# به نام خدا

محمد جواد زندیه 9831032

گزارش آزمایش شماره 7 سیستم عامل

---

در این آزمایش برای جلوگیری از dead lock از روش اجتناب یا avoidance استفاده می کنیم.

یکی از روش های اجتناب، استفاده از الگوریتم بانکداران می باشد(Banker’s Algorithm)

در این روش هر درخواستی که از طرف مشتری می آید، بررسی می شود که آیا با پاسخ دادن به آن همچنان سیستم safe می ماند یا خیر. در صورت ایمن ماندن باید منبع را تخصیص دهیم و پس از اتمام کار آن باید منبع را دوباره به منابع موجود اضافه کنیم.

3 موردی که در این آزمایش مورد استفاده قرار گرفته اند:

1. Multi-Threading
2. Dead lock and Banker’s Algorithm
3. Mutex and lock

Bankers.c > ...

```c
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

/* m : number of resources */
#define m 6
/* n : number of threads */
#define n 5
int thread_number[n] = { 0, 1, 2, 3, 4 };
/* the available amount of each resource */
int available[m];
/* the maximum demand of each thread */
int maximum[n][m] = { {2, 4, 6, 9, 10, 4},
                      {4, 8, 4, 5, 2, 0},
                      {10, 6, 7, 4, 10, 2},
                      {2, 10, 0, 3, 2, 3},
                      {3, 6, 5, 0, 2, 3}
                    };
/* the amount of currently allocated to each thread */
int allocation[n][m] = { {1, 1, 2, 2, 3, 1},
                         {2, 2, 1, 5, 0, 0},
                         {3, 2, 2, 1, 3, 0},
                         {0, 0, 0, 0, 0, 2},
                         {1, 2, 2, 0, 1, 1}
                       };
```

```c
/* the remaining need of each thread */
int need[n][m];
/* shows that each thread finished or not
    -2 : thread has exceeded its maximum calim
    -1 : (request <= need) but not (request <= available)
     0 : finish
*/
int finish[n];
/* initialize thread mutex */
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

```c
int is_safe_request(int tid, int request[]){
    for(int i = 0; i < m; i++){
        if(request[i] > need[tid][i]){
            finish[tid] = -2; // thread has exceeded its maximum calim
            return -2;
        }
    }
    // request <= need
    for(int i = 0; i < m; i++){
        if(request[i] > available[i]){
            finish[tid] = -1;
            return -1; // not request <= available
        }
    }
    // request <= available
    finish[tid] = 0;
    return 0; // success
}
```

```c
void request_resources(int tid, int request[]){
    pthread_mutex_lock(&mutex);
    int status = is_safe_request(tid, request);
    if(status == 0){ // success
        for(int i = 0; i < m; i++){
            available[i] += allocation[tid][i];
            allocation[tid][i] = 0;
            need[tid][i] = maximum[tid][i] - allocation[tid][i];
        }
        finish[tid] = 0;
        printf("thread id: %d , status: success\n", tid);
    }
    pthread_mutex_unlock(&mutex);
    return;
}
```

```c
void* thread_handler(void* args0){
    int* tid = args0;
    while(finish[*tid] != 0){
        int request[m];
        for(int i = 0; i < m; i++)
            request[i] = rand() % 4;
        request_resources(*tid, request);
        if(finish[*tid] == -2){
            printf("thread id: %d , status: has exceeded its maximum calim\n", *tid);
            finish[*tid] = 0;
        }
    }
}
```

```c
int main(int argc, char** argv){
    srand(time(NULL));

    printf("need: \n");
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            need[i][j] = maximum[i][j] - allocation[i][j];
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }
    // no thread finished in start
    for(int i = 0; i < n; i++)
        finish[i] = -1;

    // get available resources from Banker
    if(argc < m+1){
       printf("not enough arguments\n");
       return EXIT_FAILURE;
    }
    for(int i = 0; i < m; i++)
        available[i] = strtol(argv[i+1], NULL, 10);

    // make available update
    printf("available: \n");
    int sum[m];
```

```
115     for(int j = 0; j < m; j++){
116         sum[j] = 0;
117         for(int k = 0; k < n; k++)
118             sum[j] += allocation[k][j];
119     }
120     for(int i = 0; i < m; i++){
121         available[i] -= sum[i];
122         printf("%d ", available[i]);
123     }
124     printf("\n");
125 
126     // create threads
127     pthread_t thread[n];
128     for(int i = 0; i < n; i++)
129         pthread_create(&thread[i], NULL, thread_handler, &thread_number[i]);
130 
131     // wait for threads to be finished
132     for(int i = 0; i < n; i++)
133         pthread_join(thread[i], NULL);
134 
135     return 0;
136 }
```

خروجی به صورت زیر می باشد:

```
javad@javad-HP-350-G1:~/Desktop/OSLab/project7$ ./Bankers 12 14 15 12 13 10
need:
1 3 4 7 7 3
2 6 3 0 2 0
7 4 5 3 7 2
2 10 0 3 2 1
2 4 3 0 1 2
available:
5 7 8 4 6 6
thread id: 0 , status: success
thread id: 1 , status: has exceeded its maximum calim
thread id: 2 , status: success
thread id: 3 , status: has exceeded its maximum calim
thread id: 4 , status: success
javad@javad-HP-350-G1:~/Desktop/OSLab/project7$ 
```