

بسم الله الرحمن الرحيم

نام و نام خانوادگی : امیرحسین تقی زاده

رشته : فناوری اطلاعات و ارتباطات

موضوع : انواع حافظه در رم ((Stack) و (Heap) و ...)

استاد : میثاق یاریان

تعریف یک متغیر، ساخت instance، توابع، ایجاد یک thread و ... در برنامه نیاز به ذخیره سازی در حافظه دارند و همه آن ها فضایی از حافظه را اشغال می کنند. بنابراین لازم است تا یک برنامه نویس درک درستی از حافظه ای که در اختیار می گیرد داشته باشد.

• حافظه stack

در بخش user-space حافظه قرار دارد و به صورت خودکار توسط CPU مدیریت می شود. متغیرهای غیر استاتیک، پارامتر های ارسالی به توابع و آدرس های مربوط به return توابع در این حافظه ذخیره می شوند. اندازه حافظه stack ثابت است به همین دلیل به آن static memory گفته می شود.

در این حافظه اطلاعات پشت سر هم و به ترتیب قرار می گیرند به این صورت که آخرین داده ذخیره شده در بالای stack قرار می گیرد و به اصطلاح push می شود، حال اگر قصد برداشتن اطلاعات یا به اصطلاح pop کردن اطلاعات را داشته باشیم آخرین اطلاعات وارد شده در stack را در اختیار داریم. به این الگوریتم LIFO (Last In First Out) می گویند. مثال پر کاربرد در توضیح stack خشاب اسلحه (آخرین گلوله ای که در خشاب قرار داده می شود اولین گلوله ای است که شلیک می شود) و یا بشقاب های روی هم چیده شده (آخرین بشقابی که روی سایر بشقاب ها قرار داده می شود اولین بشقابی است که برداشته می شود) است.

از آنجا که در حافظه stack نیازی به پیدا کردن فضای خالی در حافظه نیست و محل قرارگیری اطلاعات مشخص است (بالای حافظه) بنابراین این حافظه سریع تر از حافظه heap است.

پارامتر ها و اطلاعات مربوط به توابع برای اجرا و کنترل آن ها در این حافظه ذخیره می شوند. تابعی که در بالای stack قرار دارد تابعی است که در حال اجراست و بعد از اتمام کار تابع یا بروز خطا در اجرای تابع، حافظه اختصاص داده شده به تابع از stack حذف می شود و حافظه اشغال شده آزاد می شود. زمانی که یک thread تعریف می شود در stack قرار می گیرد.

خطایی که ممکن است در اثر استفاده نادرست از حافظه stack رخ دهد stack overflow است. از جمله دلایل stack overflow یا سر ریز می توان به استفاده از متغیرهای محلی حجیم که منجر به کاهش فضای آزاد در stack و تخریب یا corrupt شدن بخشی از memory اشاره کرد.

• حافظه Heap

حافظه Heap در قسمت user-space حافظه مجازی قرار دارد و به صورت دستی توسط برنامه نویس مدیریت می شود. Heap مربوط به زمان اجرا (runtime) است و فضای اشغال شده در heap با اتمام کار تابع آزاد نمی شوند و تا زمانی که Garbage Collector این فضا را آزاد کند یا توسط برنامه نویس داده ها از حافظه heap پاک نشوند در این فضا باقی می ماند. اندازه حافظه heap متغیر است به همین دلیل به آن dynamic memory گفته می شود.

در این نوع از حافظه برای ذخیره مقادیر ابتدا محاسبه ای توسط سیستم عامل صورت می گیرد تا اولین فضای حافظه ای که اندازه آن متناسب با اندازه ای که مورد نیاز ماست را پیدا کند، در صورت وجود این میزان از حافظه درخواستی آن را به صورت رزرو شده درمی آورد تا بقیه برنامه ها به این فضا دسترسی نداشته باشند، سپس آدرس ابتدای این فضای محاسبه شده به صورت یک اشاره گر (pointer) در اختیارمان قرار می دهد (یا به اصطلاح allocating).

متغیر ها به صورت پیش فرض در این حافظه قرار نمی گیرند و اگر قصد ذخیره متغیر ها در این حافظه را داشته باشیم باید به صورت دستی این اقدام انجام شود. متغیر هایی که در heap ذخیره می شوند به طور خودکار حذف نمی شوند و باید توسط برنامه نویس و به صورت دستی حذف شوند. به طور کلی مدیریت حافظه heap به صورت دستی توسط برنامه نویس انجام می شود. آرایه های داینامیک در heap ذخیره می شوند.

در صورتی که داده های ما از تعداد block های پشت سر هم در حافظه بیشتر باشد یا در صورت تغییر حجم داده ها در زمان های مختلف (تغییر سایز داده ها امکان پذیر است)، سیستم عامل داده ها را به صورت تکه تکه در block های حافظه ذخیره خواهد کرد.

به دلیل محاسبات برای یافتن آدرس شروع حافظه و در اختیار گرفتن pointer حافظه heap نسبت به stack کندتر است. همچنین اگر داده ها به صورت پشت سر هم در block های حافظه قرار نگرفته باشند (این احتمال بسیار زیاد است) موجب کندی در بازیابی اطلاعات خواهد شد.

وقتی که نمونه‌ای از یک کلاس ایجاد می‌کنیم این مقدار در Heap ذخیره می‌شود و وقتی که کار آن به پایان می‌رسد garbage collector حافظه را آزاد می‌کند و اگر موفق به این کار نشود، برنامه نویس باید به صورت دستی حافظه heap را آزاد کند، در غیر این صورت Memory leak اتفاق می‌افتد که به معنی in use نگه داشتن فضای حافظه برای اشیایی است که دیگر از آن‌ها در برنامه استفاده نمی‌شود و garbage collector قادر به آزاد سازی فضایی که آن‌ها اشغال کرده اند نیست.

به طور کلی Value Type ها (primitive type) فضای زیادی اشغال نمی‌کنند و در stack ذخیره می‌شوند. برای دسترسی به متغیرهای Value Type، مقدار آن به صورت مستقیم از حافظه stack خوانده می‌شود، مثلاً زمانی که متغیری تعریف می‌کنیم آن متغیر به همراه مقدار آن در stack قرار می‌گیرد.

برای دسترسی به متغیرهای Reference Type، ابتدا با مراجعه Stack و دریافت آدرس متغیر در Heap به شیء مربوط به متغیر دسترسی خواهیم داشت. Reference Type ها در حافظه heap نگهداری می‌شوند. زمانی که یک شیء از کلاس ایجاد می‌کنیم ابتدا متغیری که شیء به آن assign شده است با مقدار null در حافظه stack قرار می‌گیرد، سپس شیء در heap ذخیره شده و پس از ذخیره سازی در heap آدرس شیء در stack جایگزین null می‌شود. همچنین reference type ها به dynamic memory و value type ها به static memory نیاز دارند. در صورت نیاز به dynamic memory، باید امکان دسترسی به heap فراهم باشد و اگر نیازمند static memory باشیم، stack محل ذخیره سازی خواهد بود.