Práctica 1

Ambiente de programación

Objetivo

Conocer el ambiente del lenguaje de programación C

Introducción

La creación de un programa ejecutable utilizando como punto de partida el lenguaje C es en buena medida, una experiencia similar a la utilización de un sistema operativo, el ambiente en el cual uno trabaja se encarga de dar la impresión de simplicidad. El ejecutable se obtiene a través de diferentes pasos. La serie de pasos involucrados son los siguientes: preprocesamiento, compilación (traducción de C a ensamblador), ensamblado, enlazado.

Creación paso a paso de un programa ejecutable mínimo

Archivo fuente

Capture el siguiente código fuente y guarde el archivo con el nombre hello.c

<code>

int main(){

return 20;

}

</code>

Preprocesamiento

El preprocesamiento se realiza con alguno de los comandos: cpp, gcc -E. El archivo preprocesado se guardará con el nombre de hello.i. El archivo hello.i se denomina archivo preprocesado obtenido a partir del archivo fuente hello.c. Preprocese el archivo fuente con el comando

$ gcc -E hello.c -o hello.i

Compilación (traducción de C a ensamblador)

El archivo compilado se guardará con el nombre hello.s y se obtendrá con el comando gcc -S. Compile el archivo hello.i con el comando

$ gcc -S hello.i -o hello.s

Ensamblado

El archivo ensamblado se obtendrá a partir del archivo hello.s, con el comando as como se indicará a continuación. Ensamble el archivo hello.s con el comando

$ as hello.s -o hello.o

Enlazado

El archivo ejecutable se obtendrá enlazando el archivo hello.o con algunos archivos que provienen de una implementación de la biblioteca estándar de lenguaje C. Enlace el archivo hello.o para obtener un archivo ejecutable hello.xtn con el comando

$ gcc hello.o -o hello.xtn

El archivo obtenido, hello.xtn, es un programa ejecutable enlazado dinámicamente, como lo indica la salida del comando

$ file hello.xtn

hello.xtn: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=ce2b228c66cf2b4cc2e46844f5cb08f4e7870a6e, for GNU/Linux 3.2.0, not stripped

Ejecución del programa hello.xtn

Ejecute y corrobore la ejecución del programa hello.xtn con los comandos

$ ./hello.xtn

$ echo $?

20

Recordemos que el código fuente de la función main del archivo hello.c, únicamente contiene la sentencia return 20;, la ejecución de esta sentencia puede corroborarse utilizando el comando echo $?

Enlazado dinámico y enlazado estático

Los cuatro pasos descritos anteriormente (preprocesamiento, compilación, ensamblado, enlazado) pueden ejecutarse automáticamente a través del comando gcc aplicado directamente al archivo fuente. Como ejemplo considere el siguiente archivo fuente hola mundo hola.c, digamos, más tradicional.

<code>

#include <stdio.h>

int main(){

printf(“Hola mundo\n”);

return 0;

}

</code>

Construcción del programa ejecutable con enlazado dinámico

Para crear un programa ejecutable utilizando enlazado dinámico podemos usar el siguiente comando

$ gcc hola.c -o hola.xtn

$ ls -hgotr

-rwxrwxrwx 1 71 feb 7 20:39 hola.c

-rwxrwxrwx 1 17K feb 7 20:39 hola.xtn

$ file hola.xtn

hola.xtn: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=02518afb53243c852aff6b78088077df57122201, for GNU/Linux 3.2.0, not stripped

El programa ejecutable, imprime el mensaje ‘Hola mundo’, como se observa:

$ ./hola.xtn

Hola mundo

$

Note que el tamaño del archivo ejecutable enlazado dinámicamente es de 17 kilobytes. A continuación, se mostrará como obtener a partir del archivo fuente hola.c, un archivo ejecutable enlazado estáticamente

$ gcc hola.c -static -o holas.xtn

$ ls -hgotr

-rwxrwxrwx 1 71 feb 7 20:39 hola.c

-rwxrwxrwx 1 17K feb 7 20:39 hola.xtn

-rwxrwxrwx 1 765K feb 7 20:49 holas.xtn

$ file holas.xtn

holas.xtn: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, BuildID[sha1]=d131ee22186bde4718cc88568784cac682554e63, for GNU/Linux 3.2.0, not stripped

$ ./ holas.xtn

Hola mundo

Como se observa en la salida del comando file holas.xtn, este archivo es un ejecutable enlazado estáticamente. Nótese que ahora el tamaño del archivo holas.xtn es de 765 kilobytes.

En los párrafos siguientes se indicará cómo construir un archivo ejecutable enlazado estáticamente con una biblioteca llamada uClibc, específicamente, utilizando la versión 1.0.39 de esa biblioteca.

Construcción de la biblioteca uClibc

Dividiremos la construcción de la biblioteca en cinco pasos:

1 Obtener el código de la biblioteca uClibc

2 Copiar archivo de configuración

3 Obtención de los archivos de cabecera del kernel en ejecución

4 Construir la biblioteca uClibc

5 Instalar la biblioteca uClibc

Ahora se detallarán cada uno de los puntos anteriores

1 Obtener el código de la biblioteca uClibc

Descarga del código fuente de la biblioteca uClibc versión 1.0.39

$ wget <https://downloads.uclibc-ng.org/releases/1.0.39/uClibc-ng-1.0.39.tar.gz>

$ cp uClibc-ng-1.0.39.tar.gz $HOME/

$ cd $HOME

$ tar xvf uClibc-ng-1.0.39.tar.gz

Con esto se obtendrá un directorio de nombre uClibc-ng-1.0.39,

drwxr-xr-x 18 4.0K feb 7 03:27 uClibc-ng-1.0.39

-rwxrwxrwx 1 3.3M oct 6 14:56 uClibc-ng-1.0.39.tar.gz

2 Copiar archivo de configuración

Descargue el archivo config-1.0.39

$ wget <url de descarga del archivo config-1.0.39>

Copie el archivo de configuración config-1.0.39 al directorio uClibc-ng-1.0.39 con el siguiente comando

$ cp config-1.0.39 uClibc-ng-1.0.39/.config

3 Obtención de los archivos de cabecera del kernel en ejecución

Descargue el código fuente del kernel que esté utilizando su sistema GNU/Linux. En este caso se supondrá que se está usando el kernel linux versión 5.10,

$ cd */usr/src*

$ sudo wget -c https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.10.tar.gz

Extraer el contenido del archivo tar.gz linux-5.10.tar.gz con el comando

$ sudo tar xvf linux-5.10.tar.gz

$ cd linux-5.10/

Ahora se extraen los archivos de cabecera del kernel 5.10 con el siguiente comando

$ sudo make INSTALL\_HDR\_PATH=/usr/src/linux-5.10-headers/ headers\_install

4 Construir la biblioteca uClibc

Ahora, debemos reingresar al directorio que se obtuvo al extraer el contenido del tar.gz de la biblioteca uClibc

$ cd $HOME/uClibc-ng-1.0.39/

$ make

5 Instalar la biblioteca uClibc

$ sudo make PREFIX=”” install

Construcción de un ejecutable enlazado estáticamente con la biblioteca uClibc

En el directorio donde se tenga el archivo fuente hola.c, deberá colocarse el siguiente archivo make (deberá guardarse con el nombre Makefile)

<code>

# Archivo make para crear un programa hola mundo enlazado est\'aticamente

# con la biblioteca uClibc (versi\'on 1.0.39), 2022.02.07.

CC = gcc

LD = ld

TARGET\_PREFIX = /usr/x86\_64-linux-uclibc

STARTUP\_FILES = ${TARGET\_PREFIX}/usr/lib/crt1.o ${TARGET\_PREFIX}/usr/lib/crti.o

END\_FILES = ${TARGET\_PREFIX}/usr/lib/crtn.o

LINKED\_FILES = ${STARTUP\_FILES} ${OBJS} ${END\_FILES}

LIBS = -lc

USER\_CFLAGS += -static -nostdlib ${LINKED\_FILES} -L${TARGET\_PREFIX}/usr/lib ${LIBS}

all: holauclibc.xtn

.c.o:

$(CC) -Wall -c $<

holauclibc.xtn:hola.o

$(LD) $< ${USER\_CFLAGS} -o $@

clean:

rm -v holauclibc.xtn

</code>

Ya con el archivo make en el mismo directorio que el archivo hola.c

$ ls -hgo

-rwxrwxrwx 1 71 feb 7 20:39 hola.c

-rwxrwxrwx 1 765K feb 7 20:49 holas.xtn

-rwxrwxrwx 1 594 feb 7 21:26 Makefile

para construir el archivo ejecutable enlazado con la biblioteca uClibc, bastará con ejecutar el comando make

$ make

gcc -Wall -c hola.c

ld hola.o -static -nostdlib /usr/x86\_64-linux-uclibc/usr/lib/crt1.o /usr/x86\_64-linux-uclibc/usr/lib/crti.o /usr/x86\_64-linux-uclibc/usr/lib/crtn.o -L/usr/x86\_64-linux-uclibc/usr/lib -lc -o holauclibc.xtn

$ ls -hgo

-rwxrwxrwx 1 71 feb 7 20:39 hola.c

-rwxrwxrwx 1 765K feb 7 20:49 holas.xtn

-rwxrwxrwx 1 604K feb 7 22:34 holauclibc.xtn

-rwxrwxrwx 1 594 feb 7 21:26 Makefile

El archivo holauclibc.xtn es un archivo ejecutable enlazado estáticamente con la biblioteca uClibc

$ file holauclibc.xtn

holauclibc.xtn: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, with debug\_info, not stripped

$ ./holauclibc.xtn

Hola mundo

$

Problema a resolver en la práctica 1

Considere los dos archivos fuente, lock.c y mycatl.c, que se muestran a continuación

<code>

/\*\* lock.c - Advisory locking, sets a writable lock on the whole of the

\* file myfile and calls pause () to wait for a SIGUSR1 signal. After the

\* signal arrives, a call is made to unlock the file.

\* REF.

\* Steve D. Pate, Steve D. Pate - Unix filesystems\_ evolution, design, and implementation-Wiley (2003).djvu, page 49.

\*/

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#if 1 //LMC 2022.02.07

#define \_\_USE\_XOPEN\_EXTENDED

#include <signal.h>

#include <stdio.h>

#endif

void

mysig(int signo)

{

return;

}

int

main()

{

struct flock lk;

int fd, err;

sigset(SIGUSR1, mysig);

fd = open("myfile", O\_WRONLY);

lk.l\_type = F\_WRLCK;

lk.l\_whence = SEEK\_SET;

lk.l\_start = 0;

lk.l\_len = 0;

lk.l\_pid = getpid();

err = fcntl(fd, F\_SETLK, &lk) ;

printf("lock: File is locked\n");

pause();

lk.l\_type = F\_UNLCK;

err = fcntl(fd, F\_SETLK, &lk);

#if 1 //LMC 2022.02.07

if (err == -1) {

perror("F\_SETLK command failed :-(");

}

#endif

printf("lock: File is unlocked\n");

return 0;

}/\*end main()\*/

</code>

<code>

/\*\* mycatl.c - Will only display the file if there are no write locks held on the

\* file. If a lock is detected, the program loops up to 5 times waiting for the

\* lock to be released. Because the lock will still be held by the lock program,

\* mycatl will extract the process ID from the flock structure returned by fcntl()

\* and post a SIGUSRl signal. This is handled by the lock program which then unlocks

\* the file.

\* REF.

\* Steve D. Pate, Steve D. Pate - Unix filesystems\_ evolution, design, and implementation-Wiley (2003).djvu, page 50.

\*/

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#if 1 //LMC 2022.02.07

#define BUFSZ 512

void catfile(pid\_t);

#endif

pid\_t

is\_locked(int fd)

{

struct flock lk;

lk.l\_type = F\_RDLCK;

lk.l\_whence = SEEK\_SET;

lk.l\_start = 0;

lk.l\_len = 0;

lk.l\_pid = 0;

fcntl(fd, F\_GETLK, &lk);

return (lk.l\_type == F\_UNLCK) ? 0 : lk.l\_pid;

}

int

main()

{

//struct flock lk;

//int i, fd, err;

int i, fd;

pid\_t pid;

fd = open("myfile", O\_RDONLY);

for (i=0;i<5;i++) {

if ((pid = is\_locked(fd)) == 0) {

catfile(fd);

exit(0);

}else{

printf("mycatl: File is locked ...\n");

sleep(1);

}

}

kill(pid, SIGUSR1);

while ((pid = is\_locked(fd)) != 0) {

printf("mycatl: Waiting for lock release\n");

sleep(1);

}

catfile(fd);

return 0;

}

#if 1 //LMC 2022.02.07

void catfile(pid\_t ifd){

int nread;

char buf [BUFSZ] ;

while ((nread = read(ifd, buf, BUFSZ)) != 0) {

write(STDOUT\_FILENO, buf, nread);

}

}

#endif

</code>

Los dos archivos anteriores usarán un archivo llamado myfile, cuyo contenido es una línea con las palabras ‘Hello World’.

Construcción de lock.xtn y mycatl.xtn

Para construir los ejecutables lock.xtn y mycatl.xtn, se utilizará el siguiente archivo make (debe guardarse con el nombre Makefile)

<code>

CC = gcc

LD = ld

TARGET\_PREFIX = /usr/x86\_64-linux-uclibc

STARTUP\_FILES = ${TARGET\_PREFIX}/usr/lib/crt1.o ${TARGET\_PREFIX}/usr/lib/crti.o

END\_FILES = ${TARGET\_PREFIX}/usr/lib/crtn.o

LINKED\_FILES = ${STARTUP\_FILES} ${OBJS} ${END\_FILES}

LIBS = -lc

USER\_CFLAGS += -static -nostdlib ${LINKED\_FILES} -L${TARGET\_PREFIX}/usr/lib ${LIBS}

all: lock.xtn mycatl.xtn

.c.o:

$(CC) -Wall -c $<

lock.xtn:lock.o

$(LD) $< ${USER\_CFLAGS} -o $@

mycatl.xtn:mycatl.o

$(LD) $< ${USER\_CFLAGS} -o $@

clean:

rm -v \*.xtn \*.o

</code>

Al ejecutar el comando make, se presentará la siguiente salida que indicará un error de enlazado:

$ make

gcc -Wall -c lock.c

ld lock.o -static -nostdlib /usr/x86\_64-linux-uclibc/usr/lib/crt1.o /usr/x86\_64-linux-uclibc/usr/lib/crti.o /usr/x86\_64-linux-uclibc/usr/lib/crtn.o -L/usr/x86\_64-linux-uclibc/usr/lib -lc -o lock.xtn

ld: lock.o: en la función `main':

lock.c:(.text+0x1f): referencia a `sigset' sin definir

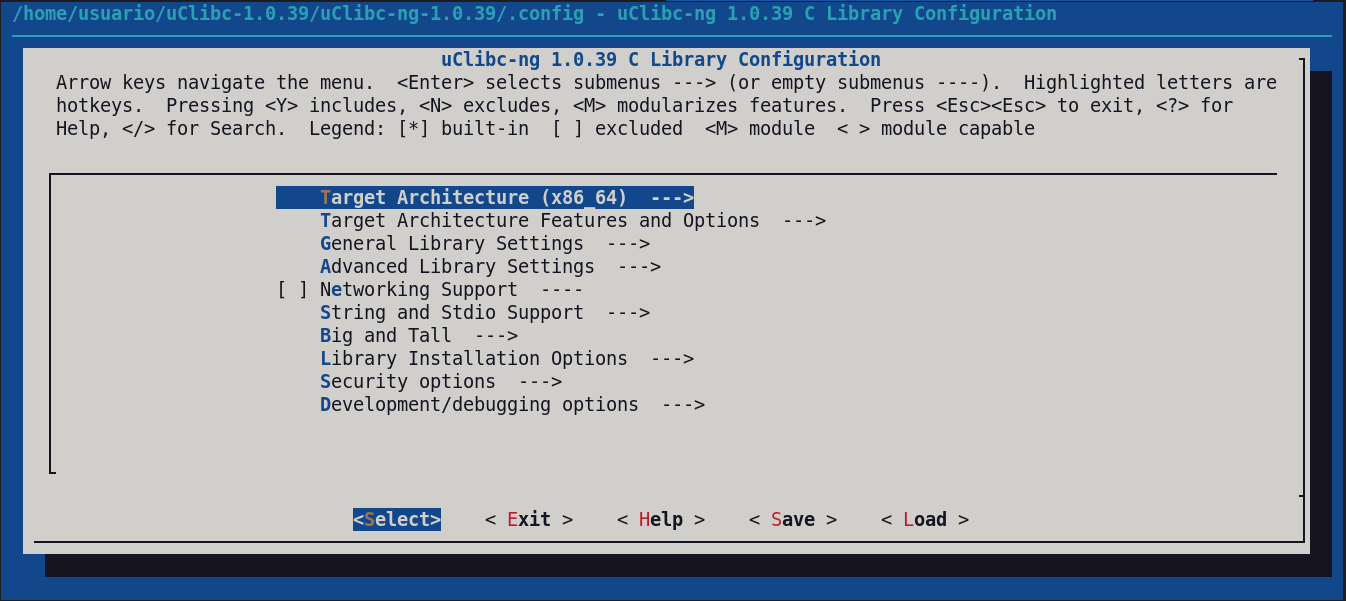
make: \*\*\* [Makefile:16: lock.xtn] Error 1

El error anterior nos indica que la función sigset() que se invoca desde el archivo lock.c, no está definida en la biblioteca uClibc. Para corregir este error, se deberá recompilar la biblioteca uClibc. Para ello, deberá usted revisar el archivo de cabecera /usr/include/signal.h, y con base en lo que se observe en ese archivo, se deberá modificar el archivo de configuración .config de las fuentes de uClibc. Para ello deberá utilizarse el comando make menuconfig,

$ cd $HOME/uclibc-ng-1.0.39/

$ make menuconfig

en respuesta a este comando, se deberá obtener la ejecución de una interface basada en ncurses como se muestra en la imagen siguiente:



Las opciones a seleccionar, deberán deducirse de la revisión del archivo de cabecera /*usr/*include/signal.h.

Después de guardar el archivo de configuración .config de las fuentes de uClibc, seleccionando la opción <Exit> tantas veces como sea necesario, se deberá recompilar e instalar la nueva versión de la biblioteca con el comando:

$ sudo make PREFIX=”” install

Cuando las opciones guardadas en el .config de uClibc sean las correctas, deberá ser posible construir los ejecutables lock.xtn y mycatl.xtn como se muestra a continuación:

$ make

ld lock.o -static -nostdlib /usr/x86\_64-linux-uclibc/usr/lib/crt1.o /usr/x86\_64-linux-uclibc/usr/lib/crti.o /usr/x86\_64-linux-uclibc/usr/lib/crtn.o -L/usr/x86\_64-linux-uclibc/usr/lib -lc -o lock.xtn

gcc -Wall -c mycatl.c

ld mycatl.o -static -nostdlib /usr/x86\_64-linux-uclibc/usr/lib/crt1.o /usr/x86\_64-linux-uclibc/usr/lib/crti.o /usr/x86\_64-linux-uclibc/usr/lib/crtn.o -L/usr/x86\_64-linux-uclibc/usr/lib -lc -o mycatl.xtn

Como evidencia de que usted ha configurado correctamente las fuentes de uClibc, deberá entregar (por el medio que se le solicite) el archivo de configuración utilizado para recompilar uClibc, con el nombre

config-1.0.39-<numero\_de\_boleta>.