Sistemas Operativos

Nachos: Introduccion

Agosto 2006

Profesor Jonathan Makuc // jmakuc@gmail.com

Índice

1. Introducción

1.1. Generalidades

El presente documento es una introducción a NachOS que explica su instalación, ejecución y arquitectura de componentes.

1.2. Qué es NachOS *(Not Another Completely Euristic Operative System)*

NachOS es una aplicación desarrollada en C++ en la Universidad de Berkley para su uso en la enseñanza del curso *Sistemas Operativos*.

En si, NachOS es un pequeño sistema operativo de kernel monolítico, diseñado para correr sobre plataformas tipo UNIX, de estructura simple, documentado; que permite a los alumnos aplicar conceptos teóricos del ramo en modificaciones reales a código. El detalle de su arquitectura se explica en el punto 3.

La versión que se trata en este documento es una modificación de la publicada en el sitio oficial http://www.cs.washington.edu/homes/tom/nachos/ y puede descargarse de http://weblogs.udp.cl/jmakuc seccion Sistemas Operativos.

2. Instalación

2.1. Generalidades

A continuación se explica como instalar y ejecutar NachOS en maquinas Intel (o similares) sobre Linux Ubuntu. Asuma que estamos trabajando en un computador cuyo nombre es "*linux*" y con nombre de usuario "*usuario*".

2.2. Requerimientos

Para la instalación de NachOS, se debe tener un computador con las siguientes características:

- Procesador: Intel o similar
- Sistema operativo Linux con kernel 2.4 o superior
- GCC 3.3 (versiones superiores no funcionan)

2.3. Instalación

2.3.1. Obtención de GCC 3.3

En sistemas Linux basados en Debian, como lo es Ubuntu, la obtención de nuevos programas se realiza simplemente a traves del comando **apt-get**. Para instalar gcc 3.3 a traves de este método, realice:

```
usuario@linux$ sudo su
password: <suclave>
root@linux:~#
```

En este momento tenemos privilegios de superusuario y se puede verificar porque el simbolo **"$"** ha cambiado por **"#"**. A continuación:

```
root@linux:~# apt-get install gcc-3.3 g++-3.3
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be installed:
 libstdc++5-3.3-dev
Suggested packages:
 gcc-3.3-doc libstdc++5-3.3-doc
The following NEW packages will be installed:
 g++-3.3 gcc-3.3 libstdc++5-3.3-dev
0 upgraded, 3 newly installed, 0 to remove and 47 not upgraded.
Need to get 0B/3094kB of archives.
After unpacking 12.9MB of additional disk space will be used.
Do you want to continue [Y/n]?

Preconfiguring packages ...
Selecting previously deselected package gcc-3.3.
(Reading database ... 71791 files and directories currently installed.)
Unpacking gcc-3.3 (from .../gcc-3.3_3.3.6-8ubuntu1_i386.deb) ...
Selecting previously deselected package libstdc++5-3.3-dev.
Unpacking libstdc++5-3.3-dev (from
.../libstdc++5-3.3-dev_3.3.6-8ubuntu1_i386.deb) ...
Selecting previously deselected package g++-3.3.
Unpacking g++-3.3 (from .../g++-3.3_3.3.6-8ubuntu1_i386.deb) ...
Setting up gcc-3.3 (3.3.6-8ubuntu1) ...
Setting up libstdc++5-3.3-dev (3.3.6-8ubuntu1) ...
Setting up g++-3.3 (3.3.6-8ubuntu1) ...
root@linux:~#
```

2.3.2. Descarga

Obtenga una copia de la distribución que trata este documento en http://weblogs.udp.cl/jmakuc/archivos/(1137)Nachos-Dist.tar.gz

Guarde el archivo en un directorio que recuerde.

2.3.3. Descompresión

El archivo viene empaquetado y comprimido. Para extraer el contenido, posicionese en el directorio donde descargó el archivo y ejecute el siguiente comando:

```
usuario@linux$ tar –xvzf "(1137)Nachos-Dist.tar.gz"
```

Modificadores utilizados:

- x – extraer
- v – verbose, lista todos los archivos extraidos
- z – descomprime el archivo mediante gzip
- f – indica que se utilizada el archivo cuyo nombre viene a continuación

Al descomprimir se creara una carpeta **nachos-dist** en el directorio.

2.3.4. Ubicación

Mueva el directorio **nachos-dist** al directorio **/usr/local** del sistema mediante el siguiente comando. Para poder escribir en ese directorio se requieren permisos de superusuario.

```
root@linux:~# mv nachos-dist /usr/local/nachos
```

A continuación para permitir que cualquier usuario pueda modificar el contenido de /usr/local/nachos, cambie los permisos de acceso a tal directorio ejecutando:

```
root@linux:~# chmod –R 777 /usr/local/nachos
```

2.4. Compilación

2.4.1. Modificación Makefile para gcc 3.3

NachOS se distribuye solo en codigo fuente, debiéndose compilar para poder ejecutarlo. NachOS utiliza el sistema de script de compilación Makefile, encontrándose este en el directorio a continuación.

Acceda al directorio /usr/local/nachos/code/build.linux:

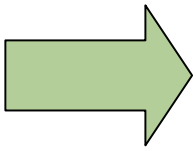
```
usuario@linux$ cd /usr/local/nachos/code/build.linux
```

Como se mencionó anteriormente, NachOS requiere de GCC version 3.3 para compilar correctamente. De manera de no afectar el resto del sistema, modificaremos el script de compilación de NachOS para que llame a la versión 3.3 en lugar de que viene con la distribución de Linux que se este utilizando.

Utilizando el editor de texto **gedit** (u otro de su preferencia), realice el siguiente cambio en el archivo **Makefile** en la linea 183:

Original

```
############################
CPP=/lib/cpp
CC = g++
LD = g++
AS = as
RM = rm
```



Cambiado

```
############################
CPP=/usr/bin/cpp-3.3
CC=/usr/bin/g++-3.3
LD=/usr/bin/g++-3.3
AS = as
RM = rm
```

Una vez realizado este cambio se esta listo para proceder con la compilación.

2.4.2. Creación de dependencias

Como primer paso, debemos establecer las dependencias que existen entre todos los componentes que forman NachOS, de manera de determinar si estan presenten todos los archivos requeridos. Para esto se ejecuta el siguiente comando, con su respectiva salida:

```
usuario@linux$ make depend

/usr/bin/g++-3.3 -g -Wall -Wno-deprecated -fwritable-strings -I../network -
I../filesys -I../userprog -I../threads -I../machine -I../lib -DRDATA -
DSIM_FIX -DFILESYS_STUB -Dx86 -DLINUX -DCHANGED -M ../lib/bitmap.cc
../lib/debug.cc ../lib/hash.cc ../lib/libtest.cc ../lib/list.cc
../lib/sysdep.cc ../machine/interrupt.cc ../machine/stats.cc
../machine/timer.cc ../machine/console.cc ../machine/machine.cc
../machine/mipssim.cc ../machine/translate.cc ../machine/network.cc
../machine/disk.cc ../threads/alarm.cc ../threads/kernel.cc
../threads/main.cc ../threads/scheduler.cc ../threads/synch.cc
../threads/thread.cc ../userprog/addrspace.cc ../userprog/exception.cc
../userprog/synchconsole.cc ../userprog/proctable.cc
../userprog/synchbitmap.cc ../userprog/table.cc ../filesys/directory.cc
../filesys/filehdr.cc ../filesys/filesys.cc ../filesys/pbitmap.cc
../filesys/openfile.cc ../filesys/synchdisk.cc  ../network/post.cc > makedep

ed - Makefile.dep < eddep

rm eddep makedep

usuario@linux$
```

2.4.3. Compilación y linkeo

Si se obtiene la salida anterior, quiere decir que no existen problemas de dependencias y se esta listo para generar los programas objeto y su posterior linkeo. Entonces ejecute:

```
usuario@linux$ make

/usr/bin/g++-3.3 -g -Wall -Wno-deprecated -fwritable-strings -I../network -
I../filesys -I../userprog -I../threads -I../machine -I../lib -DRDATA -
DSIM_FIX -DFILESYS_STUB -Dx86 -DLINUX -DCHANGED -c ../lib/bitmap.cc

/usr/bin/g++-3.3 -g -Wall -Wno-deprecated -fwritable-strings -I../network -
I../filesys -I../userprog -I../threads -I../machine -I../lib -DRDATA -
DSIM_FIX -DFILESYS_STUB -Dx86 -DLINUX -DCHANGED -c ../lib/debug.cc

/usr/bin/g++-3.3 -g -Wall -Wno-deprecated -fwritable-strings -I../network -
I../filesys -I../userprog -I../threads -I../machine -I../lib -DRDATA -
DSIM_FIX -DFILESYS_STUB -Dx86 -DLINUX -DCHANGED -c ../lib/libtest.cc

  (...)
```

```
../machine/mipssim.cc: In member function `void
   Machine::OneInstruction(Instruction*)':
../machine/mipssim.cc:142: warning: array subscript has type `char'
../machine/mipssim.cc:161: warning: array subscript has type `char'
../machine/mipssim.cc:161: warning: array subscript has type `char'
(...)
/usr/bin/cpp-3.3  -P -I../network -I../filesys -I../userprog -I../threads -
I../machine -I../lib -Dx86 -DLINUX ../threads/switch.s > swtch.s
as -o switch.o swtch.s
/usr/bin/g++-3.3 bitmap.o debug.o libtest.o sysdep.o interrupt.o stats.o
timer.o console.o machine.o mipssim.o translate.o network.o disk.o alarm.o
kernel.o main.o scheduler.o synch.o thread.o addrspace.o exception.o
synchconsole.o proctable.o synchbitmap.o table.o directory.o filehdr.o
filesys.o pbitmap.o openfile.o synchdisk.o post.o switch.o  -o nachos
usuario@linux$
```

Si se ha producido la salida anterior, entonces se tiene NachOS compilado. Se han destacado 2 porciones de la salida relevantes. La primera en celeste muestra la compilación de una clase en su programa objeto. La segunda en amarillo, muestra el comando que likea todos los programas objeto en el ejecutable **nachos**.

Como último paso crearemos un link simbólico del ejecutable en el directorio de programas de usuario, de manera de poder acceder a el sin necesidad de copiarlo cada vez que se recompile nachos. Para esto ejecute:

```
usuario@linux$ cd ../test
usuario@linux$ ln –s ../build.linux/nachos nachos
```

2.5. Ejecución

2.5.1. Ejecución Simple

La ejecución simple, sin programa de usuario, se puede interpretar como: encender la máquina simulada, iniciar el sistema operativo, determinar que no existe programa a ejecutar y apagar el sistema.

Estando en el directorio **code/build.linux**, ejecute:

```
usuario@linux$./nachos
Machine halting!

Ticks: total 10, idle 0, system 10, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Aqui NachOS entrega información sobre lo que alcanzo a hacer antes de cerrarse. Indica que transcurrieron 10 ticks de reloj CPU, siendo los 10 de sistema. No existieron accesos a disco, ni consola, ni red, ni fallas de página.

2.5.2. Ejecución de un programa de usuario

Los programas de usuario se encuentran ubicados en code/test. Estos estan programados en una version muy simplificada de C, y deben ser compilados para la maquina simulada MIPS. El detalle de esto y otros temas de compilación cruzada se encuentra en el punto 3.2 del presente documento.

Luego de compilar los programas de usuario, estos se ejecutan escribiendo:

```
usuario@linux$./nachos -x NOMBRE_PROGRAMA
```

2.5.3. Ejecución con Debug

NachOS provee una serie de opciones de debug que habilitan distintos mensajes de depuración. Las opciones de debug que vienen con NachOS son:

- **a** Address Space
- **c** System Calls
- **d** Emulación de disco
- **f** Sistema de Archivos
- **i** Interrupciones
- **m** Emulacion de maquina MIPS (imprime las instrucciones aseembler ejecutadas)
- **n** Emulación de red
- **s** Lock, semáforos y variables condición
- **t** Thread
- **+** Todo

Para ejecutar NachOS en modo debug, mostrando depuracion de hebras y address space:

```
usuario@linux$./nachos -d ta -x NOMBRE_PROGRAMA
```

3. Arquitectura

3.1. Maquina Linux v/s Maquina Simulada

La arquitectura general de funcionamiento de NachOS se basa en siguiente diagrama.

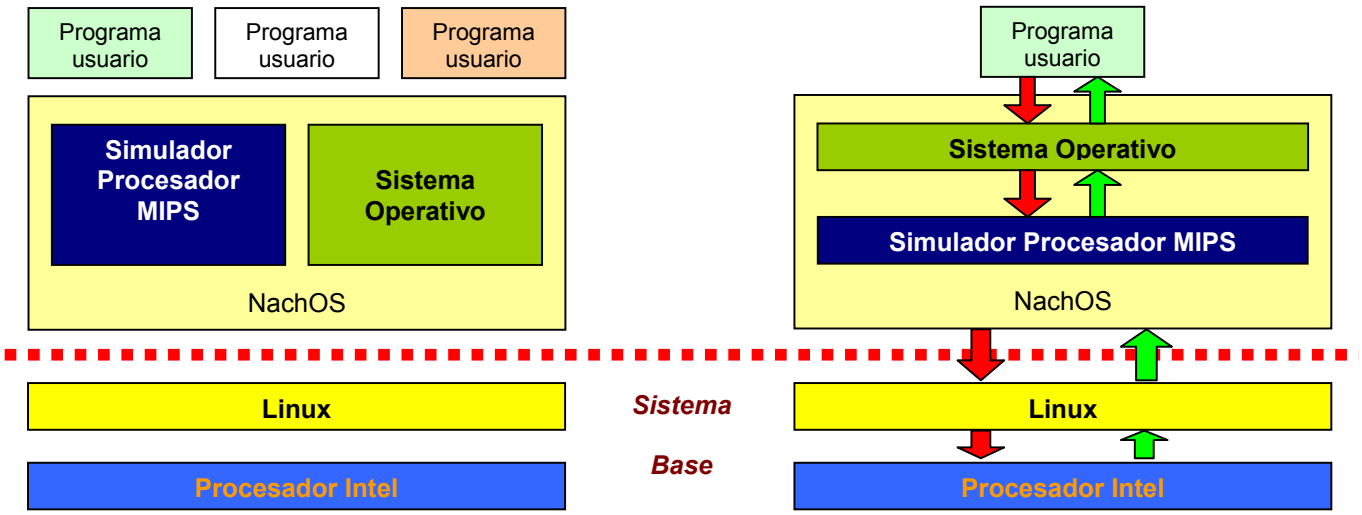


Diagrama 1
Estructura de componentes de NachOS

Diagrama 2
Diagrama de flujo

Como muestra el diagrama 1, NachOS corre sobre el sistema operativo Linux como cualquier otro programa. Tiene un número de proceso, memoria asignada, etc; como cualquier otro proceso Linux.

Por su parte NachOS consta de 2 componentes principales: un simulador de procesador MIPS y un sistema operativo básico funcional.

El simulador de procesador MIPS, escrito en una combinación de C++ y assembler propietario para el sistema base donde se ejecute, provee una maquina virtual completa para su administración por parte del sistema operativo, y para la ejecución de los programas de usuario. La simulación incluye registros, ticks de reloj, interrupciones, memoria (mapeada a la memoria del sistema base), etc.

El sistema operativo, escrito en C++, tiene dos funciones. La primera, de servicio, hace de puente entre NachOS y Linux, al ser el responsable la inicialización de NachOS en la función ***main***. La segunda es el funcionamiento como sistema operativo propiamente tal para los programas de usuario

que se carguen a NachOS, administrando el procesador simulado y sus recursos (tiempo de CPU, memoria, I/O, etc).

3.2. Compilador Cruzado

3.2.1. Descripción

Como se menciona en el punto anterior, existe la dualidad de que se esta trabajando sobre una maquina Intel, para luego correr programas de usuario sobre una máquina simulada MIPS.

Ambos procesadores no son compatibles en cuanto a programas ejecutables dado a múltiples diferencias en su arquitectura, como por ejemplo que MIPS es big-endian e Intel little-endian.

Para poder utilizar programas escritos en Intel en MIPS, se requiere de un compilador cruzado. Este es un compilador tradicional, que correra sobre el Linux en Intel, pero que producirá codigo de máquina para el procesador MIPS.

El proceso de compilación cruzada solo se lleva a cabo en el directorio code/test donde residen los programas de prueba. Para esto existe un script de compilación Makefile que se encarga de todos los pasos necesarios para la producción del ejecutable MIPS.

Se parte compilando en programa a codigo objeto. Se une con los programas que se requieran ya compilador, produciendose una salida COFF (Common Object File Formar). Esta es luego convertida en formato NOFF (NachOS Object File Format?) donde, entre otras operaciones, se invierte el encabezado del archivo ejecutable para pasar de little-endian a big-endian.

3.2.2. Compilación

Para compilar los programas de usuario que existan en el directorio **code/test**, basta ejecutar:

```
usuario@linux$ make
/usr/local/nachos/bin/decstation-ultrix-gcc -G 0 -c -I../userprog -I../lib -
I../../dec-include -I/usr/include -c halt.c
/usr/bin/cpp -I../userprog -I../lib -I../../dec-include -I/usr/include
start.s > strt.s
/usr/local/nachos/bin/decstation-ultrix-as -mips2 -o start.o strt.s
rm strt.s
/usr/local/nachos/bin/decstation-ultrix-ld -T script -N start.o halt.o -o
halt.coff
../../coff2noff/coff2noff.x86Linux halt.coff halt
numsections 3
Loading 3 sections:
        ".text", filepos 0xd0, mempos 0x0, size 0x170
        ".data", filepos 0x240, mempos 0x180, size 0x0
        ".bss", filepos 0x0, mempos 0x180, size 0x0
/usr/local/nachos/bin/decstation-ultrix-gcc -G 0 -c -I../userprog -I../lib -
I../../dec-include -I/usr/include -c shell.c
/usr/local/nachos/bin/decstation-ultrix-ld -T script -N start.o shell.o -o
shell.coff
../../coff2noff/coff2noff.x86Linux shell.coff shell
(...)
```

3.2.3. Agregar un programa de prueba

El archivo Makefile dentro de **code/test** indica que archivos deben ser compilados al hacer make. Cuando se crea un nuevo programa de usuario, se debe modificar este Makefile para que se compile.

Paso 1: Agregarlo a la lista de programas

En la fila 115 agregue su programa al final de la línea.

```
PROGRAMS = halt shell matmult (...) file_error concurrent mi_programa
```

Paso 2: Agregar una sección de compilación

Antes de la declaración:

```
clean:
        $(RM) -f *.o
        $(RM) -f *.coff
```

Incluya una sección para su programa como se muestra a continuación

```
mi_programa.o: mi_programa.c io_lib.h ../userprog/syscall.h
TAB $(CC) $(CFLAGS) -c mi_programa.c
mi_programa: mi_programa.o start.o io_lib.o
TAB $(LD) $(LDFLAGS) start.o io_lib.o mi_programa.o -o mi_programa.coff
TAB $(COFF2NOFF) mi_programa.coff mi_programa
```

IMPORTANTE: El espacio que se muestra entre el inicio de la linea y $(CC), $(LD) y $(COFF2NOFF) DEBE ser una tabulación y NO espacios en blanco, de lo contrario se arrojara error de sintaxis.

3.3. Componentes del Sistema Operativo

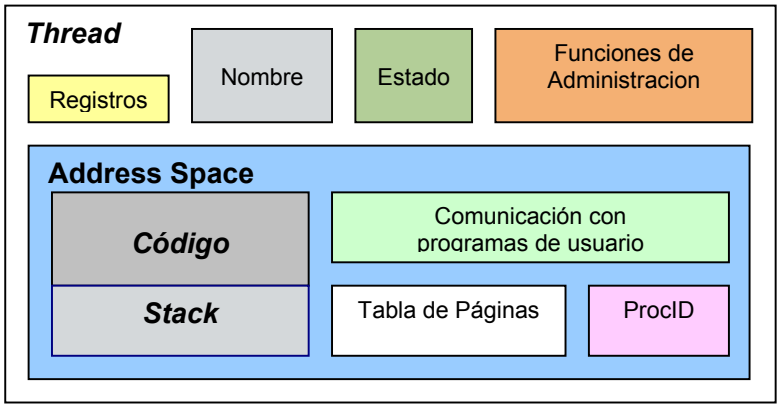
NachOS implementa los componentes base de un sistema operativo minimalista funcional. A continuación se detalla donde se encuentran las principales partes de estudio.

3.3.1. Kernel

Implementado por los archivos **kernel.h** y **kernel.cc** en ***code/thread***, concentra los punteros a todos los componentes del sistema operativo, incluyendo la representación de la maquina virtual y sus recursos.

3.3.2. Hebras y Procesos

En NachOS cada proceso es compuesto por una hebra (Thread) y el espacio de direcciones (Address Space). Estas 2 clases implementan combinadamente las definiciones formales de *Address Space* y *Process Control Block*, como se muestra en el siguiente diagrama.



NachOS separa el manejo de código y memoria, de la implementación de multiprogramación. El Address Space de NachOS se encarga de controlar que el programa de usuario este en memoria y que este pueda acceder a los recursos de memoria necesarios. Por su parte la Thread de NachOS proporciona los métodos para que el Scheduler pueda manejar el ingreso a la CPU de los procesos.

3.3.3. Scheduler

El scheduler implementado por Nachos se encuentra en ***code/thread* en scheduler.{h,cc}**. Este es round robin, ingresando los procesos uno a uno a la CPU en el orden de llegada a su *ReadyList*.

3.3.4. Sistema de Archivos

NachOS permite la implementación del sistema de archivos de 2 formas: stub y propio.

En la modalidad STUB, NachOS simplemente enlaza sus primitivas de manejo de archivos a las del sistema UNIX donde reside, permitiendo que todo se maneje directamente en los archivos UNIX.

En la modalidad propietaria, NachOS implementa el sistema de archivos en un único archivo llamado "disk" dentro del sistema de archivos UNIX. Aquí simula un sistema de archivos con una ruta única raíz "/", sin la posibilidad de directorios anidados.

En la distribución aquí trata, se utiliza el sistema de archivos vía Stub al sistema de archivos UNIX.

3.3.5. Interrupciones

De manera de tratar los problemas más importantes de los sistemas operativos reales, NachOS implementa las interrupciones en varias capas.

Por su parte la maquina MIPS simulada implementa la creación de interrupciones debido a I/O, tareas programadas, etc. Estas pueden ser apagadas y encender vía métodos que proporciona el simulador, provocando que las interrupciones se acumulen para su posterior manejo. **La única forma de proporcionar exclusión mutua a nivel de sistema operativo en NachOS, es apagando las interrupciones.**

En el sistema operativo, las únicas interrupciones que son manejadas corresponden a las programadas por el manejador de alarmas **alarm.{h,cc}**, que implementa el time slicing.

3.3.6. Exclusión Mutua

La distribución de NachOS que aquí se trata, trae implementado objetos de sincronización tradicionales como locks y semáforos, los cuales se encuentran en los archivos **synch.{h,cc}** en el directorio ***code/thread.***

3.3.7. Utilidades

NachOS incluye una pequeña librería de estructuras de datos comunes que utiliza para su implementación. Estas se encuentran en el directorio ***code/lib*** y corresponden a listas, tablas y mapas.

3.3.8. Agregar una Clase a NachOS

Si se desea incluir una clase a la estructura de Nachos y hacer que esta se compile junto con el SO, se deben realizar los siguientes pasos.

Utilizaremos la siguiente clase Ejemplo dividida en los archivos ***ejemplo.h*** y **ejemplo.cc**. Colocaremos estos archivos en el directorio ***code/userprog***.

ejemplo.h

```
class Ejemplo {

        private
                int numero;

        public:
                Ejemplo(int valor);
                int getNumero();
};
```

ejemplo.cc

```
#include "ejemplo.h"
Ejemplo::Ejemplo(int entrada) {

        numero = entrada;
}

int Ejemplo::getNumero() {

        return numero;
}
```

Luego se modifica el Makefile de code/build.linux para que se compile con el resto de las clases. En este caso, como hemos agregado los archivos al directorio userprog, modificamos la sección que corresponde a esta carpeta en la línea 259:

```
USERPROG_H = ../userprog/addrspace.h\
        ../userprog/syscall.h\
        ../userprog/synchconsole.h\
        ../userprog/noff.h\
        ../userprog/errno.h\
        ../userprog/proctable.h\
        ../userprog/synchbitmap.h\
        ../userprog/table.h\
        ../userprog/ejemplo.h

USERPROG_C = ../userprog/addrspace.cc\
        ../userprog/exception.cc\
        ../userprog/synchconsole.cc\
        ../userprog/proctable.cc\
        ../userprog/synchbitmap.cc\
        ../userprog/table.cc\
        ../userprog/ejemplo.cc

USERPROG_O = addrspace.o exception.o synchconsole.o\
        proctable.o synchbitmap.o table.o ejemplo.o
```

Nótese que debe agregarse el backslash (\ ) al final de la linea anterior, y que el espacio que hay entre en el inicio de linea y lo escrito es un **TAB** y **NO** espacios en blanco.

3.3.9. Colgar una clase al Kernel de NachOS

Si se desea tener disponible una clase para su uso por cualquiera de los componentes del sistema operativo, la mejor forma de lograrlo es colgarla del kernel. A continuación veremos como colgar la clase Ejemplo del punto anterior para su uso desde el kernel.

Primero se debe indicar que se desea utilizar la clase en el Kernel, agregando la linea en negrita en **code/thread/kernel.h.**

```
#ifndef KERNEL_H
#define KERNEL_H

#include "copyright.h"

class Thread;
class Scheduler;
class Interrupt;
class Statistics;
class Alarm;
class FileSystem;
class Machine;
class PostOfficeInput;
class PostOfficeOutput;
class SynchConsoleInput;
class SynchConsoleOutput;
class ProcTable;
class SynchDisk;
Class SynchBitmap;
class Ejemplo;

class Kernel {

(...)
  public:

    Ejemplo *ejemplo;
```

Luego se debe agregar el código que cree la instancia de la clase al momento de inicializar el Kernel. Esto se hace en el método ***Initialize*** de **code/thread/kernel.cc**. Inicializaremos nuestro ejemplo en el valor 123.

```
#include "ejemplo.h"

void
Kernel::Initialize()
{
    (...)

    ejemplo = new Ejemplo(123);

    interrupt->Enable();
}
```

En este momento se puede realizar la llamada, desde cualquier punto de Nachos:

```
kernel->ejemplo->getNumero()
```

la cual retornara el valor int 123.

3.4. Estructura de Directorios

Tomando como directorio base **/usr/local/nachos**, la estructura de archivos es la siguiente. Solo se detalla el subdirectorio ***code***, dado que las otras carpetas (bin, coff2noff, decstation-ultrix, incluye, lib) son utilidades para el compilador cruzado.

code/	Directorio base de código NachOS
code/build.linux	Directorio de compilación de NachOS. Contiene las reglas de compilación, los programa objetos al compilar y el ejecutable linkeado. **Archivos Importantes:**

	- Makefile: detalla la forma de compilar cada componente de NachOS. Al agregar una nueva clase, se debe realizar su inclusión en este script.
code/filesys	Implementación del sistema de archivos de NachOS y las primitivas de acceso al mismo.
code/lib	Utilidades (Listas, mapas, debug, hash y dependencias al sistema) **Archivos Importantes:** - debug.h: declaración de los tipos de mensajes de debug que posee NachOS.
code/machine	Emulación de máquina MIPS. **NO MODIFICAR ESTE DIRECTORIO PUES SE ESTARÍA MODIFICANDO EL PROCESADOR MISMO.**
code/network	Implementación del manejo de red de NachOS
code/test	Directorio donde se ubican los programas de usuario a ejecutarse en NachOS. Contiene los códigos fuente, programas objeto y ejecutables traducidos a MIPS. **Archivos Importantes:** - Makefile: detalla la forma de compilar cada programa de usuario de NachOS. Al incluir un nuevo programa, también debe agregarse a este archivo.
code/threads	Directorio principal de NachOS. Contiene el método main (iniciador de NachOS), kernel, scheduler, hebras y cambio de contexto.
code/userprog	Contiene las clases que manejan la interacción con los programas de usuario de NachOS: address space (addrspace.{h,cc}), manejo de excepciones (exception.cc), syscalls (syscall.h), etc.