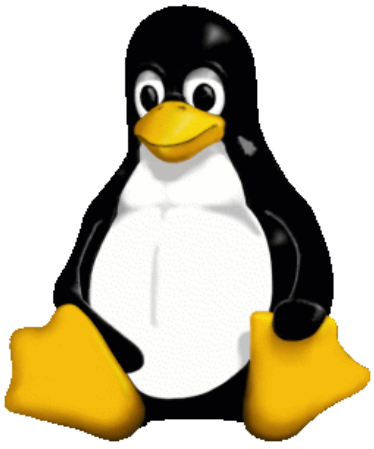
تألیف

دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

دستورکار آزمایشگاه سیستم‌های عامل

جلسه ششم: مدیریت حافظه



بهار ۱۳۹۳

مقدمه

در این جلسه از آزمایشگاه با ساختار حافظه‌ی پردازه‌ها آشنا خواهیم شد.

پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

- برنامه‌نویسی به زبان C/C++
- دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته شده‌اند.

مدیریت حافظه

همان‌طور که می‌دانید، در زبان‌های برنامه‌نویسی سطح بالا، مانند C، هنگامی که یک متغیر تعریف می‌کنید، کامپایلر فضای مورد نیاز برای آن‌ها را درنظر می‌گیرد و نیازی به تخصیص فضا به صورت دستی ندارید. متغیرهای سراسری در Data Segment پردازه و متغیرهای محلی در Stack Segment قرار می‌گیرند.

همچنین در برخی از شرایط نیاز است که حافظه به صورت پویا اختصاص یابد؛ برای مثال برای ایجاد یک داده‌ساختار مانند درخت یا لیست پیوندی. در زبان برنامه‌نویسی C فراخوانی‌های سیستمی malloc و free برای این منظور وجود دارند.

شرح آزمایش

الف) استفاده از فراخوانی‌های سیستمی malloc و free

۱. ساختار زیر را فرض کنید:

```
struct MyStruct {
	int a;
	int b;
	char name[20];
};
```

۲. به کمک malloc حافظه‌ی مورد نیاز برای یک instance از این ساختار را تخصیص دهید. خروجی دستور malloc چیست؟

۳. برای فیلدهای instance ایجاد شده مقادیری را اختصاص دهید و آن‌ها را چاپ کنید.

۴. به کمک free حافظه‌ی گرفته شده را آزاد کنید.

ب) مشاهده‌ی وضعیت حافظه‌ی پردازه‌ها

۱. به کمک دستور زیر وضعیت حافظه‌ی پردازه‌ها را مشاهده کنید:

```
ps -o user,vsz,rss,pmem,fname -e
```

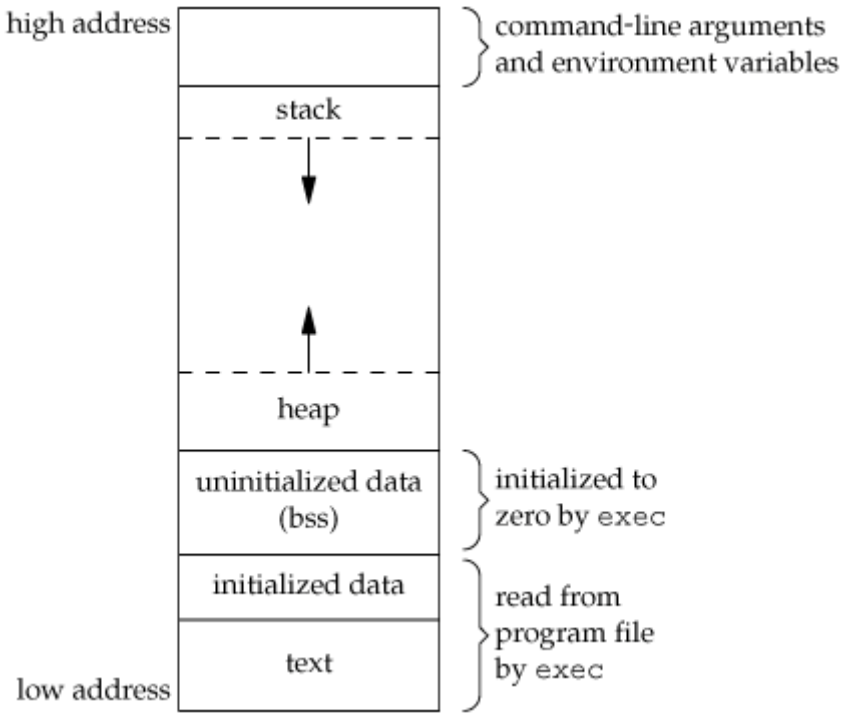
۲. به کمک دستور man ps توضیح دهید که هر کدام از ستون‌ها چه اطلاعاتی را نشان می‌دهند.

پ) اجزای حافظه‌ی یک پردازه

حافظه‌ی یک پردازه در سیستم‌عامل لینوکس به اجزای زیر تقسیم‌بندی می‌شود:

- بخش text که کد زبان ماشین پردازه را نگه‌داری می‌کند؛
- بخش data که متغیرهای سراسری پردازه در آن قرار می‌گیرد، برخی از متغیرهای دارای مقدار اولیه هستند و برخی خیر. دسته‌ی دوم در بخشی به نام bss واقع می‌شوند؛
- بخش heap که شامل حافظه‌هایی است که به صورت پویا اختصاص داده می‌شود و
- بخش stack که متغیرهای محلی، پارامترهای توابع و مقادیر بازگشتی آن‌ها در آن قرار دارد.

تصویر زیر این ساختار را نمایش می‌دهد (آدرس‌های کوچکتر در پایین تصویر قرار دارند):



۱. به کمک دستور which محل قرارگیری دستور ls را بر روی دیسک پیدا کنید.

۲. با استفاده از دستور size مشاهده کنید که چه مقدار از کد ماشین این دستور به بخش‌هایی که در بالا اشاره شد اختصاص دارد. این دستور کدام یک از بخش‌های بالا را نشان نمی‌دهد؟

ت) اشتراک حافظه

در سیستم‌عامل لینوکس، معماری حافظه به صورت صفحه‌بندی شده است؛ به این معنا که حافظه در تکه‌هایی (معمولاً ۸۱۹۲ یا ۴۰۹۶ بایتی) به پردازه‌ها اختصاص می‌یابد. سیستم‌عامل می‌تواند بعضی از این تکه‌ها را بین پردازه‌های مختلف به اشتراک بگذارد. برای مثال تمام پردازه‌هایی که یک کد یکسان را اجرا می‌کنند، بخش text مشترک دارند. کاربرد دیگر اشتراک این صفحات برای کتابخانه‌های مشترک است. برای مثال، اکثر برنامه‌ها از تابع printf استفاده می‌کنند؛ بنابراین منطقی است که تنها یکبار آن را در حافظه آورده و بین تمام پردازه‌هایی که از آن استفاده می‌کنند، حافظه را به اشتراک بگذاریم.

۱. به کمک دستور ldd کتابخانه‌های مشترکی که توسط دستور ls استفاده شده است را مشاهده کنید.
۲. این کار را برای چند برنامه‌ی دیگر (برای مثال nano) امتحان کنید و نتیجه را در گزارش کار خود بیاورید.

ث) آدرس‌های بخش‌های مختلف حافظه‌ی پردازه

به کمک استفاده از متغیرهای extern خاصی که در سیستم‌عامل تعریف شده‌اند، قادر هستیم که آدرس پایان code segment، آدرس پایان global segment و همچنین آدرس پایان بخش متغیرهای سراسری دارای مقدار اولیه را مشاهده کنیم.

برای این کار، سه متغیر زیر در دسترس هستند:

- etext: اولین آدرس بعد از پایان text در این متغیر قرار دارد.
- edata: اولین آدرس بعد از پایان متغیرهای سراسری دارای مقدار اولیه در این متغیر قرار دارد.
- end: اولین آدرس بعد از پایان بخش bss در این متغیر قرار می‌گیرد.

۱. به کمک دستور man etext، جزئیات بیشتری در مورد نحوه‌ی استفاده از این متغیرها ببینید و کدی که به عنوان مثال در انتهای این راهنما آمده است را نوشته و اجرا کنید.
۲. آیا خروجی این برنامه با ساختاری که در تصویر صفحه‌ی قبل آمده است تطابق دارد؟
۳. برای مشاهده‌ی آدرس انتهای heap از فراخوانی سیستمی sbrk استفاده می‌شود. به کمک این فراخوانی سیستمی نشان دهید که با اجرا کردن malloc و اختصاص حافظه به صورت پویا، این آدرس انتهای heap هربار تغییر می‌کند.

۴. همان‌طور که گفته شد stack در جهت عکس heap رشد می‌کند. همچنین، متغیرهای محلی و آرگومان‌های توابع در stack قرار می‌گیرند. یک تابع بازگشتی بنویسید که ابتدا یک متغیر محلی i ایجاد کند؛ سپس آدرس i را چاپ کرده و دوباره خودش را فراخوانی کند. روند تغییر آدرس متغیر i چگونه است؟ (برنامه را به گونه‌ای محدود کنید تا عملیات بازگشت مثلاً ۱۰۰ بار انجام شود).