
# موضوع: داک پروژه OS

**دانشجویان: علی شکوهی، امیرحسین راعقی**

**شماره دانشجویی:**

**400521477**

**400522373**

## پیاده سازی clone

پیاده‌سازی سیستم کال clone که شباهت زیادی به سیستم کال fork دارد، ابتدا باید فایل proc.c تغییر داده شود و این سیستم کال به آن اضافه شود. سپس همانطور که در داک پروژه گفته شده، به صورت clone(void(*worker)(void*,void*), void *arg1, void *arg2, void* stack) پیاده‌سازی می‌شود و در مرحله بعد آرگیومنت‌ها داخل استک ریخته می‌شوند. سپس آدرس استک داخل استک پوینتر ذخیره می‌شود. سپس یک استک مربوط به آن ترد ساخته می‌شود ( threadstack که داخل proc.h تعریف شده است) ومقدار استک در این استک مربوط به ترد ریخته می‌شود.

برای پیاده‌سازی کامل clone، باید سیستم کال مربوط به آن در فایل‌های زیر اضافه شود:

در فایل defs.h، در بخش مربوط به سیستم کال‌ها، آن را اضافه می‌کنیم.

در فایل syscall.c نیز باید در دو قسمت این سیستم کال تعریف شود (مانند سایر سیستم کال‌ها تعریف می‌شود).

در فایل syscall.h نیز باید همانند سایر سیستم کال‌ها آن را define کنیم.

در فایل sysproc.c مانند سایر سیستم کال‌ها، یک تابع برای سیستم کال clone تعریف می‌شود و در صورت برقراری شروط، با موفقیت تابع clone فراخوانی می‌شود. در غیر اینصورت -1 برمی‌گرداند.

در فایل ulib.c همانطور که در داک گفته شده، تابع thread_create ایجاد می‌شود که نقش wrapper را برای سیستم کال clone دارد.

در فایل user.h، سیستم کال clone و تابع thread_create ایجاد می‌شود.

همچنین در فایل usys.S نیز سیستم کال clone تعریف می‌شود.

## پیاده‌سازی join

در ابتدا ما در بخش **proc.c** سیستم کال join را تعریف می‌کنیم به این صورت که این سیستم کال به طور کلی منتظر می‌ماند تا Thread ما به اتمام برسد و در انتها وضعیت خروجی آن را دریافت می‌کنیم و با کمک آن به سنکرون کردن Thread ها می پردازیم.

به طور کلی در این قسمت ابتدا نگاه میکنیم تا آیا لاک یا قفل آزاد هست یا خیر در صورت آزاد بودن همه پروسس های موجود نگاه می‌کنیم و به دنبال یک ترد زامبی(تردی که کارش اتمام یافته) می‌گردیم سپس اگر ترد فرزند زامبی ما با ترد

اصلی برابر باشد و در یک صفحه قرار داشته باشد => شروع به پاکسازی می‌کنیم و منابع اشغال شده را آزاد می‌کند(استک و ...).و در انتها شماره ترد را به ما پس می‌دهد (۰ در صورت موقیت). و اگر ترد فعلی ما فرزندی نداشت یا ترد حذف شود(-۱ بر میگردانیم.) و مشخص می‌شود که شکست داشته ایم. حال در این پیاده‌سازی مطمین می‌شویم که توالی ترد ها حفظ شده است. این دستور شبیه سیستم کال wait است. ولی اگر قفل بود باید منتظر بمانیم تا ترد قبلی کارش به اتمام برسد در نتیجه در این موقعیت در ترد هایمان در critical section در لحظه فقط یک ترد حضور داشته باشد.

حال در بخش **sysproc.c** حال در این بخش Sys_join را تعریف می‌کنیم تا به کمک آن بتوانیم سیستم کال Join را در زمان ایجاد یک ترد(با قاعده clone) رسیدگی کنیم.

این بخش یک رابط بین برنامه‌های سطح کاربر و تابع join ما هست. داده‌های استک را می‌گیرد و سپس تابع join را فراخوانی می‌کند و داده‌های استک پوینتر را به عنوان ورودی می دهیم.

موارد تغییر داده شده:

آرگومان های ارسال شده به sys-join را دریافت کرده که یک عدد int دریافت می‌کنیم این عدد آدرس استک ما است که در زمان ساخت ترد ایجاد شده است. حال با کمک پوینتر به آن آدرس می‌رویم و داده حقیقی در استک که توسط ترد ساخته شده است را می‌بینیم و تابع join بررسی می‌کند اتمام آن ترد را و مقدار آن را بر می گرداند.

در این مرحله بعد به سراغ **Ulib.c** که برای برنامه‌های سطح کاربر هست می‌پردازیم.

یک تابع برای وصل کردن ترد و پوینتر استک آن به نام Join تعریف می‌کنیم. به صورت کلی در این بخش یک تابع تعریف می‌کنیم به نام join که این یک تابع wrapper است. که در آن join را با آدرس stackpointer می‌دهیم و در انتها مقدار بازگشتی جوین را ذخیره می کنیم.

در ادامه قفل ها را نیز تعریف می‌کنیم. تا دربخش سیستم کال join استفاده کنیم به این صورت که ۳ بخش داریم:

۱- در ابتدا خود قفل را تعریف می‌کنیم به این صورت که ما flag داریم که در ابتدای ساخت ۰ می‌گذاریم به معنای خالی بودن و اینکه از critical section ما می‌توانیم استفاده کنیم.

۲-گرفتن قفل را تعریف می‌کنیم به این صورت که اگر قفل در دست کسی نبـود مـا از آن استفاده می‌کنیم. یعنی در مرحله قبل باید ۰ باشد تا ما بتوانیم قفل را بگیریم.

۳-آزاد سازی قفل: در این بخش اگر کار ما با قفل تمام شد از آن خـارج شـده و پرچم خود را ۰ می‌کنیم تا ترد های دیگر نیز بتوانند وارد CS شوند.

مراحل اجرای کد به صورت زیر است:

1) هنگـامی کـه یـک رشـته «thread_join» را فراخـوانی می‌کنـد، آدرس متغـیر « stackPointer» را به تابع «join» می‌فرستد.

2) تابع 'join' قفل جدول را می گیرد و شروع به بررسی جدول فرآیند برای یافتن ترد فرزند زامبی می کند.

3) اگر یک ترد فرزند زامبی پیدا شود، کارهای پاکسازی لازم انجام می شود.

4) قفل جدول آزاد می شود و شناسه فرآیند فرزند متصل شده به تابع 'join' بـرمی گردد.

5) تابع 'join' شناسه فرآیند را به تابع 'thread_join' برمی گرداند، که سپس شناسه فرآیند را برمی گرداند.

## ایجاد فایل تست و درست کردن makefile

فایل test_thread را برای تست پیاده‌سازی می‌کنیم که در آن ۳ تابع ایجاد شده است که در آن ۳ ترد با استفاده از thread_create ساخته می‌شود و هر ترد هر فانکشن را ایجاد می‌کند و در هر ترد یک sleep و یک پرینت وجود دارد که باید به ترتیب اجرا شوند و در آخر نیز آن‌ها را با استفاده از thread_join جوین می‌کنیم.

`در makefile فعلی یک تست برای fork نوشته شده است که نیازی به آن نداریم و می‌توانیم آن را حذف کنیم. سپس فایل test_thread را به آن اضافه میکنیم و در نهایت با کامند make  qemu-nox آن را اجرا کنیم و پس از آن test_thread را اجرا می‌کنیم که به درستی اجرا می‌شود.