SystemCall:

یک تماس سیستمی یا "system call" روشی برای برنامه‌ها برای تعامل با هسته سیستم‌عامل است. زمانی که یک برنامه کامپیوتری درخواستی به هسته سیستم‌عامل می‌دهد، یک تماس سیستمی انجام می‌دهد. تماس‌های سیستمی برای ارائه خدمات سخت‌افزاری، ایجاد یا اجرای پردازه‌ها و برای ارتباط با خدمات هسته از جمله برنامه‌ریزی برنامه‌ها و پردازه‌ها استفاده می‌شوند.

مراحل عمومی افزودن یک systemcall جدید به سیستم عامل xv6 به شرح زیر است:

1. تعریف رابط تماس سیستمی: گام اول تعریف رابط تماس سیستمی جدید است، شامل نام، آرگومان‌ها و مقدار بازگشتی تماس سیستمی. این اطلاعات باید به فایل syscall.h اضافه شود.

2. اختصاص یک شماره تماس سیستمی: در مرحله بعدی، یک شماره یکتا به تماس سیستمی جدید اختصاص داده می‌شود. این شماره باید به فایل syscall.h و همچنین به تابع syscall() در فایل syscall.c اضافه شود.

3. اصلاح دهنده تماس‌های سیستمی: اصلاح دهنده تماس‌های سیستمی مسئول تعیین اجرای کدام تماس سیستمی براساس شماره تماس سیستمی ارسال شده توسط برنامه کاربری است. این مکانیزم باید برای مدیریت شماره تماس سیستمی جدید به‌روزرسانی شود.

4. پیاده‌سازی تماس سیستمی جدید: مرحله نهایی پیاده‌سازی تماس سیستمی جدید در کد هسته است. این شامل اصلاح تابع هسته مناسب برای پردازش تماس سیستمی جدید و اضافه کردن هر نوع ساختارهای داده یا توابع کمکی مورد نیاز می‌شود.

5.پس از انجام این مراحل، سیستم عامل xv6 تغییر یافته می‌تواند مجدداً کامپایل و تست شود تا اطمینان حاصل شود که تماس سیستمی جدید به عنوان مورد انتظار کار می‌کند.

ابتدا به توضیح مراحل پیاده سازی دو system call ، join و clone می پردازیم و سپس هر کدام از آن ها را توضیح می دهیم:

1. ابتدا در فایل syscall.h می بایست برای این دو سیستم کال جدید شماره ی جدید را اختصاص بدهیم. برای این کار SYS_clone و SYS_join به ترتیب با مقادیر 22 و 23 تعریف می نماییم.
2. Wrappers این systemcall های جدید را می بایست در فایل های syscall.c و sysproc.c پیاده سازی نماییم.
3. در فایل usys.S می بایست برای دسترسی user به آن ها می بایست این توابع را نیز تعریف نماییم.
4. در فایل user.h می بایست این توابع را در قسمت syscall همراه با وروردی آن ها تعریف نماییم.

5. در پیاده سازی wrappers ها در syscall.c می بایست توجه داشت که برای پاس دادن آن می بایست از argint استفاده نماییم تا در sysproc.c این سیستم کال ها صدا زده شوند و دو سیستم کال جدید clone و join که در proc.c پیاده سازی شده اند، اجرا شوند.

6. عملکرد clone به این گونه می باشد، با استفاده از stack داده شده به آن که در حقیقت به اندازی pagetable (4096 KB) یک ترد جدید ساخته می شود. می بایستی توجه داشت که در فایل proc.h می بایست به struct ، proc یک ویژگی جدید اضافه نماییم که بتوانیم در آن آدرس stack را دخیره نماییم، که نام آن را threadstack قرار میدهیم.

7. عملکرد join به این گونه می باشد که با دریافت id یک ترد در صورتی که این id به حالت zombie درآمده باشد، منابع اختصاص داده به آن را از آن می گیرد.
در ادامه به توضیح کامل این دو تابع می پردازیم.

8. می بایست، توجه داشت که این دو تابع و توابع کمکی آن در defs.h نیز تعریف شوند.

9. برای این که این توابع بتوانند توسط user کال شوند، می بایست در فایل ulib.c دو تابع را برای آن تعریف کنیم، این توابع به ترتیب با نام های thread_creat و thread_join و آرگمان ها لازم آن تعریف بشوند .

10. در نهایت برای تست این دو تابع یک فایل به نام test_threads.c اضافه می نماییم. که در آخر این فایل تست را به makefile اضافه می نماییم. که با استفاده از thread_create یک thread ایجاد می کنیم. هر کدام از این thread ها این وظیفه را دارند که یک counter را که مقدار اولیه را به اندازه یک واحد افزایش دهند. این عملیات برای thread آخر که اجرا می شود، عدد 10 را در خروجی نمایش دهد. این به این دلیل است که synchronization آن به درستی صورت می گیرد. در ادامه به تابع هایی که برای synchronization درست برای این systemcall اضافه کرده ایم، می پردازیم.

حال به توضیح بخش synchronization می پردازیم :

در این بخش برای synchronization درست از سه عدد wrapper استفاده می کنیم، که شباهت بسیار زیادی به قسمت به تابع های موجود در spinlock.c دارد. در این قسمت می بایست سه Wrapper به اسم های lock_init, lock_acquire, lock_release تعریف میکنیم.

این توابع نیز باید در defs.h تعریف شوند. این توابع در قسمت ulib.c پیاده سازی شده اند.

lock_init برای قسمتی می باشد، که lock را initialize می کنیم.

lock_acquire برای قسمتی می باشد که نشان می دهد، lock توسط ترد مورد نظر گرفته شده است و از ورود سایر ترد ها به بخش بحرانی جلوگیری می نماید.

lock_release برای این است که پس از این که کار ترد تمام شد، اجازه ورود سایز توابع را به نقطه بحرانی بدهد.

حال در این بخش به توضیح این که چگونه مقدار stack را برای ترد setup کرده ایم می

پردازیم :

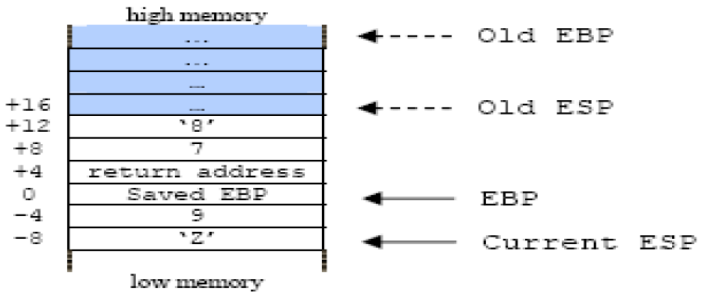
در ابتدا در این قسمت به توضح پوینتر های موجود برای stack و به شرح دهی این که چگونه این مقادیر update می شوند می پردازیم.

ما در xv6 دو پوینتر برای stack داریم که نام های آن به ترتیب عبارت است از ESP و EBP .

ESP: نشانگر اشاره‌گر پشته است که با تخصیص متغیرهای جدید در تابع افزایش می‌یابد.

EBP: به عنوان اشاره‌گر پایه نامیده می‌شود که نمایانگر آغاز پشته است.

که شکل آن به صورت زیر می باشد :



function calls and stackpointer

حال در قسمت زیر به چگونگی مقداردهی سایز stack در fork را نشان می دهد.

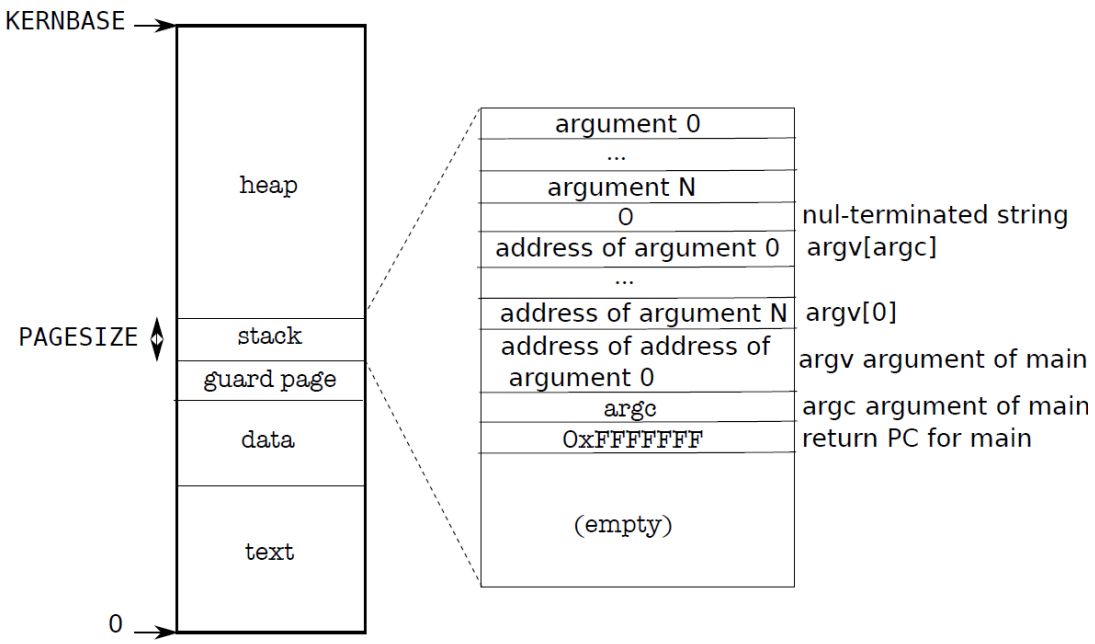
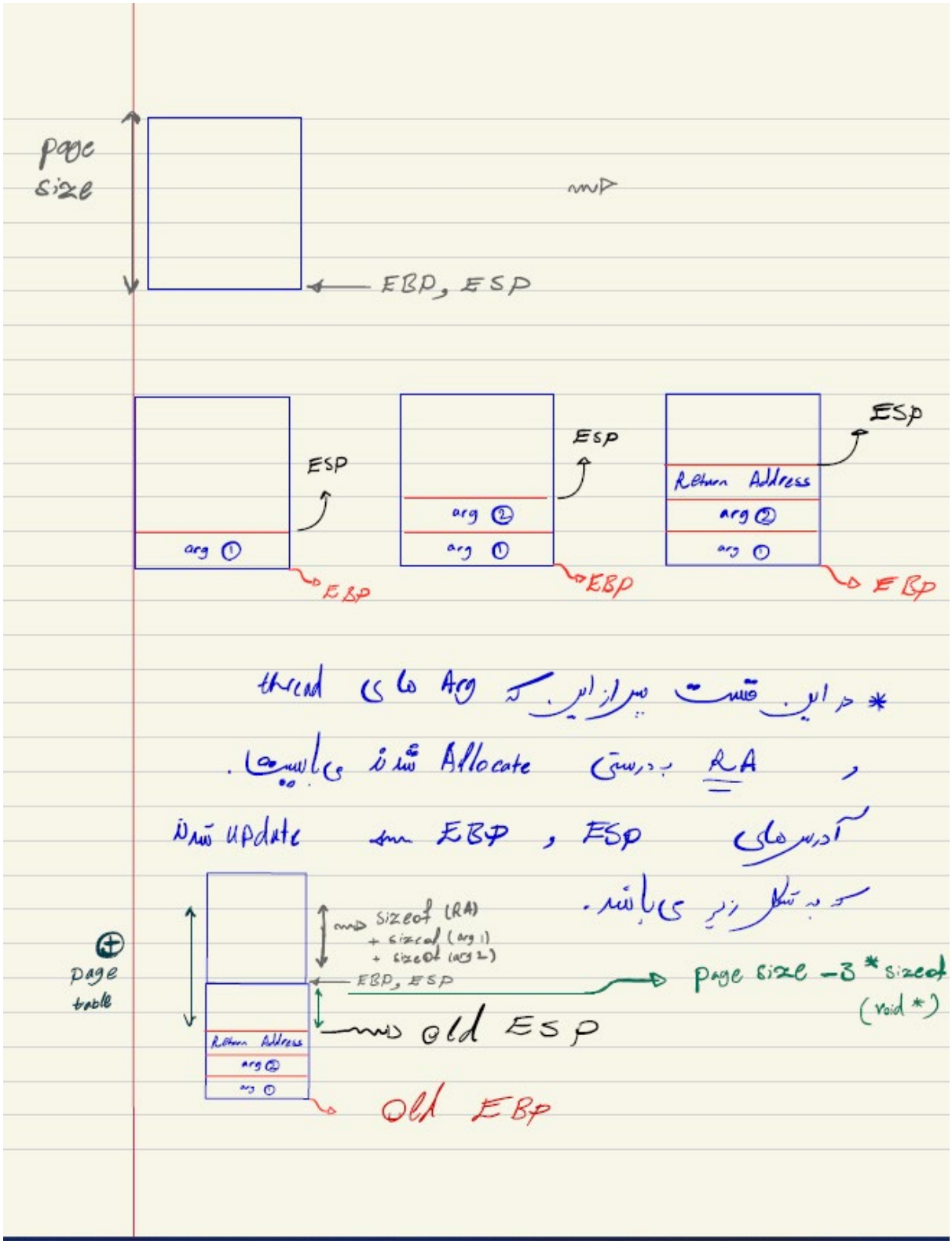


Figure 2-3. Memory layout of a user process with its initial stack.

حال در صفحه بعد که pdf جدا نیز قرار داده شده، به توضیح این که چگونه موقع ایجاد یک ترد جدید حافظه stack به آن اختصاص میابد می پردازیم.



حال به توضیح قسمت join می پردازیم. که بسیار شبیه به قسمت WAIT می باشد . در این تابع دو argument ، **void و int *pid را میگیرد، و با توجه به آن id اگر به حالت zombie درآمده بود، resourse های داده شده را از آن می گیرد.