Process Security Sequence Improvement Algorithm Based on Banker Algorithm

Li Jiang
Department of Software Engineering, Huasoft Software College, Guangzhou University, No.13, Guangcai Avenue, Conghua District, Guangzhou, Guangdong, China 510990
Corresponding author’s e-mail: jl@sise.com.cn

Abstract: By analyzing the core idea of the banker's algorithm and the essential meaning of the security state, an algorithm for applying multiple resources at a certain time in the system is proposed. The algorithm is implemented using object-oriented programming language C++. By analyzing security sequences and processes, the number of application resources can provide support for system resource allocation and process scheduling optimization. This algorithm can also be used as an improved implementation algorithm for deadlock detection algorithms or banker algorithms.

1. Introduction:
Ubuntu 12.0 is widely popular as a free operating system. It is not only open source, but also a completely free operating system. Most programs can be compiled and run on Ubuntu 12.0. At present, multi-process resource scheduling in Linux environment is widely used, and the concurrent execution and optimization of system processes developed on the basis of Linux needs to be studied in the computer field [1]. Although Microsoft's Windows operating system is a popular product, it is widely used in universities. Relatively speaking, the security and reliability of the Linux operating system have been greatly improved. In the Linux operating system environment, in the field of network communication and database system application, concurrent execution of processes, when the operating system is scheduling and sharing resources, when the multi-process advancement order is unreasonable, or the Linux system resource allocation order causes the system to die. Lock [2]. The system in deadlock state seriously affects the reliability and security of the system. The improved banker algorithm avoids excessive waiting resources between processes, which can ensure that multiple processes continue to move forward reasonably. Advance.

2. Analysis of multi-resource deadlock problem
In the Ubuntu 12.0 operating system, MySql6.3 database system and local area network communication, due to the concurrent execution of the process and the sharing of resources, the competition for resources between multiple processes, the advancement of the unreasonable can never continue to advance. When the deadlock state, the reliability and security of the system are seriously affected.

In the Ubuntu 12.0 operating system, in the case of multi-resource scheduling, the four necessary conditions for the process deadlock are generated:
(1) Mutual exclusion: The current resources in the operating system environment can only be scheduled to a specific process or idle. Resources cannot be assigned to multiple processes at the same time.

(2) Occupy and wait: The process of a resource in the operating system environment requests new resources, and these resources are occupied by other processes and cannot be released [2][3].

(3) Cyclic waiting: At some time in the Linux operating system environment, there are N resources P1, P2, P3, ..., ..., Pn, Pn waiting for the resources occupied by P1, P2 waiting for the resources occupied by P3, P1 waits for resources occupied by P2.

(4) Non-preemption: It means that resources are inalienable. A process that can only be occupied by it voluntarily releases (the state of the process is blocked or ready).

Common strategies for solving deadlocks in the operating system:

(1) Deadlock prevention: It is impossible to make a deadlock by destroying one of the four necessary conditions of deadlock.

(2) Deadlock detection and recovery: When a resource is requested or released in the system, it is necessary to check whether the current state has a deadlock. If there is a deadlock, you need to adopt the appropriate strategy to recover the system from the deadlock [5][6].

(3) Deadlock avoidance: Banker is an algorithm to avoid deadlock during operation. In the operating system environment, security detection algorithm is implemented through dynamic resource allocation, so as to ensure that the operating system environment does not reach death. Lock status.

The so-called security state is that in the operating system environment, the process can allocate the required amount of resources for each process in a reasonable order that can be advanced according to a certain process order p1, p2, ..., pn until the process pi is satisfied. The maximum demand of resources enables each process pi to be successfully completed. In this state, the system is in a safe state, and the process advances the sequences p1, p2, ..., pn as security sequences [5][6][8].

3. Banker algorithm overall design

3.1 Algorithm description

The algorithm studied in this paper is an improved algorithm for the existing banker algorithm. When a new process declares its maximum resource requirement, in the Linux operating system environment, if other processes apply for resources, the number of resources available to the operating system may be In this case, the original safe system environment becomes unsafe. The author studies the banker's algorithm against this problem, and compares the amount of resources requested by the process with the available resources of the system before the allocation of operating system resources. To ensure that the operating system allocates available resources in a secure sequence state. Ensure that the system allocates resources in a secure state.

3.2 Data structure of banker algorithm

There are three important matrices in the banker's algorithm, namely the maximum demand matrix Max[M,N], the assigned matrix Allocation[M,N], and the demand matrix Need[M,N]. Their relationship is the demand matrix Need[M,N]= Max[M,N]- Allocation[M,N], the data structure of the banker algorithm is described as follows:

(1) Maximum demand matrix Max[M,N], Max[i][j] in the operating system environment is expressed as the maximum demand quantity of Rj resources for process Pi, where the largest demand matrix Max[M,N] is an M a matrix of rows N;

(2) Allocation matrix Allocation[M,N], Allocation[i][j] in the operating system environment indicates that the number of R j resources has been obtained for the process Pi, where the allocation matrix Allocation[M,N] is an M row N a matrix of columns;

(3) Demand matrix Need[M,N], Need[i][j] in the operating system environment indicates the number of R j resources required for the process Pi, where the demand matrix Need[M,N] is an M row N a matrix of columns;
(4) The available resource vector Available, one containing M elements. Array vector, the value of the array vector represents the amount of resources available for a class; (5) Finish[i] indicates whether the resource has an identification vector, and the Linux operating system environment indicates whether the system has sufficient resources to allocate to the process Pi; (6) Work[i] represents the number of resources vector. In the Linux operating system environment, the number of Rj resources required to continue the operation of the process Pi can be supplied. Set Requesti Is the Pi request vector for the process. [8]

3.3 Banker algorithm code implementation

```
// Banker algorithm global variable definition process
int Max[60][60];   //Banker algorithm maximum demand matrix Max
int Allocation[60][60]; // Banker algorithm allocation matrix Allocation
int Need[60][60];         // Banker Algorithm Demand Matrix Need
int p[50];  // N resources, M processes
int Available[80]; // Banker algorithm can use resource array Available
int Request[60][80];    // The banker algorithm process Pi also needs the resources of Rj resources.
int finish[60];
// Banker security detection algorithm
int Bank_Safe()
{
    int i;  int j;  int l=0;   int k;
    int Work_bank[80];    for (int i=0; i<n; i++)
        Work_bank [i] = Available[i];    for (int i=0; i<m; i++)
    finish[i]=0;    for (int i=0; i<n; i++)
    {
        if (1==finish[i])    continue;   else
        {for (int j=0;j<n;j++)
            if (Work_bank [j]<Need[i][j])
            break;    if (n == j)
            {int finish[i]=1;
            for(int k=0;k<;k++)
                Work_bank [k]+=Allocation[i][k];    p[i]=i;   i=-1;
            } else    continue;
        } if (m == l)  { printf("The system is safe \n");    printf("System security sequence\n");
    for (i=0;i<l+1)    printf("%d",p[i]);
    if (i!=l-1)    printf("-->");
    } printf("\n");    return 1;   }
}
```

4. Algorithm validation

Banker algorithm application example: Assume that there are three resources of A, B, and C in the operating system. The number of Class A resources is 17, the number of Class B resources is five, and the number of Class C resources is 20. The five processes are p0, p1, p2, p3, and p4. The state at T0 is shown in the table.

| Table 1 Number of system resources at time T0 |
|----------------------------------------------|
| Max A, B, C | Allocation A, B, C | Need A, B, C | Available A, B, C |
| P0 | 5 5 9 | 2 1 2 | 3 4 7 | 2 3 3 |
| P1 | 5 3 6 | 4 0 2 | 1 3 4 | 2 3 3 |
Running the security detection algorithm on the Linux operating system can obtain a secure process sequence p3, p1, p2, p4, p0, so it can be concluded that the operating system T0 is safe at all times.

### Table 2 Security status assignment sequence

| Process | Work | Need | Allocation | Available | Finish |
|---------|------|------|------------|-----------|--------|
| P3      | 233  | 221  | 204        | 437       | T      |
| P1      | 437  | 134  | 402        | 839       | T      |
| P2      | 839  | 006  | 405        | 12314     | T      |
| P4      | 12314| 110  | 314        | 15418     | T      |
| P0      | 15418| 347  | 212        | 17520     | T      |

Suppose the p1 process issues a new resource request, request vector Request[1]=(0,3,4). Since Request[1]=(0,3,4)≤Need[1]=(1,3,4), the request is legal, and further Request[1]=(0,3,4)≤Available=(2,3,3), p1 is blocked, P1 application resources are rejected, resources cannot be allocated, and no security detection algorithm is required.

At time T0, it is assumed that process P3 requests Request[3]=(2,0,1), Request[3]=(2,0,1)≤Need[3]=(2,2,1), Request[3]≤Available=(2,3,3), the system has sufficient resources available to perform the security check algorithm.

### Table 3 Security sequence

| Process | Work | Need | Allocation | Available | Finish |
|---------|------|------|------------|-----------|--------|
| P3      | 032  | 020  | 405        | 437       | T      |
| P1      | 437  | 134  | 402        | 839       | T      |
| P2      | 839  | 006  | 405        | 12314     | T      |
| P4      | 12314| 110  | 314        | 15418     | T      |
| P0      | 15418| 347  | 212        | 17520     | T      |

### 5. Conclusion

This paper proposes a deadlock avoidance algorithm for Linux operating systems. This algorithm is an improved algorithm for the existing banker algorithm. When a new process declares its maximum resource requirement, in the Linux operating system environment, if another process Pi requests resources, the number of operating system resources will change. The banker algorithm is improved for this problem. Before the operating system allocates resources to the process Pi, the number of resources required by the process Pi is re-compared with the available resources of the Linux operating system. After obtaining the security sequence, the Linux operating system is guaranteed. All available resources are allocated in the secure sequence state. System simulation and experimental verification show that this algorithm ensures that the system will not deadlock and improve the security and reliability of the system.

### References:

[1] He Yanxiang, Li Fei, Li Ning. Computer Operating System [M]. Revised Edition. Beijing: Tsinghua University Press, 2001.

[2] Men Weijiang, Yang Yanjiao. Method of designing computer network operating system [J]. Journal of Gansu Science, 2004, 16(1): 84-86.

[3] Wang Xiaodong, Computer Algorithm Design and Analysis [M]. 2nd ed. Beijing: Publishing House of Electronics Industry, 2004. Author
[4] Li Wei, Chen Wanghu. Banker Algorithm Based on Generalized Table [J]. Journal of Northwest Normal University (Natural Science Edition), 2002, 38(3): 30-33.

[5] Xu Gang, Wu Zhiming. Improvement and application of banker algorithm in flexible manufacturing system [J]. Computer Integrated Manufacturing Systems, 2004, 10(1): 70-76.

[6] Hou Gang. In-depth analysis of banker algorithm [J]. Journal of Weifang University, 2006, 6(2): 46-48.

[7] Tie Jun, Jiang Tianfa. Security Sequence Analysis in Banker Algorithm [J]. Journal of Wuhan University of Technology, 2007, 29(6): 114-117.

[8] Zuo Wanli, Wang Lazhu. Improvement of banker algorithm [J]. Journal of Natural Science of Jilin University, 1997, (1): 35-37.

[9] Tang Zikai, Zhe Fengping, Tang Xiaodan, Computer Operating System [M] 2nd Edition, Xi'an: Xi'an University of Electronic Science and Technology Press. 1996

[10] Smith J E, Nair R. Universal Platform for Virtual Machine Systems and Processes [M] Beijing: Mechanical Industry Press, 2009

[11] Zhang Wei, Liu Qingkun, Yin Yuanyou, et al. PXE-based Linux parallelism Rapid and automatic deployment and configuration of clusters[J]. Journal of Liaoning Normal University: Natural Science Edition, 2008(1): 51-53.

[12] Huang Guanli, Jin Yan, Go Chuanjing, et al. Dynamic score based on PXE technology Research on Storage Security of Fabric Diskless Network[J]. Computing Science, 2010(9): 297-301.