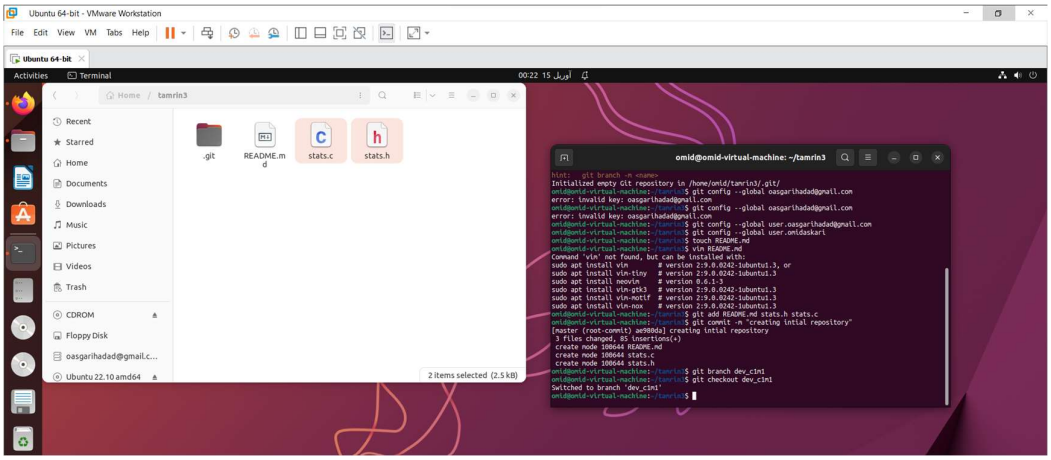
این کد هم دقیقا کد V6-V7 و V8 است فقط یک سری باگ ها را حل کرده است.



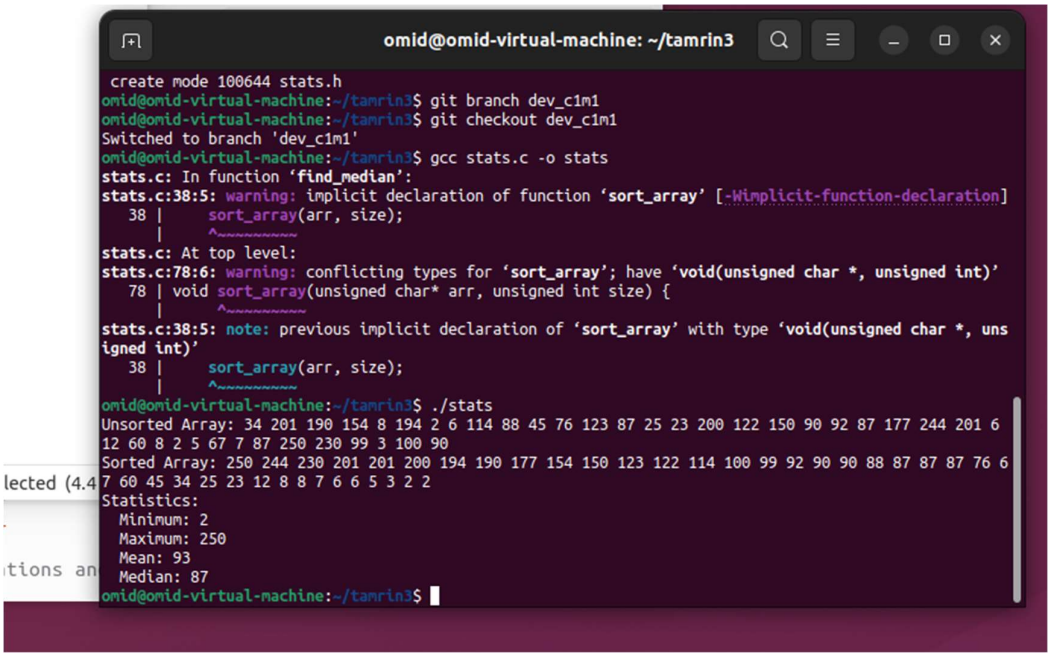
قسمت ب) assessments

1) week-1-application-assignmen

برای حل این تکلیف ابتدا روی ماشین مجازی یک ریپازیتوری محلی میسازیم :

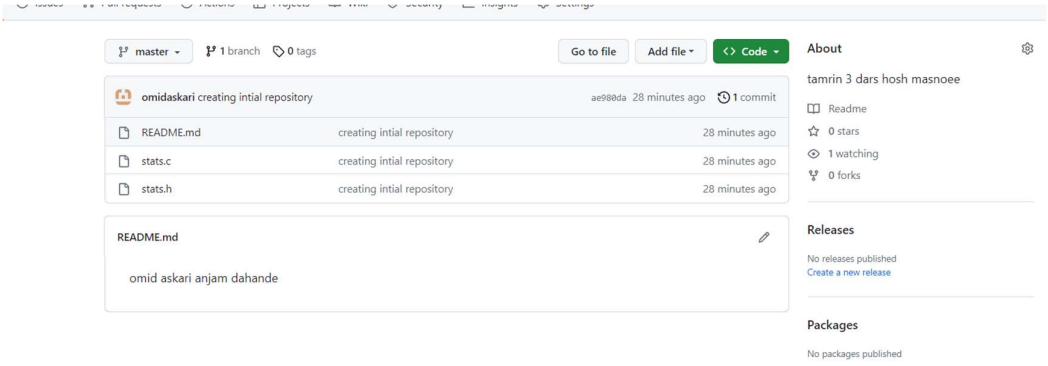


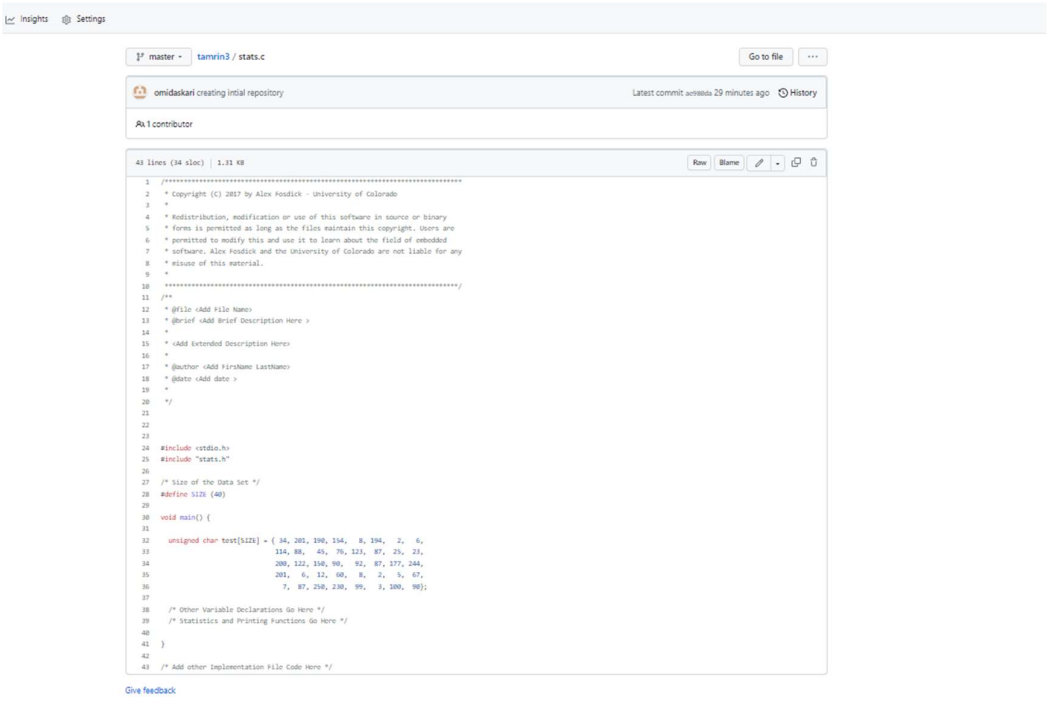
حال کد مورد نیاز برای stats.h , stats.c را مینویسیم. من این کد ها را ضمیمه میکنم.



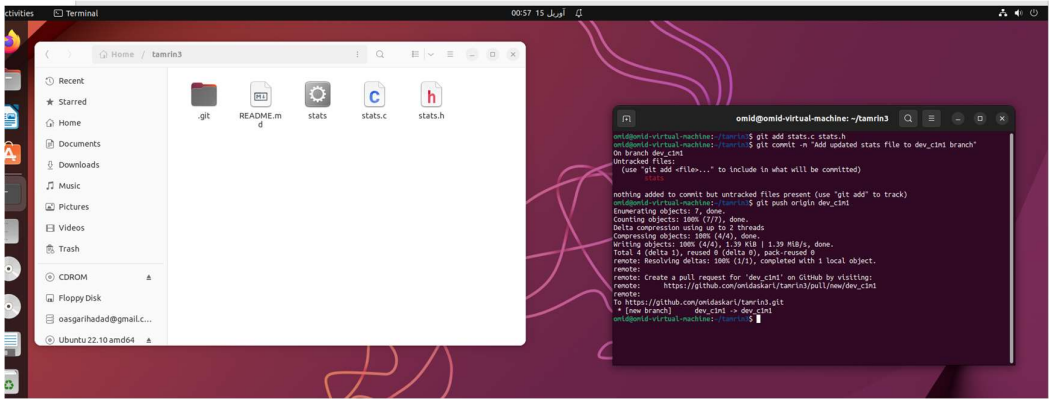
حالا در گیت هاب ریپازیتوری تمرین را میسازم: در کامیت اول همان فایل های خام را داریم

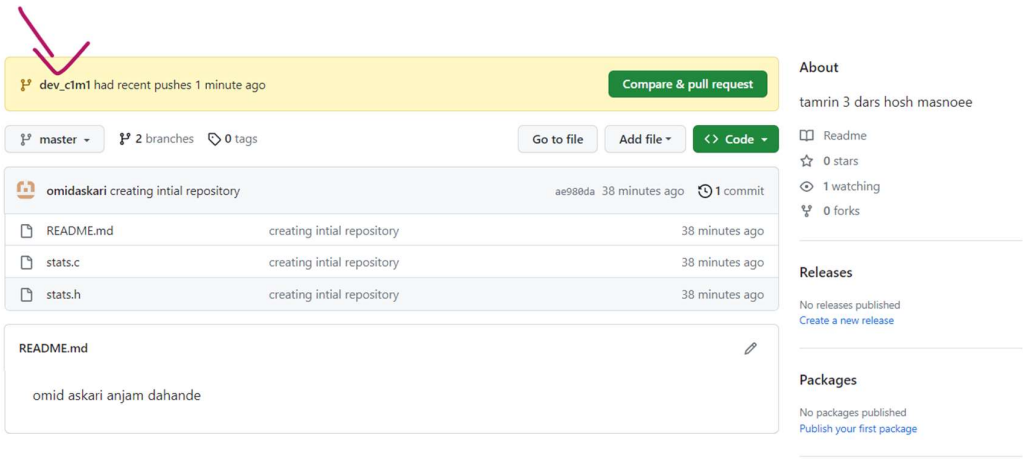
```
omid@omid-virtual-machine:~/tamrin3$ git remote set-url origin https://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@github.com/omidaskari/tamrin3.git
omid@omid-virtual-machine:~/tamrin3$ git push --set-upstream origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 1.11 KiB | 1.11 MiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/omidaskari/tamrin3.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
omid@omid-virtual-machine:~/tamrin3$
```



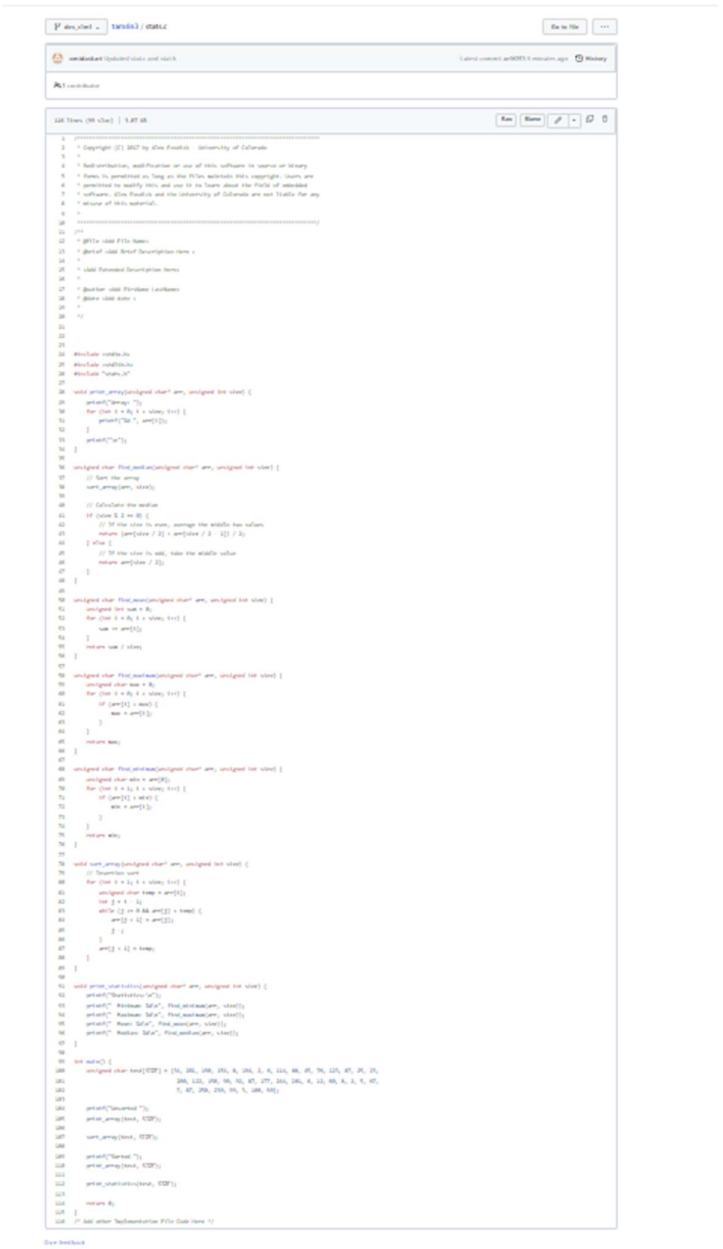


در کامیت بعدی فایل هایی که نوشتم را پوش میکنم:
https://github.com/omidaskari/tamrin3
https://github.com/omidaskari/tamrin3/tree/dev_c1m1
(1)https://github.com/omidaskari/tamrin3/tree/master/tamrin3-dev_c1m1

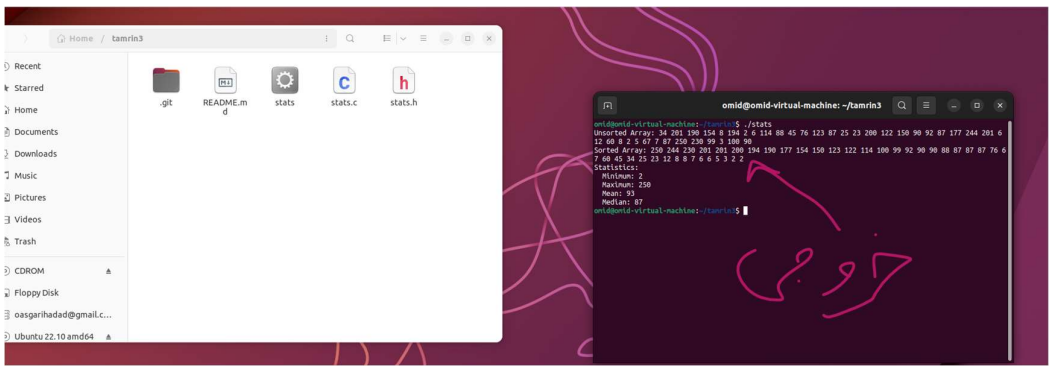




مشاهده میشود که ریپازیتوری دو برنچ دارد که در برنچ دوم تغیرات ایجاد شده را میتوان مشاهده کرد عکس فایل جدید stats.c را در زیر مشاده میکنید.

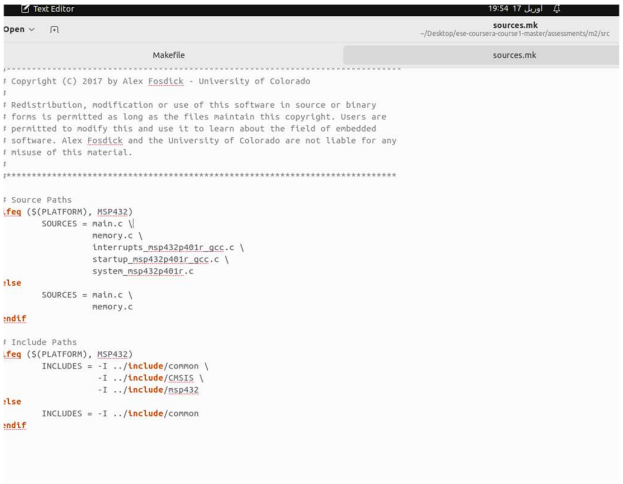


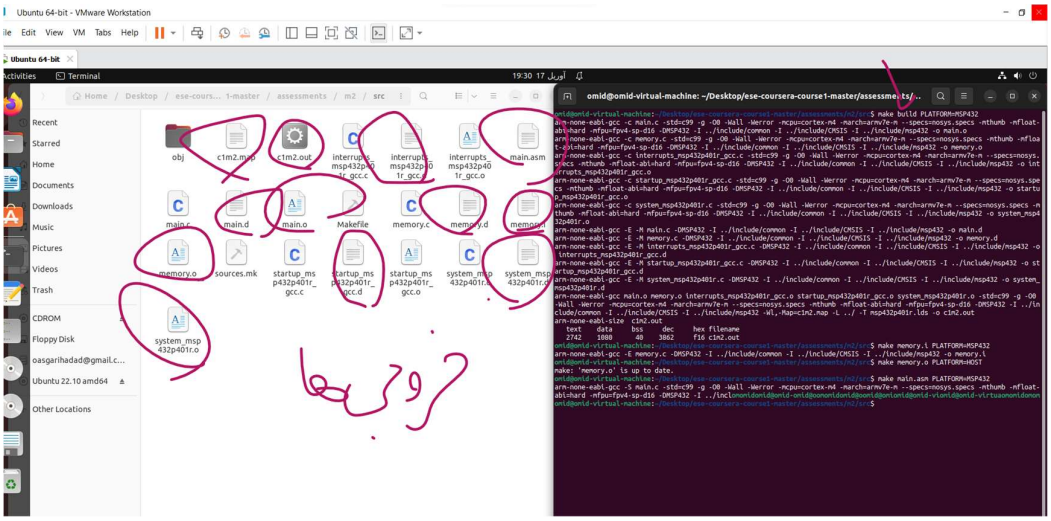
ضمنا خروجی کد تولیدی من در عکس زیر آورده شده و مشخص میشود که کد صحت دارد:



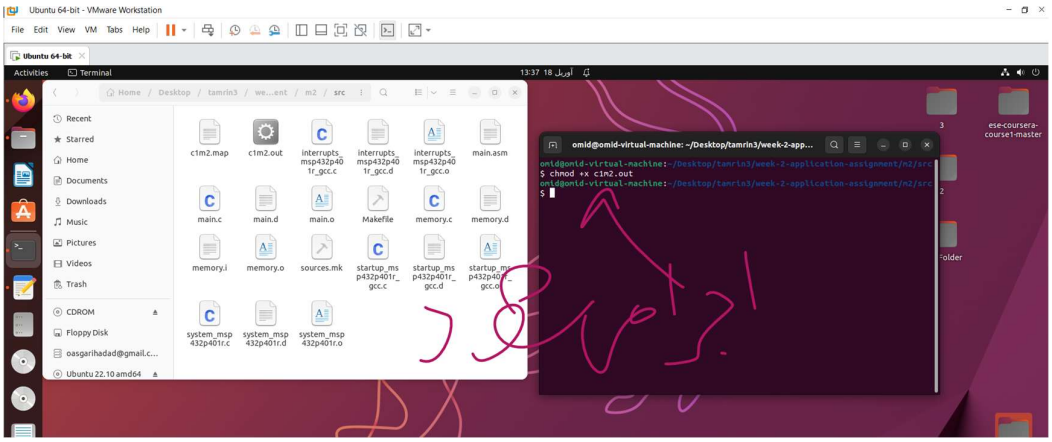
اول باید makefile وsources.mk را تغییردهیم:

```
include sources.mk


# Platform Overrides.
PLATFORM ?= HOST

# General flags for both platforms.
G_FLAGS = -std=c99 \
          -g \
          -O0 \
          -Wall \
          -Werror

# Architectures Specific Flags for MSP432.
LINKER_FILE = -T msp432p401r.lds
CPU = cortex-m4
ARCH = armv7e-m
SPECS = nosys.specs
EXTRA = -mthumb \
        -mfloat-abi=hard \
        -mfpu=fpv4-sp-d16


TARGET = c1m2


# Conditional If for select between Platforms (HOST & MSP432).
# When Platform selected, Specific Flags will be added.
ifeq ($(PLATFORM), MSP432)
        # Compiler Flags & Defines
        CC = arm-none-eabi-gcc
        LD = arm-none-eabi-ld
        CFLAGS = $(G_FLAGS) \
                 -mcpu=$(CPU) \
                 -march=$(ARCH) \
                 --specs=$(SPECS) \
                 $(EXTRA)
        CPPFLAGS = -DMSP432 $(INCLUDES)
        LDFLAGS = -Wl,-Map=$(TARGET).map \
                  -L ../ $(LINKER_FILE)
        OBJDUMP = arm-none-eabi-objdump
        SIZE = arm-none-eabi-size

else ifeq ($(PLATFORM), HOST)
        # Compiler Flags & Defines
        CC = gcc
        LD = ld
        CFLAGS = $(G_FLAGS)
```



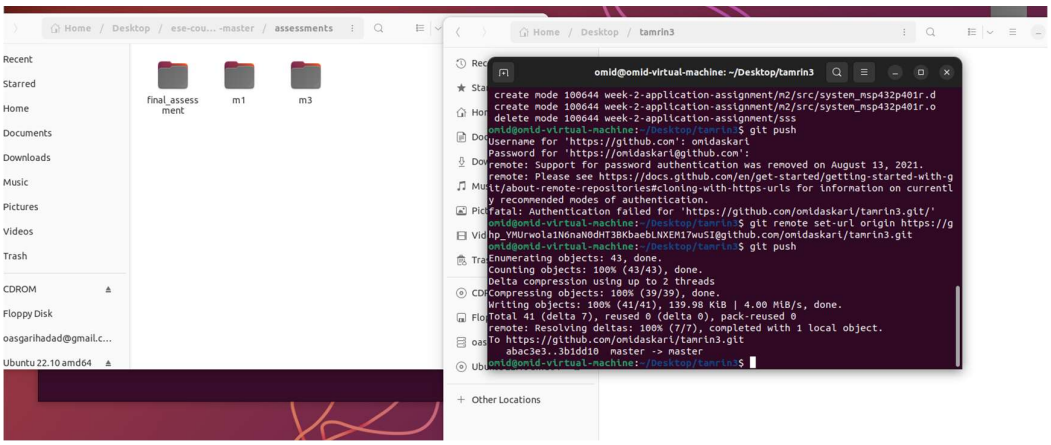


در زیر عکس اجرای خروجی را گذاشتم: چون خروجی برای آرم 32 بیتی طراحی شده است و سیستم من64 بیتی است باید از دستور زیر برای اجرا با رفع این محدودیت استفاده کنم.



همه فایل های پروژه را در گیت هابم آپلود کردم که چک کنید:

https://github.com/omidaskari/tamrin3/tree/master/week-2-application-assignment-assesment%202/m2



3- تمرین 3 از assessment ها

در این تمرین یک کد ازیک اسکریپت لینکر داریم و کد هایی برای اجرای برنامه هایی روی این ماژول داریم که در این برنامه ها یک سری نماد ها تعریف میشوند و هدف از این تمرین به دست آوردن برخی از اطلاعات در مورد این نماد ها که در زیر نام میبرم هستش:

1- نوع متغیر و مکان در تاپ سگمنت

2-مکان در ساب سگمنت (مانند استک و هیپ یا دیتا)

3-مجوزهای دسترسی (مانند خواندن (R)، نوشتن (W) خواندن-نوشتن (RW)یا هیچ کدام)

4-طول عمر(مانند عملکرد/بلاک، برنامه، نامحدود، هیچکدام)

برای تعیین مکان حافظه، بخش ساب سگمنت، مجوزهای دسترسی و طول عمر نمادهای موجود در فایل برنامه ، می توان از ترکیب ابزارها و روش های زیر استفاده کنید:

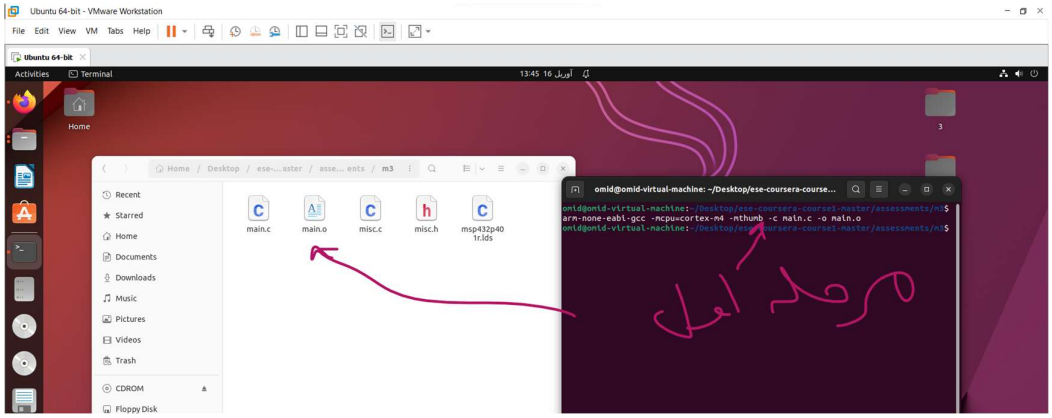
1- با استفاده از پرچم پیوند دهنده -Wl,MAP = main.map یک فایل .map ایجاد میکنیم و اطلاعات نماد ها را بررسی میکنیم تا فضای حافظه مورد استفاده هر نماد تعیین شود.

2- از ابزار باینری GCC nm برای بررسی اطلاعات کلی نمادها و ویژگی های حافظه آنها استفاده میکنیم. این کار را می توان با اجرای دستور زیر در ترمینال انجام داد:

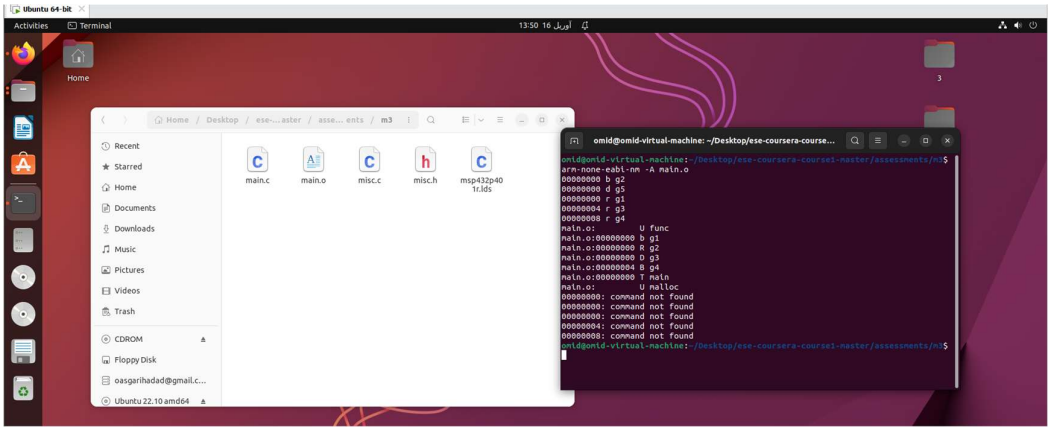
nm main.o

در زیر اسکرین شات های مراحل به دست اوردن اطلاعات مربوط به نماد ها را قرار میدهم و سپس اطلاعات خواسته شده را مینویسم.

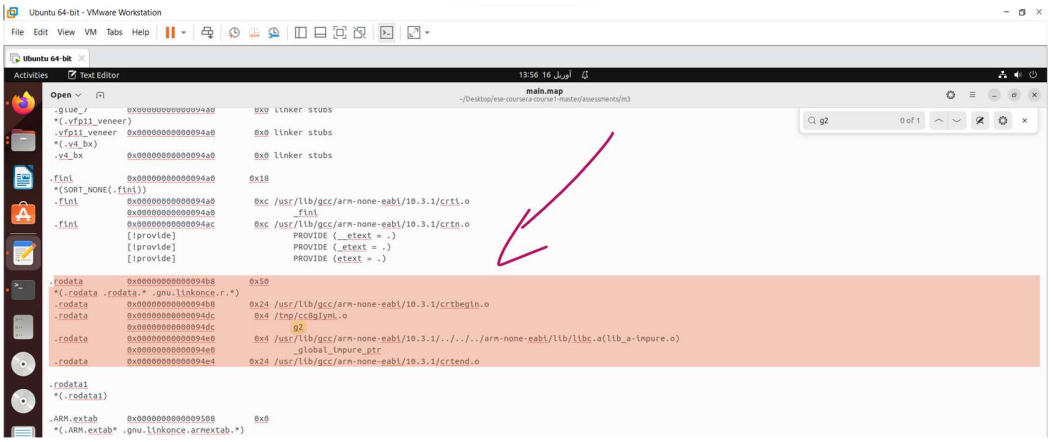
تصویر زیر برای کامپایل کردن فایل main است



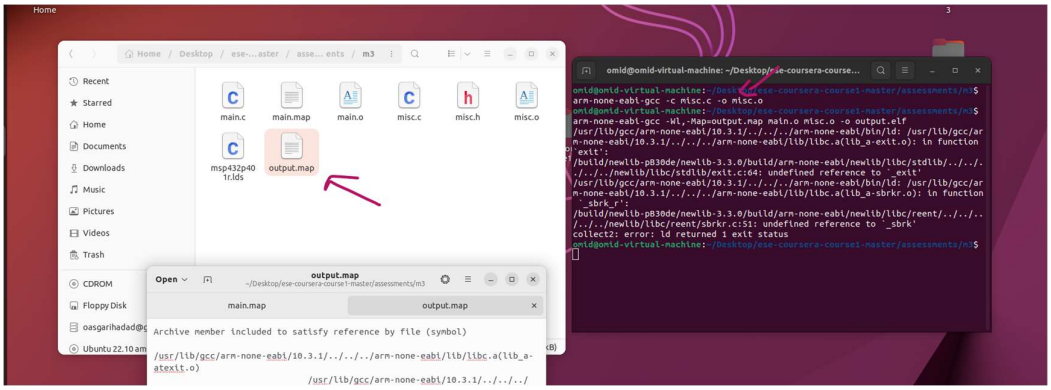
تصویر زیر استفاده از دستور nm برای به دست اوردن اطلاعات مربوط به نماد های g است.



در فایل map. ایجاد شده میتوان اطلاعتی که در مورد g2 داریم را مشاهده کنیم ، مثلا این نماد در قسمت data از حافظه جا میگیرد.



در اینجا هم ابتدا فایل misc را کامپایل میکنیم و سپس فایل map. را میسازیم تا اطلاعات مربوط به نماد های خواسته شده در این برنامه را به دست آوریم.



اطلاعات مربوط به نماد ها:

G1:

- Location Top segment or Type: .rodata (read-only data segment)
- Location Sub-segment: bss
- Access Permissions: RW
- Lifetime: program

g2:

- Location Top segment or Type: .rodata (read-only data segment)
- Location Sub-segment: const
- Access Permissions: Read
- Lifetime: program

G3:

- Location Top segment or Type: .data (initialized data segment)
- Location Sub-segment: data
- Access Permissions: Read/Write (RW)
- Lifetime: program

G4:

- Location: Top segment
- Location Sub-segment: bss
- Access Permissions: Read-Write (RW)
- Lifetime: program

G5:

- Location: Top segment
- Location Sub-segment: data
- Access Permissions: Read-Write (RW)
- Lifetime: program

F1,F2,F3:

name	Segment	Sub-segment	Permissions	Lifetime
F1	data	stack	RW	function
F2	data	bss	RW	program
F3	data	stack	RW	function

I1,I2,I3:

- Location Top segment or Type: Register (I1), Data (I2), Data (I3)
- Location Sub-segment: None (I1), stack (I2)), stack (I3)
- Access Permissions: Read-write (I1,I2,I3)
- Lifetime: Function (I1, I2,I3)

Hello World!(string):

- Location Top segment: code
- Sub-segment: rodata segment.
- Access Permissions: Read
- Lifetime: Program

این تمرین را نیز در گیت هاب آپلود میکنم

https://github.com/omidaskari/tamrin3/tree/master/programming-assignment-instructions(3)

فایل های map. حاوی اطلاعات مربوطه هستند.

4) تمرین نهایی assessment ها

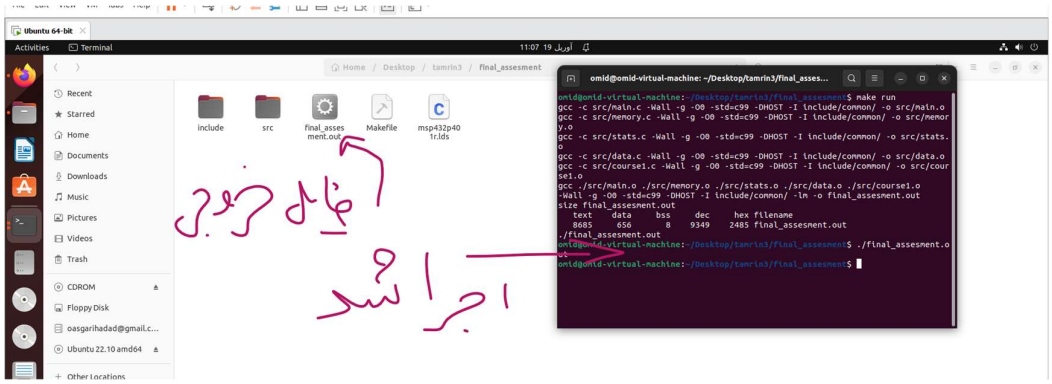
در این تکلیف برنامه نویسی، تجربه بیشتری از کار با Git پیدا خواهیم کرد، کدهای برنامه نویسی C راجب با حافظه و دیتا مینویسیم و کد خود را با کد های قبلی که نوشتیم ادغام می کنیم. ما از ریپازیتوری که تا به حال با آن کار کرده ایم مجددا استفاده خواهیم کرد و برخی از توابع جدید را اضافه خواهیم کرد که حافظه را دستکاری می کنند. برای تمام کد ها کامنت گذاری شده است.کد را روی دستگاه میزبان خود تست خواهم کرد، اما کد باید برای پلتفرم هدف هم کامپایل شود.

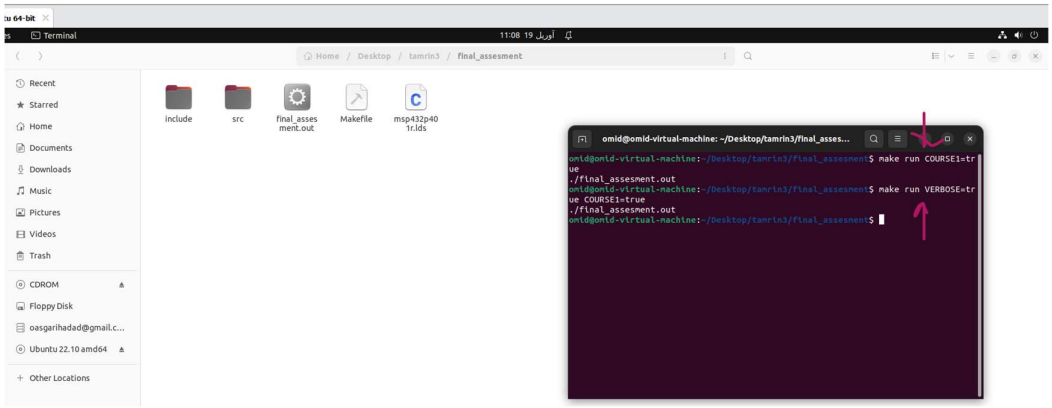
تمامی خواسته هایی که در تمرین داده شد از جمله اضافه کردن uint8_t * my_memmove و غیره انجام شد.

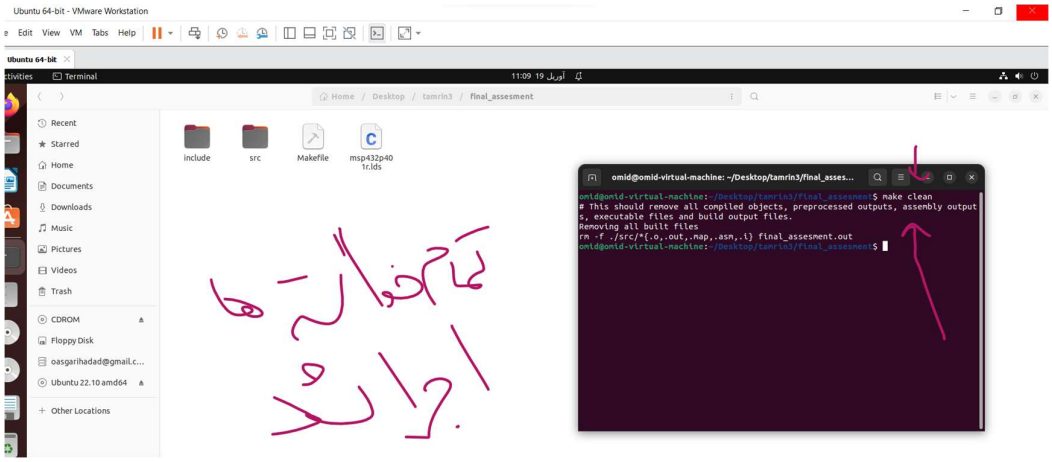
تمام فایل ها در ریپازیتوری من در گیت هاب برای چک کردن موجود است.

هم چنین کل پوشه گیت هاب را ضمیمه میکنم.

https://github.com/omidaskari/tamrin3/tree/master/final_assesment






Ubuntu 64-bit - VMware Workstation
Edit View VM Tabs Help
Ubuntu 64-bit
Terminal
Home / Desktop / tamrin3 / final_assesment
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
CDROM
Floppy Disk
Ubuntu 22.10 amd64
Other Locations
include
src
Makefile
msp432p401r.lds
omid@omid-virtual-machine: ~/Desktop/tamrin3/final_asses...
omid@omid-virtual-machine:~/Desktop/tamrin3/final_assesment$ make clean
This should remove all compiled objects, preprocessed outputs, assembly outputs, executable files and build output files.
Removing all built files
rm -f ./src/*{.o,.out,.map,.asm,.i} final_assesment.out
omid@omid-virtual-machine:~/Desktop/tamrin3/final_assesment$