Implementación de divacks en el kernel Linux v2.6.35.7

Omar A. Diaz, Andrés Arcia-Moret
andres.arcia@ula.ve
Universidad de Los Andes, Mérida, Venezuela.

21 de noviembre de 2013

El objetivo de esta práctica es guiar al estudiante paso a paso en el proceso de implementación de la técnica de *divacks* en un kernel linux, desde la obtención de las fuentes, hasta la compilación del nuevo y modificado kernel.

Lo primero que debe quedar claro es que el kernel linux es solo un programa más, escrito en lenguaje C, como cualquier otro que hayas visto, solo que quizás más grande y compuesto por muchos archivos diferentes. Esto quiere decir que si quieres que hacer alguna modificación en el kernel y que ésta este activa, necesitas agregar tu modificación, volver a compilar el kernel y arrancar el sistema con el kernel modificado. También quiere decir que si guardas la última compilación del kernel y tu modificación tiene algún error que no te permite arrancar el sistema, solo tienes que arrancar desde la última compilación funcional. La intención de este comentario es que se pierda el miedo a revisar y modificar el kernel, si has revisado y modificado código ajeno (en C), entonces estás listo para modificar un kernel linux.

Nota: Para el desarrollo de la práctica se utilizará el kernel v2.6.35.7.

0.1. Paso 1: Obtener del código fuente

Las fuentes del kernel linux la podemos conseguir en el sitio http://www.kernel.org, aquí se encuentran actualizadas las últimas versiones del kernel. Descarga la última versión del kernel o la que desees utilizar, lo que obtendrás aquí es un paquete comprimido de contiene todos los archivos que conforman el kernel.

Cópialo en el directorio `/usr/src`:

```
~$ sudo cp ~/Descargas/linux-2.6.35.7.tar.bz2 /usr/src/
```

Ahora descomprímelo:

```
~$ cd /usr/src
/usr/src$ sudo tar -xvf linux-2.6.35.7.tar.bz2
```

Es recomendable hacer un enlace simbólico[1] a este directorio y trabajar sobre él, de tal manera que podamos realizar *scripts* hacia una dirección y si necesitamos cambiar de kernel, solo es necesario cambiar apuntar el enlace al nuevo directorio:

```
/usr/src$ sudo ln -s linux-2.6.35.7/ linux
```

0.2. Paso 2: Configurar el kernel

Antes de poder configurar y compilar el kernel es necesario contar con algunos programas:

```
~$ sudo aptitude install build-essential libncurses5-dev
```

Configurar el kernel significa habilitar las opciones que queremos que tenga disponible y deshabilitar aquellas que no utilizaremos. El kernel posee muchas opciones, por lo tanto que caso de no saber es preferible dejar las opciones que ya están definidas.

Como cualquier otro programa, el kernel linux se compila por medio de un *Makefile*, esto presenta la ventaja, además de no necesitar conocer todas las dependencias entre archivos, de que después de compilar una primera vez, en las demás ocasiones solo se compilaran aquellos archivos que hayan sido modificados o que dependan de los archivos modificados, por lo tanto el proceso de compilación tardará considerablemente menos tiempo.

Existen tres formas de configurar el kernel, las cuales se llaman a través de los siguientes comandos:

1. `Make config`.
2. `Make menuconfig`.

[1] Es un enlace o acceso a un directorio o archivo que está en otro lugar pero cualquier modificación realizada en el enlace se reflejará en el original, sin embargo al eliminar el enlace el original se conserva.

3. `Make xconfig`.

La primera opción nos permite modificar las opciones en modo texto, una a una. La desventaja que presenta es que tenemos que pasar por todas las opciones del kernel secuencialmente y no podemos retroceder a las opciones ya pasadas.

La segunda opción (que utilizaremos en esta práctica), utiliza la librería ncurses, que instalamos anteriormente. Usando `Make menuconfig` podemos movernos entre las opciones de configuración a través de una interfaz más amigable.

La última opción ofrece una interfaz gráfica, lo que nos permite usar el ratón para seleccionar y buscar las opciones de configuración de nuestra preferencia. Pero para utilizar esta opción es necesario tener instalado el entorno de ventanas.

Cada parámetro de configuración tiene dos o tres opciones que podemos elegir:

- 'y' o '*', significa cuando se arranque con este kernel esta opción estará soportada.
- 'n' o '' (vacio), significa que el kernel no soportará esta opción.
- 'm', significa que la opción estará disponible como módulo, es decir, que para utilizarla es necesario montar este módulo en el kernel (ver cómo montar módulos).

Para empezar la configuración ejecutamos el siguiente comando:

```
/usr/src$ cd linux
/usr/src/linux$ sudo make menuconfig
```

Para esta práctica solo cambiaremos el algoritmo de control de congestión que trae por defecto el kernel, para eso debemos seguir esta ruta: `Networking support ->Netoworking options ->TCP: advanced congestion control ->Default TCP congestion control.`

Deberías poder ver algo como esto:

¿Qué algoritmos de control de congestión hay disponibles en el kernel y cuál viene por defecto?

Una vez guardada la configuración se creará el archivo `.conf`.

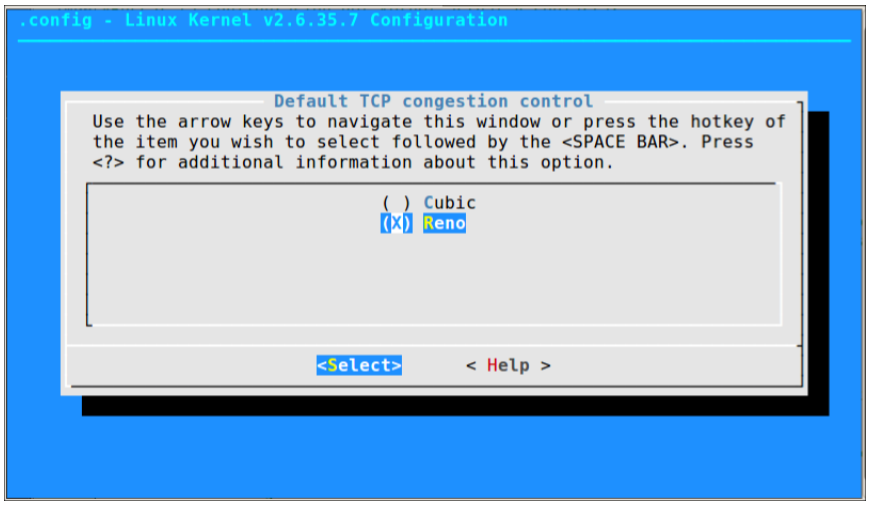

Figura 1: Captura de pantalla menuconfig: Selección del algoritmo de control de congestión.

0.3. Paso 3: Implementación de la técnica de *divacks*

0.3.1. Modificaciones en el cliente:

Las siguientes modificaciones deberán realizarse únicamente en el kernel que será implementado en el cliente.

Implementación del parámetro del kernel `tcp_divack`

tcp_divack será un parámetro del kernel que permitirá controlar el envío de *divacks* recibiendo como valor números enteros, donde:

- 0: Desactiva el mecanismo de envío de *divacks*, es decir, se envía un ACK por paquete de datos.
- 1: Divide el ACK saliente en dos partes, es decir, se envía un *divack* y el *full-ACK*.
- 2: Divide el ACK saliente en tres partes, es decir, se envían dos *divacks* y el *full-ACK*.

Y así en adelante hasta *SMSS - 1*, donde que cada *divack* y el *full-ACK* reconocerían un solo byte del paquete de datos.

Como cualquier variable, **tcp_divack** debe ser declarada antes de poder ser usada. En el archivo net/ipv4/tcp_input.c agregamos el siguiente código. Se utiliza el prefijo *sysctl_* para indicar que es un parámetro de control del sistema:

```
98 int sysctl_tcp_divack __read_mostly;
```

El siguiente paso es modificar la estructura de control `ctl_table ipv4_table[]` definida en el archivo net/ipv4/sysctl_net_ipv4.c. En esta tabla se especifican el 'procname'(nombre de

la variable en el árbol de control del sistema), la dirección de la variable que contiene el dato ('data'), el tamaño de la variable ('maxlen'), los permisos que posee ('mode') y la función que la maneja ('proc_handler'). En este caso (Cuadro 0.3.1) el 'procname'(nombre de la variable en el directorio /proc/sys/net/ipv4) es, como ya se había expuesto, **tcp_divack**, la variable que contiene el dato es la ya declarada `sysctl_tcp_divack`, tiene el tamaño de un entero, para los permisos se específica que el dueño del archivo es *root* y que tiene permisos de lectura y escritura, mientras que el grupo y otros solo tienen permisos de lectura (0644), y por último el 'proc_handler'se asigna a `proc_dointvec` el cual lee y escribe valores enteros desde y hacia el *buffer* de usuario.

Debemos entonces agregar el siguiente código en el archivo net/ipv4/sysctl_net_ipv4.c, en la estructura ipv4_table[ ] (línea 493):

```
493     {  .procname = "tcp_divack",
494        .data     = &sysctl_tcp_divack,
495        .maxlen   = sizeof(int),
496        .mode     = 0644,
497        .proc_handler = proc_dointvec,
498     }
```

En el archivo include/linux/sysctl.h se definen las interfaces de control del sistema de Linux, aquí se definirá el nombre *sysctl* a través de un *enum* (línea 428):

```
428  NET_TCP_divack = 126;
```

Ahora debemos agregar la variable en la estructura `bin_net_ipv4_table[]` del archivo kernel/sysctl_binary.c, en la cual se debe especificar la función de modificación[2] (CTL_INT para enteros), el nombre *sysctl* que acabamos de definir y el *procname* (línea 392):

```
392  {CTL_INT, NET_TCP_divack, "tcp_divack"}
```

Finalmente se agrega `sysctl_tcp_divack` (como variable externa) junto al resto de las variables *sysctl* para TCP en el archivo include/net/tcp.h (línea 241):

```
241  extern int sysctl_tcp_divack;
```

Implementación del algoritmo de *divacks*:

El mecanismo de envío de `divacks` se implenta en la función `__tcp_ack_snd_check()` (en el archivo net/ipv4/tcp_input.c), la cual decide si enviar un ACK o esperar al envío de un ACK retrasado. Lo que se hará es guardar el número de secuencia del ACK (`rcv_nxt_original`) y en lugar de enviarlo, se le resta `sysctl_tcp_divack` y se entra en un lazo donde, si el número de secuencia del ACK es menor a `rcv_nxt_original`, se envía y aumenta en uno este número de secuencia. Al salir del lazo se habrán enviado tantos *divacks* como lo indica `sysctl_tcp_divack`, donde el primer *divack* reconoce:

[2]Las funciones de modificación son las encargadas de actualizar el valor de los parámetros de configuración del kernel.

bytes, luego cada *divack* reconoce solo un nuevo byte del paquete, hasta que finalmente se envía el *full-ACK* para reconocer por completo el último byte recibido. Cuando la variable `sysctl_tcp_divack` sea cero (0), no se modificará el número de secuencia del ACK, no se entrará en el lazo y se enviará únicamente el *full-ACK* que reconocerá todo el paquete de datos recibido.

Debemos entonces sustituir la función `__tcp_ack_snd_check()` (línea 4916), por ña siguiente modificada con el algoritmo de *divacks*:

```
4916 static void __tcp_ack_snd_check(struct sock *sk, int ofo_possible)
4917 {
4918    struct tcp_sock *tp = tcp_sk(sk);
4919    u32 rcv_nxt_original = tp->rcv_nxt; //Proyecto divacks
4920
4921    /* More than one full frame received... */
4922    if (((tp->rcv_nxt - tp->rcv_wup) > inet_csk(sk)->icsk_ack.rcv_mss
          &&
4923        /* ... and right edge of window advances far enough.
4924        * (tcp_recvmsg() will send ACK otherwise). Or...
4925        */
4926        __tcp_select_window(sk) >= tp->rcv_wnd) ||
4927        /* We ACK each frame or... */
4928        tcp_in_quickack_mode(sk) ||
4929        /* We have out of order data. */
4930        (ofo_possible && skb_peek(&tp->out_of_order_queue))) {
4931           tp->rcv_nxt-=sysctl_tcp_divack;  //Proyecto divacks
4932           /*Proyecto divacks: Envio de divacks */
4933           while(tp->rcv_nxt < rcv_nxt_original){
4934              tcp_send_ack(sk);
4935              tp->rcv_nxt++;
4936           }
4937           /*Envio del full-ACK*/
4938           tcp_send_ack(sk);
4939    } else {
4940         /* Else, send delayed ack. */
4941        tcp_send_delayed_ack(sk);
4942    }
4943 }
```

La siguiente modificación permitirá enviar en la ventana anunciada (*rwnd*) todo el el espacio disponible en el *buffer* de recepción, lo que vamos a hacer es eliminar una validación que limita el espacio disponible que el cliente puede anunciar. Comentemos entonces las siguientes líneas en la función __tcp_select_window() en el archivo net/ipv4/tcp_output.c (a partir de la línea 1910):

```
1909 ...
1910    /*if (free_space > tp->rcv_ssthresh)
1911       free_space = tp->rcv_ssthresh;*/
1912 ...
```

0.3.2. Modificaciones en el servidor:

Las siguientes modificaciones deberán realizarse únicamente en el kernel que será implementado en el servidor.

El kernel linux está protegido contra los *divacks*, mediante la siguiente modificación logramos que cada *divack* enviado provoque la actualización del la ventana de congestión (*cnwd*). Vamos a cambiar la función tcp_ack() (a partir de la línea 3701), ubicada en el archivo net/ipv4/tcp_input.c, para que quede de la siguiente forma:

```
3701  ...
3702  if (tcp_ack_is_dubious(sk, flag)) {
3703     /* Advance CWND, if state allows this. */
3704     /*Eliminamos la validacion de la bandera FLAG_DATA_ACKED
3705      if ((flag & FLAG_DATA_ACKED) && !frto_cwnd && */
3706     if (!frto_cwnd && tcp_may_raise_cwnd(sk, flag))
3707        tcp_cong_avoid(sk, ack, prior_in_flight);
3708     tcp_fastretrans_alert(sk, prior_packets - tp->packets_out,
3709                           flag);
3710  } else {
3711     /*Eliminamos la validacion de la bandera FLAG_DATA_ACKED
3712      if ((flag & FLAG_DATA_ACKED) && !frto_cwnd)*/
3713     if (!frto_cwnd)
3714        tcp_cong_avoid(sk, ack, prior_in_flight);
3715  }
3716  ...
```

Para terminar vamos a comentar otra validación que limita el crecimiento de la *cnwd*, de manera que ésta pueda crecer por cada *divack* recibido. Modificaremos la función tcp_reno_cong_avoid() en el archivo net/ipv4/tcp_cong.c (línea 360) para que quede la siguiente manera:

```
360  void tcp_reno_cong_avoid(struct sock *sk, u32 ack, u32 in_flight)
361  {
362      struct tcp_sock *tp = tcp_sk(sk);
363
364     /*Eliminamos la validacion, sabemos que necesitamos enviar muchos
           datos*/
365      /*if (!tcp_is_cwnd_limited(sk, in_flight))
366        return;*/
367  ...
```

0.4. Paso 4: Compilación del kernel modificado

Finalmente se muestran los pasos para la compilación del kernel modificado (tanto en el cliente como en el servidor):

El siguiente comando compila el kernel, la primera vez este proceso llevará bastante tiempo (no desesperes), utilizaremos la opción `-jn` para usar varios procesadores, donde n es el número de núcleos de tu procesador + 1:

```
/usr/src$ sudo make -j3
```

Abra el monitor del sistema y vea la gráfica de utilización del procesador. ¿Qué observa?

_____

_____

_____

Al finalizar la compilación se debe hacer creado el binario del kernel. Ahora instalamos los módulos del kernel con el comando:

```
/usr/src$ sudo make modules_install -j3
```

Veamos lo que hay dentro del directorio /boot:

```
/usr/src$ ls /boot
```

Ahora ejecutemos el comando:

```
/usr/src$ sudo make install
```

Vuelva a observar lo que hay en el directorio /boot. ¿Qué observa?

_____

_____

_____

El siguiente comando genera los archivos de arranque, luego de la opción `-k` va la versión del kernel utilizado en nuestro caso 2.6.35.7. Cuando se vuelva a compilar el kernel ya no se debe usar la opción `-c` (*create*), sino la opción `-u` (*update*):

```
/usr/src$ sudo update-initramfs -c -k 2.6.35.7
```

Los módulos quedarán instalados en el directorio /lib/modules/2.6.35.7, el kernel y el archivo `initrd` en /boot/vmlinuz2.6.35.7 y /boot/initrd.img-2.6.35.7 respectivamente.

Ahora debemos actualizar el gestor de arranque para que aparezca la opci´on de iniciar con el kernel que acabamos de instalar, eso lo hacemos con el comando:

```
/usr/src$ sudo update-grub
```

Y con esto hemos finalizado la implementación de la técnica de *divacks* en el kernel linux, tanto en el cliente como en el servidor. Para utilizarlo debemos reiniciar el equipo y elegir el kernel modificado en el *grub*:

```
/usr/src$ sudo reboot
```