Universal TUN/TAP device driver.
Copyright (C) 1999-2000 Maxim Krasnyansky <max_mk@yahoo.com>

  Linux, Solaris drivers
  Copyright (C) 1999-2000 Maxim Krasnyansky <max_mk@yahoo.com>

  FreeBSD TAP driver
  Copyright (c) 1999-2000 Maksim Yevmenkin <m_evmenkin@yahoo.com>

  Revision of this document 2002 by Florian Thiel <florian.thiel@gmx.net>

1. Description
  TUN/TAP provides packet reception and transmission for user space programs.
  It can be seen as a simple Point-to-Point or Ethernet device, which,
  instead of receiving packets from physical media, receives them from
  user space program and instead of sending packets via physical media
  writes them to the user space program.

  In order to use the driver a program has to open /dev/net/tun and issue a
  corresponding ioctl() to register a network device with the kernel. A network
  device will appear as tunXX or tapXX, depending on the options chosen. When
  the program closes the file descriptor, the network device and all
  corresponding routes will disappear.

  Depending on the type of device chosen the userspace program has to read/write
  IP packets (with tun) or ethernet frames (with tap). Which one is being used
  depends on the flags given with the ioctl().

  The package from http://vtun.sourceforge.net/tun contains two simple examples
  for how to use tun and tap devices. Both programs work like a bridge between
  two network interfaces.
  br_select.c - bridge based on select system call.
  br_sigio.c - bridge based on async io and SIGIO signal.
  However, the best example is VTun http://vtun.sourceforge.net :))

2. Configuration
  Create device node:
     mkdir /dev/net (if it doesn't exist already)
     mknod /dev/net/tun c 10 200

  Set permissions:
     e.g. chmod 0666 /dev/net/tun
     There's no harm in allowing the device to be accessible by non-root users,
     since CAP_NET_ADMIN is required for creating network devices or for
     connecting to network devices which aren't owned by the user in question.
     If you want to create persistent devices and give ownership of them to
     unprivileged users, then you need the /dev/net/tun device to be usable by
     those users.

  Driver module autoloading

     Make sure that "Kernel module loader" - module auto-loading
     support is enabled in your kernel. The kernel should load it on
     first access.

  Manual loading

    insert the module by hand:
        modprobe tun

  If you do it the latter way, you have to load the module every time you
  need it, if you do it the other way it will be automatically loaded when
  /dev/net/tun is being opened.

3. Program interface
  3.1 Network device allocation:

  char *dev should be the name of the device with a format string (e.g.
  "tun%d"), but (as far as I can see) this can be any valid network device name.
  Note that the character pointer becomes overwritten with the real device name
  (e.g. "tun0")

  #include <linux/if.h>
  #include <linux/if_tun.h>

  int tun_alloc(char *dev)
  {
      struct ifreq ifr;
      int fd, err;

      if((fd = open("/dev/net/tun", O_RDWR)) < 0)
         return tun_alloc_old(dev);

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

      /* Flags: IFF_TUN   - TUN device (no Ethernet headers)
       *   IFF_TAP   - TAP device
       *
       *   IFF_NO_PI - Do not provide packet information
       */
      ifr.ifr_flags = IFF_TUN;
      if(*dev)
         strncpy(ifr.ifr_name, dev, IFNAMSIZ);

      if((err = ioctl(fd, TUNSETIFF, (void *) &ifr)) < 0){
         close(fd);
         return err;
      }
      strcpy(dev, ifr.ifr_name);
      return fd;
  }

  3.2 Frame format:
  If flag IFF_NO_PI is not set each frame format is:
      Flags [2 bytes]
      Proto [2 bytes]
      Raw protocol(IP, IPv6, etc) frame.

Universal TUN/TAP device driver Frequently Asked Question.

1. What platforms are supported by TUN/TAP driver ?
Currently driver has been written for 3 Unices:
    Linux kernels 2.2.x, 2.4.x

    FreeBSD 3.x, 4.x, 5.x
    Solaris 2.6, 7.0, 8.0

2. What is TUN/TAP driver used for?
As mentioned above, main purpose of TUN/TAP driver is tunneling.
It is used by VTun (http://vtun.sourceforge.net).

Another interesting application using TUN/TAP is pipsecd
(http://perso.enst.fr/~beyssac/pipsec/), a userspace IPSec
implementation that can use complete kernel routing (unlike FreeS/WAN).

3. How does Virtual network device actually work ?
Virtual network device can be viewed as a simple Point-to-Point or
Ethernet device, which instead of receiving packets from a physical
media, receives them from user space program and instead of sending
packets via physical media sends them to the user space program.

Let's say that you configured IPX on the tap0, then whenever
the kernel sends an IPX packet to tap0, it is passed to the application
(VTun for example). The application encrypts, compresses and sends it to
the other side over TCP or UDP. The application on the other side decompresses
and decrypts the data received and writes the packet to the TAP device,
the kernel handles the packet like it came from real physical device.

4. What is the difference between TUN driver and TAP driver?
TUN works with IP frames. TAP works with Ethernet frames.

This means that you have to read/write IP packets when you are using tun and
ethernet frames when using tap.

5. What is the difference between BPF and TUN/TAP driver?
BPF is an advanced packet filter. It can be attached to existing
network interface. It does not provide a virtual network interface.
A TUN/TAP driver does provide a virtual network interface and it is possible
to attach BPF to this interface.

6. Does TAP driver support kernel Ethernet bridging?
Yes. Linux and FreeBSD drivers support Ethernet bridging.