期末PROJECT

評分標準

- (Q1) GEM5 + NVMAIN BUILD-UP (40%)
- (Q2) Enable L3 last level cache in GEM5 + NVMAIN (15%) (看到 log 裡面有 L3 cache 的資訊)
- (Q3) Config last level cache to 2-way and full-way associative cache and test performance (15%)
  - 必須跑benchmark quicksort在 2-way跟 full way (直接在 L3 cache implement，可以用 miss rate 判斷是否成功)
- (Q4) Modify last level cache policy based on frequency based replacement policy (15%)
- (Q5) Test the performance of write back and write through policy based on 4-way associative cache with isscc_pcm(15%)
  - 必須跑 benchmark multiply 在 write through 跟 write back (gem5 default 使用 write back，可以用 write request 的數量判斷 write through 是否成功)
- Bonus (10%)
  - Design last level cache policy to reduce the energy consumption of pcm_based main memory
  - Baseline:LRU

繳交方式

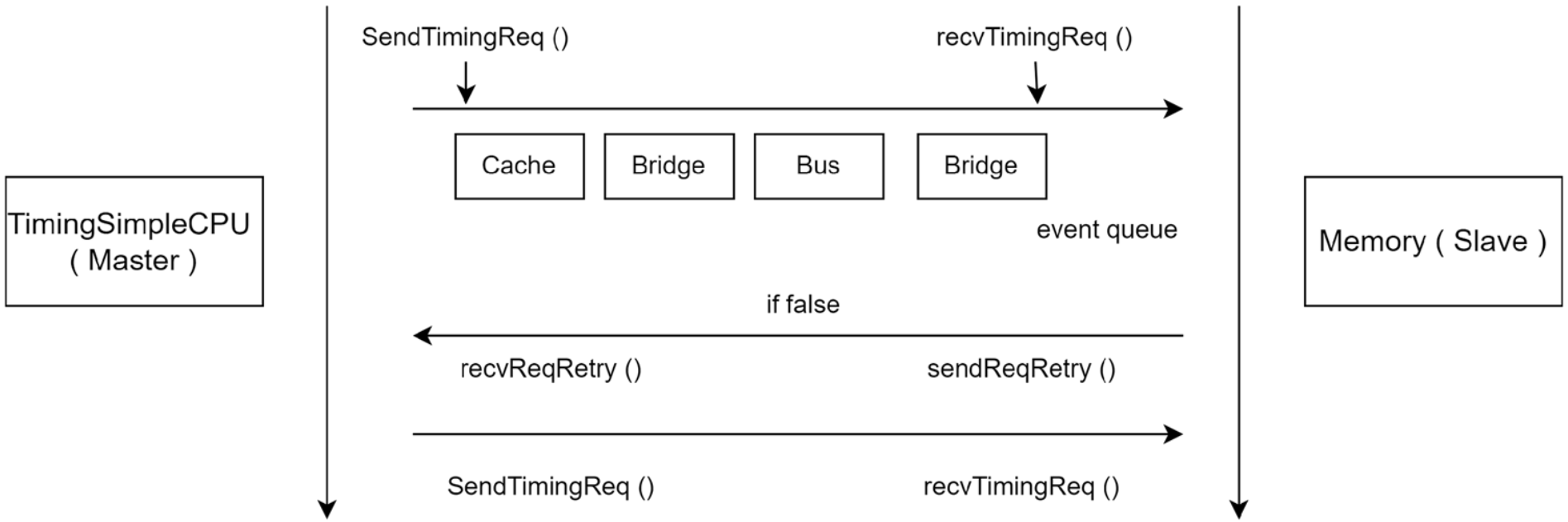
- (Q3) 直接在 l3 cache implement，並繳交兩份 log 檔案，一份是 2-way， 一份是 full-way，需在 log 檔裡看到 l3 cache 的資訊
- (Q4) 繳交兩份 log 檔案，一份是原始 policy，一份是 frequency based policy
- (Q5) 繳交兩份 log 檔案，一份 write through，一份是 write back
- (bonus) 繳交兩份 log 檔案，一份是 baseline (hello)，一份是修改過後的方法
- log 檔案包含 m5out 中的 stat.txt 檔案，以及 nvmain 輸出的 log
- “修改過後” 的 code 需要繳交，可以上傳 github，或是一起放在壓縮檔繳交，不需要額外的書面報告
- demo 時會額外問與 gem5，nvmain 的相關問題，且為了測試 code 是否能正常執行，會要求 demo 時編譯一次

常見問題

- 記憶體空間不足
- Ubuntu 版本不對，有問題的話建議用 18.04
- 執行 benchmark 的時候，因為 gem5 本身速度緩慢，會誤以為程式卡住，但其實他仍然有在執行
- 跑完程式後，可以從 m5out 的資料夾去查看 stat.txt
- 更改完 gem5 的檔案記得要編譯再重新執行
- gem5 有 debug flag 的功能

```
build/X86/gem5.opt --debug-flags=Exec configs/learning_gem5/part1/simple.py | head -n 50
```

Gem5 Memory system



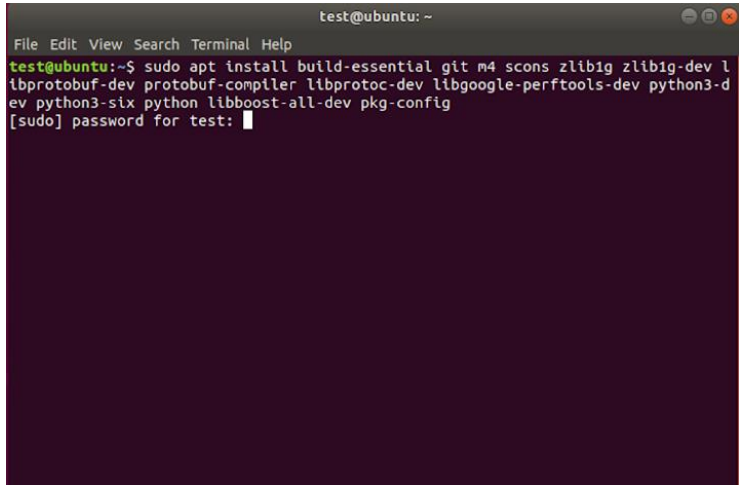
NVMAIN+GEM5教學

VIRTUAL MACHINE

- 推薦使用Ubuntu 18.04
- Ubuntu 20.04 g++太新 nvmain編譯會出問題
- Ubuntu 16.04 gem5不支援
- 其他硬體規格
  - 記憶體:4GB
  - 硬碟:20GB

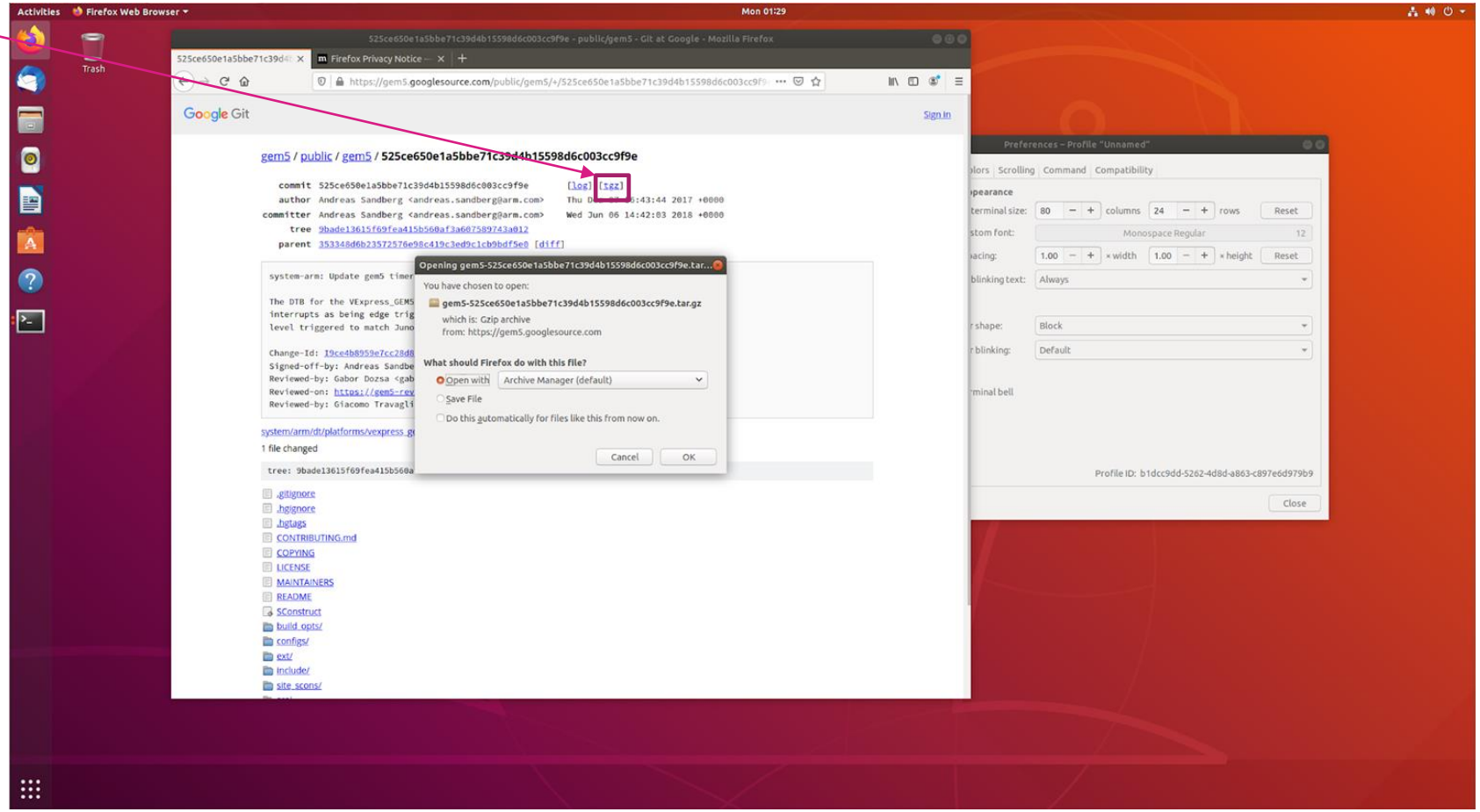
安裝編譯工具

- sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python3-dev python3-six python libboost-all-dev pkg-config

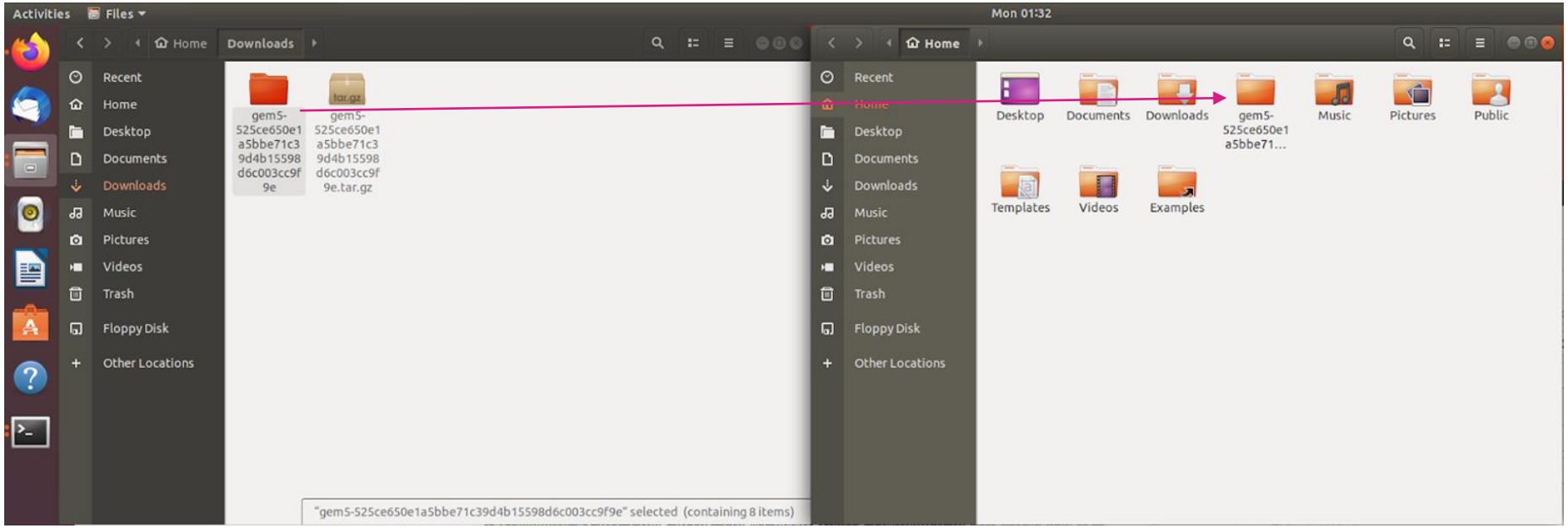


安裝GEM5

- https://gem5.googlesource.com/public/gem5/+/525ce650e1a5bbe71c39d4b15598d6c003cc9f9e
- 點tgz下載(不推薦下載最新版的GEM5，後面編譯會出問題)

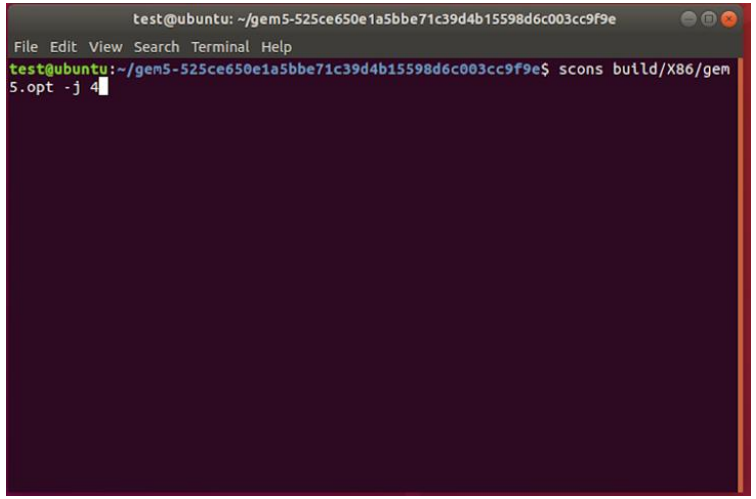


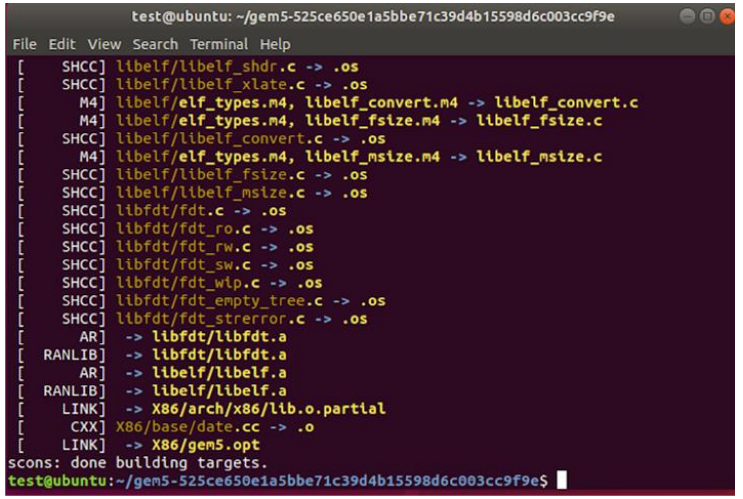
把解壓後的GEM5資料夾丟進HOME



編譯 GEM5

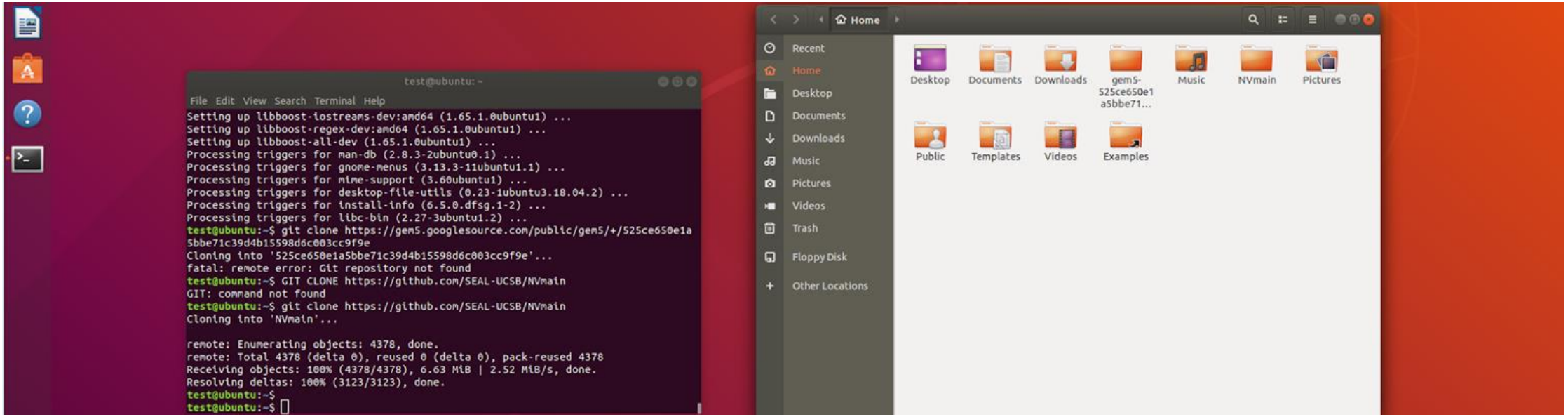
- 在GEM5目錄下scons build/X86/gem5.opt (後面可以加 –j num　num為multithread數量來加速)
- 這安裝要大概20分鐘





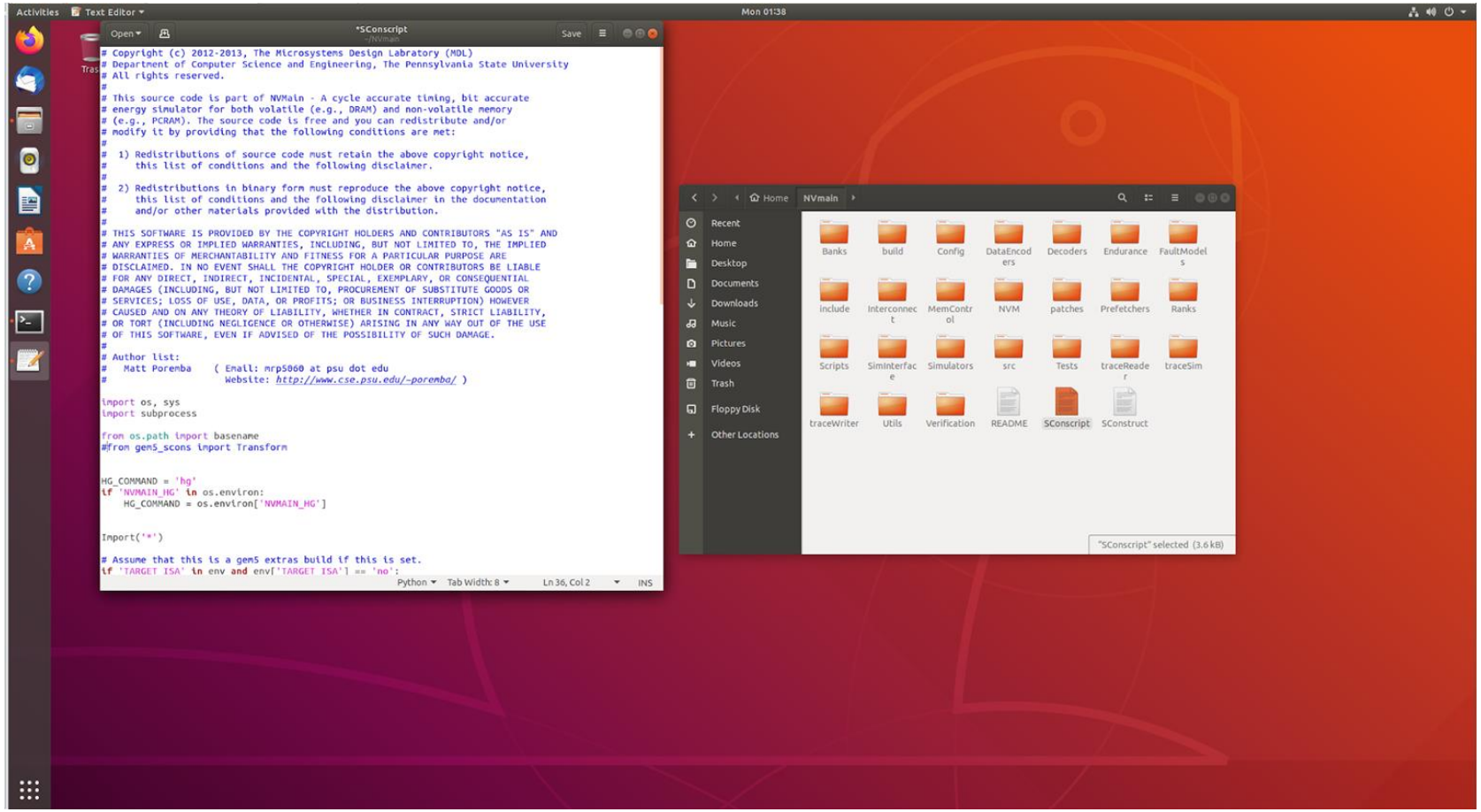
用GIT 安裝 NVMAIN

- git clone https://github.com/SEAL-UCSB/NVmain
- 把nvmain跟gem5放到同一個目錄下



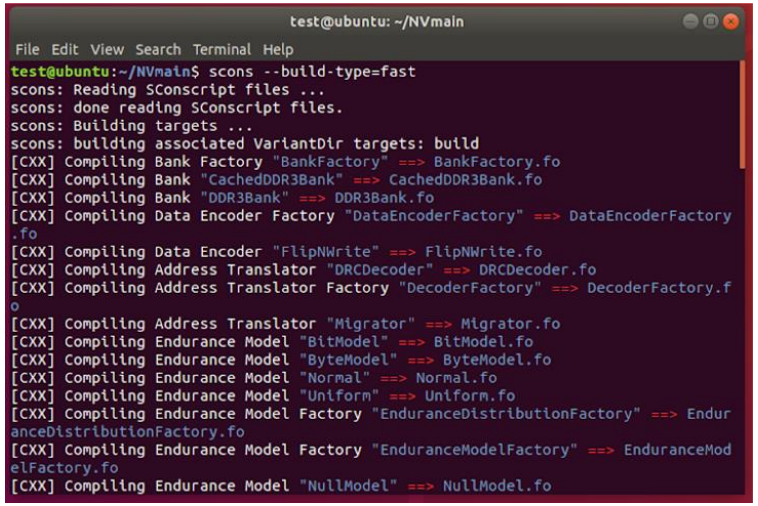
修改 SCONSCRIPT

- 進入nvmain資料夾 點開 SConscript 把36行的from gem5_scons import Transform註解掉(記得要存檔)



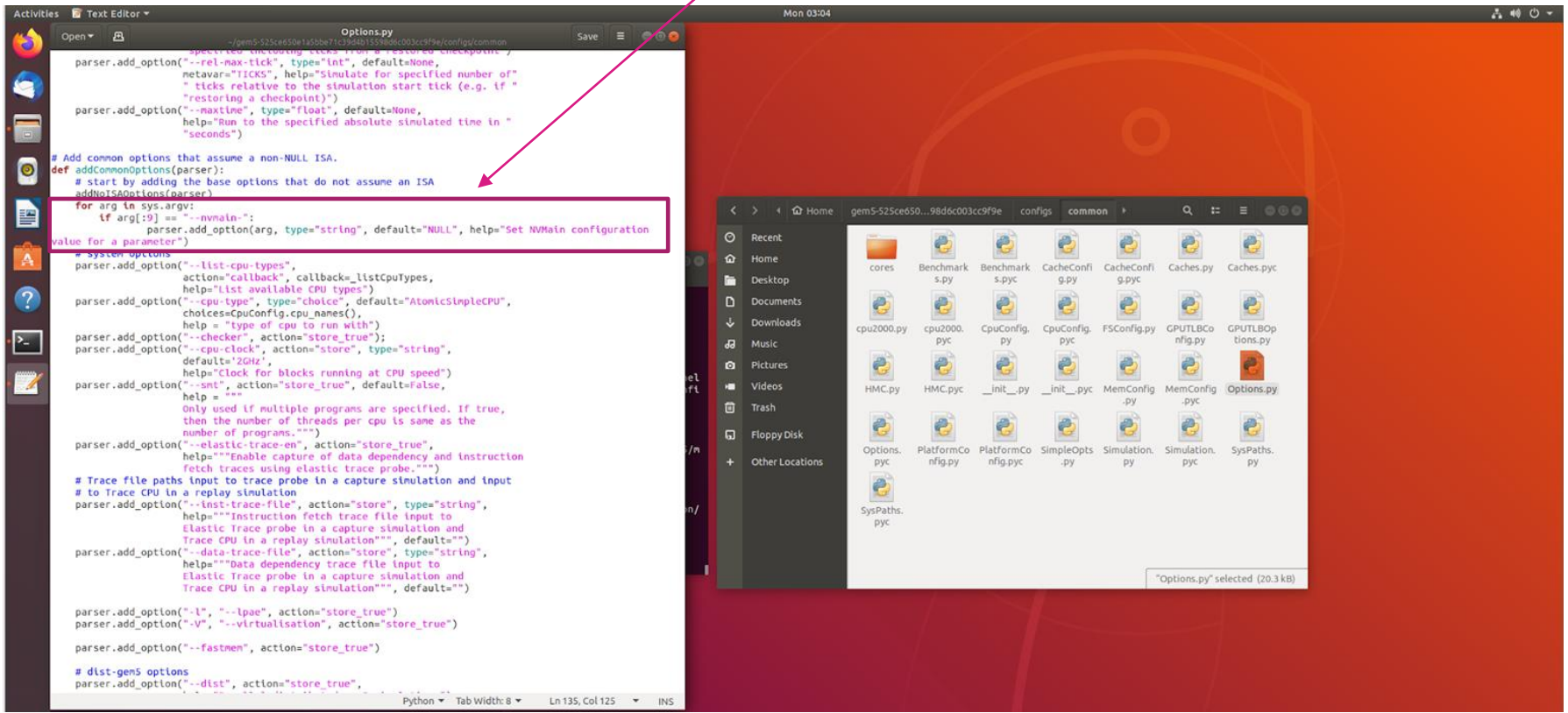
編譯NVMAIN

- 在NVmain目錄下 scons --build-type=fast



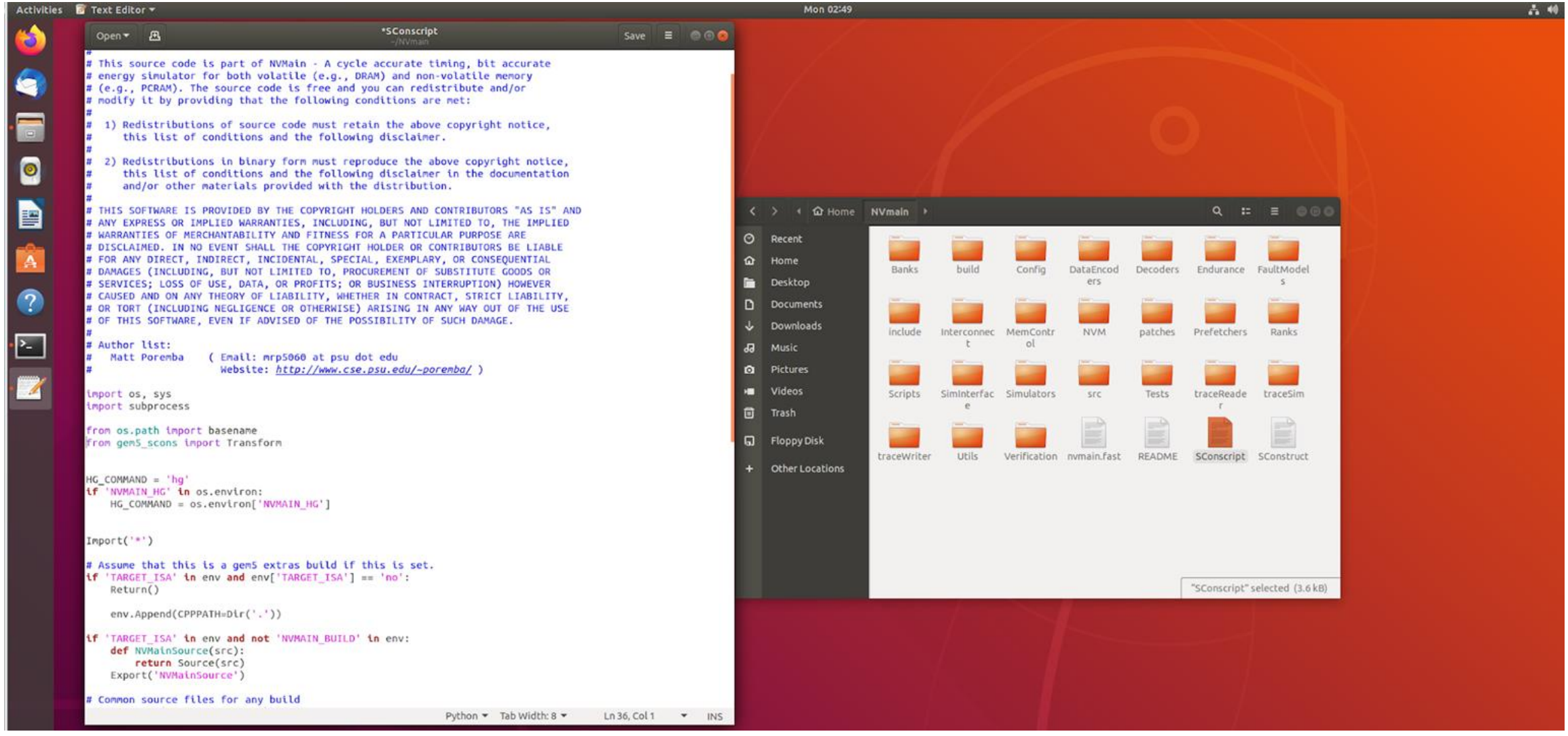
修改GEM5 OPTIONS

- 在gem5/configs/common/Options.py中第133行加入下段程式(記得要存檔)
- for arg in sys.argv:
- 	if arg[:9] == "--nvmain-":
- 		parser.add_option(arg, type="string", default="NULL", help="Set NVMain configuration value for a parameter")



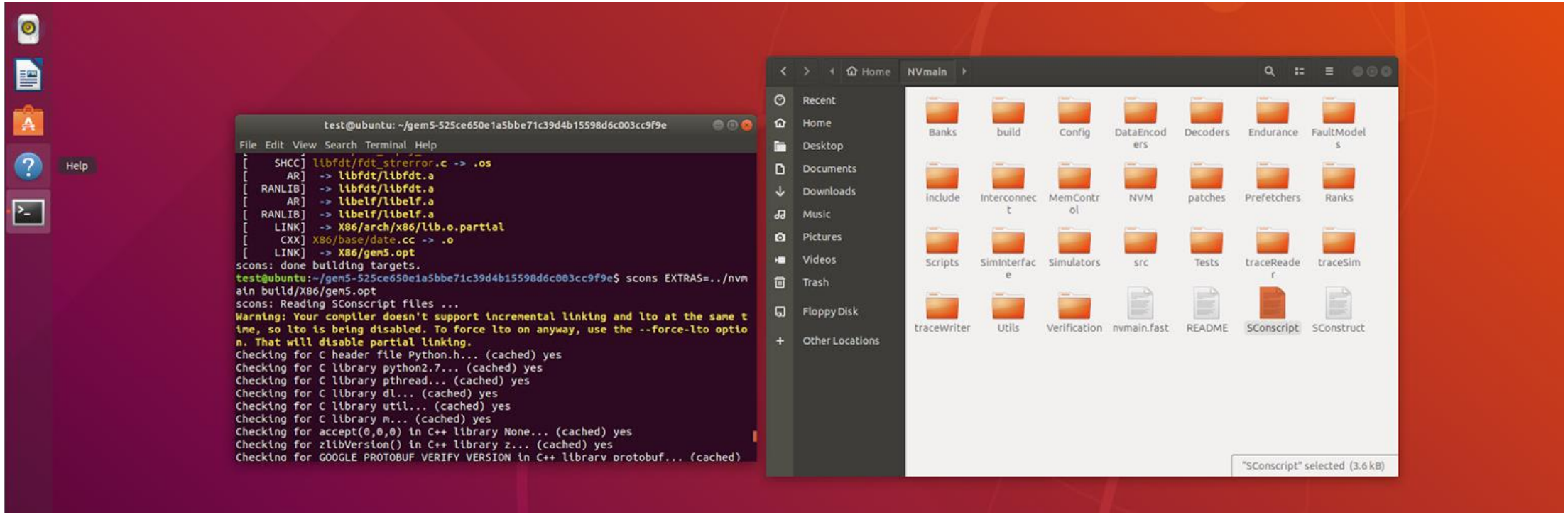
還原 SCONSCRIPT

- 還原前面nvmain sconscript註解掉的指令



混和編譯 GEM5

- 在GEM5目錄下 混合NVMAIN編譯GEM5
- scons EXTRAS=../NVmain build/X86/gem5.opt



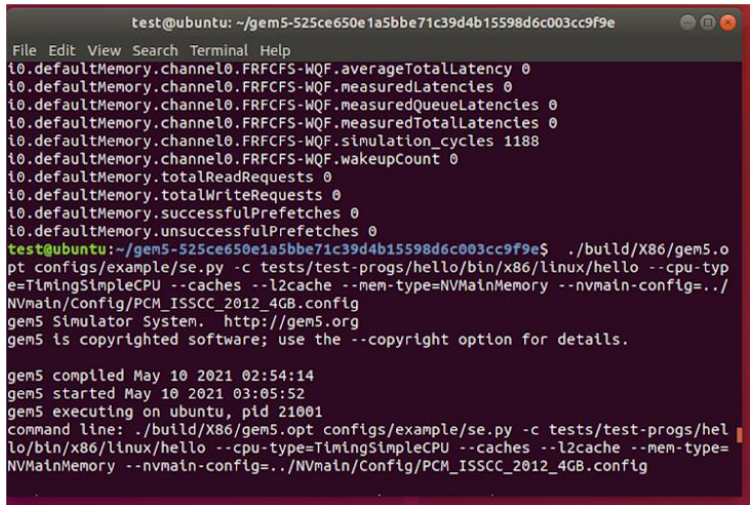
測試HELLO WORLD

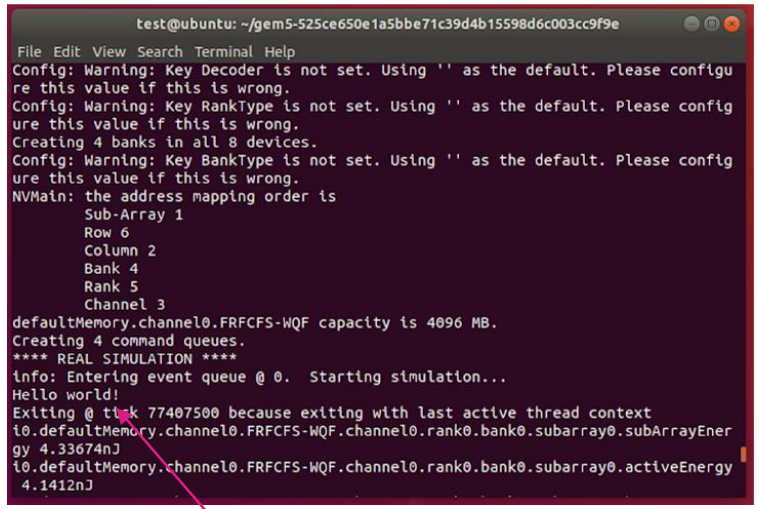
Gem5 執行檔

程式執行檔

- ./build/X86/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/x86/linux/hello --cpu-type=TimingSimpleCPU --caches --l2cache --mem-type=NVMainMemory --nvmain-config=../NVmain/Config/PCM_ISSCC_2012_4GB.config

nvm

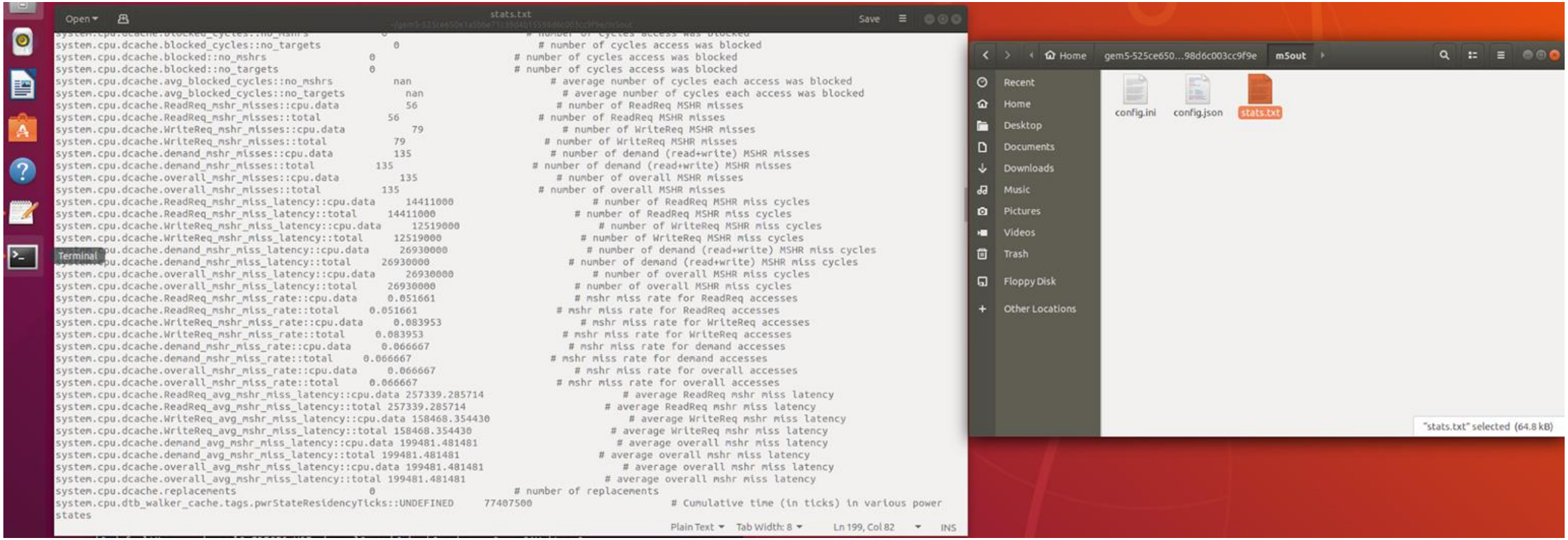




Hello world!

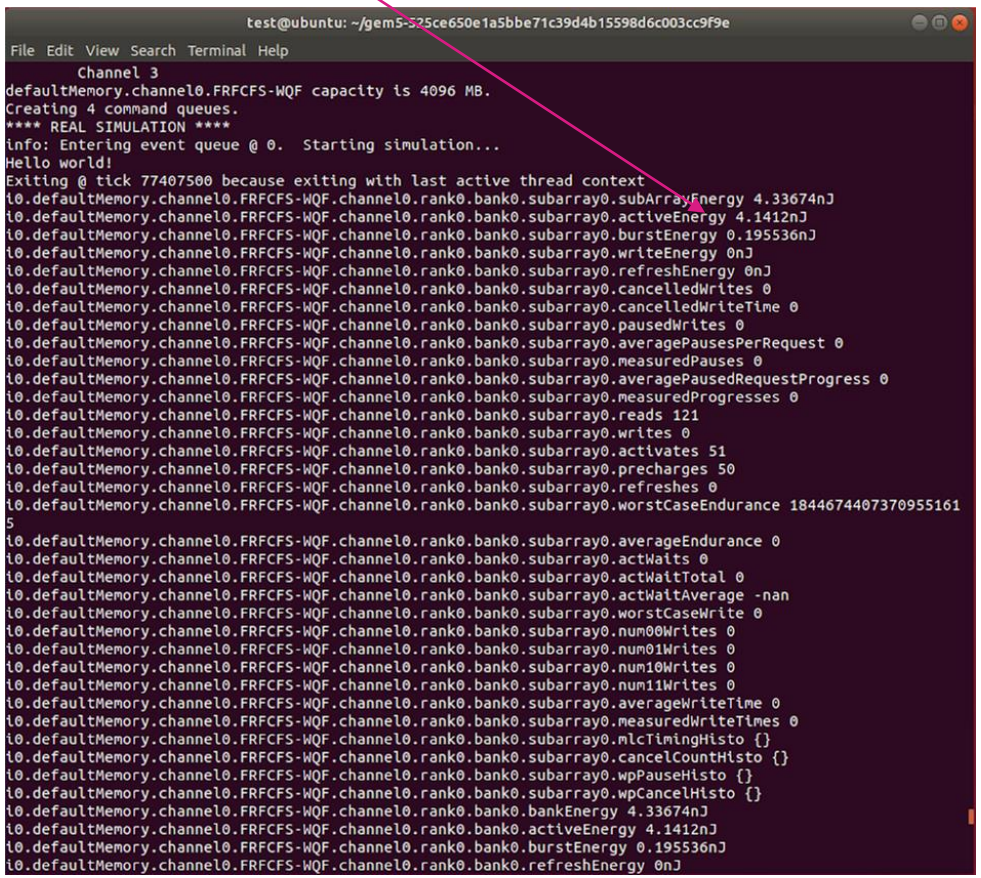
OUTPUT

- 可以由gem5/m5out中的stat.txt查看cache hit 的次數



ACTIVE ENERGY

- 在執行完程式後就可以看到



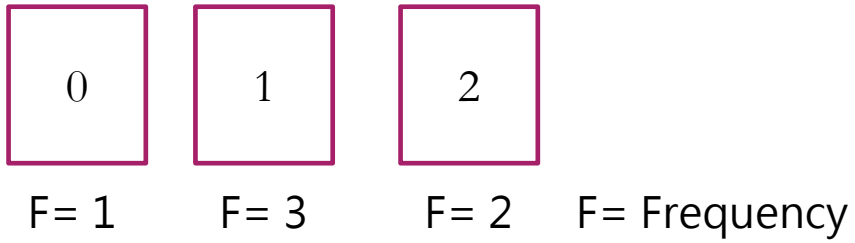
ENABLE L3 CACHE簡易提示

- 需要修改的檔案 在gem5資料夾中的
  - Options.py
  - Caches.py
  - Xbar.py
  - BaseCPU.py
  - CacheConfig.py
- 前面四個檔案只是增加L3 cache的parameter ，照著L2 cache的設定去做模仿就可以。
- CacheConfig.py需要讓L3 cache連接整個Gem5系統，這邊要注意 L2跟L3這兩個cache的關係，要讓系統在已使用L2 cache的情況下才能使用L3 cache，所以修要注意修改的時候有沒有滿足這個條件。

- 細節code 的部分，同學可以上網找資源關鍵字 Gem5 L3 cache之類的。

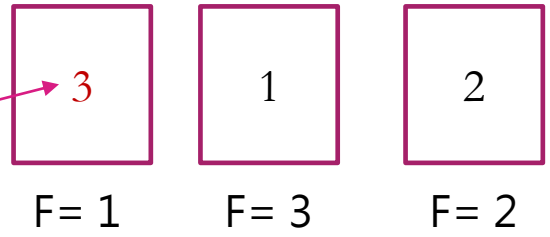
FREQUENCY BASED REPLACEMENT POLICY

- Cache size = 3 full associative blocks
- block request 0 , 1 , 2 , 1, 2, 1

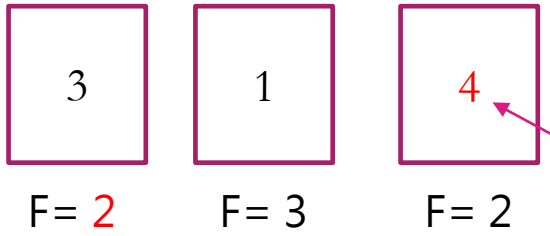


- block request 3

Select victim as min F



- block request 3, 4



Select victim as min F and the Oldest data.

WRITE BACK AND WRITE THROUGH 提示

- 先了解什麼是write back 和 write through (應該上課會教)
- 可以去mem/cache的資料夾下找到兩個檔案一個叫做cache.cc另一個叫做base.cc，可以研究一下gem5底層的code是如何把資料寫入Memory。