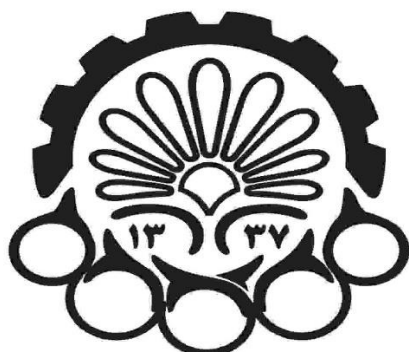


به نام خدا



دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

آزمایشگاه سیستم های عامل

آزمایش دهم : DeadLock

اعضای گروه :

محمد امین فرح بخش - (40131029)

حسین تاتار - (40133014)

اردیبهشت 1404

این برنامه پیاده‌سازی الگوریتم بانکدار (Banker's Algorithm) برای مدیریت منابع در سیستم عامل است. الگوریتم بانکدار برای جلوگیری از بن‌بست (deadlock) در تخصیص منابع استفاده می‌شود. برنامه ما یک شبیه‌ساز چندرسمانی (multi-threaded) است که در آن مشتری‌ها (یا فرایندها) منابعی را درخواست و آزاد می‌کنند و الگوریتم تصمیم می‌گیرد که آیا اعطای منابع امن است یا نه. حالا به سراغ توضیح می‌رویم:

```
7
8 #define NUMBER_OF_RESOURCES 5
9 #define NUMBER_OF_CUSTOMERS 5
10
```

تعداد کل منابع: ۵ نوع مختلف

تعداد مشتری: ۵ فرایند یا ترد

آرایه‌های مهم:

معنی	نام متغیر
تعداد منابع آزاد از هر نوع	available[]
بیشترین تعداد منابعی که هر مشتری ممکن است نیاز داشته باشد	maximum[][]
منابعی که هم اکنون به هر مشتری تخصیص داده شده	allocation[][]
نیاز باقیمانده‌ی هر مشتری = maximum - allocation	need[][]

```
17
18 // Initialize the data structures
19 void initialize() {
20     srand(time(NULL));
21     for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
22         for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
23             allocation[i][j] = 0;
24             maximum[i][j] = rand() % (available[j] + 1);
25             need[i][j] = maximum[i][j];
26         }
27     }
28 }
29
```

مقداردهی اولیه تصادفی برای نیاز مشتری‌ها:

• allocation = 0

- $\text{maximum} = \text{rand}() \% (\text{available}[j] + 1)$ حداکثر درخواست تصادفی کمتر یا مساوی مقدار

موجود

- $\text{need} = \text{maximum}$

```

C BankersAlgorithm.c X
C: > Users > lenovo > Downloads > Eitaa Desktop > C BankersAlgorithm.c
31 bool is_safe() {
32     int work[NUMBER_OF_RESOURCES];
33     bool finish[NUMBER_OF_CUSTOMERS] = { false };
34
35     memcpy(work, available, sizeof(available));
36
37     for (int count = 0; count < NUMBER_OF_CUSTOMERS; count++) {
38         for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
39             if (!finish[i]) {
40                 bool can_allocate = true;
41                 for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
42                     if (need[i][j] > work[j]) {
43                         can_allocate = false;
44                         break;
45                     }
46                 }
47                 if (can_allocate) {
48                     for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
49                         work[j] += allocation[i][j];
50                     }
51                     finish[i] = true;
52                 }
53             }
54         }
55     }
56
57     for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
58         if (!finish[i]) return false;
59     }
60     return true;
61 }

```

الگوریتم ایمنی:

- تقلید می‌کند که آیا می‌توان به ترتیب خاصی منابع را آزاد کرد که هیچ بن‌بستی رخ ندهد.
- از آرایه `work` برای بررسی منابع فعلاً قابل استفاده استفاده می‌کند.
- از `finish[]` برای مشخص کردن مشتری‌هایی که کارشان تمام شده استفاده می‌کند.
- هر مشتری‌ای که $\text{need} \leq \text{work}$ داشته باشد، منابعش آزاد می‌شود.
- در انتها اگر همه مشتری‌ها `finish = true` شده باشند، سیستم در وضعیت ایمن است.

```

C BankersAlgorithm.c X
C:\Users\lenovo\Downloads\Eitaa Desktop > C BankersAlgorithm.c
62
63 int request_resources(int customer_num, int request[]) {
64     pthread_mutex_lock(&lock);
65
66     for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
67         if (request[i] > need[customer_num][i]) {
68             pthread_mutex_unlock(&lock);
69             return -1; // Request exceeds need
70         }
71         if (request[i] > available[i]) {
72             pthread_mutex_unlock(&lock);
73             return -1; // Not enough resources available
74         }
75     }
76
77     // Pretend to allocate
78     for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
79         available[i] -= request[i];
80         allocation[customer_num][i] += request[i];
81         need[customer_num][i] -= request[i];
82     }
83
84     if (is_safe()) {
85         pthread_mutex_unlock(&lock);
86         return 0;
87     } else {
88         // Rollback
89         for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
90             available[i] += request[i];
91             allocation[customer_num][i] -= request[i];
92             need[customer_num][i] += request[i];

```

بررسی می‌کند که آیا درخواست معتبر است:

- بیشتر از نیاز نباشد.

- بیشتر از منابع موجود نباشد.

سپس به صورت «موقتی» منابع را تخصیص می‌دهد.

اگر سیستم ایمن باقی بماند، تخصیص نهایی می‌شود.

اگر نه، «rollback» می‌کند.

```

98
99 int release_resources(int customer_num, int release[]) {
100     pthread_mutex_lock(&lock);
101     for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
102         if (release[i] > allocation[customer_num][i])
103             release[i] = allocation[customer_num][i];
104         available[i] += release[i];
105         allocation[customer_num][i] -= release[i];
106         need[customer_num][i] += release[i];
107     }
108     pthread_mutex_unlock(&lock);
109     return 0;
110 }
111

```

منابع تخصیص‌یافته توسط مشتری را آزاد می‌کند.

اگر مقدار release بیشتر از allocation باشد، آن را محدود می‌کند.

سپس:

- available را زیاد می‌کند.

- allocation را کم می‌کند.
- need را دوباره افزایش می‌دهد.

```

111
112 void* customer_thread(void* arg) {
113     int id = *(int*)arg;
114     int request[NUMBER_OF_RESOURCES];
115
116     while (1) {
117         for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
118             request[i] = rand() % (need[id][i] + 1);
119         }
120         if (request_resources(id, request) == 0) {
121             printf("Customer %d request granted\n", id);
122             sleep(1);
123             release_resources(id, request);
124             printf("Customer %d released resources\n", id);
125         }
126         sleep(rand() % 3 + 1);
127     }
128     return NULL;
129 }
130

```

تابع مربوط به اجرای هر مشتری در ترد جداگانه.

در حلقه‌ی بی‌نهایت:

1. درخواست تصادفی از `need[]`
2. تلاش برای گرفتن منابع (`request_resources`)
3. اگر موفق شد، بعد از کمی خواب منابع را آزاد می‌کند.
4. سپس منتظر می‌ماند تا دوباره تلاش کند.

```

130
131 int main(int argc, char* argv[]) {
132     if (argc != NUMBER_OF_RESOURCES + 1) {
133         printf("Usage: %s <r1> <r2> <r3> <r4> <r5>\n", argv[0]);
134         return EXIT_FAILURE;
135     }
136
137     for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
138         available[i] = strtol(argv[i + 1], NULL, 10);
139     }
140
141     initialize();
142     pthread_mutex_init(&lock, NULL);
143
144     pthread_t threads[NUMBER_OF_CUSTOMERS];
145     int ids[NUMBER_OF_CUSTOMERS];
146
147     for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
148         ids[i] = i;
149         pthread_create(&threads[i], NULL, customer_thread, &ids[i]);
150     }
151
152     for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
153         pthread_join(threads[i], NULL);
154     }
155
156     pthread_mutex_destroy(&lock);
157     return 0;
158 }
159

```

1. ورودی را بررسی می‌کند:

• باید `NUMBER_OF_RESOURCES` مقدار به عنوان آرگومان داده شود.

2. مقداردهی اولیه‌ی `available[]` از آرگومان‌ها

3. مقداردهی اولیه‌ی ساختارها با `initialize()`

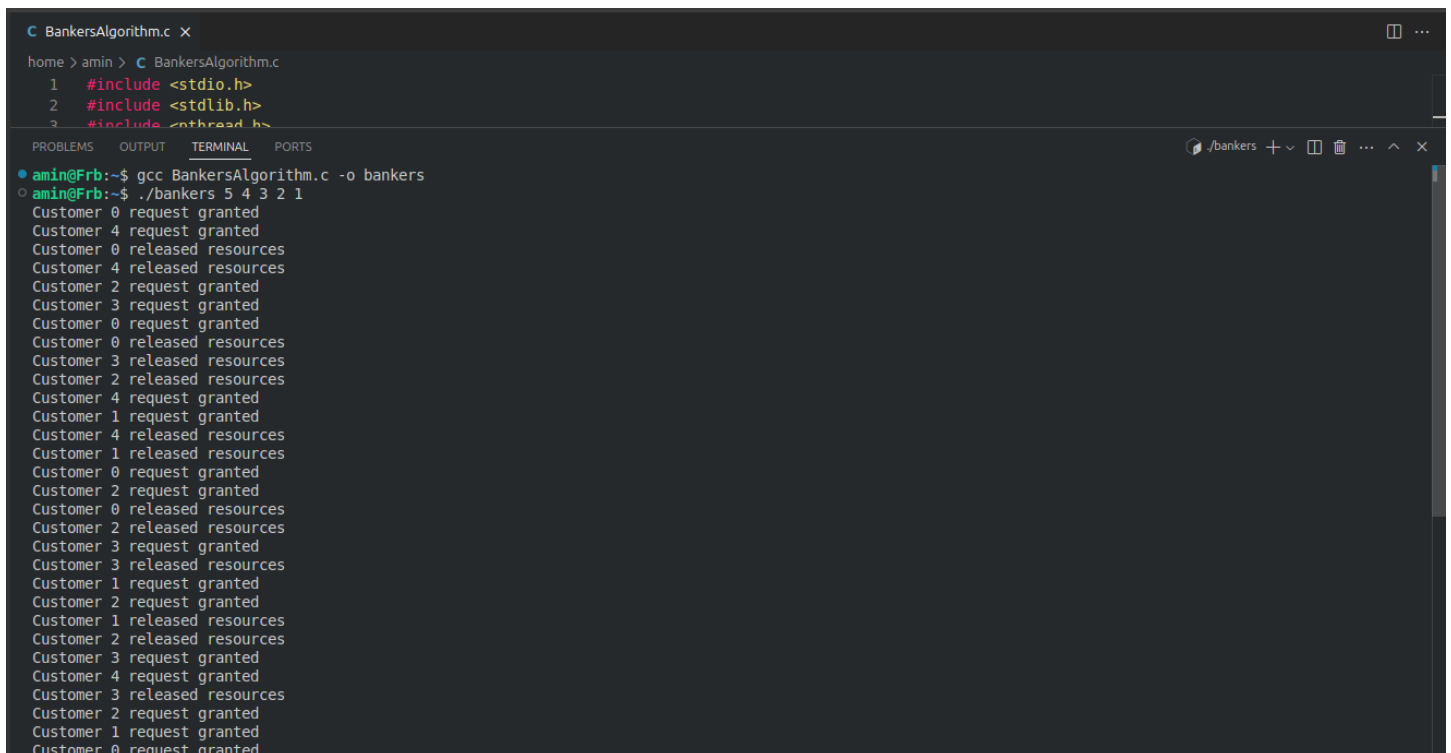
4. مقداردهی اولیه قفل `mutex`

5. ایجاد تردها برای هر مشتری

6. صبر برای اتمام تردها با `pthread_join`

7. نابود کردن `mutex` در پایان

در ادامه عکس نتیجه اجرای برنامه به این صورت است :



```
C BankersAlgorithm.c X
home > amin > C BankersAlgorithm.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>

PROBLEMS OUTPUT TERMINAL PORTS
• amin@Frb:~$ gcc BankersAlgorithm.c -o bankers
• amin@Frb:~$ ./bankers 5 4 3 2 1
Customer 0 request granted
Customer 4 request granted
Customer 0 released resources
Customer 4 released resources
Customer 2 request granted
Customer 3 request granted
Customer 0 request granted
Customer 0 released resources
Customer 3 released resources
Customer 2 released resources
Customer 4 request granted
Customer 1 request granted
Customer 4 released resources
Customer 1 released resources
Customer 0 request granted
Customer 2 request granted
Customer 0 released resources
Customer 2 released resources
Customer 3 request granted
Customer 3 released resources
Customer 1 request granted
Customer 2 request granted
Customer 1 released resources
Customer 2 released resources
Customer 3 request granted
Customer 4 request granted
Customer 3 released resources
Customer 2 request granted
Customer 1 request granted
Customer 0 request granted
```

و مدیریت منابع بین چند مشتری (یا ترد) به گونه‌ای انجام میشود که سیستم به وضعیت نا امن یا بن بست (Deadlock) نرسد. هر خط نشان می‌دهد که یک مشتری منابع مورد نظر خود را با موفقیت دریافت و پس از مدتی آزاد کرده است. در صورتی که درخواست رد شود (مثلاً سیستم ایمن نباشد یا منابع کافی نباشد)، هیچ پیامی چاپ نمی‌شود، ولی برنامه به تلاش ادامه می‌دهد.