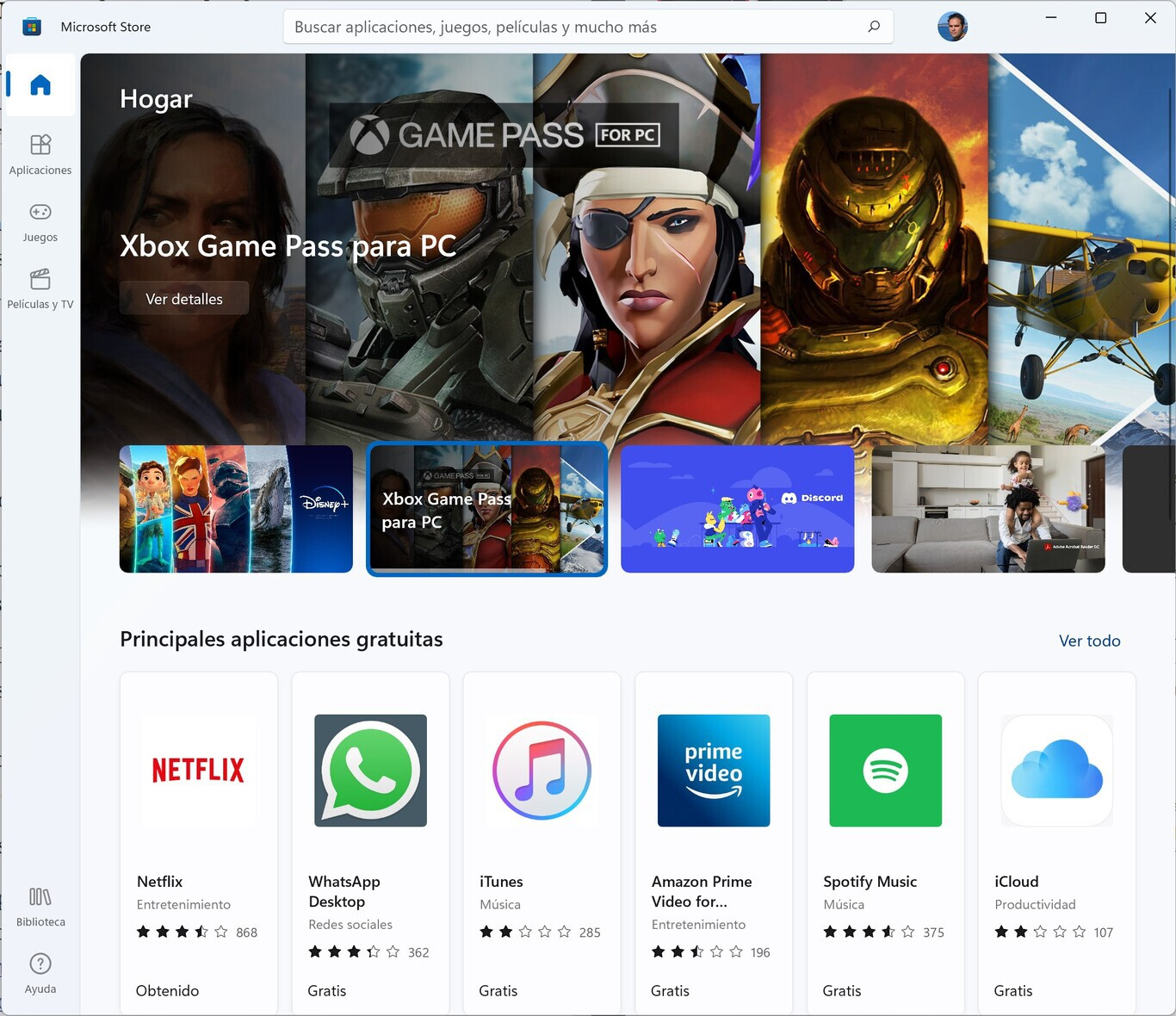
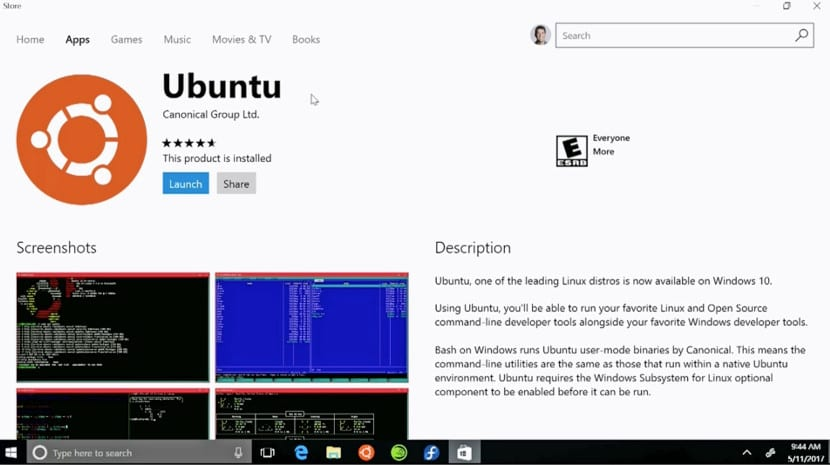
**Grupo: CCOMP6-1**

**Profesor: Julio Santisteban**

1. Familiarizarse con Linux (Ubuntu) y WSL versión 2

<https://learn.microsoft.com/es-es/windows/wsl/install>





1. Familiarizarse con los mecanismos de compilación en la línea de comando

FILE: hw.c

/\* header files go up here \*/

/\* note that C comments are enclosed within a slash and a star, and

may wrap over lines \*/

// if you use gcc, two slashes will work too (and may be preferred)

#include <stdio.h>

/\* main returns an integer \*/

int main(int argc, char \*argv[]) {

/\* printf is our output function;

by default, writes to standard out \*/

/\* printf returns an integer, but we ignore that \*/

printf("hello, world\n");

/\* return 0 to indicate all went well \*/

return(0);

}

**Compilation and Execution**

prompt> gcc hw.c

prompt> ./a.out

prompt> gcc -o hw.exe hw.c

prompt> ./hw.exe

**Useful Flags**

prompt> gcc -o hw hw.c # -o: to specify the executable name

prompt> gcc -Wall hw.c # -Wall: gives much better warnings

prompt> gcc -g hw.c # -g: to enable debugging with gdb

prompt> gcc -O hw.c # -O: to turn on optimization

**Linking with Libraries**

Note that fork() is a system call, and not just a library routine. However, the C library

provides C wrappers for all the system calls, each of which simply trap into the operating

system.

#include <math.h>

...

tan()

...

// Link with -lm.

prompt> gcc -o hw.exe hw.c -Wall –lm

The -lXXX flag tells the linker to look for libXXX.so or libXXX.a,

probably in that order.

if you want the compiler to search for headers in a different path than the usual places, or want it to link with libraries that you

specify, you can use the compiler flag -I/foo/bar to look for headers in the directory /foo/bar, and the -L/foo/bar flag to look for libraries in the /foo/bar directory.

-L/foo/bar;/foo/bar333;/foo/barv55

The -c flag tells the compiler just to produce an object file — in this case, files called hw.o and helper.o. These files are not executables, but just machine-level representations of the code within each source file. To combine the object files into an executable, you have to “link”

them together; this is accomplished with the third line gcc -o hw hw.o helper.o).

1. Crear una libreria dinamica y estatica

<http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html>

**Static Libraries: (.a)**

ctest1.c

void ctest1(int \*i)

{

\*i=5;

}

ctest2.c

void ctest2(int \*i)

{

\*i=100;

}

prog.c

#include <stdio.h>

void ctest1(int \*);

void ctest2(int \*);

int main()

{

int x;

ctest1(&x);

printf("Valx=%d\n",x);

return 0;

}

* Compile: cc -Wall -c ctest1.c ctest2.c  
  Compiler options:
  + -Wall: include warnings. See man page for warnings specified.
* Create library "libctest.a": ar -cvq libctest.a ctest1.o ctest2.o
* List files in library: ar -t libctest.a
* Linking with the library:
  + cc -o *executable-name* prog.c libctest.a
  + cc -o *executable-name* prog.c -L/path/to/library-directory -lctest

**Dynamically Linked "Shared Object" Libraries: (.so)**

**Library creation example:**

gcc -Wall -fPIC -c \*.c

gcc -shared -Wl,-soname,libctest.so.1 -o libctest.so.1.0 \*.o

mv libctest.so.1.0 /opt/lib

ln -sf /opt/lib/libctest.so.1.0 /opt/lib/libctest.so.1

ln -sf /opt/lib/libctest.so.1.0 /opt/lib/libctest.so

This creates the library libctest.so.1.0 and symbolic links to it.

It is also valid to cascade the linkage:

ln -sf /opt/lib/libctest.so.1.0 /opt/lib/libctest.so.1

ln -sf /opt/lib/libctest.so.1 /opt/lib/libctest.so

**Compile main program and link with shared object library:**

Compiling for run-time linking with a dynamically linked libctest.so.1.0:

gcc -Wall -I*/path/to/include-files* -L*/path/to/libraries* prog.c -lctest -o prog

Use:

gcc -Wall -L/opt/lib prog.c -lctest -o prog

**List Dependencies:**

The shared library dependencies of the executable can be listed with the command: [**ldd**](http://man.yolinux.com/cgi-bin/man2html?cgi_command=ldd) *name-of-executable*

Example: ldd prog

libctest.so.1 => /opt/lib/libctest.so.1 (0x00002aaaaaaac000)

libc.so.6 => /lib64/tls/libc.so.6 (0x0000003aa4e00000)

/lib64/ld-linux-x86-64.so.2 (0x0000003aa4c00000)

1. Tarea, Investigar e implementar el uso de Makefile y/o auto make

Las tareas se entregan con Makefile o automake