Achten Sie besonders darauf, dass keine Race Conditions auftreten können, synchronisieren sie, wo nötig!

Entwickeln Sie ausgehend von der Lösung des Beispiels aus Übungsblatt 6 eine weitere Version des Character Device Treibers mit folgendem geändertem Verhalten:

1) Unterstützung von ioctl (falls noch nicht vorhanden)
2) Unterstützung des O_NDELAY bzw. O_NONBLOCK Flags. (Unter Linux haben die zwei Flags denselben Wert und sind gleichbedeutend. Unter Linux ist O_NDELAY nur ein anderer Name für O_NONBLOCK. Sprich, Sie können in ihrem Code auf O_NDELAY oder auf O_NONBLOCK überprüfen. Überprüfen auf beide Flags ist unter Linux sinnlos.)

ioctl:
Mit ioctl kann

1. abgefragt werden, wie oft das Gerät gerade zum Lesen offen ist,
2. abgefragt werden, ob das Gerät gerade zum Schreiben offen ist,
3. der Inhalt des Geräts gelöscht werden. Andere Prozesse, die das Gerät gerade offen haben sollen sich wie in Write und Read beschrieben (siehe vorige Übung + akt. Änderungen), anschließend verhalten.

read:
Wenn genügend viele Bytes zum Lesen vorliegen, ist das Leseverhalten unverändert. Sprich, der Aufrufer erhält seine gewünschte Anzahl an Bytes.
Stehen zu wenige Bytes im Puffer des Geräts soll sich das Gerät wie folgt verhalten:

1. Device **ohne** O_NDELAY geöffnet → warten/blockieren erlaubt:
   Falls jemand das Device zum Schreiben offen hält, wird gewartet bis genügend Daten von diesem geschrieben wurden, um die restlichen Daten zu lesen.
   Erreicht der Leser das Ende des Puffers, wird nicht weiter gewartet, sondern es werden die gelesenen Bytes bis zum Pufferende zurückgegeben.
   Hat niemand das Device zum Schreiben offen, wird nicht gewartet.
   Verwenden Sie für diese Synchronisation eine **WAITQUEUE**!
2. Device **mit** O_NDELAY geöffnet → **kein** warten/blockieren erlaubt:
   Das Device soll sich unverändert wie in Übung 6 verhalten. Sprich, es werden die restlich verfügbaren Bytes ausgelesen.

write:
Ist genügend freier Platz im Puffer vorhanden, ist das Schreibverhalten unverändert. Das bedeutet, alle Bytes werden geschrieben. Erreicht der Schreiber das Pufferende soll sich das Gerät wie folgt verhalten:

1. Device **ohne** O_NDELAY geöffnet → warten/blockieren erlaubt:
   Der Schreiber wartet (unabhängig wie viele das Device offen halten) bis jemand den Inhalt per ioctl löscht und der Schreiber somit am Beginn seine restlichen Bytes weiterschreiben kann. Erreicht der Schreiber erneut das Ende des Puffers, muss sich der Vorgang wiederholen, bis der Schreiber alle Bytes geschrieben hat.
2. Device **mit** O_NDELAY geöffnet → **kein** warten/blockieren erlaubt:
   Unverändert zur vorigen Übung.
   Schreiber schreibt die mögliche Anzahl an Bytes und returniert die Anzahl (bzw. falls 0 Bytes geschrieben wurden evt. einen Error Code).

Testprogramm: Erweitern Sie ihr Testprogramm, um das neue Verhalten des Devices testen zu können.

Tipp: Testen mit der Shell/Bash (Damit Sie auch ohne Testprogramm das Gerät offenhalten können):

```
exec 7< /dev/mydevX       # Oeffnen zum Lesen mit fd 7. > fuer Schreiben und <> fuer Lesen und Schreiben
head -c Byteanzahl <&7    # Lesen von Byteanzahl Bytes
echo asdf >&7             # Schreibe asdf in offenen Filedescriptor 7
exec 7<&-                 # Schliessen des Filedescriptors 7 bzw. exec 7>&-
```

Zu entwickeln und abzugeben sind:
Der gut dokumentierte Kerneltreiber samt Makefile und Shellskripts zum Laden und Entladen des Moduls und das Testprogramm.