作業系統期末報告

Banker Problem

103703006 何世馨

103703038 林廷韋

目錄

一、實作主題、內容簡介

此次期末多執行緒報告主題，將實作作業系統典型的同步問題：
銀行家問題（Banker Problem ）。
其內容與多執行緒同步有關，大致內容如下：

在銀行中，客戶申請貸款的數量是有限的，每個客戶在第一次申請貸款時要聲明完成該項目所需的最大資金量，在滿足所有貸款要求時，客戶應及時歸還。銀行家在客戶申請的貸款數量不超過自己擁有的最大值時，都應盡量滿足客戶的需要。銀行家就好比操作系統，資金就是資源，客戶就相當於要申請資源的進程。

銀行家算法是一種最有代表性的避免死鎖的算法。在避免死鎖方法中允許進程動態地申請資源，但系統在進行資源分配之前，必須先計算此次分配資源的安全性，若分配不會導致系統進入不安全狀態，則分配，否則等待。為實現銀行家算法，系統必須設置數據結構(後面會有更詳細的解釋)。

名詞定義:

(1)安全狀態:
如果存在一個由系統中所有進程構成的安全序列 P1，…，Pn，則系統處於安全狀態。安全狀態一定是沒有死鎖發生。
(2)不安全狀態:
不存在一個安全序列。不安全狀態不一定導致死鎖。
(3)數據結構

a）可利用資源向量 Available

是個含有 m 個元素的數組，其中的每一個元素代表一類可利用的資源數目。如果 Available[j]=K，則表示在銀行系統中現 j 類有資源 K 個。

b）最大需求矩陣 Max

這是一個 n×m 的矩陣，它定義了系統中 n 個顧客中的每一個顧客對 m 類資源的最大需求。如果 Max[i,j]=K，則表示顧客 i 需要 j 類資源的最大數目為 K。

c）分配矩陣 Allocation

這也是一個 n×m 的矩陣，它定義了系統中每一類資源當前已分配給每一顧客的資源數。如果 Allocation[i,j]=K，則表示進程 i 當前已分得 j 類資源的數目為 K。

d）需求矩陣 Need

這也是一個 n×m 的矩陣，用以表示每一個顧客仍需要的各類資源數。如果 Need[i,j]=K，則表示顧客 i 還需要 j 類資源 K 個，方能完成其

任務。Need [i, j]=Max[i, j]-Allocation[i, j]。
（需要資源數 = 最大需求數 - 現在持有數）

二、算法原理

把操作系統看作是銀行家，操作系統管理的資源相當於銀行家管理的資金（實作中因考慮現實銀行借貸關係、資源類型上我們只取一種(金錢)），進程向操作系統請求分配資源相當於用戶向銀行家貸款。

為保證資金的安全，銀行家規定：

(1) 當一個顧客對資金的最大需求量不超過銀行家現有的資金時就可接納該顧客；

(2) 顧客可以分期貸款，但貸款的總數不能超過最大需求量；

(3) 當銀行家現有的資金不能滿足顧客尚需的貸款數額時，對顧客的貸款可推遲支付，但總能使顧客在有限的時間裡得到貸款；

(4) 當顧客得到所需的全部資金後，一定能在有限的時間裡歸還所有的資金.

操作系統按照銀行家製定的規則為進程分配資源，當進程首次申請資源時，要測試該進程對資源的最大需求量，如果系統現存的資源可以滿足它的最大需求量則按當前的申請量分配資源，否則就推遲分配。當進程在執行中繼續申請資源時，先測試該進程本次申請的資源數是否超過了該資源所剩餘的總量。若超過則拒絕分配資源，若能滿足則按當前的申請量分配資源，否則也要推遲分配。

三、執行緒（ Threads ）類型需求與各自功能

在這次實作中，我們執行緒的類型只有一種，那就是顧客，每一位顧客都代表著一個 Thread，其功能為獨立與銀行借貸。其內容如下:

I. 在交易過程前，顧客會先判斷自己還需要的金錢為多少，如果不為零，則會產生一個隨機值金錢值去和銀行借錢。

```
if (request_resources(pNum)){
  fullResCount = 0;
  /* Check whether all needs are met. */
  for (int i=0; i < TOT_RESOURCES; i++){
      fullResCount = fullResCount + (needState[pNum][i] == 0);
  }
```

II. 在交易過程前，顧客會自己先判斷自己需要金額是否可以被銀行滿足，即銀行是否有足夠的金額可以借貸給此顧客，如果可以則在往下做更進一步的判斷，如果不行則放棄此借貸，將借貸金額

轉為銀行可以提供的最大金額。

```
for (int i=0; i < TOT_RESOURCES; i++){

    if (needState[pNum][i] != 0){
        rRequest[i] = rand() % needState[pNum][i] + 1;
    }
    else
    {rRequest[i] = 0;}
    /* If the request is too high, set as max avail.*/
    if (rRequest[i] > availableResources[i])
    {rRequest[i] = availableResources[i];}

    if (availableResources[i] == 0)
    {emptyResources ++;}
}
```

III. 顧客在決定自己借貸的金額後，且等待時間結束後，顧客則可以執行交易，在交易的過程中，銀行會判斷是否有死結產生，其在後面會有更詳細的說明。

IV. 在顧客的需求都有被滿足時，則其會釋放金錢給銀行(還錢)，並

```
buffer_time += 1; // Increment the time.
printf("\nCustomer# %d 歸還 ", pNum);

isExit[pNum] = 1;//Customer leave after release_resources

/* Add resources back to availableResources. */
for (int i=0; i < TOT_RESOURCES; i++){

    printf("%d million", maxWorkState[pNum][i]);

    availableResources[i] = availableResources[i] + maxWorkState[pNum][i];
    maxWorkState[pNum][i] = 0;
    workState[pNum][i] = 0;
}

printf("\n\n");

printState();
```

離開銀行，此時，銀行的可用金額必須增加顧客釋放的金額。

V. 在交易前或交易後但是仍未滿足自己所需的金額，則顧客必須回到等待狀態並且可以讓下一個等待時間結束的顧客交易。

```c
void *customer(void *param) {
    int pNum = *((int*) param);
    //int resReq[TOT_RESOURCES];
    int r;
    int fullResCount = 0;
    while(TRUE)
    {
        r = rand() % MAX_SLEEP + 1;

        printf("Customer# %d 等待中(%d)...\n", pNum, r);
        sleep(r); // Sleep for both holding and requesting resources.
        printf("Customer# %d 完成等待...\n", pNum);

        /* Release resources if maximum resources are satisfied. */
        if (fullResCount == TOT_RESOURCES){

            if (release_resources(pNum)){

                fullResCount = 0;

                pthread_exit(NULL);
            }
            else
            {
                fprintf(stderr, "Error releasing resources");
            }
        }
        /* Still need resources, create a random request of resources */
        else{
              if (request_resources(pNum)){
                fullResCount = 0;
                /* Check whether all needs are met. */
                for (int i=0; i < TOT_RESOURCES; i++){
                    fullResCount = fullResCount + (needState[pNum][i] == 0);
                }
            }
        }
        if (buffer_time >= runtime)
        {
            break;
        }
    }
```

四、共有資源與共享變數需要進行 Multi-Thread 之間的同步與合作

我們在實作所用的共同變數為下列兩項:

(1)銀行的可用資源(銀行現有金額)

(2)銀行已沒有資源的種類數

```c
int availableResources[TOT_RESOURCES];
int emptyResources = 0;
```

在實作同步的過程中我們主要是用 pthread_mutex_lock(&mutex);和 pthread_mutex_unlock(&mutex);利用這兩個行的區間來綁定 thread。以下分為顧客借貸與顧客還款兩個狀況來探討:

I. 顧客借貸:

當綁定後其他顧客並不能進入櫃台並與正在交易的顧客爭奪共享資源(銀行裡的錢)除非其顧客解除綁定。而在解除綁定時有三個狀況(1)發現自己的金額銀行無法支付，退出後並解除綁定。

```
int request_resources(int pNum)
{
    /* Acquire the mutex lock. */
    pthread_mutex_lock(&mutex);

    int rRequest[TOT_RESOURCES];
    emptyResources = 0; // Counts the number of rec. at 0.

    buffer_time += 1; // Increment the time.

    for (int i=0; i < TOT_RESOURCES; i++){

        if (needState[pNum][i] != 0){
            rRequest[i] = rand() % needState[pNum][i] + 1;
        }
        else
        {rRequest[i] = 0;}
        /* If the request is too high, set as max avail.*/
        if (rRequest[i] > availableResources[i])
        {rRequest[i] = availableResources[i];}

        if (availableResources[i] == 0)
        {emptyResources ++;}
    }
    if (emptyResources == TOT_RESOURCES){
        /* Release mutex lock */
        pthread_mutex_unlock(&mutex);
        return 0;
    }
    /* Generate the proposed new work state */
    int tmpWorkState[TOT_PROCESSES][TOT_RESOURCES];
    int tmpAvailRec[TOT_RESOURCES];
    /* Create a temporary work state and resource request. */
    for (int i=0; i < TOT_PROCESSES; i++){
```

(2)如果顧客的前置判斷成功，則銀行會做進階判斷，確認是否此交易會造成死結，會則拒絕交易，顧客解除綁定。(3)否，則執行

交易，交易完成後解除綁定。

```
if (safety_test(tmpWorkState, tmpAvailRec)){
    int fullResCount = 0; // Count the number of full resources.
    for (int i=0; i < TOT_RESOURCES; i++){
        availableResources[i] = availableResources[i] - rRequest[i];
        workState[pNum][i] = workState[pNum][i] + rRequest[i];
        needState[pNum][i] = maxWorkState[pNum][i] - workState[pNum][i];
        fullResCount = fullResCount + (needState[pNum][i] == 0);
    }
    /* Print the result. */
    printf("        處理中...\n");
    //printf("      Allocated:  %d %d %d\n", rRequest[0], rRequest[1], r
    //printf("      Available:  %d %d %d\n", availableResources[0], avai
    printf("        銀行借出: %d million\n", rRequest[0]);
    printf("        銀行餘額: %d million\n", availableResources[0]);
    printf("\n");
    printf("Customer# %d 處理結束...\n", pNum);
    printState();
    /* Release the mutex lock. */
    pthread_mutex_unlock(&mutex);
    return 1;
}
else{
    printf("        銀行金額不足\n");
    printf("        Customer# %d denied to avoid deadlock\n", pNum)
    printf("Customer# %d 處理結束...\n", pNum);
    /* Release the mutex lock. */
    pthread_mutex_unlock(&mutex);
    return 0;
}
```

II. 顧客還款:

當顧客要進行還錢前，會先綁定顧客，等還款完畢，銀行確認金錢後，再將顧客的綁定解除。

```
/* Acquire the mutex lock. */
pthread_mutex_lock(&mutex);

buffer_time += 1; // Increment the time.

//printf("\nTime %d Customer# %d releasing ", buffer_time, pNum);

printf("\nCustomer# %d 歸還 ", pNum);

isExit[pNum] = 1;//Customer leave after release_resources

/* Add resources back to availableResources. */
for (int i=0; i < TOT_RESOURCES; i++){

    printf("%d million", maxWorkState[pNum][i]);

    availableResources[i] = availableResources[i] + maxWorkState[pNum][i]
    //needState[pNum][i] = maxWorkState[pNum][i];
    maxWorkState[pNum][i] = 0;
    workState[pNum][i] = 0;
}

printf("\n\n");
printState();

/* Release the mutex lock. */
pthread_mutex_unlock(&mutex);
```

五、關鍵區域程式碼(使用 C)

這次實作 Banker Problem 中，最關鍵的程式碼莫過於判斷是否有死結，並且可以避免死結:

在實作死結的部分，銀行必須判斷此交易是否處於一個安全的狀態，而不會造成其他後續顧客無法借到金錢的問題如果經判斷為一個安全狀態，則成功交易，並將交易紀錄顯示出來。

```
/* If the new work state is safe, then allocate resources. */
if (safety_test(tmpWorkState, tmpAvailRec)){

    int fullResCount = 0; // Count the number of full resources.
    for (int i=0; i < TOT_RESOURCES; i++){
        availableResources[i] = availableResources[i] - rRequest[i];
        workState[pNum][i] = workState[pNum][i] + rRequest[i];
        needState[pNum][i] = maxWorkState[pNum][i] - workState[pNum][i];
        fullResCount = fullResCount + (needState[pNum][i] == 0);
    }
    /* Print the result. */
    printf("        處理中...\n");
    //printf("      Allocated:  %d %d %d\n", rRequest[0], rRequest[1], rRe
    //printf("      Available:  %d %d %d\n", availableResources[0], availa
    printf("        銀行借出: %d million\n", rRequest[0]);
    printf("        銀行餘額: %d million\n", availableResources[0]);
    printf("\n");
    printf("Customer# %d 處理結束...\n", pNum);
    printState();
    /* Release the mutex lock. */
    pthread_mutex_unlock(&mutex);

    return 1;
}
```

如果不行，銀行則拒絕此顧客的交易，並將拒絕內容顯示出來，顧客回到等待的狀態。

```
else{

    printf("        銀行金額不足\n");
    printf("        Customer# %d denied to avoid deadlock\n", pNum);

    printf("Customer# %d 處理結束...\n", pNum);

    /* Release the mutex lock. */
    pthread_mutex_unlock(&mutex);

    return 0;
}
```

六、文字顯示狀態與結果

Start

```
------------------------------------------------------------------
------------------------------------------------------------------

銀行金額:15 million
---------------------------------------------------------
                <已借金額>        <差額>        <需借金額> |
Customer# 0     0 million       7 million       7 million  |
Customer# 1     0 million       3 million       3 million  |
Customer# 2     0 million       9 million       9 million  |
Customer# 3     0 million       2 million       2 million  |
Customer# 4     0 million       4 million       4 million  |
---------------------------------------------------------

Customer# 0 等待中(5)...
Customer# 1 等待中(2)...
Customer# 2 等待中(2)...
Customer# 3 等待中(1)...
Customer# 4 等待中(2)...
Customer# 3 完成等待...

Customer# 3
          顧客需要: 2 million
          銀行金額: 15 million
          處理中...
          銀行借出: 2 million
          銀行餘額: 13 million

Customer# 3 處理結束...

銀行金額:13 million
---------------------------------------------------------
                <已借金額>        <差額>        <需借金額> |
Customer# 0     0 million       7 million       7 million  |
Customer# 1     0 million       3 million       3 million  |
Customer# 2     0 million       9 million       9 million  |
Customer# 3     2 million       0 million       2 million  |
Customer# 4     0 million       4 million       4 million  |
---------------------------------------------------------
```

```
Customer# 3 等待中(3)...
Customer# 1 完成等待...
Customer# 2 完成等待...
Customer# 4 完成等待...

Customer# 1
        顧客需要: 3 million
        銀行金額: 13 million
        處理中...
        銀行借出: 3 million
        銀行餘額: 10 million

Customer# 1 處理結束...

銀行金額:10 million
-----------------------------------------------------
              <已借金額>      <差額>      <需借金額> |
Customer# 0   0 million      7 million     7 million  |
Customer# 1   3 million      0 million     3 million  |
Customer# 2   0 million      9 million     9 million  |
Customer# 3   2 million      0 million     2 million  |
Customer# 4   0 million      4 million     4 million  |
-----------------------------------------------------

Customer# 1 等待中(1)...

Customer# 2
        顧客需要: 9 million
        銀行金額: 10 million
        處理中...
        銀行借出: 9 million
        銀行餘額: 1 million

Customer# 2 處理結束...

銀行金額:1 million
-----------------------------------------------------
```

```
                <已借金額>        <差額>        <需借金額> |
Customer# 0     0 million       7 million       7 million  |
Customer# 1     3 million       0 million       3 million  |
Customer# 2     9 million       0 million       9 million  |
Customer# 3     2 million       0 million       2 million  |
Customer# 4     0 million       4 million       4 million  |
-----------------------------------------------------------

Customer# 2 等待中(4)...

Customer# 4
          顧客需要: 1 million
          銀行金額: 1 million
          處理中...
          銀行借出: 1 million
          銀行餘額: 0 million

Customer# 4 處理結束...

銀行金額:0 million
-----------------------------------------------------------
                <已借金額>        <差額>        <需借金額> |
Customer# 0     0 million       7 million       7 million  |
Customer# 1     3 million       0 million       3 million  |
Customer# 2     9 million       0 million       9 million  |
Customer# 3     2 million       0 million       2 million  |
Customer# 4     1 million       3 million       4 million  |
-----------------------------------------------------------

Customer# 4 等待中(5)...
Customer# 1 完成等待...

Customer# 1 歸還 3 million


銀行金額:3 million
-----------------------------------------------------------
                <已借金額>        <差額>        <需借金額> |
```

```
Customer# 0    0 million       7 million       7 million  |
Customer# 2    9 million       0 million       9 million  |
Customer# 3    2 million       0 million       2 million  |
Customer# 4    1 million       3 million       4 million  |
-----------------------------------------------------------

Customer# 3 完成等待...

Customer# 3 歸還 2 million


銀行金額:5 million
-----------------------------------------------------------
               <已借金額>        <差額>        <需借金額> |
Customer# 0    0 million       7 million       7 million  |
Customer# 2    9 million       0 million       9 million  |
Customer# 4    1 million       3 million       4 million  |
-----------------------------------------------------------

Customer# 0 完成等待...

Customer# 0
         顧客需要: 5 million
         銀行金額: 5 million
         處理中...
         銀行借出: 5 million
         銀行餘額: 0 million

Customer# 0 處理結束...

銀行金額:0 million
-----------------------------------------------------------
               <已借金額>        <差額>        <需借金額> |
Customer# 0    5 million       2 million       7 million  |
Customer# 2    9 million       0 million       9 million  |
Customer# 4    1 million       3 million       4 million  |
-----------------------------------------------------------
```

```
Customer# 0 等待中(3)...
Customer# 2 完成等待...

Customer# 2 歸還 9 million


銀行金額:9 million
-----------------------------------------------------------
                <已借金額>        <差額>        <需借金額> |
Customer# 0     5 million       2 million       7 million  |
Customer# 4     1 million       3 million       4 million  |
-----------------------------------------------------------

Customer# 4 完成等待...

Customer# 4
          顧客需要: 3 million
          銀行金額: 9 million
          處理中...
          銀行借出: 3 million
          銀行餘額: 6 million

Customer# 4 處理結束...

銀行金額:6 million
-----------------------------------------------------------
                <已借金額>        <差額>        <需借金額> |
Customer# 0     5 million       2 million       7 million  |
Customer# 4     4 million       0 million       4 million  |
-----------------------------------------------------------

Customer# 4 等待中(2)...
Customer# 0 完成等待...

Customer# 0
          顧客需要: 1 million
          銀行金額: 6 million
          處理中...
```

```
        銀行借出: 1 million
        銀行餘額: 5 million

Customer# 0 處理結束...

銀行金額:5 million
-------------------------------------------------------
              <已借金額>      <差額>      <需借金額> |
Customer# 0   6 million     1 million     7 million  |
Customer# 4   4 million     0 million     4 million  |
-------------------------------------------------------

Customer# 0 等待中(4)...
Customer# 4 完成等待...

Customer# 4 歸還 4 million


銀行金額:9 million
-------------------------------------------------------
              <已借金額>      <差額>      <需借金額> |
Customer# 0   6 million     1 million     7 million  |
-------------------------------------------------------

Customer# 0 完成等待...

Customer# 0
        顧客需要: 1 million
        銀行金額: 9 million
        處理中...
        銀行借出: 1 million
        銀行餘額: 8 million

Customer# 0 處理結束...

銀行金額:8 million
-------------------------------------------------------
              <已借金額>      <差額>      <需借金額> |
```

```
Customer# 0     7 million        0 million        7 million   |
-------------------------------------------------------------

Customer# 0 等待中(1)...
Customer# 0 完成等待...

Customer# 0 歸還 7 million

銀行金額:15 million
-------------------------------------------------------------
                  <已借金額>          <差額>          <需借金額> |
-------------------------------------------------------------
```

七、組員分工

- 103703006 資科三 何世馨
  主程式撰寫、與主程式測試

- 103703038 資科三 林廷韋
  主程式邏輯轉換與協助與測試、報告撰寫

- Github 版本控管
  https://github.com/jackyjackylin/OS_Project

八、參考資料

1. http://spl.wikidot.com/pthread(pthread 基本用法)
2. http://www.syscom.com.tw/ePaper_Content_EPArticledetail.aspx?id=213&EPID=176&j=5&HeaderName=%E6%8A%80%E8%A1%93%E5%88%86%E4%BA%AB (同步實作用法)
3. http://notewhu.rhcloud.com/2015/11/24/semaphore%E7%9A%84%E7%94%A8%E6%B3%95%E8%88%87%E7%89%B9%E8%89%B2/ (Semaphore 的用法與特色)