

malloc アルゴリズムについて

1998 年 9 月 18 日金曜日
セガ・エンタープライゼス
ソフトウェア技術開発部

Dreamcast SDK Ver.1.00 より、syMalloc および、malloc のアルゴリズムが大きく変わりました。

古いアルゴリズムについては old ディレクトリにある、malloc.doc をご覧ください。

ここでは、Dreamcast SDK Ver.1.00 で新規導入されたアルゴリズムについてのみ解説致します。

malloc と syMalloc の関係

syMalloc は以下の宣言によりマクロ化されております。

```
#define syMalloc(ptr) malloc(ptr)
```

つまり、両者は全く同じ物です。

以後は、malloc のアルゴリズムについて詳しく説明をしていきます。

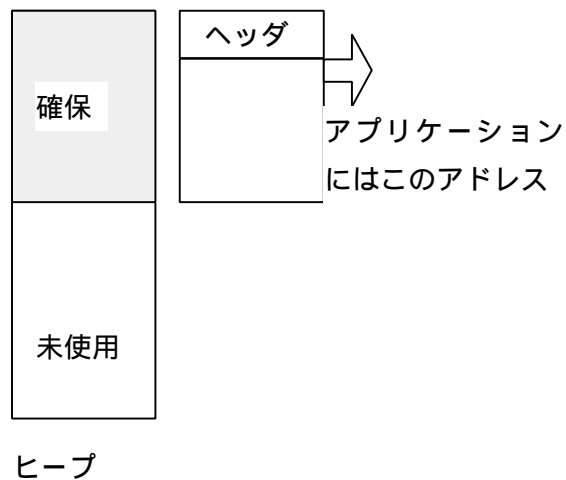
malloc の仕組み

syMalloc によって、アプリケーションが宣言したヒープは、malloc 関数によって直接管理されます。

malloc 関数には、内部で、連続した空きメモリのブロックのリストを管理しており、アプリケーションから malloc にメモリ領域の割り当て要求があった場合、その空きメモリリストからアプリケーションの要求したサイズと 32 バイトの malloc 用のヘッダを加えたメモリを確保しようとします。つまり、一回の malloc で

アプリケーションが要求したサイズ + 32 バイト

のデータがヒープから取られることになります。



< 要求領域とヘッダ >

32 バイトの malloc 用ヘッダには、要求したメモリのサイズなどの情報が保存され、free する際に使用されます。

次に malloc の管理する空きメモリブロックのリストについてももう少し詳しく見ていきましょう。

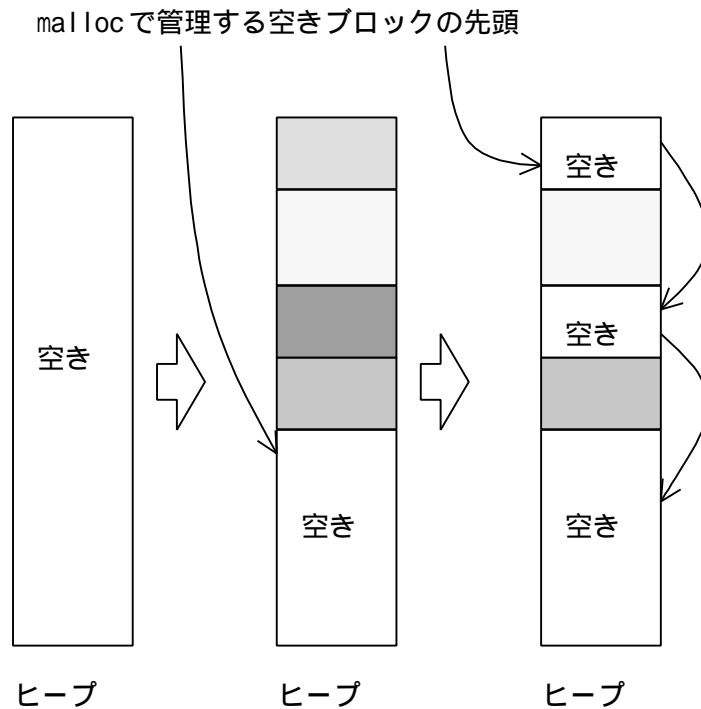
malloc 関数は、メモリ管理と、アプリケーションに連続したメモリブロックを供給できるように、連続したメモリブロックの情報を連結リストにして管理しています。

syMallocInit 関数が実行され、アプリケーションがヒープ領域を確保した時点では、ヒープの全領域が連続であることが保証されますので、リストは一つになります。

何回か、malloc が行われると、ヒープの中身はいくつかのメモリブロックにわかれ、それぞれ別の目的で使用されています。

さらにそのうちのいくつかが free されると、ヒープ領域は分断されいくつかのまとまった空きメモリのブロックに別れていきます。

malloc は free されたメモリのブロックのうち最初の 32 バイトを使用し、メモリブロックの連結情報を加え管理します。



＜メモリの確保と分断＞

malloc を繰り返し、上図の三番目のような状態になった時、malloc は空きブロックリストの先頭からアプリケーションで要求する連続したメモリブロックがあるかを調べて、最初に見つけたものから、必要とするサイズを取得し、アプリケーションに返します。

このアルゴリズムをFirst Fitといいます。

このアルゴリズムに関する詳しい説明が、共立出版社から発行されている

「プログラミング言語 C(通称 K&R)の 8.7 記憶割り当て」

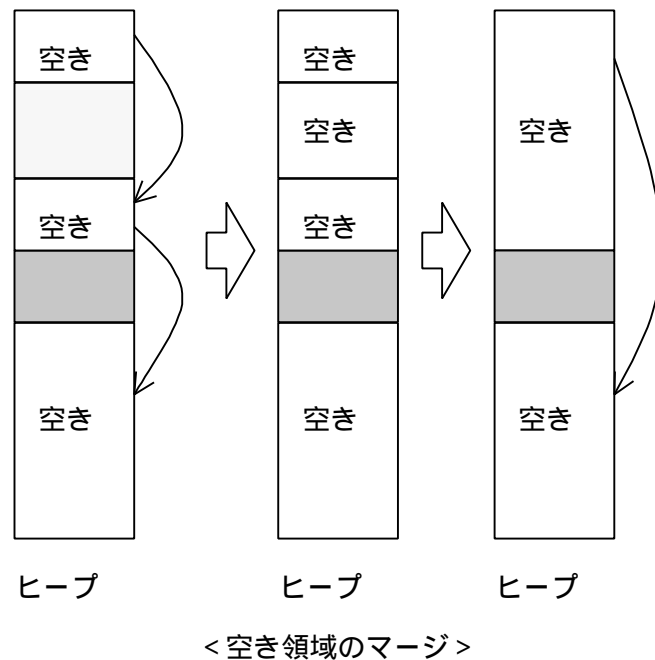
にありますので、興味のある方はご覧いただくとよいでしょう。

free の仕組み

次に、freeによってヒープの状態がどう変わっていくのかを見ていくことにします。

上の図で、1と書かれたエリアを free した場合、この図では、全部で、4つの空きメモリブロックが生まれます。

ところが、1の前後には空きがあるため、実際には1を含め前後の空きメモリブロックをマージし、二つのメモリブロックが生成されることになります。



この様に malloc アルゴリズムでは、極力分断の起こらない方法で、メモリを確保、開放します。

malloc と new

C++言語を使用すると、new により動的に変数空間を割り当てることが出来ますが、この際に new 演算子は内部で malloc関数を呼び出します。

また、SHCPP の場合、同時に malloc で確保したメモリブロックのヘッダに new 用の特別な情報を書き込みます。

SEGA ライブラリの malloc 関数は、この SHCPP 独自のメモリアロケーションの動作をサポートしていますが、**現状では C++に対する検証が十分でないため SEGA ライブラリ利用環境下での C++のご利用は極力避けて頂くようお願いいたします。**

SDRAM の特性と malloc

Dreamcast のメインメモリに搭載されている SDRAM には、同一バンク間のアクセスよりも、異なるバンク間でのアクセスのほうがメモリアクセスが高速であるという特徴を持っていますが、現在の malloc 関数は、複数ヒープの同時管理をサポートしておりませんので、この特性を生かすことは出来ません。

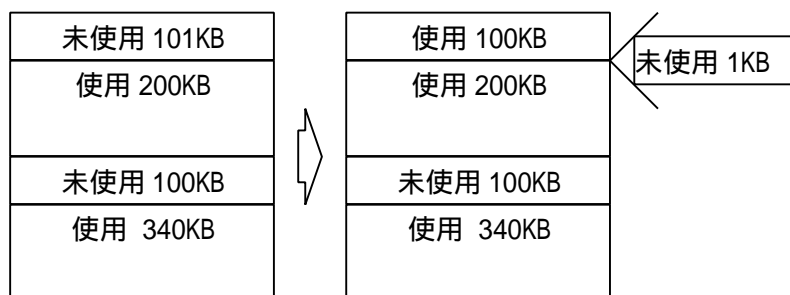
SEGA ライブラリの malloc の問題点

先にも説明しましたように、SEGA ライブラリでは First Fit アルゴリズムを用いて、メモ

リブロックを獲得しています。

しかしながら、First Fit アルゴリズムは非常に高速に検索できる反面、メモリの分断化が起こりやすく、又、効率が悪いという欠点を持っています。

下図のようなヒープの状態を例にとって、実際に見ていきましょう。



<First Fitの問題点>

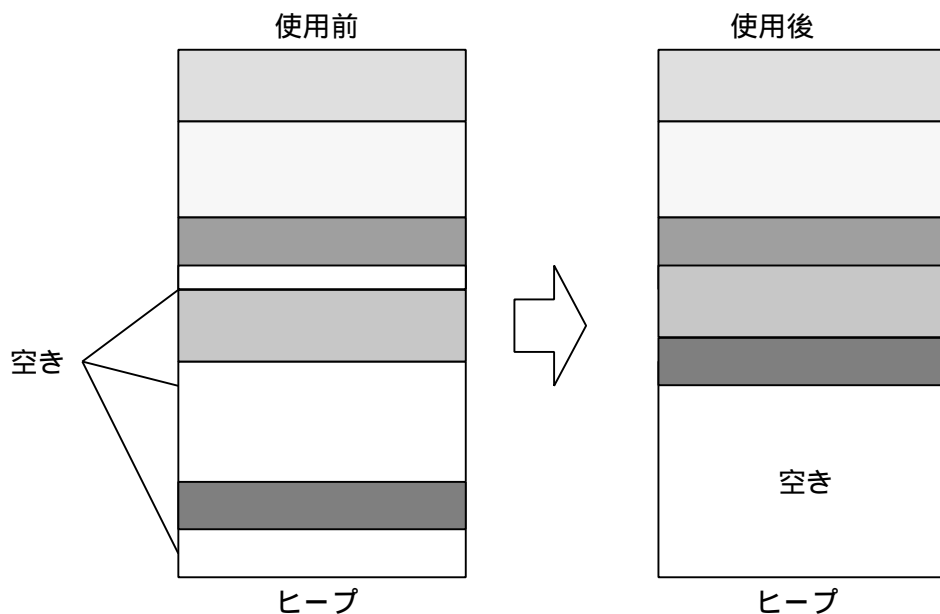
この例で、malloc に対して 100KB を要求した場合、First Fit では、最初に候補にあがった、101KB の未使用領域から 100KB を確保し未使用領域 1KB が残ります。

もちろん、未使用領域の前後が free されれば、未使用 1KB は、他のメモリブロックとマージされますが、First Fit アルゴリズムにはこのような微少なメモリブロックがヒープ上に散在する可能性があるのです。

malloc 関数を使用する場合、アルゴリズムを熟知し、それに合わせたメモリ確保の方法を取ることが非常に大切です。

ガーベージコレクションについて

ヒープ領域の分断された状態で、空き領域と、使用領域をグループ化して一個所にまとめてしまう事を、「ガーベージコレクション」といいます。



< ガーベージコレクション >

ガーベージコレクションをすることで、分断されていた空きメモリを連続した空間に再配置できるため、連続空き領域が増えるという特徴があります。

しかしながら SEGA ライブラリではメモリのガーベージコレクションはサポートいたしません。

理由としては、ガーベージコレクションによって使用中のメモリブロックの先頭アドレスが変わるためです。

MMU を使用していない SEGA ライブラリ環境下では、一旦割り当てたメモリ領域の先頭アドレスに変更があったことを知らせる手段が無いため、アプリケーションが誤動作します。