Operating system Lab 5 report

Bank synchronize problem

Yi Lin

Class 34293

Table of Contents

Initial output

Initial output of bank.c will have the output like this:

```
Dad's Pid: 66744
Dad is requesting to view the balance.
First Son's Pid: 66745
Dad reads balance = 0
SON_1 is requesting to view the balance.
Dad needs 5 sec to prepare money
Attempt remaining: 20.
Second Son's Pid: 66746
SON_1 reads balance. Available Balance: 0
SON_2 is requesting to view the balance.
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: -20
Attempt remaining: 20.
Number of attempts remaining:19
SON_2 reads balance. Available Balance: -20
SON_2 wants to withdraw money. SON_2 withdrawed 20. New Balance: -40
Number of attempts remaining: 19
Dad writes new balance = 60
Dad will deposit 4 more time
Dad is requesting to view the balance.
Dad reads balance = 60
Dad needs 2 sec to prepare money
Dad writes new balance = 120
Dad will deposit 3 more time
SON_2 is requesting to view the balance.
Attempt remaining: 19.
SON_2 reads balance. Available Balance: 120
SON_2 wants to withdraw money. SON_2 withdrawed 20. New Balance: 100
Number of attempts remaining: 18
SON_1 is requesting to view the balance.
Attempt remaining: 18.
SON_1 reads balance. Available Balance: 100
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: 80
Number of attempts remaining:17
SON_1 is requesting to view the balance.
SON_2 is requesting to view the balance.
Attempt remaining: 17.
Attempt remaining: 17.
SON_2 reads balance. Available Balance: 80
SON_1 reads balance. Available Balance: 80
SON_2 wants to withdraw money. SON_2 withdrawed 20. New Balance: 60
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: 60
Number of attempts remaining: 16
Number of attempts remaining:16
Dad is requesting to view the balance.
Dad reads balance = 60
Dad needs 3 sec to prepare money
SON_1 is requesting to view the balance.
SON_2 is requesting to view the balance.
Attempt remaining: 16.
```

As we can see, the output result is in a mess and not well synchronized, then we need to add several semaphores and change some code to solve the problem.

Solution:

1. Restrict 2 process work at the same time:

Since there are only 2 ATMs, we need a semaphore semATM to restrict that only maximum 2 processes can run at the same time, hence we need to create a semaphore and initialize it to 2

```
// Create semaphore
int semMutex = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semWrAtt = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semZero = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semATM = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);

sem_create(semMutex, 1); // semMutex avoid 2 process deposit/withdraw at the same time
sem_create(semWrAtt, 1); // semWrAtt avoid 2 chile process access attempt at the same time
sem_create(semZero, 0); // semZero avoid zero balance
sem_create(semATM, 2); // semATM restrict that only 2 processes can run at the same time
```

To restrict 2 process work at the same time, the critical section would be the code of dad's action and son's actions, then before dad or son do anything, we call P(semATM) to reduce the value of semATM by 1, and if the semaphore is 0, the process will be blocked until it is greater than 0, and at the end of these processes, increase the value of semATM by V(semATM).

Critical section of dad process:

```
72          P(semATM);
73          printf("Dad is requesting to view the balance.\n"); //Dad is requesting to get hold of an ATM.
74          fp1 = fopen("balance.txt", "r+"); //Dad successfully got hold of the ATM.
75          fscanf(fp1, "%d", &bal2);
76          printf("Dad reads balance = %d \n", bal2);
77          int r = rand()%5+1;
78          printf("Dad needs %d sec to prepare money\n", r);
79          sleep(r); //Dad Process is sleeping for r sec. You need to make sure that other processes can work in the mean time.
80
81          P(semMutex);
82          fp1 = fopen("balance.txt", "r+");
83          fscanf(fp1, "%d", &bal2);
84          fseek(fp1,0L,0); //Dad will now deposit the money. For this Dad will access the ATM again. And update the current balance.
85          bal2 += DepositAmount;
86          V(semZero); // semEmpty to be 3
87          V(semZero);
88          V(semZero);
89          fprintf(fp1, "%d \n", bal2);
90          fclose(fp1);
91
92          printf("Dad writes new balance = %d \n", bal2);
93          printf("Dad will deposit %d more time\n",N-1); //Dad depostited the money.
94          V(semMutex);
95          V(semATM);
96
97          sleep(rand()%10+1); /* Dad will wait some time for requesting to see balance again.*/
```

Critical section of son processes:

```
176         if (pid == CHILD)
177         {
178             printf("Second Son's Pid: %d\n",getpid());
179             //Third child process. Second poor son tries to do updates.
180             flag1 = FALSE;
181
182             while(flag1 == FALSE)
183             {
184                 P(semATM);
185                 P(semWrAtt);
186                 printf("SON_2 is requesting to view the balance.\n"); //Son_2 is requesting to get hold of the ATM.
187
188                 fp3 = fopen("attempt.txt" , "r+"); //Son_2 successfully got hold of the ATM.
189                 fscanf(fp3, "%d", &N_Att); // Son_2 Checks if he has more than 0 attempt remaining.
190                 printf("Attempt remaining: %d.\n", N_Att);
191                 if(N_Att == 0)
192                 {
193                     fclose(fp3);
194                     flag1 = TRUE;
195                 }
196                 else
197                 {
198                     P(semMutex);
199                     fp2 = fopen("balance.txt", "r+"); //Son_2 reads the balance.
200                     fscanf(fp2,"%d", &bal2);
201                     printf("SON_2 reads balance. Available Balance: %d \n", bal2);
202                     printf("SON_2 wants to withdraw money. "); //And if balance is greater than Withdraw amount, then son can withdraw money.
203                     if (bal2 >= 20)
204                     {
205                         P(semZero);
206                         fseek(fp2,0L, 0);
207                         bal2 -=WithdrawAmount;
208                         fprintf(fp2,"%d\n", bal2);
209                         fclose(fp2);
210                         printf("SON_2 withdrawed %d. New Balance: %d \n",WithdrawAmount, bal2);
211                     }
212                     else
213                     {
214                         printf("SON_2 not enough balace to withdraw! \n");
215                     }
216
```

```
217
218                     V(semMutex);
219
220
221                     fseek(fp3,0L, 0); //SON_2 will write the number of  attempt remaining in the attampt.txt file.
222                     N_Att -=1;
223                     fprintf(fp3, "%d\n", N_Att);
224                     fclose(fp3);
225                     printf("Number of attempts remaining: %d \n", N_Att);
226                 }
227                 V(semWrAtt);
228                 V(semATM);
229                 sleep(rand()%10+1);//SON_2 will wait some time before the next request.
230             }
231         }
```

2. Solve race conditions:

The race condition is that the outcome of the execution of one process depends on the outcome of another process. When the two processes are executed in the meantime, unexpected outcomes can occur.

In this case, to solve the race conditions, we need to add a semaphore that prevent any 2 processes deposit/withdraw at the same time or a process deposit/withdraw while another process tries to read the balance. In the code, I called the semaphore to be semMutex, and initialize to be 1.

```
// Create semaphore
int semMutex = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semWrAtt = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semZero = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semATM = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);

sem_create(semMutex, 1); // semMutex avoid 2 process deposit/withdraw at the same time
sem_create(semWrAtt, 1); // semWrAtt avoid 2 chile process access attempt at the same time
sem_create(semZero, 0); // semZero avoid zero balance
sem_create(semATM, 2); // semATM restrict that only 2 processes can run at the same time
```

To avoid any 2 processes deposit/withdraw at the same time or a process deposit/withdraw while another process tries to read the balance, the critical section would be the code of dad deposit and son withdraw, then before dad deposit or son read balance, we call P(semMutex) to block other processes deposit/withdraw action, and after current process deposit/withdraw action is done, call V(semMutex) to free the semaphore. In this case, I did not put dad's read balance to the critical section since I don't want to block the sons' process for read when dad preparing money to deposit.

Critical section of dad:

```
65      if (pid == CHILD){
66      //First Child Process. Dad tries to do some updates.
67          printf("Dad's Pid: %d\n",getpid());
68          N=NumOfDepositAttempt;
69          for(i=1;i<=N; i++)
70          {
71
72              P(semATM);
73              printf("Dad is requesting to view the balance.\n"); //Dad is requesting to get hold of an ATM.
74              fp1 = fopen("balance.txt", "r+"); //Dad successfully got hold of the ATM.
75              fscanf(fp1, "%d", &bal2);
76              printf("Dad reads balance = %d \n", bal2);
77              int r = rand()%5+1;
78              printf("Dad needs %d sec to prepare money\n", r);
79              sleep(r); //Dad Process is sleeping for r sec. You need to make sure that other processes can work in the mean time.
80
81              P(semMutex);
82              fp1 = fopen("balance.txt", "r+");
83              fscanf(fp1, "%d", &bal2);
84              fseek(fp1,0L,0); //Dad will now deposit the money. For this Dad will access the ATM again. And update the current balance.
85              bal2 += DepositAmount;
86              V(semZero); // semEmpty to be 3
87              V(semZero);
88              V(semZero);
89              fprintf(fp1, "%d \n", bal2);
90              fclose(fp1);
91
92              printf("Dad writes new balance = %d \n", bal2);
93              printf("Dad will deposit %d more time\n",N-i); //Dad depostited the money.
94              V(semMutex);
95              V(semATM);
96
97              sleep(rand()%10+1); /* Dad will wait some time for requesting to see balance again.*/
```

Critical section of sons:

```
131             else
132             {
133 
134                 P(semMutex);
135                 fp2 = fopen("balance.txt", "r+");//Son_1 reads the balance.
136                 fscanf(fp2,"%d", &bal2);
137                 printf("SON_1 reads balance. Available Balance: %d \n", bal2);
138                 printf("SON_1 wants to withdraw money. "); //And if balance is greater than Withdraw amount, then son can withdraw money.
139 
140                 if (bal2 >= 20)
141                 {
142                     P(semZero);
143                     fseek(fp2,0L, 0);
144 
145                     bal2 -=WithdrawAmount;
146                     fprintf(fp2,"%d\n", bal2);
147                     fclose(fp2);
148                     printf("SON_1 withdrawed %d. New Balance: %d \n",WithdrawAmount, bal2);
149                 }
150                 else{
151                     printf("SON_1 not enough balace to withdraw! \n");
152                 }
153                 V(semMutex);
154 
155                 fseek(fp3,0L, 0); //SON_1 will write the number of  attempt remaining in the attamnt.txt file.
```

3. Prevent 2 sons accessing attempts at the same time
   To implement this, we need to add a semaphore in the code, I called the semaphore to be semWrAtt, and initialize to be 1.

```
// Create semaphore
int semMutex = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semWrAtt = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semZero = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semATM = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);

sem_create(semMutex, 1); // semMutex avoid 2 process deposit/withdraw at the same time
sem_create(semWrAtt, 1); // semWrAtt avoid 2 chile process access attempt at the same time
sem_create(semZero, 0); // semZero avoid zero balance
sem_create(semATM, 2); // semATM restrict that only 2 processes can run at the same time
```

To prevent 2 sons accessing attempts at the same time, the critical section would be from sons read attempt to sons write attempt, so before sons read attempt, we increase the semWrAtt by 1 by calling P(semWrAtt), and it will block another son to access attempt, and after write attempt, call V(semWrAtt) and it will free another son to access attempt.

Critical section:

```
111         if (pid == CHILD)
112         {
113             printf("First Son's Pid: %d\n",getpid());
114             //Second child process. First poor son tries to do updates.
115             flag = FALSE;
116 
117             while(flag == FALSE)
118             {
119                 P(semATM);
120                 P(semWrAtt);
121                 printf("SON_1 is requesting to view the balance.\n"); //Son_1 is requesting to get hold of the ATM.
122 
123                 fp3 = fopen("attempt.txt" , "r+"); //Son_1 successfully got hold of the ATM.
124                 fscanf(fp3, "%d", &N_Att); // Son_1 Checks if he has more than 0 attempt remaining.
125                 printf("Attempt remaining: %d.\n", N_Att);
126                 if(N_Att == 0)
127                 {
128                     fclose(fp3);
129                     flag = TRUE;
130                 }
131                 else
132                 {
133 
134                     P(semMutex);
135                     fp2 = fopen("balance.txt", "r+");//Son_1 reads the balance.
136                     fscanf(fp2,"%d", &bal2);
137                     printf("SON_1 reads balance. Available Balance: %d \n", bal2);
138                     printf("SON_1 wants to withdraw money. "); //And if balance is greater than Withdraw amount, then son can withdraw money.
139 
140                     if (bal2 >= 20)
141                     {
```

```
139
140                 if (bal2 >= 20)
141                 {
142                     P(semZero);
143                     fseek(fp2,0L, 0);
144
145                     bal2 -=WithdrawAmount;
146                     fprintf(fp2,"%d\n", bal2);
147                     fclose(fp2);
148                     printf("SON_1 withdrawed %d. New Balance: %d \n",WithdrawAmount, bal2);
149                 }
150                 else{
151                     printf("SON_1 not enough balace to withdraw! \n");
152                 }
153                 V(semMutex);
154
155                 fseek(fp3,0L, 0); //SON_1 will write the number of  attempt remaining in the attampt.txt file.
156                 N_Att -=1;
157                 fprintf(fp3, "%d\n", N_Att);
158                 fclose(fp3);
159                 printf("Number of attempts remaining:%d \n", N_Att);
160             }
161             V(semWrAtt);
162             V(semATM);
163             sleep(rand()%10+1); //SON_1 will wait some time before the next request.
164         }
165     }
```

4. Prevent undefined output (output < 0)
   The program would gives the output of negative value since sons process may withdraw money even if the balance is not enough.
   A way to solve this problem is creating a semaphore called semZero, and initialize to to be 0.

```
// Create semaphore
int semMutex = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semWrAtt = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semZero = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semATM = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);

sem_create(semMutex, 1); // semMutex avoid 2 process deposit/withdraw at the same time
sem_create(semWrAtt, 1); // semWrAtt avoid 2 chile process access attempt at the same time
sem_create(semZero, 0); // semZero avoid zero balance
sem_create(semATM, 2); // semATM restrict that only 2 processes can run at the same time
```

   Then when the dad deposit money, it will increase semZero by 3 times, since a single deposit supports 3 times of withdraws. Then in the child process, every time it withdraw money, it will reduce semZero by 1, if semZero is already 0, then it will block the son process until dad complete a deposit and semZero to be greater than 0 again, here is how I prevent it from not enough balance.

   For dad:

```
69      for(i=1;i<=N; i++)
70      {
71
72          P(semATM);
73          printf("Dad is requesting to view the balance.\n"); //Dad is requesting to get hold of an ATM.
74          fp1 = fopen("balance.txt", "r+"); //Dad successfully got hold of the ATM.
75          fscanf(fp1, "%d", &bal2);
76          printf("Dad reads balance = %d \n", bal2);
77          int r = rand()%5+1;
78          printf("Dad needs %d sec to prepare money\n", r);
79          sleep(r); //Dad Process is sleeping for r sec. You need to make sure that other processes can work in the mean time.
80
81          P(semMutex);
82          fp1 = fopen("balance.txt", "r+");
83          fscanf(fp1, "%d", &bal2);
84          fseek(fp1,0L,0); //Dad will now deposit the money. For this Dad will access the ATM again. And update the current balance.
85          bal2 += DepositAmount;
86          V(semZero); // semZero to be 3
87          V(semZero);
88          V(semZero);
89          fprintf(fp1, "%d \n", bal2);
90          fclose(fp1);
91
92          printf("Dad writes new balance = %d \n", bal2);
93          printf("Dad will deposit %d more time\n",N-i); //Dad depostited the money.
94          V(semMutex);
95          V(semATM);
96
97          sleep(rand()%10+1); /* Dad will wait some time for requesting to see balance again.*/
98      }
99  }
```

For son:

```
125             printf("Attempt remaining: %d.\n", N_Att);
126             if(N_Att == 0)
127             {
128                 fclose(fp3);
129                 flag = TRUE;
130             }
131             else
132             {
133
134                 P(semMutex);
135                 fp2 = fopen("balance.txt", "r+");//Son_1 reads the balance.
136                 fscanf(fp2,"%d", &bal2);
137                 printf("SON_1 reads balance. Available Balance: %d \n", bal2);
138                 printf("SON_1 wants to withdraw money. "); //And if balance is greater than Withdraw amount, then son can withdraw money.
139
140                 if (bal2 >= WithdrawAmount)
141                 {
142                     P(semZero);
143                     fseek(fp2,0L, 0);
144
145                     bal2 -=WithdrawAmount;
146                     fprintf(fp2,"%d\n", bal2);
147                     fclose(fp2);
148                     printf("SON_1 withdrawed %d. New Balance: %d \n",WithdrawAmount, bal2);
149                 }
150                 else{
151                     printf("SON_1 not enough balace to withdraw! \n");
152                 }
153                 V(semMutex);
```

5. Solve son's withdrawal while dad is preparing money

   To allow son to withdraw from account while dad is preparing money, since a withdrawal will change the account balance, a simplest solution is that dad read the balance before dad deposit. And since deposit/withdraw is synchronized, dad will get the balance up-to-date.

```
68          N=NumOfDepositAttempt;
69          for(i=1;i<=N; i++)
70          {
71
72              P(semATM);
73              printf("Dad is requesting to view the balance.\n"); //Dad is requesting to get hold of an ATM.
74              fp1 = fopen("balance.txt", "r+"); //Dad successfully got hold of the ATM.
75              fscanf(fp1, "%d", &bal2);
76              printf("Dad reads balance = %d \n", bal2);
77              int r = rand()%5+1;
78              printf("Dad needs %d sec to prepare money\n", r);
79              sleep(r); //Dad Process is sleeping for r sec. You need to make sure that other processes can work in the mean time.
80
81              P(semMutex);
82              fp1 = fopen("balance.txt", "r+");
83              fscanf(fp1, "%d", &bal2);
84              fseek(fp1,0L,0); //Dad will now deposit the money. For this Dad will access the ATM again. And update the current balance.
85              bal2 += DepositAmount;
86              V(semZero); // semZero to be 3
87              V(semZero);
88              V(semZero);
89              fprintf(fp1, "%d \n", bal2);
90              fclose(fp1);
91
92              printf("Dad writes new balance = %d \n", bal2);
93              printf("Dad will deposit %d more time\n",N-i); //Dad depostited the money.
94              V(semMutex);
95              V(semATM);
96
97              sleep(rand()%10+1); /* Dad will wait some time for requesting to see balance again.*/
98          }
99      }
```

6. Reduce unnecessary semaphore
   We can reduce some unnecessary semaphore in this case (2 ATM)

   semATM is unnecessary, since we have semWrAtt to control that only one child process can actually did some action on the account, therefore, totally will not have more than 2 processes actually take actions on the account.

   semZero is also not required, since we can use a simple if statement to control the not enough balance case.

   (I still keep these two semaphores in the source code.)

Final output:

Final output will be like this, it solve the problem of synchronize.

```
Dad's Pid: 70047
Dad is requesting to view the balance.
First Son's Pid: 70048
SON_1 is requesting to view the balance.
Dad reads balance = 0
Dad needs 5 sec to prepare money
Attempt remaining: 20.
Second Son's Pid: 70049
SON_1 reads balance. Available Balance: 0
SON_1 wants to withdraw money. SON_1 not enough balace to withdraw!
Number of attempts remaining:19
SON_2 is requesting to view the balance.
Attempt remaining: 19.
SON_2 reads balance. Available Balance: 0
SON_2 wants to withdraw money. SON_2 not enough balace to withdraw!
Number of attempts remaining: 18
Dad writes new balance = 60
Dad will deposit 4 more time
SON_1 is requesting to view the balance.
Attempt remaining: 18.
SON_1 reads balance. Available Balance: 60
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: 40
Number of attempts remaining:17
SON_2 is requesting to view the balance.
Attempt remaining: 17.
SON_2 reads balance. Available Balance: 40
SON_2 wants to withdraw money. SON_2 withdrawed 20. New Balance: 20
Number of attempts remaining: 16
Dad is requesting to view the balance.
Dad reads balance = 20
Dad needs 4 sec to prepare money
Dad writes new balance = 80
Dad will deposit 3 more time
SON_1 is requesting to view the balance.
Attempt remaining: 16.
SON_1 reads balance. Available Balance: 80
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: 60
Number of attempts remaining:15
SON_2 is requesting to view the balance.
Attempt remaining: 15.
SON_2 reads balance. Available Balance: 60
SON_2 wants to withdraw money. SON_2 withdrawed 20. New Balance: 40
```

```
Dad reads balance = 40
Attempt remaining: 8.
Dad needs 3 sec to prepare money
SON_2 reads balance. Available Balance: 40
SON_2 wants to withdraw money. SON_2 withdrawed 20. New Balance: 20
Number of attempts remaining: 7
SON_1 is requesting to view the balance.
Attempt remaining: 7.
SON_1 reads balance. Available Balance: 20
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: 0
Number of attempts remaining:6
Dad writes new balance = 60
Dad will deposit 0 more time
child(pid = 70047) exited with the status 0.
SON_1 is requesting to view the balance.
Attempt remaining: 6.
SON_1 reads balance. Available Balance: 60
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: 40
Number of attempts remaining:5
SON_2 is requesting to view the balance.
Attempt remaining: 5.
SON_2 reads balance. Available Balance: 40
SON_2 wants to withdraw money. SON_2 withdrawed 20. New Balance: 20
Number of attempts remaining: 4
SON_1 is requesting to view the balance.
Attempt remaining: 4.
SON_1 reads balance. Available Balance: 20
SON_1 wants to withdraw money. SON_1 withdrawed 20. New Balance: 0
Number of attempts remaining:3
SON_2 is requesting to view the balance.
Attempt remaining: 3.
SON_2 reads balance. Available Balance: 0
SON_2 wants to withdraw money. SON_2 not enough balace to withdraw!
Number of attempts remaining: 2
SON_1 is requesting to view the balance.
Attempt remaining: 2.
SON_1 reads balance. Available Balance: 0
SON_1 wants to withdraw money. SON_1 not enough balace to withdraw!
Number of attempts remaining:1
SON_2 is requesting to view the balance.
Attempt remaining: 1.
SON_2 reads balance. Available Balance: 0
SON_2 wants to withdraw money. SON_2 not enough balace to withdraw!
Number of attempts remaining: 0
SON_1 is requesting to view the balance.
Attempt remaining: 0.
SON_2 is requesting to view the balance.
Attempt remaining: 0.
child(pid = 70048) exited with the status 0.
child(pid = 70049) exited with the status 0.
```