به نام خداوند بخشنده مهربان

دانشگاه صنعتی شریف
مدرس: مهدی خرازی

تمرین سوم*
سیستم عامل

دانشکده‌ی مهندسی کامپیوتر
پاییز ۱۳۹۶

۱. مقدمه

هدف این تمرین **پیاده‌سازی دستورات مدیریت حافظه** در کتابخانهٔ استاندارد C است. در انجام این تمرین شما با واسط POSIX و ساختار حافظه مجازی پردازه‌ها[1] آشنا شده و با چالش‌های الگوریتمی جذابی روبه‌رو خواهید شد. صفحات راهنمای رسمی `malloc` و `sbrk` مراجع خوبی برای انجام این تمرین هستند.

بدیهی است استفاده از دستورهای استاندارد مدیریت حافظه در C مانند malloc، free و realloc در این تمرین مجاز **نیست** و با هدف آن در تناقض خواهد بود.

۲. راه‌اندازی

ابتدا می‌بایست قالب انجام تمرین را از مخزن تمرین‌های درس دریافت کنید:

```
cd ~/code/ce424-961-handouts
git pull
cd hw3
```

پس از دریافت فایل‌ها در مسیرِ `hw3` پرونده‌ای با نام `mm_alloc.c` خواهید یافت که قالبی ساده برای انجام پروژه است. در این پرونده سه دستور `mm_malloc`، `mm_free` و `mm_realloc` تعریف شده‌اند که شما می‌بایست آن‌ها را پیاده‌سازی کنید. از تغییر نام این توابع خودداری کنید!

* این تمرین برگرفته از تمرین ارائه شده در دانشگاه برکلی، مربوط به درسِ CS۱۶۲ می‌باشد.
با تشکر از تیم دستیاران آموزشی

[1] Process

همچنین در این پوشه، پرونده‌ی دیگری با نام `mm_test.c` وجود دارد که می‌توانید آن را برای بار کردن[۲] و تست ابتدایی کدهای خود استفاده کنید. از آنجا که این پرونده در نمره‌دهی تاثیر ندارد، شما می‌توانید آن را به طور دلخواه تغییر دهید.

۳. پیش‌زمینه: گرفتن حافظه از سیستم‌عامل

۱.۳. حافظهٔ پردازه

می‌دانیم هر پردازه در سیستم‌عامل دارای فضای آدرس‌دهی [۳] مخصوص به خود است. بخش‌هایی از این فضای آدرس‌دهی به هنگام تبدیل آدرس (Address Translation) توسط MMU[۴] و هستهٔ سیستم‌عامل به حافظه فیزیکی[۵] نگاشته می‌شوند. برای ساختن یک تخصیص‌دهنده حافظه (Memory Allocator)، می‌بایست ابتدا ساختار حافظه heap را به درستی درک کرد.

حافظه heap فضایی پیوسته از آدرس‌های مجازی است که برای آن ۳ مرز تعریف می‌شود:

- «پایین» یا شروع heap،
- «بالای» heap که به آن ***Break*** (*وقفه*) گفته می‌شود. break پایان قسمتی از فضای حافظهٔ heap را مشخص می‌کند که به حافظهٔ فیزیکی نگاشته شده و به کمک فراخوانی‌های سیستمی [۶] brk و sbrk تغییر داده می‌شود. آدرس‌های مجازی بالاتر از break توسط سیستم‌عامل به حافظه فیزیکی نگاشته نشده‌اند.
- مرز سخت حافظه heap که break نمی‌تواند از آن بگذرد و باید پایین‌تر از آن باشد. فضای بالاتر از این آدرس قابل اختصاص به heap نیست و دسترسی به آن موجب خطا می‌شود.

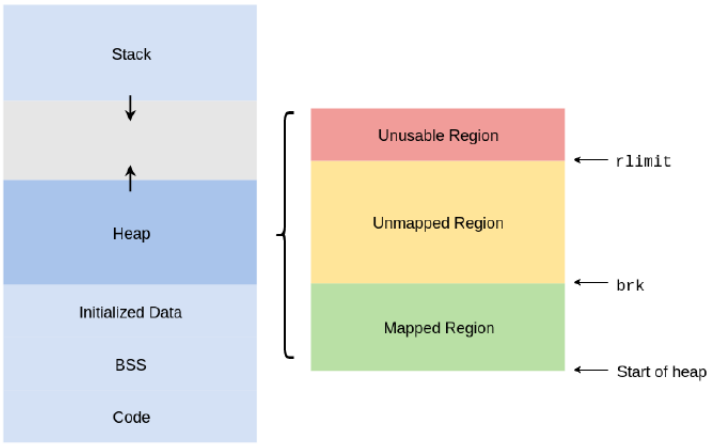
  این مرز توسط تابع‌های `getrlimit` و `setrlimit` تعریف شده در فایل `sys/resource.h` مدیریت می‌شود.

در انجام این تمرین شما باید قطعه‌های **نگاشته شدهٔ** حافظه را به هنگام فراخوانی دستور allocate به فراخواننده تخصیص دهید. همچنین هنگامی که لازم شد ناحیهٔ نگاشته‌شده را گسترش دهید، محل break را به کمک دستور sbrk به میزان مناسب تغییر دهید.

---

[۲] Load
[۳] Address Space
[۴] Memory Management Unit
[۵] Physical Address
[۶] system call

شکل ۱: ساختار قسمت نگاشته‌شده حافظه heap هنگام پیاده‌سازی اختصاص‌دهنده با لیست پیوندی



۲.۳ sbrk

اندازهٔ قسمت نگاشته‌شدهٔ حافظه heap در ابتدا صفر است. برای گسترش قسمت نگاشته شده لازم است محل break تغییر داده شود. همانطور که گفته شد فراخوانی سیستمی که برای این کار پیشنهاد می‌شود، sbrk است:

```
1 void *sbrk(int increment);
```

sbrk محل فعلی break را به اندازه ورودی آن (increment) افزایش می‌دهد و آدرس محل قبلی break را برمی‌گرداند. بنابراین برای گرفتن محل break کافیست به آن صفر را پاس دهید. (در واقع می‌توانید مقداری که sbrk برمی‌گرداند را به صورت محل شروع حافظه‌ای که بعد از فراخوانی sbrk به ناحیهٔ نگاشته‌شده افزوده می‌شود ببینید)

برای اطلاعات بیشتر می‌توانید صفحات راهنمای (manual) مربوطه را مطالعه کنید.

۳.۳ داده‌ساختار Heap

برای مدیریت حافظه، لازم است مشخصات قطعه‌های آزاد (و یا اشغال شدهٔ) حافظه را در داده‌ساختار مناسبی نگهداری کنیم تا هنگام درخواست حافظه بدانیم با توجه به مقدار درخواست شده، کدام قطعه حافظه قابل اختصاص به درخواست‌کننده است.

به این منظور، روش‌های متعددی وجود دارد. سه داده‌ساختار پیشنهادی زیر از جمله آنها هستند:

۱. یک لیست پیوندی (Linked List) از قطعه‌های حافظه

۲. یک لیست از **اندازه های حافظه**، که هر یک شامل یک لیست پیوندی از قطعه‌های حافظه با آن اندازه باشند. (در واقع می‌توان آن را به لیستی از سبدهای حافظه تعبیر کرد که در هر سبد تکه‌هایی هم‌اندازه از حافظه نگه‌داری می‌شود)

۳. یک درخت بازه (Interval Tree). برگ های این درخت قطعه‌های آزاد حافظه هستند و هر گره از درخت بازه‌ای را به صورت (شروع، اندازه) بیان می‌کند. به این ترتیب اگر N بایت حافظه درخواست شود و درخت به خوبی متوازن شده باشد (مثلاً از Red-Black Tree به جای BST عادی استفاده شود) می‌توان درخت را برای قطعه‌هایی با اندازه بزرگتر از N در زمان $O(\log n)$ جست‌وجو کرد.

شما در این تمرین باید ساده‌ترین این داده‌ساختارها، یعنی مورد اول را پیاده‌سازی نمایید.

۴.۳. سازماندهی حافظه

همانطور که در قسمت قبل گفته شد قصد داریم پیاده‌سازی خود را در این تمرین به کمک یک لیست پیوندی ساده انجام دهیم. عناصر این لیست، قطعه‌های حافظه هستند که ممکن است آزاد یا مورد استفاده باشند.

به این منظور، درست قبل از هر قطعهٔ حافظه تعداد مشخصی بایت برای نگهداری ابرداده (metadata) آن کنارگذشته می‌شود که به منزله سرآیند (header) آن است.

در این سرآیند، مقادیر زیر نگهداری می‌شود:

- prev و next: اشاره‌گرهایی به عناصر قبلی و بعدی لیست پیوندی (که همان metadata یا سرآیند قطعه‌های قبلی و بعدی حافظه هستند)
- free: مقداری دودویی که بیانگر آزاد یا مورداستفاده بودن قطعه حافظه است
- size: اندازه قطعه حافظه

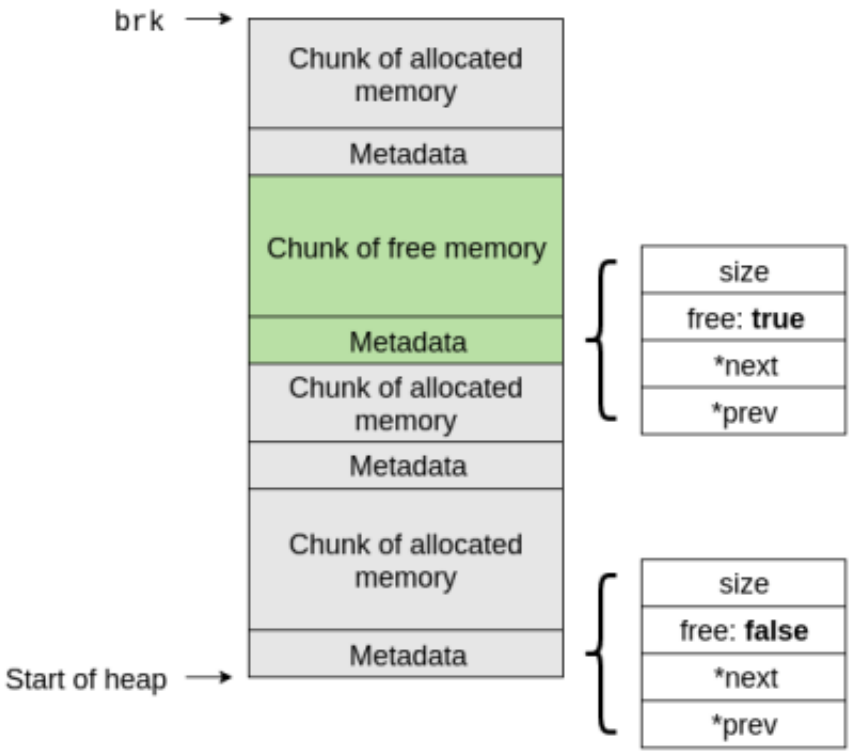
۴. پیاده‌سازی

در قسمت قبل توضیح دادیم روش‌های متعددی برای پیاده‌سازی اختصاص‌دهنده حافظه وجود دارد. همچنین با چند داده‌ساختار مختلف برای سازماندهی حافظه heap آشنا شدید و گفته‌شد در این تمرین قصد داریم از یک لیست پیوندی ساده استفاده کنیم.

تنوع روش‌های پیاده‌سازی اختصاص‌دهنده به داده‌ساختار مورداستفاده برای نگهداری قطعات حافظه محدود نمی‌شود و الگوریتم های متعددی نیز برای اختصاص، بازپس‌گیری [۷] و اختصاص مجدد [۸] قطعه‌های حافظه وجود دارد که یک مورد از آنها در اینجا توصیف خواهد شد. [۹]

[۷] Deallocation
[۸] Reallocation
[۹] در این تمرین برای سهولت بیشتر ساده‌ترین داده‌ساختار heap و الگوریتم تخصیص و بازتخصیص حافظه در نظر گرفته شده‌است. برای اطلاعات بیشتر از ساده‌سازی های انجام‌شده می‌توانید قسمت ۲.۶ را مطالعه کنید.

شکل ۲: ساختار قسمت نگاشته‌شده حافظه heap هنگام پیاده‌سازی اختصاص‌دهنده با لیست پیوندی



۱.۴. اختصاص حافظه

```
void *mm_malloc(size_t size);
```

کاربر مقدار حافظه مورد درخواست خود به بایت را به صورت ورودی ***size*** پاس می‌دهد. تابع `mm_malloc` یک قطعه حافظه با اندازه خواسته شده را به کاربر اختصاص داده و اشاره‌گر به آن را برمی‌گرداند.

توجه داشته‌باشید مقداری که برمی‌گردانید باید نقطه شروع حافظه قابل استفاده باشد، نه سرآیند قطعه حافظه اختصاص داده‌شده.

یکی از الگوریتم‌های ساده برای اختصاص حافظه، **first-fit** نام دارد. در این روش هنگامی که تابع اختصاص فراخوانی می‌شود، قطعه‌های حافظه را به ترتیب مرور می‌کند تا قطعه‌ای که به اندازه کافی بزرگ باشد را پیدا کند:

- اگر چنین قطعه‌ای پیدا نشود، sbrk را صدا می‌کنیم تا فضای heap را گسترش دهیم.
- اولین قطعه حافظه‌ای که به اندازه کافی بزرگ باشد به کاربر اختصاص داده می‌شود. در صورتی که این قطعه آنقدر بزرگ باشد که علاوه بر مقدار مورد درخواست کاربر بتواند قطعه دیگری را نیز در خود جای دهد، قطعه به دو قسمت تقسیم می‌شود که یکی دقیقاً به اندازه مورد درخواست کاربر است و دیگری شامل قسمتی از قطعه اولیه است که اضافه آمده.

در انجام محاسبات یادشده به وجود سرآیند metadata توجه کنید. برای مثال ممکن است قطعه‌ای پیدا کنید که از مقدار درخواست‌شده توسط کاربر بزرگتر باشد ولی نتواند علاوه بر آن سرآیند یک قطعه جدید را در خود جای دهد. در این صورت در این روش از مقدار اضافه صرف‌نظر می‌کنیم و آن را بدون استفاده خواهیم گذاشت.

- اگر نمی‌توانیم قطعه‌ای با اندازه خواسته شده را به کاربر اختصاص دهیم، مقدار NULL را برمی‌گردانیم.
- اگر مقدار درخواست شده صفر باشد، NULL را برمی‌گردانیم.
- برای ساده‌تر کردن نمره‌دهی، از شما خواسته می‌شود مقدار بایت‌های اختصاص داده شده را قبل از تحویل به کاربر صفر کنید. (عملاً دستور calloc را پیاده‌سازی می‌کنید! می‌توانید از دستور `memset` کتابخانه استاندارد C استفاده کنید)

۲.۴. بازپس‌گیری حافظه

```
void mm_free(void *ptr);
```

کاربر زمانی که دیگر به یک قطعه از حافظه نیازی نداشته باشد، اختصاص‌دهنده حافظه را فراخوانی می‌کند تا آن قطعه را آزاد کند. به این منظور کاربر همان آدرسی که از `mm_malloc` دریافت کرده (شروع قطعه موردنظر) را به `mm_free` پاس می‌دهد.

دقت داشته‌باشید عمل deallocate به این معنا نیست که لازم است حافظه بازپس‌گرفته‌شده را به سیستم‌عامل برگردانیم، بلکه فقط باید بتوان آن را مجدداً برای درخواست دیگری اختصاص داد. به این ترتیب شما هیچ گاه break را پایین‌تر نخواهید برد.

- از آنجا که هنگام اختصاص گاهی قطعات حافظه را تقسیم می‌کنیم، پس از مدتی با مسئله **fragmentation** روبرو می‌شویم: ممکن است قطعه‌ای آزاد از حافظه به مقدار N بایت موجود باشد اما چون در چند قطعه مجاور شکسته شده نمی‌توانیم آن را به یک درخواست $> N$ بایتی اختصاص دهیم.

  برای جلوگیری از این موضوع، هنگام فراخوانی دستور free قطعه آزاد شده را در صورت وجود به قطعات آزاد مجاور ملحق می‌کنیم. یعنی اگر قطعه‌ای که آزاد شده در همسایگی قطعه آزاد دیگری باشد، آن دو قطعه را با یکدیگر merge کرده و یک قطعه آزاد بزرگتر ایجاد می‌کنیم.

  در اینجا نیز باید به سرآیند قطعات حافظه توجه داشته باشید و عمل حذف سرآیند میانی از لیست پیوندی را به درستی انجام دهید.

- اگر اشاره‌گر NULL به deallocator شما پاس داده شود، نباید هیچ کاری انجام دهید.

۳.۴. اختصاص مجدد حافظه

```
void *mm_realloc(void* ptr, size_t size);
```

دستور Reallocation باید اندازه قطعه حافظه واقع در آدرس ***ptr*** را به ***size*** تغییر دهد.

ptr همان مقدار دریافت شده از `mm_malloc` هنگام اختصاص حافظه است و size می‌تواند بزرگتر یا کوچکتر از اندازه قطعه داده شده باشد.

برای سادگی، realloc را با آزاد کردن قطعه داده‌شده به کمک `mm_free` و اختصاص یک قطعه جدید با اندازه درخواست شده به کمک `mm_malloc` و در نهایت کپی کردن داده‌های قدیمی به محل جدید با کمک دستور `memcpy` کتابخانه استاندارد C پیاده‌سازی کنید.

اگر اندازه درخواست شده از اندازه قطعه اولیه بزرگتر باشد، بایت های جدید در انتهای قطعه باید همگی صفر شوند. در انجام این عملیات به حالت فضای بدون استفاده در انتهای قطعه‌ها که قبلاً توضیح داده‌شد نیز توجه داشته باشید.

- اگر نمی‌توانید قطعه‌ای با اندازه درخواست شده را به کاربر اختصاص دهید، مقدار NULL را برگردانید.
- `realloc(ptr, 0)` هم ارز دستور `mm_free(ptr)` است و مقدار NULL را برمی‌گرداند.
- `realloc(NULL, n)` هم ارز دستور `mm_malloc(n)` است.
- `realloc(NULL, 0)` هم ارز دستور `mm_malloc(0)` است و باید NULL را برگرداند.

۴.۴. تحویل‌دادنی‌ها

تحویل‌دادنی شما در این تمرین صرفاً **پیاده‌سازی شما از دستورات malloc، realloc و free** به شیوه توصیف شده در بالا است. از شما انتظار می‌رود در پیاده‌سازی خود از داده‌ساختار و الگوریتم توصیف شده در صورت تمرین پیروی کنید اما در طراحی ساختار کد خود آزاد هستید.

برای مثال دستورات دیگر تعریف شده در فایل header مانند `split_block` و `fusion` پیشنهادی هستند و پیاده‌سازی آنها تا جایی که الگوریتم توصیف شده رعایت شود جزو تحویل‌دادنی‌های شما محسوب نمی‌شود.

این تمرین تحویل‌دادنی غیر کد (مانند مستند و یا گزارش) **ندارد**.

۵. ارسال پاسخ

برای ارسال پاسخ تمرین، کافیست تغییرات خود را push کنید تا نمره‌دهی خودکار انجام شود.

```
git push personal master
```

پس از حداکثر نیم ساعت، شما می‌توانید با به‌روز کردن مخزن خود، نمره‌ی خود را در پرونده‌ی `grade.txt` مشاهده نمایید.

همچنین تمامی پرسش‌ها و مشکلات خود را از طریق **ثبت Issue در مخزن git** مطرح نمایید.

۶. اطلاعات اضافه

۱.۶. قسمت نگاشته‌نشده و فضای بدون مالک

همانطور که گفته شد break انتهای قسمت نگاشته شده فضای آدرس مجازی به فضای آدرس فیزیکی را مشخص می‌کند. با این فرض، دسترسی به آدرس‌های بالاتر از break می‌بایست منجر به خطا شود. (معمولاً "bus error" یا "segmentation fault")

اما این قاعده همیشه درست نیست. می‌دانیم فضای آدرس مجازی دارای پیمانه هایی به نام page است که معمولاً اندازه آنها مضربی از ۴۰۹۶ بایت می‌باشد. هنگامی که sbrk صدا شود، سیستم عامل باید حافظه بیشتری را به heap اختصاص دهد. به این منظور، سیستم عامل یک page کامل از حافظه فیزیکی را به heap اختصاص داده و قسمت نگاشته شدهٔ heap را گسترش می‌دهد.

بنابراین همواره این احتمال وجود دارد که break دقیقاً در انتهای یک page قرار نگیرد. در این حالت، وضعیت فضای بین break و انتهای page حافظه چه خواهدبود؟

این فضا، *فضای بدون مالک* (***No man's land***) نامیده می‌شود و به لحاظ منطقی به heap اختصاص ندارد چرا که بالاتر از break قرار دارد ولی دسترسی به آن منجر به خطا نیز نمی‌شود چرا که در page ای از حافظه فیزیکی قرار خواهد داشت که به حافظه heap پردازه اختصاص داده شده‌است.

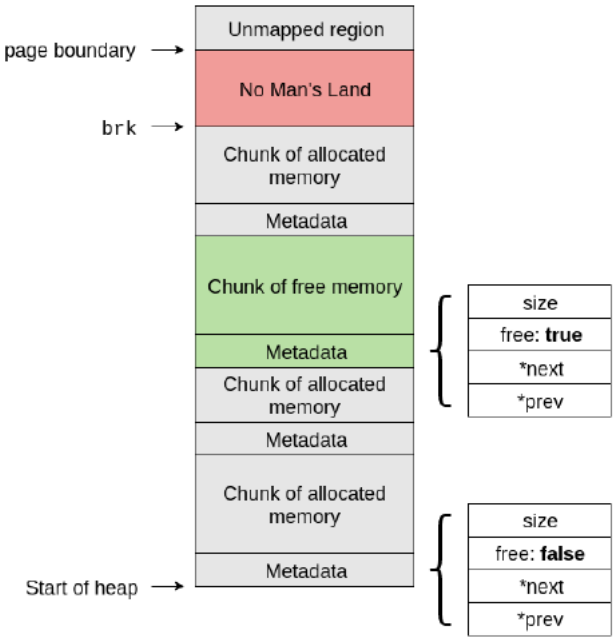
این موضوع می‌تواند به باگ‌های عجیبی در نرم‌افزار منجر شود. دسترسی به فضایی بیرون از heap (مثلاً به علت شماره درایه نادرست در استفاده از آرایه‌ها) منجر به بروز خطانمی‌شود و برنامه به عملکرد نادرست خود ادامه می‌دهد. یافتن منشأ چنین باگ‌هایی می‌تواند بسیار دشوار باشد.

برای مثال ممکن است برنامه دارای ایرادی باشد که داده‌هایی را به اشتباه خارج از محل صحیح خود در heap بنویسد اما برای ورودی‌های کوچک این داده‌ها در فضای بدون مالک قرار بگیرند و خطایی رخ ندهد. اما با بزرگ شدن ورودی به تدریج از فضای بدون مالک نیز بیرون بزنیم و به فضای بیرون از page وارد شویم و خطای segfault رخ دهد. نتایج از این دست می‌تواند بسیار گیج‌کننده باشد.

۲.۶. مقایسه با Standard C

همان‌طور که گفته‌شد با توجه به پیچیدگی اختصاص‌دهنده‌های استاندارد حافظه که در زبان‌های متداول استفاده می‌شوند، در این تمرین ساده‌سازی های متعددی صورت گرفته‌است. به عنوان یک تمرین خوب می‌توانید بخش‌هایی از متن پیاده‌سازی استاندارد دستور malloc را در این مخزن git مطالعه کنید.

شکل ۳: ساختار قسمت نگاشته‌شده حافظه heap هنگام پیاده‌سازی اختصاص‌دهنده با لیست پیوندی



از جمله تفاوت‌های پیاده‌سازی شما با کتابخانه استاندارد C می‌توان به موارد زیر اشاره کرد:

- ساختار struct مورد استفاده برای نگهداری اطلاعات قطعه‌های حافظه در libc کمی پیچیده‌تر است. در این ساختار اطلاعاتی مانند اندازه قطعه حافظه قبلی نیز نگهداری می‌شود.
- برای کارآیی بالاتر، در پیاده سازی رسمی توجه می‌شود قطعه‌های حافظه در even-word boundary آغاز شوند.
- قطعه‌های آزاد حافظه در یک لیست پیوندی دایره‌ای دوطرفه [۱۰] نگهداری می‌شوند.
- برای سرعت بالاتر و مصرف حافظه کمتر، از استراتژی‌های متعددی استفاده می‌شود. سبد (bin) های قطعه‌های حافظه که براساس اندازه مرتب شده‌اند، نگه‌داری قطعه‌های به‌تازگی آزادشده برای اختصاص‌دهی سریع‌تر و trade-off های متعدد بین سرعت و مصرف اضافه حافظه از جمله این استراتژی‌ها هستند.
- پیاده‌سازی malloc در کتابخانه استاندارد، thread-safe است.
- در پیاده‌سازی free تنظیماتی وجود دارد که با کاهش سرعت، مقدار مصرف اضافی حافظه به طرز قابل‌توجهی کاهش یابد.
- در پیاده‌سازی realloc به جای درخواست حافظه جدید، کپی کردن داده‌ها به محل جدید و آزادسازی حافظه قبلی تلاش می‌شود درصورت امکان قطعه حافظه بدون انتقال به مکان جدید گسترش یابد. برای مثال اگر

[۱۰] Circular Doubly-linked List

قطعه بعدی آزاد باشد گاهی می‌توان با الحاق آن به قطعه قعلی با سرعت بسیار بالاتری عمل اختصاص دوباره حافظه را انجام داد.

۳.۶. حتی بیشتر

این قسمت اختیاری و فاقد نمره امتیازی است و صرفاً مخصوص علاقه‌مندان است.

شما می‌توانید اختصاص‌دهنده حافظه خود را از نقطه‌نظرهای بسیاری بهبود دهید. توجه داشته باشید که نمره‌دهنده خودکار (جاج) انتظار دارد شما الگوریتم first-fit را پیاده‌سازی کنید. بنابراین اگر تصمیم به پیاده‌سازی قسمت های اضافه دارید، آن را پس از ارسال قسمت اصلی برای تصحیح و گرفتن نمره کامل انجام دهید.

- اختصاص‌دهنده خود را Thread Safe کنید! منظور این نیست که یک قفل دور کل دستور malloc خود قرار دهید، بلکه باید داده‌ساختارهای خود را طوری طراحی کنید که دسترسی چند ریسه به صورت هم‌زمان به آنها امکان‌پذیر باشد.

  یک راه خوب برای این مقصود، این است که داده‌ساختاری استفاده کنید که قطعه‌های هم‌اندازه حافظه را در یک لیست (سبد) قرار دهد و برای هر سبد یک قفل جدا در نظر بگیرید.

  به این ترتیب دو ریسه که malloc را هم‌زمان صدا کنند تنها در صورتی بلاک می‌شوند که قطعه‌هایی هم‌اندازه را درخواست کنند.

- الگوریتم اختصاص‌دهی خود را بهبود دهید. الگوریتم first-fit یکی از ساده‌ترین روش‌های اختصاص‌دهی حافظه است. یکی از روش‌های پیشرفته‌تر، Buddy Allocator نام دارد که می‌توانید درباره آن تحقیق کنید.

- پیاده‌سازی realloc را بهبود دهید به طوری که در صورت امکان از قطعه حافظه فعلی استفاده کند و از آزاد کردن، اختصاص دوباره و کپی کردن بی‌مورد اجتناب کند.