**Compiling with GCC**

**(*main.c*)**

#include <stdio.h>

#include “reciprocal.hpp”

int main (int argc, char \*\*argv)

{

int i;

i = atoi (argv[1]);

printf (“The reciprocal of %d is %g\n”, i, reciprocal (i));

return 0;

}

**(*reciprocal.cpp*)**

#include <cassert>

#include “reciprocal.hpp”

double reciprocal (int i) {

// I should be non-zero.

assert (i != 0);

return 1.0/i;

}

**(*reciprocal.hpp*)**

#ifdef \_\_cplusplus

extern “C” {

#endif

extern double reciprocal (int i);

#ifdef \_\_cplusplus

}

#endif

**Compiling a Single Source File**

To compile a C source file

gcc -c main.c

The C++ compiler is called g++.

g++ -c reciprocal.cpp

g++ -c -I ../include reciprocal.cpp

**Linking Object Files**

g++ -o reciprocal main.o reciprocal.o

g++ -o reciprocal main.o reciprocal.o –lpam

g++ -o reciprocal main.o reciprocal.o -L/usr/local/lib/pam –lpam

**Using *getopt\_long***

const struct option long\_options[] = {

{ “help”, 0, NULL, ‘h’ },

{ “output”, 1, NULL, ‘o’ },

{ “verbose”, 0, NULL, ‘v’ },

{ NULL, 0, NULL, 0 }

};

**Create Archives**

ar cr libtest.a test1.o test2.o

**(*test.c*)**

int f ()

{

return 3;

}

**(*app.c*)**

int main ()

{

return f ();

}

gcc -o app app.o -L. –ltest

**Shared Libraries**

gcc -c -fPIC test1.c

PIC stands for position-independent code.

Then you combine the object files into a shared library, like this:

gcc -shared -fPIC -o libtest.so test1.o test2.o

gcc -o app app.o -L. –ltest

gcc -static -o app app.o -L. –ltest`

**Pasos para agregar librerias.**

**Primero asignar los .o**

gcc -fPIC -c adios.cpp

**Crear el .so**

gcc -shared -fPIC -o libadios.so adios.o

compilar el main.

g++ main.cpp -o holaadios -ldl

**Using *LD\_LIBRARY\_PATH***

gcc -o app app.o -L. -ltest -Wl,-rpath,/usr/local/lib

**Library Dependencies**

**(*tifftest.c*)**

#include <stdio.h>

#include <tiffio.h>

int main (int argc, char\*\* argv)

{

TIFF\* tiff;

tiff = TIFFOpen (argv[1], “r”);

TIFFClose (tiff);

return 0;

}

gcc -o tifftest tifftest.c –ltiff

**Dynamic Loading and Unloading**

You could open a shared library named libtest.so by calling dlopen like this

**dlopen (“libtest.so”, RTLD\_LAZY)**

void\* handle = dlopen (“libtest.so”, RTLD\_LAZY);

void (\*test)() = dlsym (handle, “my\_function”);

(\*test)();

dlclose (handle);

Para compilar necesitamos poner -ldl

g++ holaadios.cpp -o holaadios –ldl

**Processes**

**Looking at Processes**

Los procesos hijo heredan las tuberías padre.

Los paréntesis abren un subproceso: (Se duplica la terminal). Por ejemplo: **(cd pepe).**

**Ejecutar el proceso en segundo plano se utiliza al final el &**

**(echo “Hola” && sleep 3 && echo “adios”)&**

Un proceso que se ejecuta en segundo plano se llama **job**, que tienen un number

**Jobs** nos muestra un listado de todos los detenidos y ejecutándose.

Detener un proceso con ctrl + z.

Como levantar un proceso que está en segundo plano (fg y el número del proceso).

**Process IDs**

#include <stdio.h>

#include <unistd.h>

int main ()

{

printf (“The process ID is %d\n”, (int) getpid ());

printf (“The parent process ID is %d\n”, (int) getppid ());

return 0;

}

**Viewing Active Processes**

Para ver un proceso tenemos que poner **ps**, **pid** numero de proceso.

**Using *fork* and *exec***

**Calling *fork***

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main ()

{

pid\_t child\_pid;

printf (“the main program process ID is %d\n”, (int) getpid ());

child\_pid = fork ();

if (child\_pid != 0) {

printf (“this is the parent process, with id %d\n”, (int) getpid ());

printf (“the child’s process ID is %d\n”, (int) child\_pid);

}

else

printf (“this is the child process, with id %d\n”, (int) getpid ());

return 0;

}

**Using the *exec* Family**

The exec functions replace the program running in a process with another program.

Exec sustituye el programa en ejecución.

**p:** buscar en el patch actual.

Respecto los argumentos:

**v:** varargs (Array de argumentos).

**l:** lista de parámetros (Como en el printf).

En entorno (environment)

**e:** Se puede pasar una lista de variables de entorno.

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

/\* Spawn a child process running a new program. PROGRAM is the name

of the program to run; the path will be searched for this program.

ARG\_LIST is a NULL-terminated list of character strings to be

passed as the program’s argument list. Returns the process ID of

the spawned process. \*/

int spawn (char\* program, char\*\* arg\_list)

{

pid\_t child\_pid;

/\* Duplicate this process. \*/

child\_pid = fork ();

if (child\_pid != 0)

/\* This is the parent process. \*/

return child\_pid;

else {

/\* Now execute PROGRAM, searching for it in the path. \*/

execvp (program, arg\_list);

/\* The execvp function returns only if an error occurs. \*/

fprintf (stderr, “an error occurred in execvp\n”);

abort ();

}

}

int main ()

{

/\* The argument list to pass to the “ls” command. \*/

char\* arg\_list[] = {

“ls”, /\* argv[0], the name of the program. \*/

“-l”,

“/”,

NULL /\* The argument list must end with a NULL. \*/

};

/\* Spawn a child process running the “ls” command. Ignore the

returned child process ID. \*/

spawn (“ls”, arg\_list);

printf (“done with main program\n”);

return 0;

}

**Process Scheduling**

nice -n 10 sort input.txt > output.txt

**Signals**

Una señal es un numero

Librería es <signal.h>.

#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <sys/types.h>

#include <unistd.h>

sig\_atomic\_t sigusr1\_count = 0;

void handler (int signal\_number)

{

++sigusr1\_count;

}

int main ()

{

struct sigaction sa;

memset (&sa, 0, sizeof (sa));

sa.sa\_handler = &handler;

sigaction (SIGUSR1, &sa, NULL);

/\* Do some lengthy stuff here. \*/

/\* ... \*/

printf (“SIGUSR1 was raised %d times\n”, sigusr1\_count);

return 0;

}

**Process Termination**

Manda la señal a otro programa

kill -KILL pid

haces una llamada a la función kill.

kill (child\_pid, SIGTERM);