Linux USB Printer Gadget Driver

06/04/2007

General

This driver may be used if you are writing printer firmware using Linux as the embedded OS. This driver has nothing to do with using a printer with your Linux host system.

You will need a USB device controller and a Linux driver for it that accepts a gadget / "device class" driver using the Linux USB Gadget API. After the USB device controller driver is loaded then load the printer gadget driver. This will present a printer interface to the USB Host that your USB Device port is connected to.

This driver is structured for printer firmware that runs in user mode. The user mode printer firmware will read and write data from the kernel mode printer gadget driver using a device file. The printer returns a printer status byte when the USB HOST sends a device request to get the printer status. The user space firmware can read or write this status byte using a device file /dev/g_printer . Both blocking and non-blocking read/write calls are supported.

Howto Use This Driver

To load the USB device controller driver and the printer gadget driver. The following example uses the Netchip 2280 USB device controller driver:

```
modprobe net2280
modprobe g_printer
```

The follow command line parameter can be used when loading the printer gadget (ex: modprobe g_printer idVendor=0x0525 idProduct=0xa4a8):

idVendor
: This is the Vendor ID used in the device descriptor. The default is the Netchip vendor id 0x0525. YOU MUST CHANGE TO YOUR OWN VENDOR ID BEFORE RELEASING A PRODUCT. If you plan to release a product and don't already have a Vendor ID please see www.usb.org for details on how to get one.

idProduct
: This is the Product ID used in the device descriptor. The default is 0xa4a8, you should change this to an ID that's not used by any of your other USB products if you have any. It would be a good idea to start numbering your products starting with say 0x0001.

bcdDevice
: This is the version number of your product. It would be a good idea to put your firmware version here.

iManufacturer
: A string containing the name of the Vendor.

iProduct
: A string containing the Product Name.

iSerialNum
: A string containing the Serial Number. This should be changed for each unit of your product.

iPNPstring
: The PNP ID string used for this printer. You will want to set either on the command line or hard code the PNP ID string used for your printer product.

qlen
: The number of 8k buffers to use per endpoint. The default is 10, you should tune this for your product. You may also want to tune the size of each buffer for your product.

Using The Example Code

This example code talks to stdout, instead of a print engine.

To compile the test code below:

1. save it to a file called prn_example.c
2. compile the code with the follow command:

```
gcc prn_example.c -o prn_example
```

To read printer data from the host to stdout:

```
# prn_example -read_data
```

To write printer data from a file (data_file) to the host:

```
# cat data_file | prn_example -write_data
```

To get the current printer status for the gadget driver::

```
# prn_example -get_status

Printer status is:
     Printer is NOT Selected
     Paper is Out
     Printer OK
```

To set printer to Selected/On-line:

```
# prn_example -selected
```

To set printer to Not Selected/Off-line:

```
# prn_example -not_selected
```

To set paper status to paper out:

```
# prn_example -paper_out
```

To set paper status to paper loaded:

```
# prn_example -paper_loaded
```

To set error status to printer OK:

```
# prn_example -no_error
```

To set error status to ERROR:

```
# prn_example -error
```

Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <linux/poll.h>
#include <sys/ioctl.h>
#include <linux/usb/g_printer.h>

#define PRINTER_FILE                    "/dev/g_printer"
#define BUF_SIZE                        512


/*
 * 'usage()' - Show program usage.
 */

static void
usage(const char *option)               /* I - Option string or NULL */
{
      if (option) {
              fprintf(stderr,"prn_example: Unknown option \"%s\"!\n",
                              option);
      }

      fputs("\n", stderr);
      fputs("Usage: prn_example -[options]\n", stderr);
      fputs("Options:\n", stderr);
      fputs("\n", stderr);
      fputs("-get_status    Get the current printer status.\n", stderr);
      fputs("-selected      Set the selected status to selected.\n", stderr);
      fputs("-not_selected  Set the selected status to NOT selected.\n",
                      stderr);
      fputs("-error         Set the error status to error.\n", stderr);
      fputs("-no_error      Set the error status to NO error.\n", stderr);
      fputs("-paper_out     Set the paper status to paper out.\n", stderr);
      fputs("-paper_loaded  Set the paper status to paper loaded.\n",
                      stderr);
      fputs("-read_data     Read printer data from driver.\n", stderr);
      fputs("-write_data    Write printer sata to driver.\n", stderr);
      fputs("-NB_read_data  (Non-Blocking) Read printer data from driver.\n",
                      stderr);
      fputs("\n\n", stderr);
```

```
	exit(1);
}


static int
read_printer_data()
{
	struct pollfd	fd[1];

	/* Open device file for printer gadget. */
	fd[0].fd = open(PRINTER_FILE, O_RDWR);
	if (fd[0].fd < 0) {
		printf("Error %d opening %s\n", fd[0].fd, PRINTER_FILE);
		close(fd[0].fd);
		return(-1);
	}

	fd[0].events = POLLIN | POLLRDNORM;

	while (1) {
		static char buf[BUF_SIZE];
		int bytes_read;
		int retval;

		/* Wait for up to 1 second for data. */
		retval = poll(fd, 1, 1000);

		if (retval && (fd[0].revents & POLLRDNORM)) {

			/* Read data from printer gadget driver. */
			bytes_read = read(fd[0].fd, buf, BUF_SIZE);

			if (bytes_read < 0) {
				printf("Error %d reading from %s\n",
					fd[0].fd, PRINTER_FILE);
				close(fd[0].fd);
				return(-1);
			} else if (bytes_read > 0) {
				/* Write data to standard OUTPUT (stdout). */
				fwrite(buf, 1, bytes_read, stdout);
				fflush(stdout);
			}

		}

	}

	/* Close the device file. */
	close(fd[0].fd);

	return 0;
}


static int
write_printer_data()
{
	struct pollfd	fd[1];

	/* Open device file for printer gadget. */
	fd[0].fd = open (PRINTER_FILE, O_RDWR);
	if (fd[0].fd < 0) {
		printf("Error %d opening %s\n", fd[0].fd, PRINTER_FILE);
		close(fd[0].fd);
		return(-1);
	}

	fd[0].events = POLLOUT | POLLWRNORM;

	while (1) {
		int retval;
		static char buf[BUF_SIZE];
		/* Read data from standard INPUT (stdin). */
		int bytes_read = fread(buf, 1, BUF_SIZE, stdin);

		if (!bytes_read) {
			break;
		}

		while (bytes_read) {
```

```
                        /* Wait for up to 1 second to sent data. */
                        retval = poll(fd, 1, 1000);

                        /* Write data to printer gadget driver. */
                        if (retval && (fd[0].revents & POLLWRNORM)) {
                                retval = write(fd[0].fd, buf, bytes_read);
                                if (retval < 0) {
                                        printf("Error %d writing to %s\n",
                                                        fd[0].fd,
                                                        PRINTER_FILE);
                                        close(fd[0].fd);
                                        return(-1);
                                } else {
                                        bytes_read -= retval;
                                }

                        }

                }

        }

        /* Wait until the data has been sent. */
        fsync(fd[0].fd);

        /* Close the device file. */
        close(fd[0].fd);

        return 0;
}


static int
read_NB_printer_data()
{
        int             fd;
        static char     buf[BUF_SIZE];
        int             bytes_read;

        /* Open device file for printer gadget. */
        fd = open(PRINTER_FILE, O_RDWR|O_NONBLOCK);
        if (fd < 0) {
                printf("Error %d opening %s\n", fd, PRINTER_FILE);
                close(fd);
                return(-1);
        }

        while (1) {
                /* Read data from printer gadget driver. */
                bytes_read = read(fd, buf, BUF_SIZE);
                if (bytes_read <= 0) {
                        break;
                }

                /* Write data to standard OUTPUT (stdout). */
                fwrite(buf, 1, bytes_read, stdout);
                fflush(stdout);
        }

        /* Close the device file. */
        close(fd);

        return 0;
}


static int
get_printer_status()
{
        int     retval;
        int     fd;

        /* Open device file for printer gadget. */
        fd = open(PRINTER_FILE, O_RDWR);
        if (fd < 0) {
                printf("Error %d opening %s\n", fd, PRINTER_FILE);
                close(fd);
                return(-1);
        }

        /* Make the IOCTL call. */
```

```
	retval = ioctl(fd, GADGET_GET_PRINTER_STATUS);
	if (retval < 0) {
		fprintf(stderr, "ERROR: Failed to set printer status\n");
		return(-1);
	}

	/* Close the device file. */
	close(fd);

	return(retval);
}


static int
set_printer_status(unsigned char buf, int clear_printer_status_bit)
{
	int	retval;
	int	fd;

	retval = get_printer_status();
	if (retval < 0) {
		fprintf(stderr, "ERROR: Failed to get printer status\n");
		return(-1);
	}

	/* Open device file for printer gadget. */
	fd = open(PRINTER_FILE, O_RDWR);

	if (fd < 0) {
		printf("Error %d opening %s\n", fd, PRINTER_FILE);
		close(fd);
		return(-1);
	}

	if (clear_printer_status_bit) {
		retval &= ~buf;
	} else {
		retval |= buf;
	}

	/* Make the IOCTL call. */
	if (ioctl(fd, GADGET_SET_PRINTER_STATUS, (unsigned char)retval)) {
		fprintf(stderr, "ERROR: Failed to set printer status\n");
		return(-1);
	}

	/* Close the device file. */
	close(fd);

	return 0;
}


static int
display_printer_status()
{
	char	printer_status;

	printer_status = get_printer_status();
	if (printer_status < 0) {
		fprintf(stderr, "ERROR: Failed to get printer status\n");
		return(-1);
	}

	printf("Printer status is:\n");
	if (printer_status & PRINTER_SELECTED) {
		printf("     Printer is Selected\n");
	} else {
		printf("     Printer is NOT Selected\n");
	}
	if (printer_status & PRINTER_PAPER_EMPTY) {
		printf("     Paper is Out\n");
	} else {
		printf("     Paper is Loaded\n");
	}
	if (printer_status & PRINTER_NOT_ERROR) {
		printf("     Printer OK\n");
	} else {
		printf("     Printer ERROR\n");
	}
```

```
	return(0);
}


int
main(int  argc, char *argv[])
{
	int	i;		/* Looping var */
	int	retval = 0;

	/* No Args */
	if (argc == 1) {
		usage(0);
		exit(0);
	}

	for (i = 1; i < argc && !retval; i ++) {

		if (argv[i][0] != '-') {
			continue;
		}

		if (!strcmp(argv[i], "-get_status")) {
			if (display_printer_status()) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-paper_loaded")) {
			if (set_printer_status(PRINTER_PAPER_EMPTY, 1)) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-paper_out")) {
			if (set_printer_status(PRINTER_PAPER_EMPTY, 0)) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-selected")) {
			if (set_printer_status(PRINTER_SELECTED, 0)) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-not_selected")) {
			if (set_printer_status(PRINTER_SELECTED, 1)) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-error")) {
			if (set_printer_status(PRINTER_NOT_ERROR, 1)) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-no_error")) {
			if (set_printer_status(PRINTER_NOT_ERROR, 0)) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-read_data")) {
			if (read_printer_data()) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-write_data")) {
			if (write_printer_data()) {
				retval = 1;
			}

		} else if (!strcmp(argv[i], "-NB_read_data")) {
			if (read_NB_printer_data()) {
				retval = 1;
			}

		} else {
			usage(argv[i]);
			retval = 1;
		}
	}

	exit(retval);
}
```