الگوریتم Banker

بردیا نیک بخش

سیستم‌های عامل

استاد استادزاده

فهرست مطالب

الگوریتمBanker

الگوریتم Banker یک الگوریتم است که در سیستم‌های عامل و مدیریت منابع استفاده می‌شود تا از وقوع قحطی منابع جلوگیری کند. این الگوریتم به صورت خاص در محیط‌های چندپردازنده و چندکاربره مورد استفاده قرار می‌گیرد.

هدف اصلی الگوریتمBanker ، اجازه دادن به فرآیندها برای اجرای همزمان و ایمن استفاده از منابع را با حفظ وجودی منابع کافی در سیستم است. یکی از ویژگی‌های مهم این الگوریتم، این است که به صورت پیشبینی‌گر است و بر اساس وضعیت فعلی سیستم و نیازمندی‌های فرآیندها، تصمیم‌گیری می‌کند.

برای کار با الگوریتمBanker ، ابتدا باید نیازمندی‌های هر فرآیند را مشخص کنید. سپس برای هر منبع، تعداد منابع موجود در سیستم را وارد کرده و تخصیص آن‌ها به فرآیندها را مشخص می‌کنید. الگوریتم سپس بررسی می‌کند که آیا با تخصیص پیشنهادی از منابع، سیستم در وضعیت امن باقی می‌ماند یا خیر.

اگر تخصیص منابع باعث وارد شدن سیستم به وضعیت ناامن می‌شود (مثلاً باعث قحطی منابع می‌شود)، تخصیص رد می‌شود و فرآیند به صف انتظار می‌رود تا منابع لازم را در دسترس داشته باشد. اما اگر تخصیص منابع به صورت ایمن انجام شود و وضعیت سیستم پس از تخصیص منابع همچنان امن باشد، تخصیص انجام می‌شود و فرآیند مجاز به ادامه اجرا است.

با استفاده از الگوریتم Banker، به صورت عملی می‌توانیم از وقوع قحطی منابع در سیستم‌های چندپردازنده جلوگیری کنیم و اجرای همزمان و بدون تداخل فرآیندها را تضمین کنیم.

الگوریتم Banker بر اساس دو مفهوم اصلی کار می‌کند: منابع تخصیص یافته و منابع درخواستی .

منابع تخصیص یافته، مجموعه‌ای از منابع است که به هر فرآیند تخصیص داده شده است. این منابع می‌توانند شامل مثلاً حافظه، پردازنده و دستگاه‌های ورودی/خروجی باشند. هر فرآیند در زمان شروع خود، نیازمندی‌های منابع را برای اجرای خود اعلام می‌کند.

منابع درخواستی، نیازمندی‌های منابعی هستند که فرآیند در طول اجرای خود ممکن است داشته باشد. یعنی در طول زمان، فرآیند برخی منابع را درخواست می‌کند و سیستم باید بتواند این درخواست‌ها را برآورده کند.

الگوریتم Banker برای هر فرآیند، یک بردار نشانگر استفاده شده برای نشان دادن تعداد منابعی که آن فرآیند در حال استفاده از آن‌ها است، و یک بردار درخواستی برای نشان دادن تعداد منابعی که آن فرآیند قصد استفاده از آن‌ها را دارد، در نظر می‌گیرد.

زمانی که یک فرآیند درخواستی جدید دارد، الگوریتم Banker با بررسی این درخواست و بررسی وضعیت سیستم، تصمیم می‌گیرد که آیا این درخواست باید تخصیص داده شود یا خیر. الگوریتم بررسی می‌کند که آیا پس از تخصیص منابع به فرآیند، سیستم در وضعیت امن باقی می‌ماند یا نه. اگر وضعیت سیستم پس از تخصیص هنوز امن است، درخواست تأیید می‌شود و منابع به فرآیند اختصاص پیدا می‌کنند. اگر وضعیت سیستم ناامن می‌شود، درخواست رد می‌شود و فرآیند به صف انتظار منابع اضافه می‌شود تا زمانی که منابع مورد نیاز در دسترس قرار بگیرند.

با استفاده از الگوریتم Banker، سیستم می‌تواند با هوشمندانه تخصیص منابع به فرآیندها، از وقوع قحطی منابع جلوگیری کند و همزمانی و ایمنی در اجرای فرآیندها را تضمین کند.

به منظور به کارگیری الگوریتم Banker، سه چیز لازم به ذکر است:

- هر فرایند چقدر از هر منبع می تواند نیاز داشته باشد.
- هر فرایند چقدر از هر منبع را دردست دارد.
- هر سیستم چقدر از هر منبع را موجود دارد.

مراحل الگوریتم Banker

1. محاسبه ماتریس Need یا نیاز آتی فرایندها طبق فرمول زیر:

$$Need = MAX - Allocation$$

2. پیدا کردن سطری از ماتریس Need که در آن رابطه زیر برقرار است و اضافه کردن فرایند مربوطه به دنباله جواب.

$$Need_i \leq Available$$

***i*: شماره سطر مربوطه در ماتریس Need**

3. به‌روزرسانی ماتریس Available مطابق با فرمول زیر و تکرار گام 2 تا این که همه سطرها حذف شوندیا دیگر هیچ سطری در رابطه بالا صدق نکند.

$$Available_{New} = Availabe_{Old} + Allocation$$

***i*: شماره سطری که در گام 2 پیدا شد.**

4. به‌روزرسانی ماتریس Available مطابق با فرمول زیر و تکرار گام 2 تا این که همه سطرها حذف شوندیا دیگر هیچ سطری در رابطه بالا صدق نکند.

5. اگر همه فرایندها در دنباله جواب وجود داشته‌باشند، دنباله حاصل شده یک دنباله امن است ← حالت امن سیستم ← احتمال بن‌بست صفر است. (بن‌بست در سیستم وجود ندارد)

پیاده سازی الگوریتم Banker با زبان پایتون

برای محاسبه ماتریس need از تابع زیر استفاده شده:

```
def calculate_need(n,m,max_need,allocated):
    need = []
    for i in range(n):
        row = []
        for j in range(m):
            row.append(max_need[i][j] - allocated[i][j])
        need.append(row)
    return need
```

این تابع به ازای هر یک از پراسس های داده شده و متناسب با ریسورس مشخص شده جدول need را با فرمول Need=MAX-Allocation محاسبه کرده و در متغیری به نام need که دیتاتایپ آن از نوع آرایه دوبُعدی است ذخیره می‌کند.

تابع بعدی، تابع اصلی برنامه است که مشخص میکند که آیا سیستم در حالت امن یا Safe قرار دارد یا نه.

```
def is_safe(processes, available, max_need, allocated):
    n = len(processes)
    m = len(available)
    finish = [False] * n
    work = available[:]
    need = calculate_need(n,m,max_need,allocated)
```

n متغیری است که تعداد process ها را در خود ذخیره میکند.

m متغیری است که تعداد available ها را در خود ذخیره میکند.

Finish آرایه ای است که همه ی مقادیر اولیه آن false مقداردهی شده.

پس از تعریف کردن متغیر ها و مقدار دهی آنها تابع calculate_need() که توضیح داده شد فراخوانی می‌شود تا جدول need رسم گردد.

```
count = 0
while count < n:
    found = False
    for p in range(n):
        if not finish[p]:
            flag = True
            for j in range(m):
                if need[p][j] > work[j]:
                    flag = False
                    break
            if flag:
                for j in range(m):
                    work[j] += allocated[p][j]
                finish[p] = True
                found = True
                count += 1
    if not found:
        return False
return True
```

سپس در قدم بعدی با استفاده از حلقه while فرایندی که باید اجرا شود از طریق فرمول $Need_i \leq Available$ پیدا میگردد. سپس این کار تا زمانی که همه سطر ها حذف شوند یا هیچ سطری در این رابطه صدق نکند.

نحوه پیاده سازی نیز به این شکل است که اگر رابطه need[p][j] > work[j] صدق کند به دنباله جواب افزوده میشود در غیر این صورت متغیر flag برابر True میشود و در این حالت ماتریس available بر اساس مقادیر موجود و با استفاده از رابطه work[j]+=allocated[p][j] بازنویسی میشود.

در نهایت اگر متغیر found برابر با False بود (یعنی دنباله جواب خالی باشد) به این معنی است که سیستم در حالت غیرامن یا unsafe قرار دارد

تست برنامه

مثال : سیستمی با ۵ فرایند P_0 تا P_4 و سه منبع A، B و C را درنظر بگیرید. منبع نوع A دارای ۱۰ نمونه، نوع B دارای ۵ نمونه و نوع C دارای ۷ نمونه است. اگر وضعیت سیستم در ابتدای کار مطابق با جدول زیر باشد، وضعیت سیستم را از لحاظ وجود بن‌بست در آن بررسی نمایید.

	Allocation			MAX		
	A	B	C	A	B	C
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3

test.py > ...

```
processes = [0, 1, 2, 3, 4]
max_need = [
    [7, 5, 3],
    [3, 2, 2],
    [9, 0, 2],
    [2, 2, 2],
    [4, 3, 3]
]
allocated = [
    [0, 1, 0],
    [2, 0, 0],
    [3, 0, 2],
    [2, 1, 1],
    [0, 0, 2]
]

available = [3, 3, 2]

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Python

```
PS D:\Desktop\Bardia\Banker Algorithm> & D:/apps/Python311/python.exe "d:/Desktop/Bardia/Banker Algorithm/test.py"
The system is in a safe state.
PS D:\Desktop\Bardia\Banker Algorithm>
```