به نام خدا

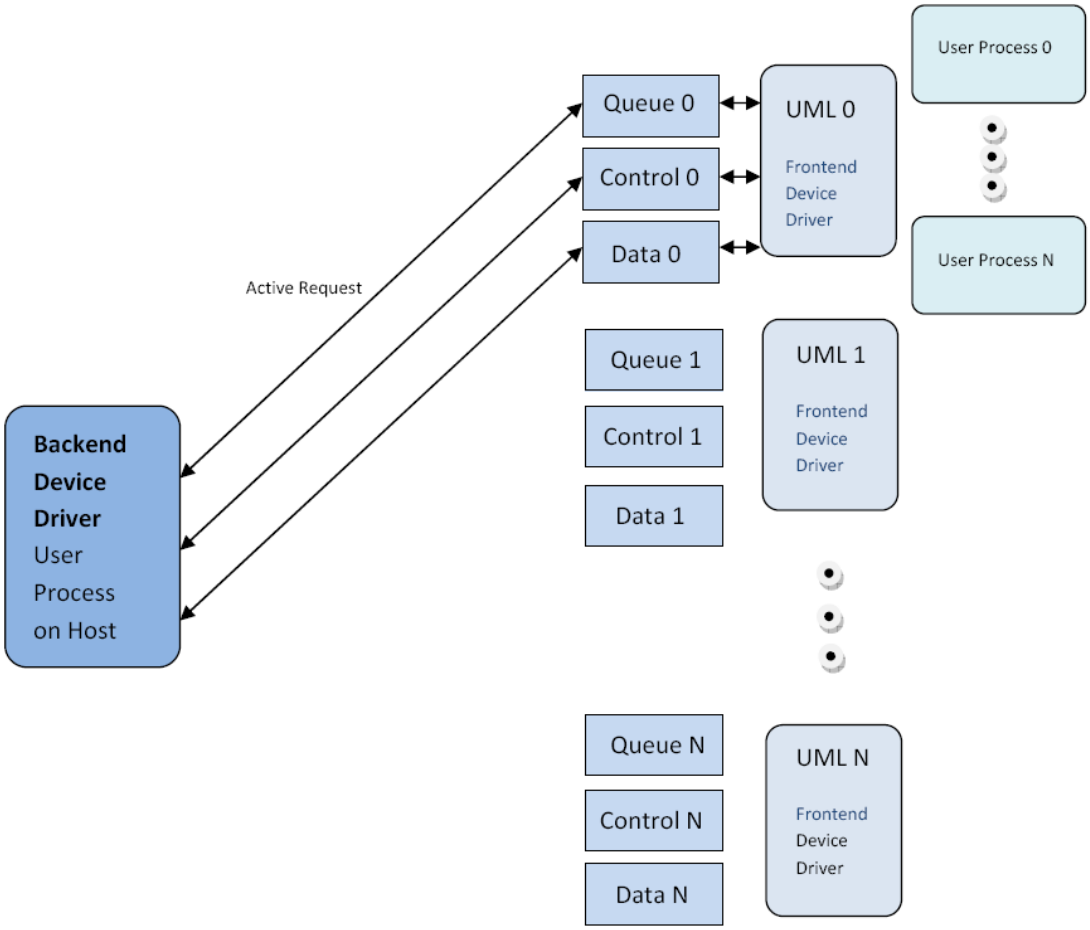
سند پروژه نهایی درس سیستم های توزیع شده

دانشجو:محمد تنهایی ۸۷۲۰۲۴۴۶	tanhaei@ce.sharif.edu
استاد : دکتر رسول جلیلی	jalili@sharif.ir
دستیار استاد : مهندس محروقی	Mahrooghi@ce.sharif.edu

هدف پروژه : پیاده سازی یک مکانیزم اشتراک داده ها بین چند سیستم ، با استفاده از UML و در فضای کرنل

طراحی سیستم : سیستمی که ما طراحی کرده ایم دارای شمایی مانند شکل زیر است :

(توجه : عکس از سایت دانشگاه جورجیا برداشته شده است ، و با توجه به مناسب بودن در این سند قرار داده شده است)



طراحی ما به این صورت است که از یک مکانیزم filesharing برای انتقال داده ها استفاده می کنیم ، (در این پروژه با توجه به نکات مطرح شده و اینکه تنها می خواهیم کاراکتر انتقال دهیم اندکی کار را ساده تر کرده ایم)

هر یک از N پردازه که در قسمت کاربر قرار دارند می توانند از Frontend Device Driver کارهای Open ، Read ، ... را انجام دهند ، Frontend Device Driver برای ارتباط با Backend Device Driver چاره ای ندارد جز استفاده از socket برای این منظور از socket های کرنلی استفاده کرده و یک ارتباط بین Frontend Device Driver و Backend Device Driver ایجاد می کنیم ، هر

Tanhaei@ce.sharif.edu

کدام از Frontend Device Driver ها صفی از درخواست ها از کاربر دارند که برای Backend ارسال مـی کننـد و یـک سـری کارهـای کنترلی را انجام می دهند ، (در اینجا تنها کار کنترلی ما کنترل خواندن بافر است) ، از طرفی باید از طریق ارتباط ایجاد شده داده ها را منتقل کرد ، در این پروژه ما داده کاراکتری Tanhaei را به صورت stream ای انتقال می دهیم .

در قسمت سرور یک مکانیزم polling حکم فرماست و به درخواست های Frontend Device Driver پاسخ مناسـب را مـی دهـد ، از آن جمله فرستادن داده ، بستن اتصال و ... است .

با توجه به اشتراک یک بافر کاراکتری پروژه ما بسیار راحت می شود ، در صورتی که بخواهیم یک درایور برای دوربین مجازی تهیه کنـیم بایـد دسـتوراتی ماننـد scale کـه گفتـه شـده را پیـاده سـازی کنـیم ، ایـن دسـتور بـه وسـیله کانـال socket کـه سـاخته ایـم بـه صـورت <imageID,hight,width> فرستاده شده و در جواب تصویر scale شده دریافت می شود ، با توجه به ساختار طراحی شده افزودن چنـین امکانی بسیار ساده است .

اما با توجه به نکات گفته شده در داخل کلاس ، در حال حاضر سرور برای درخواست های Frontend Device Driver یک سری داده می فرستد و از این طریق می توان یک مکانیزم اشتراک پیاده سازی کرد .

پیاده سازی :

گام اول : آشنایی با UML ، برای کار با UML کافی است که یک filesystem و یک کرنل UML دانلود کرده و به صورت زیر اجرا کنیم :

```
host% ./linux-2.6.24-rc7 ubda=FedoraCore5-x86-root_fs mem=128M
```

چون که در این پروژه نیاز به شبکه هم داریم می توانیم UML ای که backend روی آن اجرا می شود را به صورت زیر به یک ip متصل کنیم :

۱. اجرای UML
۲. Mount کردن شاخه حاوی slrip
۳. کامپایل slrip در UML که باعث ساخته شدن یک فایل اجرایی می شود.
۴. اجرای UML با اضافه کردن دستور option eth0=slrip,,<path of the slrip exe>

همچنین می توانیم از plugin هایی که توسط UML utility تهیه می شود استفاده کرد.

[به شخصه ترجیح می دهم که به جای این کار نسخه backend را روی لینوکس اصلی و Client ها را روی UML اجرا کنیم]

گام دوم : پیاده کردن یک Device Driver در لینوکس :

برای پیاده کردن یک Device Driver در لینوکس چند کار اساسی باید انجام داد ، به صورت خیلی خلاصه کارهای زیر را انجام می دهیم :

۱. با توابع module_init(Client_init_module) و module_exit(Client_exit_module) توابعی را که به زمان load و یا unload ماجول باید انجام شود را معرفی می کنیم .
۲. یکی از مهمترین کارهای تابع init اجرای دستور زیر است : register_chrdev(0, DEVICE_NAME, &fops) که نام device و تابع های آن را ثبت نام می کند ، و یک عدد بر می گرداند که به Major معروف است ، این عدد شماره ID درایور ماست ، و نباید آنرا اشتباه گرفت .
۳. در exit باید دستور unregister_chrdev(Major, DEVICE_NAME) را اجرا کرد که برعکس بالا عمل می کند.
۴. یک ساختار داده هر کدام از فراخوانی های سیستم را روی فایل درایور به یک تابع invoke می کند :

```
static struct file_operations fops = { .read = device_read,
                .write = device_write, .open = device_open, .release = device_release ;}
```

گام سوم : نوشتن یک socket برای ارتباط :

برای پیاده کردن Network Socket Program در مد کرنل باید از مجموعه دستورات زیر استفاده کنیم :

```
sock_create(...)
```


Tanhaei@ce.sharif.edu

```
socket->ops->connect(...)

sock->ops->bind(...)

sock->ops->listen(...)

sock_recvmsg(..)

sock_sendmsg(...)
```

این قسمت یکی از سخت ترین و کلافه کننده ترین بخش های پیاده سازی است!! و با توجه به پیچیدگی سعی شد که از کدهای کرنلی کلاس های کرنل و غیره با مقداری تغییرات استفاده شود ، یکی از مشکلات کار ، connect شدن به یک socket بود کـه در حالـت کرنلـی بسـیار محدود شده و منجر به خطاهای متعدد NULL references در محیط می شد ، با توجه به اینکه UML به سخت افزار خاصی متصل نیست این آزمایشات ضربه ای به سخت افزار سیستم وارد نکرد! و در نهایت راهی برای ارتباط به دست آمد که در کد وجود دارد.

گام چهارم: کامپایل کردن برنامه ها :

همان طور که شاید بدانید برای کامپایل برنامه های کرنلی باید حتماً یک Makefile تدارک ببینیم ، ما این کار را انجـام داده و بـرای هـر ۲ درایور یک Makefile ساخته ایم ، برای کامپایل برنامه ها کافی است که در Shell در مکان برنامه ها بنویسیم : make

اجرای پروژه :

پس از طی کردن گام های فوق آماده ایم که برنامه را تست کنیم برای این منظور چند کار باید انجام شود : (ابتدا مطمئن شوید کـه در مـود root هستید ، دستور su را بزنید و رمز خود را وارد کنید)

- **برای اجرای Backend Driver :**
  - ابتدا شاخه حاوی Backend Driver را Mount می کنیم.
  - بعد از کامپایل کردن درایور در محیط UML و یا لینوکس اصلی :
  - با دستور insmod tanhaei-server.ko ماجول درایور را در کرنل load می کنیم.
  - هر موقع که خواستیم می توانیم با دستورrmmod tanhaei-server ماجول را از کرنل unload کنیم.

- **برای اجرای Frontend Device Driver :**
  - شاخه حاوی Frontend Device Driver را Mount می کنیم .
  - آدرس ip سرور را در خط ۲۵۱ ، server.sin_addr.s_addr = htonl(create_address(127,0,0,1)) فایـل tanhaei-dev.c وارد می کنیم ، به صورت پیش فرض از localhost استفاده می کنیم ، اما بـرای تسـت روی UML باید آدرس backend Driver ip را وارد کرد.
  - کامپایل کردن درایور در محیط UML (با دستور make)
  - با دستور insmod tanhaei-dev.ko ماجول درایور را در کرنل load می کنیم.
  - حال باید نگاهی کوتاه به فایل /var/log/messages داشته باشیم ، با دستور tail –f /var/log/messages ایـن فایل را دنبال می کنیم ، در این فایل پیغام های Kernel_INFO که در داخل برنامه می نویسیم نوشته مـی شـود : بـه چنین چیزی بر می خوریم (بعد از load کردن ماجول) :

```
I was assigned major number 247. To talk to
the driver, create a dev file with
mknod /dev/tanhaei-dev c 247 0
....
```

همان طور کـه در اینجـا هـم پیشـنهاد داده ام ، مـی توانیـد از دسـتور mknod /dev/tanhaei-dev c major 0 استفاده کنید ، این دستور یک فایل برای ارتباط با درایور در شاخه /dev می سازد.

  - با استفاده از دستور cat /dev/tanhei-dev فایل درایور را بخوانید ، یک جریان stream می بینید که مـدام کلمـه Tanhaei را به کاربر می دهد (کاربر مدام داده ها را دریافت می کند) (عمدا برنامه طوری نوشته شده است کـه مـدام داده ها را بخواند ، زیرا می خواستیم stream بودن دریافت را نمایش دهیم) ، به عبارت بهتر backend مدام داده ها را به صورت polling برای کاربرانی که درخواست کرده اند ارسال می کند.

- برای تست تقلبی نبودن سرور می توان با telnet به پورت 5060 متصل شد ، می بینیم که همین داده دریافت می شود.(البته تنها یک بار ، زیرا فقط یک بار درخواست دریافت داده ایم و مانند حالت قبل مدام در حال درخواست نیستیم)
- در پایان با دستور rmmod tanhaei-dev و نیز rm /dev/tanhaei-dev به ماجول خاتمه می دهیم.

توجه : به همراه این سند یک فایل zip نیز ارسال شده است ، فایل حاوی شاخه های **back-end** و **frond-end** است ، این شاخه ها هر کدام یک فایل Makefile و یک فایل c دارند ، برای کامپایل می توانید از دستور make استفاده کنید. یک فایل install.sh نیز وجود دارد که برای خالی نبودن عریضه و همچنین نصب همیشگی ماجول ها روی کرنل می توانید از آن استفاده کنید .

بدیهی است : کد به صورت کامل **مستند سازی** شده و جاهایی که کمبودی حس می شود در داخل این سند کمبودها جبران شده است ، سعی بر آن بوده که بهترین کد و واضح ترین شیوه انتخاب شود ، هر چند که پیچیدگی کار با Kernel و C بودن کد ها فهم آنرا اندکی دشوار می نمایاند.

و من الله توفیق

محمد تنهایی ۲۷ دی ماه ۱۳۸۷

Tanhaei@ce.sharif.edu