

به نام خدا

گزارش پروژه سیستم های عامل

اعضای تیم:

پوریا کرمی: 400522121

محمدرضا تاجیک: 400521198

هدف پروژه ایجاد دو سیستم کال جدید به نام های join و clone است. برای این منظور اقدامات زیر صورت گرفته است:

- **syscall.h**
سیستم کال های join و clone را به این فایل اضافه کرده و به آنها یک عدد یکتا اختصاص داده ایم.
- **syscall.c**
در این فایل فرمت سیستم کال های join و clone را تعریف کرده و آنها را به لیست سیستم کال ها اضافه می کنیم.
- **sysproc.c**
در این فایل محتوای سیستم کال های join و clone را پیاده سازی می کنیم.
در این فایل با استفاده از argint ورودی های پاس داده شده به توابع را دریافت می کنیم.
- **usys.s**
برای برقراری دسترسی کاربر به سیستم کال های جدید ما باید آنها را با فرمت `SYSCALL(system_call_name)` به این فایل اضافه کنیم.
- **Begin.sh**
در این فایل یک اسکریپت ساده جهت سهولت در اجرای برنامه نوشته شده است؛ دقت کنید ممکن است نیاز باشد با دستور `chmod +x ./begin.sh` به این فایل اجازه ی اجرا شدن داده شود.

- user.h

به منظور دسترسی کاربر به سیستم کال های ما علاوه بر افزودن آنها در `usys.s` باید آنها را در این فایل نیز اضافه کنیم؛ همچنین توابعی که قرار است به عنوان کتابخانه های جدید مورد استفاده قرار بگیرد و ما آنها را پیاده سازی خواهیم کرد را نیز باید به این فایل اضافه کرد.

این توابع عبارت اند از:

- `thread_create`
- `thread_join`
- `lock_init`
- `lock_acquire`
- `lock_release`

- برای استفاده از مفهوم `lock` و استفاده از آن در فایل `user.h` نیازمند تعریف یک `struct` با نام `lock_t` هستیم که با استفاده از متغیر `flag` عمل `lock` کردن را انجام می دهد.

- ulib.c

پیاده سازی توابعی که برای کتابخانه ی جدید در نظر گرفتیم و در بالا لیست شده اند باید در این فایل قرار بگیرند.

- proc.c

در این فایل سیستم کال های `join` و `clone` را پیاده سازی می کنیم. محتوی این دو سیستم کال به شرح زیر است:

- `clone`

در این تابع با استفاده از اطلاعات داده شده توسط `stack` یک `process` جدید ساخته می شود.

- `join`

این تابع در `process table` به دنبال `process` های فرزندی می گردد که به حالت `zombie` (`process` ای که `parent` آن مرده باشد ولی خود آن زنده باشد) را `zombie` می نامند) درآمده اند تا آنها را از جدول پاک کند.

- **proc.h**

در این فایل یک struct به نام `proc_info` ایجاد می کنیم که شامل دو عدد از نوع `int` است که یکی بیانگر `process id (pid)` و دیگری بیانگر `memory size (memsize)` است؛ همچنین در این فایل به struct `proc` یک property جدید به نام `threadstack` اضافه می کنیم که در آن آدرس `thread stack` را نگه می داریم همچنین یک property دیگر به نام `vruntime` به آن اضافه می کنیم که در قسمت ایجاد `fair share scheduler` از آن به عنوان سنج ای برای تعداد دفعات اشغال CPU توسط `process` استفاده می شود.

- **defs.h**

توابع `join` و `clone` پیاده سازی شده در فایل `proc.c` را در اینجا تعریف می کنیم.

- **Makefile**

فایل تست را که در ادامه توضیح خواهیم داد به `Makefile` اضافه می کنیم که در هنگام صدا زدن دستور `make qemu-nox` این فایل را هم به همراه باقی فایل ها `compile` شود.

- **Testing**

برای تست کردن سیستم کال های `join` و `clone` که به شرح بالا پیاده سازی شده اند به این شکل عمل کرده ایم:

در فایل `test_threads.c` کد مربوط به ایجاد سه `thread` با استفاده از سیستم کال `clone` بر روی سه فانکشن مجزا که هر کدام به متغیر `counter` که در ابتدا برابر صفر است یک واحد اضافه می کنند و سپس مقدار `counter` را با ذکر شماره `thread` در خروجی چاپ می کنند پیاده سازی شده است. (طبیعتاً بعد از سه مرتبه صدا زدن سیستم کال `clone` سه مرتبه نیز سیستم کال `join` صدا زده می شود).

از آنجایی که ما از `lock` و فانکشن های تعریفی خودمان به منظور باقی ماندن در ناحیه امن (thread safety) استفاده کرده ایم مطمئن هستیم که `counter` در پایان عملیات مقدار 3 را خواهد داشت.

خروجی اجرای تست:

```
SeaBIOS (version 1.15.0-1)
```

```
ipXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00
```

```
Booting from Hard Disk..xv6...
```

```
cpu0: starting 0
```

```
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
```

```
init: starting sh
```

```
$ test_threads
```





















```
1. First thread : 1
```

```
2. Second Thread: 2
```

```
3. Third Thread: 3
```

```
$
```

:Git graph

Graph	Description	Date	Author	Commit
	 documentation  origin documentation	31 Jan 2024 01:32	pourya22334415	f6931a30
	 main  origin  origin/HEAD Merge pull request #5 from pourya22334415/add-commit	31 Jan 2024 04:58	pourya	206afb1c
	 origin/add-commit add-commit	31 Jan 2024 01:28	pourya22334415	d5c240a1
	Merge pull request #4 from pourya22334415/update-code	30 Jan 2024 20:29	pourya	5dae91a2
	 origin/update-code update code	30 Jan 2024 20:26	pourya22334415	6a2e600d
	Merge pull request #3 from pourya22334415/implement-testing	30 Jan 2024 10:25	MohammadReza Ta...	a1962577
	 origin/implement-threading Merge pull request #2 from pourya22334415/implement-testing	30 Jan 2024 10:25	MohammadReza Ta...	746ff72c
	 origin/implement-testing add and edit some files for testing project	30 Jan 2024 10:22	MRezaT	0a0a6b07
	Merge pull request #1 from pourya22334415/implement-threading	29 Jan 2024 22:16	pourya	11885607
	implement threading	29 Jan 2024 22:13	pourya22334415	e3e08253
	first-push	25 Jan 2024 15:23	pourya22334415	abb219