|  |  |
| --- | --- |
| 14.Januar 2014  Betriebssysteme Praktikum 4 Gruppe 2 Gerätetreiber | Labor Protokoll  Dies ist das Protokoll zum vierten Laborversuch und umfasst eine kurze Beschreibung der Aufgabe, sowie eine Darstellung des Entwurfs und Diskussion des Ergebnisses.  Steffen Giersch & Maria Lüdemann  HAW Hamburg |

Inhaltsverzeichnis

[Das Problem der Speisenden Philosophen 2](#_Toc377461910)

[Aufgabenstellung 2](#_Toc377461911)

[Der Entwurf 2](#_Toc377461912)

[Anmerkung-Nachtrag 6](#_Toc377461913)

# Das Problem der Speisenden Philosophen

### Aufgabenstellung

Schreiben sie einen Gerätetreiber

Das Treibermodul implementiert zwei MinorDevices welche dazu genutzt werden um User Eingaben zu decodieren oder encodieren. Dies kann mittels einer eigenen Erstezungstabelle geschehen der allerdings einigen Regeln unterliegt. Das Modul muss gleichzeitig sicherstellen, dass jeweils nur ein Prozess Lese oder Schreibrechte hat und nicht mehr Rechte heraus gegeben werden können.

### Der Entwurf

Steffen Giersch, Maria Luedemann

Betriebssysteme Praktikum Nummer 4 - Pseudocode

function translate\_read (file \*filp, char \_\_user \* buf, size\_t count, loff\_t \* f\_pos)

speichere Translate-Informationen aus dem Aufruf in dev

belege den Semaphoren dev->sem

solange der Buffer leer ist

gib dev->sem frei

reihe dich in dev->queue ein und warte auf Elemente im Buffer

belege den Semaphoren dev->sem wieder

wenn der WritePointer hinter ReadPointer steht

setze count auf das Minimum der Anzahl der Elemente zwischen dev->wp und dev->rp und den an angeforderten Elementen

sonst ist ein Wrap-Around beim ReadPointer geschehen, also

setze count auf das Minimum der Anzahl der Elemente zwischen dem WritePointer und dem Ende des Buffers und den angeforderten Elementen

wenn translate1 aufgerufen wurde

decodiere count Elemente im Buffer ab dem ReadPointer und kopiere sie zum Benutzer

sonst

kopiere count Elemente im Buffer ab dem ReadPointer zum Benutzer

inkrementiere den ReadPointer um count

wenn der ReadPointer am Ende des Buffers angekommen ist

Setze den ReadPointer an den Anfang des Buffers

dekrementiere den fillcount um count

gib dev->sem wieder frei und benachrichtige die wartenen Prozesse in dev->queue

gib count zurueck, damit der lesende Prozess weis, wie viele Elemente tatsaechlich uebertragen werden konnten

end function

function translate\_write (file \*filp, count char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

speichere Translate-Informationen aus dem Aufruf in dev

belege den Semaphoren dev->sem

solange der Buffer voll ist

gib dev->sem frei

reihe dich in dev->queue ein und warte auf freien Platz im Buffer

belege den Semaphoren dev->sem wieder

setze count auf das Minimum des verbleibenen Platzes im Buffer und count

wenn der WritePointer hinter oder auf dem ReadPointer steht

setze Count auf das Minimum von Count und den Elementen bis zum Ende der Liste

sonst ist ein Wrap-Around beim ReadPointer geschehen, also

setze Count auf das Minimum von Count und den Elementen bis zum ReadPointer

wenn translate0 aufgerufen wurde

codiere count Elemente im Buffer ab dem WritePointer und kopiere sie in den Buffer sonst

kopiere count Elemente im Buffer ab dem WritePointer in den Buffer

inkrementiere den WritePointer um count

wenn der WritePointer am Ende des Buffers angekommen ist

Setze den WritePointer an den Anfang des Buffers

inkrementiere den fillcount um count

gib dev->sem wieder frei und benachrichtige die wartenen Prozesse in dev->queue

gib count zurueck, damit der lesende Prozess weis, wie viele Elemente tatsaechlich ouebertragen werden konnten

end function

function translate\_open (inode \*inode, file \*filp)

speicher die Referenz auf das jeweilige MinorDevice und die MinorNumber in

filp->private\_data, bzw dev->minor\_number ab

wenn der oeffnende Prozess lesen will, aber schon ein anderer Prozess auf diesem MinorDevice liest

gib EBUSY zurueck

sonst, wenn kein anderer Prozess liest, aber der oeffnende Prozess lesen will

inkrementiere die Anzahl der lesenden Prozesse um 1

wenn der oeffnende Prozess schreiben will, aber schon ein anderer Prozess auf diesem MinorDevice schreibt

gib EBUSY zurueck

sonst, wenn kein anderer Prozess schreibt, aber der oeffnende Prozess schreiben will

inkrementiere die Anzahl der schreibenden Prozesse um 1

Informiere den Prozess darueber, dass nicht im Buffer gesucht werden kann

end function

function translate\_close (inode \*inode, file \*filp)

wenn der oeffnende Prozess gelesen hat

dekrementiere die Anzahl der lesenden Prozesse

wenn der oeffnende Prozess geschrieben hat

dekrementiere die Anzahl der schreibenden Prozesse

end function

function init\_module

wenn der vom Nutzer eingegebene String translate\_subst zu lang ist

brich mich einem Fehlerwert ab

wenn translate\_subst zu kurz ist

ergaenze translate\_subst bis TRANSLATE\_SUBST\_LENGTH Symbole und die terminierende 0 in darin sind

wenn der zur Verfuegung gestellte Buffer <= 0 ist

brich mit einem Fehlerwert ab

registriere translate als CharDevice und speichere die erhaltene Major-Number

Allokiere Speicher fuer die Minor-Devices und nulle ihn zunaechst aus

fuer jeden erzeugten Speicherblock fuer die MinorDevices

initialisiere den Semaphoren

initialisiere die Queue

fordere Kernel-Speicher fuer den Buffer an

setze die Write-, Read-, BufferAnfangs- und BufferEndPointer

setze den fillcount auf 0

end function

function cleanup\_module

wenn keine Geraete gefunden wurden

brich ab

rufe kfree fuer jeden erzeugten Buffer der Minor-Devices auf

rufe kfree auf die translate\_devices auf

deregistriere das CharDevice

end function

function encode\_char (char c)

wenn c zwischen 'A' und 'Z' liegt

gib translate\_subst[(c - 'A') + Laenge des Alphabets] zurueck

sonst, wenn c zwischen 'a' und 'z' liegt

gib translate\_subst[c - 'a'] zurueck

wenn keiner der Faelle eingetreten ist, gib c unveraendert zurueck

end function

function decode\_char (char c)

wenn c ein Buchstabe ist

speichere den index dieses Buchstaben aus translate\_subst

wenn der Buchstabe in der ersten Haelfte des Substitutionsstrings vorkam

gib 'a' + index zurueck

sonst

gib 'A' + (index - Laenge des Alphabets) zurueck

wenn c kein Buchstabe war gib c unveraendert zurueck

end function

### Anmerkung-Nachtrag

Anzumerken ist, dass der kritische Abschnitt beim Open und Close in dem geprüft wird ob der öffnende oder schließende Prozess liest und anschließend die Anzahl der nReader und nWriter erhöht wird durch eine Semaphore geschützt werden müsste. Dies haben wir leider versäumt aus den Beispielen zu übernehmen.