CEng 536 Advanced Unix
Fall 2014
Take Home Final/Kernel Project 2
Due: 19/01/2015

1 Description

In this project you are going to implement a pseudo character device driver for Linux kernel supporting priority based communication. Each data block written on device will have a priority value as the first long in the data as:

```
struct buffer {
        long prio;
        char data[1]
};
```

Actual buffer content follows the priority value. Data in the same priority value works as FIFO, however when priorities different, smaller value will be read regardless of insertion order. (file offset is ignored in operations)

```
int fd = open("/dev/pqueue0","O_RDWR"), i;
struct buffer { long prio; char data[100]
        } l[] = [{ 3, "hello"},{0,"world"},{1,"whatzup"}];

for (i=0; i < 3; i++)
        write(fd, l+i, sizeof(struct buffer));

for (i=0; i < 3; i++)
        read(fd, l+i, sizeof(struct buffer));

/* read values back as: "world", "whatzup", "hello" */
```

2 Implementation

Download and change one of the scull devices under:
`https://github.com/duxing2007/ldd3-examples-3.x`. `scullc()` is simplest but has enough features for you.
Each minor device should work as a separate priority queue. Device driver can define a default maximum number of messages. By default it is 0, meaning there is no limit. When maximum is defined, following writes will block until some process reads device and opens capacity. Maximum message length is also configurable with default value 1024 bytes. These two values are module parameters that can be set on module load.
`modprobe pqdev maxmsgnum=100000 maxmsglen=4096 numminors=8`
`numminors` is the number of minor devices created. Default value is 4.
You can use your favorite Linux distribution as long as kernel is new (¿ 3.6). Install `linux-headers` package, compile and test your device.
Your module should allocate a character device and register the following operations:
`open(), read(), write(), release(), poll(), unlocked_ioctl()`
Your reads are blocked by default (unless `O_NONBLOCK` flag given), and writes are blocked when max number of messages is reached. Your device should have wait/wakeup functionality for blocks.
Each read and write operation defines a single message. Messages are considered discontinuous, they do not span multiple reads or writes. If read buffer is shorter than message, message is truncated. `read()` will return the number of bytes actually read (message length or truncated value). Writes are similarly truncated in case of written message is longer than `maxmsglen`. Actual number of bytes written is returned. For all operations priority value (sizeof (long)) is considered as a part of message length.

Your device should support following ioctl functions:

```
#include <linux/ioctl.h>

#define PQDEV_IOC_MAGIC  'Q'

#define PQDEV_IOCRESET    _IO(PQDEV_IOC_MAGIC, 0)
#define PQDEV_IOCQMINPRI  _IO(PQDEV_IOC_MAGIC, 1)
#define PQDEV_IOCHNUMMSG  _IO(PQDEV_IOC_MAGIC, 2)
#define PQDEV_IOCTSETMAXMSGS  _IO(PQDEV_IOC_MAGIC, 3)
#define PQDEV_IOCGMINMSG  _IOR(PQDEV_IOC_MAGIC, 4, int)

#define PQDEV_IOC_MAXNR 4
```

`PQDEV_IOCRESET` deletes all existing messages cleans the priority queue. Usage is:
`ioctl(pqfd, PQDEV_IOCRESET);`

`PQDEV_QMINPRI` returns the minimum priority value existing in the priority queue. Usage is:
`minpri = ioctl(pqfd, PQDEV_IOCQMINPRI);`

`PQDEV_IOCHNUMMSG` returns the number of messages in given priority level in the parameter. Usage is:
`nmessages = ioctl(pqfd, PQDEV_IOCHNUMMSG, plevel);`
If `plevel` is given as -1, total number of messages in all levels is returned.

`PQDEV_IOCTSETMAXMSGS` sets the maximum number of messages of the queue (`maxmsgnum` module parameter). If current number of messages is larger, it will not allow writes until it gets below this value. Usage is:
`ioctl(pqfd, PQDEV_IOCTSETMAXMSGS, 1000000);`

`PQDEV_IOCGMINMSG` gets the minimum priority message in the priority queue without an actual read. This call will get first 100 bytes of message (including priority) without deleting the message from queue. Caller should provide a buffer with length at least 100 bytes. If message is longer, it is truncated, if shorter, only message length bytes are filled. It returns the length of actual message. Usage is:
`char buf[100]; mlen = ioctl(pqfd, PQDEV_IOCGMINMSG, buf);`

3 Bonus

There are two bonuses in the project.

1. Efficient implementation: The simplest implementation is a sorted linked list of priorities and each priority level has a linked list, FIFO, of messages. When last message is deleted, priority level is deleted. When number of distinct priority levels is high, this implementation drops systems performance. If you implement priority queue in a heap structure, a bonus of 30 points will be given. Note that recursion is not allowed in kernel.

2. Implementation of priority based reads: By default, read always read first message with the smallest priority value. You can add a `ioctl` operation `PQDEV_IOCTSETREAPRIO` that will change this behaviour temporarily.
   `ioctl(pqfd, PQDEV_IOCTSETREAPRIO, pval)`

   If `pval == 0`, default read behaviour is followed. If `pval > 0`, only a fixed priority level is read. If no message in `pval`, read will block even if there are messages in other priority levels. If `pval < 0`, the smallest priority level which is greater than absolute value of pval is read. If no such message in $\geq$`abs(pval)`, read will block even if there are messages in other priority levels.

   Implementing this ioctl operation will be rewarded as 20 bonus points.

Please specify in a `README` file in your submission which bonuses you implemented.

4 Submission and External libraries

You need to submit a 'tar.gz' archive (no zip) containing scull like Makefile and your source. Makefile, a .c file, a .h file, load and unload scripts are sufficient.
Please ask all questions to:
`news://news.ceng.metu.edu.tr:2050/metu.ceng.course.536/`
`https://cow.ceng.metu.edu.tr/News/thread.php?group=metu.ceng.course.536`