General info:

* We need to write functions for read, write, open, and release.
  + These need file pointers, pointer to a buffer, buffer size and an offset
* Using file operations to store the functions
* Register devices we want to create using register\_chrdev
  + Takes in arguments major, device name and the file operation
  + major numbers tell you which driver handles which device file
  + A device file is an interface to a device driver
  + Device files 🡨 major numbers🡪 drivers
* Adding a driver to your system means registering it with the kernel. This is synonymous with assigning it a major number during the module's initialization. You do this by using the register\_chrdev function, defined by linux/fs.h.

Installing Linux headers:

* These give an interface for the kernel functions
* Uname -r
  + Shows kernel version installed
* Apt search linux-headers-$(uname-r)
  + Shows kernel headers and versions
* ls -l /usr/src/linux-headers-$(uname -r)
  + all kernel header files can be found under the /usr/src directory.
* sudo apt install linux-headers-$(uname -r)
  + installs the Linux kernel headers package for your kernel version
* apt update -y && apt upgrade -y && apt dist-upgrade
  + If it breaks use this to update kernel

Making the program:

* Any text editor
* Nano/ vim/ Pico/ emacs hello.c
* Include code
* We are using the “printk” function because this allows us to take a string and print it to the kernel log. If we were to use “printf” or another way of outputting then it would not recognise the command and not print to the kernel log.

When using “open()” in C:

* Need to define an variable
* Set variable to open the document = int x;
* X = open(const char \*pathname, int flags, mode\_t mode);
* Const char is the name of the file you want to open and then you can provide the file path for it.
* Flags are used based on the permissions of the program (read only, write only etc)
* If the file doesn’t exist you can use the O\_CREAT flag to create it and pipe (|) this to the permissions you want it set as e.g. 0\_WRONLY.
* The mode is the permissions for the file for user groups and world e.g. 777



Making a make file:

* Use any texteditor you like
* I use nano
* Nano Makefile
* Inside your directory you will write the C program
* Inside the Makefile you write a shell script to run the C code
* Makefiles care about the use of tabs (do not use spaces) like how Python cares about tabs
* Include code
  + Geneal layout for a makefile would be something like:

Target: dependencies:

gcc dependencies -o nameOfFile

program: program.o mylib.o

gcc -o program program.o mylib.o



* What that code will do is compile the “all” and “clean” functions from our hello.o program.
* The reason it is a .o extension is because it is an object file. Object files contain C and C++ code to be compiled.

Insert modules into the kernel:

