بخش چهارم

مدیریت تخصیص حافظه در سیستم عامل الگوریتم تخصیص حافظه همراه با مثال

مبحث **مدیریت تخصیص حافظه در سیستم عامل** یکی از مباحث محوری در علوم کامپیوتر به شمار می رود. هر فرآیند برای اجراشدن باید دارای حافظه مناسب و اختصاصی باشد که مدیریت تخصیص حافظه در سیستم‌عامل در این حوزه عمل می کند و با ارائه الگوریتم های تخصیص حافظه برای افزایش کارایی، این مهم را برآورده می سازد. در این پست با مدیریت تخصیص حافظه در سیستم عامل بیشتر آشنا می‌شویم.

فهرست مطالب

مقدمه

مهم‌ترین عملکرد یک سیستم عامل این است که حافظه اصلی را مدیریت کند . این عمل به فرآیندها کمک می کند تا بین حافظه اصلی و دیسک در حین اجرا به عقب و جلو حرکت کنند و به سیستم عامل کمک می کند تا هر مکان حافظه را ردیابی کند، صرف نظر از اینکه به یک فرآیند اختصاص داده شده یا آزاد باشد. در یک تعریف کلی، تخصیص حافظه فرآیندی است که طی آن به برنامه‌های رایانه‌ای حافظه یا فضا اختصاص می‌یابد.

وقتی فرآیندها در یک **لیست مرتب‌شده** بر اساس نشانی قرار می گیرند، الگوریتم‌های مختلفی جهت تخصیص حافظه به یک فرآیند وجود دارد که از آن ها با عنوان **الگوریتم های تخصیص حافظه در سیستم عامل** یاد می‌شود که در ادامه به بررسی آن ها خواهیم پرداخت.

تخصیص حافظه چیست؟

منظور از حافظه ی کامپیوتر، مجموعه‌ای از داده هاست که به شکل باینری نشان داده می شوند و در یک حافظه‌ی فیزیکی در داخل کامپیوتر قرار دارند . حافظه‌ی اصلی که به‌عنوان حافظه‌ی RAM هم شناخته می‌شود بخشی از این حافظه است که جدا از سایر دستگاه های ذخیره‌سازی انبوه خارجی مانند درایو دیسک می‌باشد. هر برنامه‌ای که می خواهیم اجرا شود یا هر فایلی که می خواهیم به آن دسترسی پیدا کنیم، باید از دستگاه‌های ذخیره‌سازی به حافظه‌ی اصلی کپی شود تا اجرا گردد.

به عبارتی تمام برنامه ها برای اجرا در حافظه ی اصلی بارگذاری می شوند که در این هنگام با چند حالت روبرو می‌شویم:

- **حالت اول :**برنامه کامل در حافظه‌ی اصلی بارگذاری می‌شود.
- **حالت دوم :**قسمتی یا روال خاصی از برنامه، آن هم تنها زمانی که توسط برنامه فراخوانی می‌شود در حافظه ی اصلی بارگذاری می گردد، این مکانیسم را **(Dynamic Loading) بارگذاری پویا** می‌نامند که باعث افزایش کارایی می‌شود.

- **حالت سوم :**گاهی اوقات یک برنامه به برنامه دیگری وابسته است . در چنین حالتی، CPU به‌جای بارگیری همه برنامه های وابسته، برنامه های وابسته را در صورت نیاز به برنامه اصلی اجراکننده پیوند می‌دهد. این مکانیسم به **پیوند پویا** معروف است.

مدیریت تخصیص حافظه در سیستم عامل چیست؟

برای بهینه کردن عملکرد سیستم و همچنین هماهنگی و کنترل حافظه ی کامپیوتر فرآیندی صورت می‌گیرد، به این‌ترتیب که حافظه بخش هایی را به عنوان بلوک به برنامه های مختلف در حال اجرا اختصاص می‌دهد، این فرآیند را **مدیریت تخصیص حافظه در سیستم عامل** می‌نامند.

مدیریت تخصیص حافظه در سیستم عامل شامل چهار وظیفه زیر است:

۱- نظارت بر وضعیت هر یک از مکان های حافظه اصلی :یعنی نظارت بر اینکه کدام مکان تخصیص‌یافته و کدام یک تخصیص نیافته (آزاد) است.

۲- تعیین سیاست تخصیص حافظه :یعنی تصمیم‌گیری در مورد اینکه حافظه به کدام فرآیند، چه مقدار از آن، چه هنگام و کجا باید اختصاص یابد؟ چنانچه حافظه اصلی باید به طور هم‌روند بین چند فرآیند تقسیم شود، در این صورت مدیریت حافظه باید تعیین کند که تقاضای کدام فرآیندها اجابت گردد.

۳- شیوه تخصیص :پس‌ازآنکه تصمیم به تخصیص حافظه گرفته شد، نشانی های خاص باید انتخاب شده و اطلاعات مربوط به تخصیص به‌روزرسانی شوند.

۴- شیوه و سیاست بازیابی حافظه :یعنی اقدام در مورد بازیابی حافظه . فرآیند ممکن است حافظه تخصیص‌یافته از پیش را، خود آزاد کند و یا اینکه مدیریت حافظه به طور یک جانبه و برمبنای یک سیاست بازیابی آن را آزاد کند . پس از بازیابی، اطلاعات مربوط به وضعیت حافظه باید به روزرسانی شود.

الگوریتم های تخصیص حافظه در سیستم عامل

۱. الگوریتمFirst Fit

۲. الگوریتمNext fit
۳. الگوریتمBest fit
۴. الگوریتمWorst fit
۵. الگوریتمQuick fit

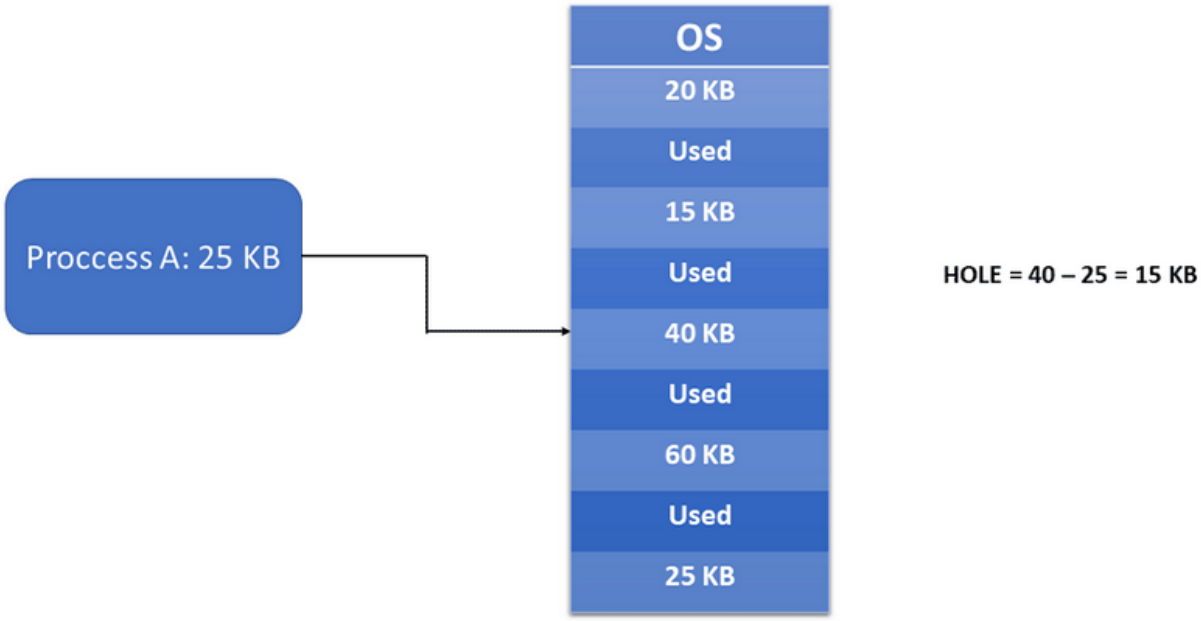
۱ -الگوریتم (First Fit) اولین برازش

در این الگوریتم از ابتدای حافظه شروع به اسکن می کند، اولین حفره‌ای که به‌اندازه‌ی کافی بزرگ باشد و بتواند در آن جای بگیرد را انتخاب کرده و تخصیص می دهد. این الگوریتم ساده بوده و سریع ترین الگوریتم می باشد زیرا تا حد امکان کمتر جستجو م ی‌کند، همچنین کارایی آن نیز مناسب است . درعین‌حال معایبی هم دارد ازجمله اینکه، مناطقی از حافظه که پس از تخصیص استفاده نشده باقیمانده‌اند اگر خیلی کوچک تر باشد، تبدیل به زباله می شوند. بنابراین درخواست برای نیاز به حافظه بزرگ تر نمی‌تواند انجام شود.

مثلا درخواست‌های فرآیندها را به ترتیب 300K، 25K، 125Kو ۵۰ Kدر نظر بگیرید. بگذارید دو بلوک حافظه با اندازه 150Kو سپس یک بلوک با اندازه ۳۵۰ Kموجود باشد.

300K درخواست به بلوک ۳۵۰ K اختصاص داده می‌شود، ۵۰ K باقیمانده حذف می‌شود. ۲۵ K به بلوک ۱۵۰ K اختصاص داده می شود، 125Kمابقی حذف می شود. سپس ۱۲۵ K و ۵۰ K به پارتیشن‌های باقی‌مانده اختصاص داده می‌شوند.

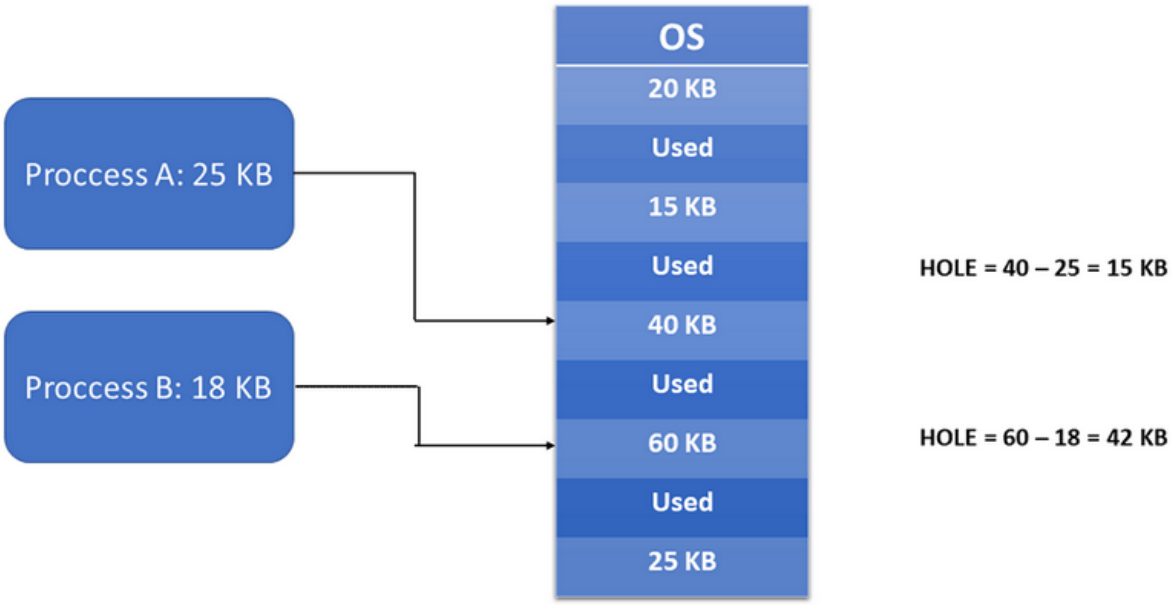
مثالی دیگر با شکل:



۲ الگوریتم(Next fit)برازش بعدی

دومین الگوریتم تخصیص حافظه در سیستم عامل Next fit می‌باشد که نسخه اصلاح شده first fit است. برای یافتن یک پارتیشن آزاد با همان اندازه مناسب شروع می شود. هنگامی‌که دفعه بعد فراخوانی می‌شود، جستجو را از ابتدا شروع نمی‌کند بلکه از همان‌جایی که توقف کرده شروع می‌کند.

همانطوریکه در شکل زیر مشاهده می کنید فرآیند A بر اساس الگوریتم First Fit از ابتدا شروع به اسکن کرده و در اولین حفره مناسب قرارگرفته است . فرآیند B که بعد از فرآیند A واردشده است از همان‌جایی که A در جستجو مانده بود ادامه داده و در اولین حفره مناسب قرار می گیرد که این عملکرد با عنوان الگوریتم Next Fit شناخته می‌شود.



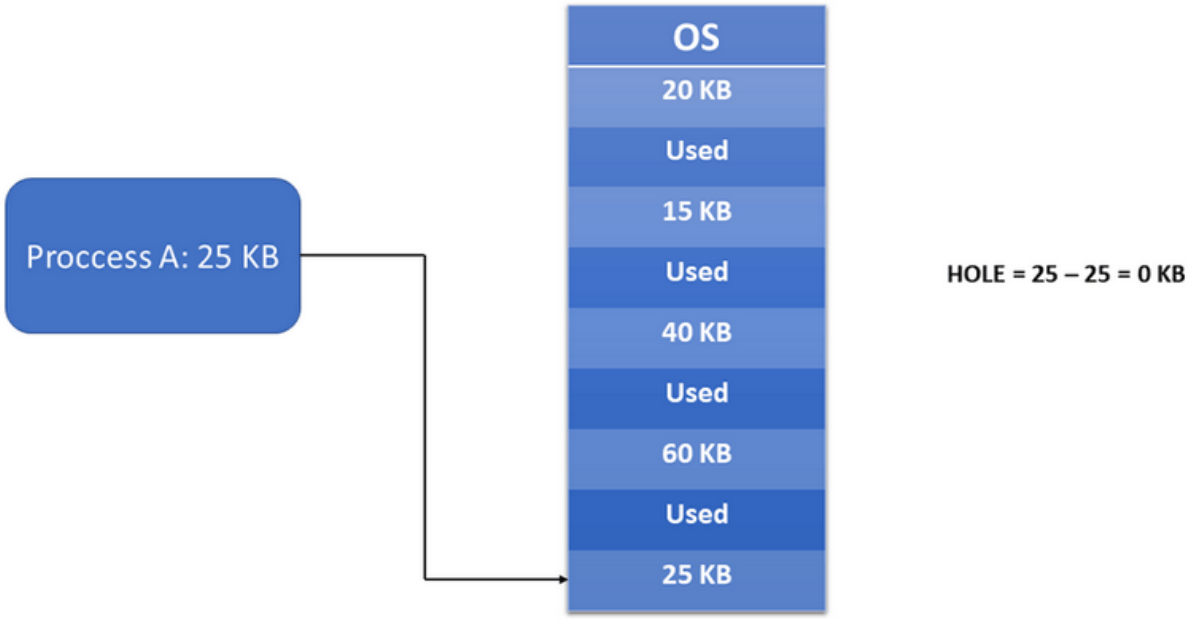
۳ الگوریتم(Best Fit)بهترین برازش

نوع دیگری از الگوریتم های تخصیص حافظه در سیستم عامل Best Fit یا بهترین برازش یا بهترین تناسب است. این الگوریتم، کوچک ترین پارتیشن آزادی را که نیازهای فرآیند درخواست شده را برآورده می کند، تخصیص می‌دهد. این الگوریتم ابتدا کل لیست پارتیشن های آزاد را جستجو می کند و کوچک‌ترین حفره‌ای را که برای فرآیند کافی است در نظر می گیرد. سپس سعی می کند حفره‌ای را پیدا کند که با اندازه واقعی موردنیاز فرآیند یکی باشد.

مزیت آن نسبت به First Fit این است که استفاده از حافظه در آن بسیار بهتر است، زیرا ابتدا کوچک‌ترین پارتیشن آزاد موجود را جستجو می کند اما نسبت به آن کندتر است و حتی ممکن است حافظه را با حفره‌های کوچک بی‌فایده پر کند.

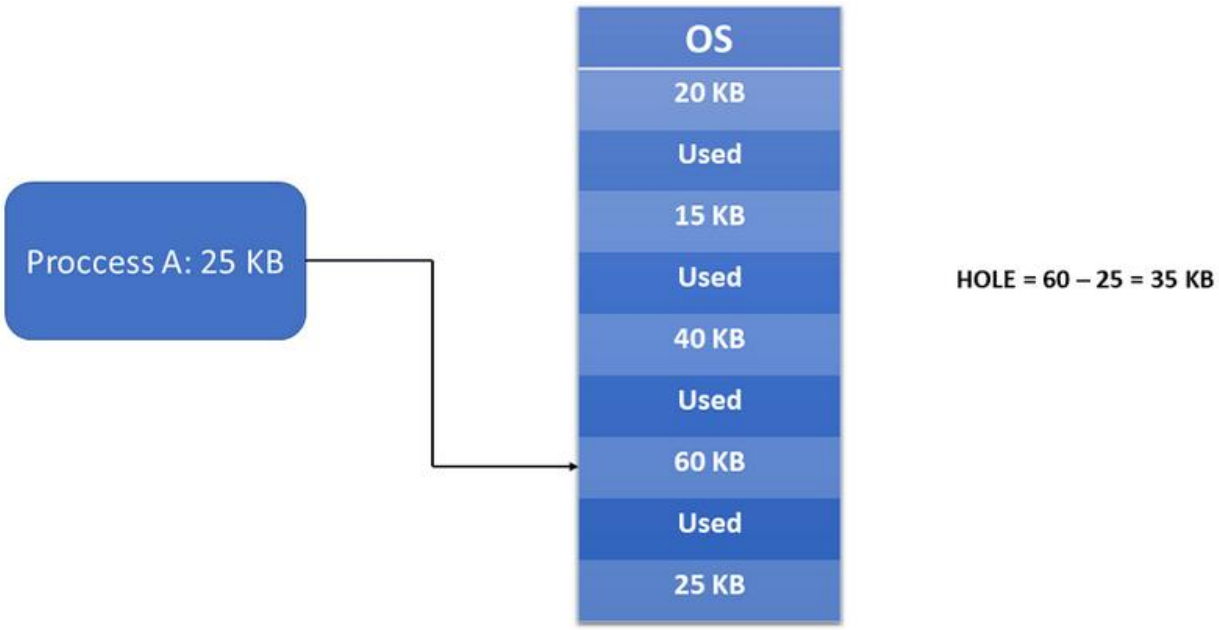
مثلا درخواست‌های فرآیندها را به ترتیب 300K، 25K، 125K و ۵۰ Kدر نظر بگیرید. بگذارید دو بلوک حافظه با اندازه 150K و سپس یک بلوک با اندازه ۳۵۰ Kموجود باشد.

300Kبه یک بلوک با اندازه ۳۵۰ Kاختصاص داده می‌شود. ۵۰ kدر بلوک باقی خواهد ماند. ۲۵ Kبه ۵۰ Kباقیمانده از بلوک قبل تخصیص داده می شود. ۲۵ kدر بلوک باقی خواهد ماند . ۱۲۵ Kبه بلوک ۱۵۰ Kاختصاص داده می شود. ۲۵ kدر این بلوک نیز باقی خواهد ماند . ۵۰ Kنمی‌تواند اختصاص داده شود حتی اگر فضای ۲۵ K + 25K در دسترس باشد.



۴ الگوریتم (Worst Fit) بدترین برازش

این الگوریتم کاملاً برعکس الگوریتم Best Fit بوده و بدترین نوع الگوریتم های تخصیص حافظه در سیستم عامل است. در این الگوریتم تمام لیست را جستجو می کند و بزرگ‌ترین حفره‌ی آزاد موجود را پیداکرده و تخصیص می دهد. قسمت بلقیمانده آن به اندازه کافی بزرگ است که بتوان بعدها از آن دوباره استفاده مفید کرد.



۵ الگوریتم (Quick Fit) برازش سریع

پنجمین الگوریتم تخصیص حافظه در سیستم عامل، الگوریتم Quick Fit (برازش سریع) رویکرد متفاوتی نسبت به الگوریتم هایی دارد که تاکنون در نظر گرفته ایم. لیست‌های جداگانه‌ای برای برخی از اندازه‌های حافظه رایج درخواست شده، نگهداری می شود. برای مثال، می توانیم فهرستی برای حفره های K۴، فهرستی برای حفره هایی با اندازه ۸ Kو غیره داشته باشیم. یک فهرست را می توان برای حفره های بزرگ یا حفره‌هایی که در هیچ یک از لیست‌های دیگر قرار نمی گیرند، نگه داشت.

درQuick Fit ، قبل از اینکه تصمیم بگیرید از سیستم عامل فضای ذخیره‌سازی بیشتری بخواهید، ادغام انجام می شود. سپس ببینید آیا اکنون یک بلوک به اندازه کافی بزرگ دارید یا خیر؟ اگر این کار را نکنید، فضای بیشتری از سیستم دریافت خواهید کرد . توجه داشته باشید که Quick Fit، بر خلاف First Fitو Best Fit ، تمایل دارد تا تکه تکه شدن حافظه آزاد را به حداقل برساند که عملکرد کلی سیستم را بهبود می بخشد. اگرچه یافتن حفره مناسب سریع است اما هر گاه یک فرآیند خاتمه می یابد، عمل ترکیب حفره‌ها بسیار وقت گیر خواهد بود.

۶ الگوریتم (Buddy’s system) سیستم دوستان

در سیستم دوستان، انداز ه بلوک‌های آزاد به‌صورت توان یکپارچه ۲ است: ۲، ۴، ۸، ۱۶ و غیره تا اندازه حافظه . هنگامی‌که یک بلوک آزاد با اندازه 2^k درخواست می شود، یک بلوک آزاد از لیست بلوک‌های آزاد با اندازه 2^k اختصاص داده می شود. اگر بلوک آزاد با اندازه $2k$ در دسترس نباشد، بلوک با اندازه بزرگ تر بعدی، یعنی 2^{k+1} به دو نیمه به نام دوستان تقسیم می شود تا درخواست را برآورده کند.

مثال:

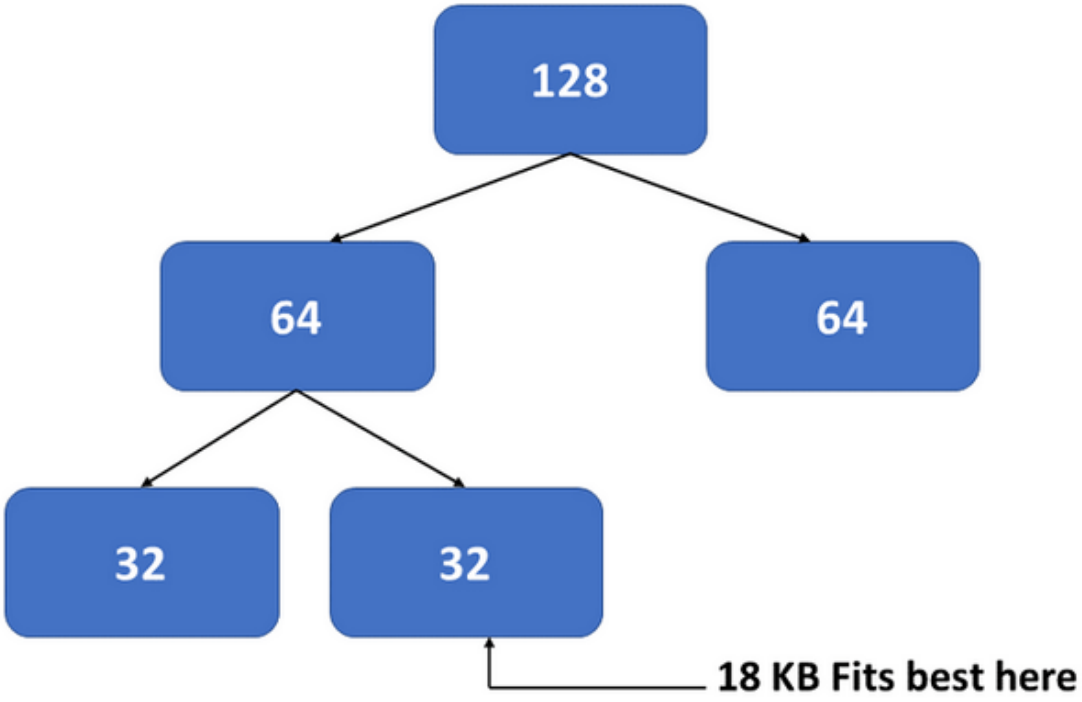
فرض کنید اندازه کل حافظه ۵۱۲ کیلوبایت باشد و فرآیند P1 ، به ۷۰ کیلوبایت نیاز دارد تا تخصیص داده شود .ازآنجایی‌که لیست های حفره فقط برای توان های ۲ هستند، ۱۲۸ کیلوبایت به اندازه کافی بزرگ خواهد بود . در ابتدا هیچ ۱۲۸ کیلوبایتی وجود ندارد و بلوک های ۲۵۶ کیلوبایتی نیز وجود ندارند. بنابراین بلوک ۵۱۲ کیلوبایتی به دو دوست با حجم ۲۵۶ کیلوبایت تقسیم می‌شود، یکی از آن‌ها به دو بلوک ۱۲۸ کیلوبایتی تقسیم می‌شود و یکی به فرآیند اختصاص می یابد. فرآیند بعدی با نام P2 به ۳۵ کیلوبایت نیاز دارد . برای گرد کردن ۳۵ کیلوبایت تا توان ۲، یک بلوک ۶۴ کیلوبایتی موردنیاز است.

وقتی بلوک ۱۲۸ کیلوبایتی به دو دوست ۶۴ کیلوبایتی تقسیم و دوباره فرآیندی دیگر مثل P3 (130KB)در کل ۲۵۶ KBتنظیم می‌شود. پس از ارضای درخواست به این صورت، زمانی که چنین بلوکی آزاد است، می توان دو بلوک /رفیق را دوباره باهم ترکیب کرد تا بلوک اصلی دو برابر بزرگ تر را تشکیل دهند، وقتی‌که نیمه دوم هم آزاد است.

مزیت این الگوریتم تخصیص حافظه در س یستم عامل این است که سیستم دوستان نسبت به سایر الگوریتم‌ها سریع‌تر است .هنگامی‌که یک بلوک با اندازه $2k$ آزاد می‌شود، یک حفره با اندازه حافظه $2k$ جستجو می‌شود تا بررسی شود که آیا امکان ادغام وجود دارد یا خیر؟، درحالی‌که در الگوریتم‌های دیگر تمام لیست حفره‌ها باید جستجو شوند.

آخرین الگوریتم تخصیص حافظه در سیستم عامل (Buddy's system)، معایبی هم دارد ازجمله اینکه، اغلب ازنظر استفاده از حافظه به تدریج ناکارآمد می شود. ازآنجایی‌که همه درخواست ها باید به توان ۲ گرد شوند، یک فرآیند ۳۵ کیلوبایتی به ۶۴ کیلوبایت اختصاص داده می شود، بنابراین ۲۹

کیلوبایت اضافی هدررفته و باعث تکه تکه شدن داخلی می شود. ممکن است حفره هایی هم بین دوستان وجود داشته باشد که باعث تکه‌تکه شدن خارجی شود.



برنامه الگوریتم Buddy System در C# به صورت زیر است:

```
using System;
using System.Collections.Generic;
public class Buddy
{
// Inner class to store lower
// and upper bounds of the
// allocated memory
class Pair
```

```
{
public int lb, ub;
public Pair(int a, int b)
{
lb = a;
ub = b;
}
}
// Size of main memory
int size;
// Array to track all
// the free nodes of various sizes
List<Pair> []arr;
// Else compiler will give warning
// about generic array creation
Buddy(int s)
{
size = s;
// Gives us all possible powers of 2
int x = (int)Math.Ceiling(Math.Log(s) /
Math.Log(2));
// One extra element is added
// to simplify arithmetic calculations
arr = new List<Pair>[x + 1];
for (int i = 0; i <= x; i++)
arr[i] = new List<Pair>();
// Initially, only the largest block is free
// and hence is on the free list
arr[x].Add(new Pair(0, size - 1));
}
void allocate(int s)
{
// Calculate which free list to search
// to get the smallest block
// large enough to fit the request
int x = (int)Math.Ceiling(Math.Log(s) /
Math.Log(2));
int i;
Pair temp = null;
```

```
// We already have such a block
if (arr[x].Count > 0)
{
// Remove from free list
// as it will be allocated now
temp = (Pair)arr[x][0];
arr[x].RemoveAt(0);
Console.WriteLine("Memory from " + temp.lb +
" to " + temp.ub + " allocated");
return;
}
// If not, search for a larger block
for (i = x + 1; i < arr.Length; i++)
{
if (arr[i].Count == 0)
continue;
// Found a larger block, so break
break;
}
// This would be true if no such block
// was found and array was exhausted
if (i == arr.Length)
{
Console.WriteLine("Sorry, failed to" +
" allocate memory");
return;
}
// Remove the first block
temp = (Pair)arr[i][0];
arr[i].RemoveAt(0);
i--;
// Traverse down the list
for (; i >= x; i--)
{
// Divide the block in two halves
// lower index to half-1
Pair newPair = new Pair(temp.lb, temp.lb +
(temp.ub - temp.lb) / 2);
// half to upper index
```

```
Pair newPair2 = new Pair(temp.lb + (temp.ub -
temp.lb + 1) / 2, temp.ub);
// Add them to next list which is
// tracking blocks of smaller size
arr[i].Add(newPair);
arr[i].Add(newPair2);
// Remove a block to continue
// the downward pass
temp = (Pair)arr[i][0];
arr[i].RemoveAt(0);
}
// Finally inform the user
// of the allocated location in memory
Console.WriteLine("Memory from " + temp.lb +
" to " + temp.ub + " allocated");
}
// Driver Code
public static void Main(String []args)
{
int initialMemory = 0;
initialMemory = 128;
// Initialize the object with main memory size
Buddy obj = new Buddy(initialMemory);
obj.allocate(32);
obj.allocate(7);
obj.allocate(64);
obj.allocate(56);
}
}
// This code is contributed by 29AjayKumar
```

سخن پایانی درمورد مدیریت تخصیص حافظه در سیستم عامل

مهم‌ترین وظیفه سیستم عامل مدیریت حافظه ی اصلی و حافظه ی دیسک است. این مدیریت با استفاده از الگوریتم‌های تخصیص حافظه در سیستم عامل صورت می گیرد و منجر به عملکرد بهینه سیستم می شود. هر برنامه و فرآیندی برای اجراشدن باید در حافظه اصلی کپی گردد و این وظیفه سیستم عامل است که

تعیین می‌کند کدام فرآیند به چه ترتیبی اجرا شود. هر یک از الگوریتم‌ها هم متناسب با نیازهای سیستم و بر اساس معایب و مزایای خود به‌کار گرفته شده و برنامه یا فرآیندها بر اساس آن‌ها اجرا می‌شوند.