The City College of New York

Computer Science Department

Operating Systems

CSC 33500

Lab 5

By Nafis Khan

Submit a report showing the critical section of the code (and logical errors) and explain your solution in detail:

Semaphore was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as a semaphore. Semaphore is simply an integer variable that is shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.

In the case of this Lab, an easy-to-use header file has been provided to work with the semaphores. Initially, the semcall is used to initialize the file and at the end, semkill is used to kill and remove everything from the memory. The P function is used to wait or halt other processes from using or entering the critical section and V function is used to release the memory and share access to that memory location again.

In this lab, three main critical sections were found, and P and V operations were implemented to handle the withdrawal and deposits of balances into the bank account so that there aren't any ghost copies of older values being used by a different process. One of them is for the dad depositing money, and the other two for son 1 and son 2 withdrawing money. Screenshots of the critical section of the code along with other adjustments can be seen at the end of the report.

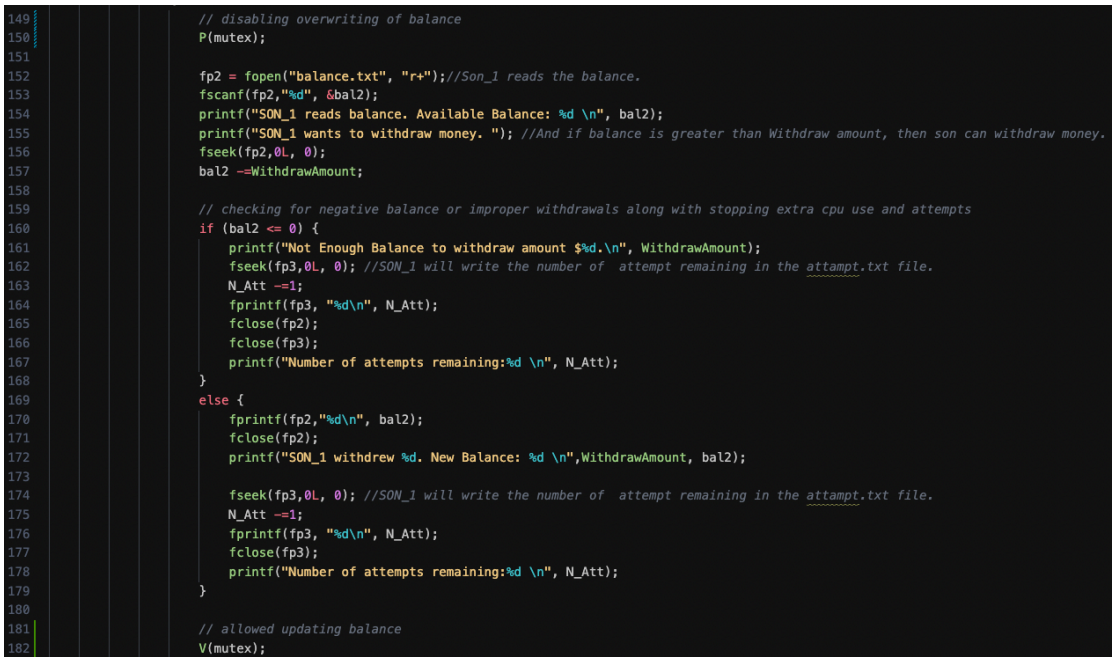
While going through the project, there were no compilation issues, but the output seemed a little strange. After taking a deeper look into it, the initial realization was to fix the shared bank balance resource so that all the code is synchronized and does not cause any balance issues. But two other logical errors were also caught after that. One was to handle a negative bank balance and the other to handle attempts even if the balance is negative. This could have been handled using a semaphore method, but I chose a simpler route of just checking the balance to see if a withdrawal was possible or not with the amount of money in the bank. If not, then it is counted as an attempt like regular, but no money is actually drawn since there is not enough or none at

all. If there is enough balance, then everything is done like before and money is withdrawn from the account and the balance in the bank is subtracted accordingly. Therefore, taking care of the negative balance issues along with handling any unneeded attempts to access or write to the file.

Screenshots:

```
68          //First Child Process.
69          //Dad tries to do some updates.
70          /**you need to identify the logical issues and synchronization issues and solve them using logic and semaphores.*/
71          printf("Dad's Pid: %d\n",getpid());
72          N=NumOfDepositAttempt;
73          for(i=1;i<=N; i++)
74          {
75              //Dad process need some time to go to the bank.
76              int r = rand()%5+1;
77              sleep(r);
78              //After r second Dad process reached the Bank.
79
80              // disabling overwriting of balance
81              P(mutex);
82
83              printf("Dad is requesting to view the balance.\n"); //Dad is requesting to get hold of an ATM.
84              fp1 = fopen("balance.txt", "r+"); //Dad successfully got hold of the ATM.
85              fscanf(fp1, "%d", &bal2);
86              printf("Dad reads balance = %d \n", bal2);
87              r = rand()%5+1;
88              printf("Dad wants to deposit money\n");
89              printf("Dad needs %d sec to prepare money.\n", r);
90              sleep(r); //Dad Process is sleeping for r sec. You need to make sure that other processes can work in the mean time.
91
92              //After some time Dad process wakes up.
93              //It is possible that the balance has changed during the time dad process is sleeping
94              //Dad process starts to deposit the money. So Dad process needs the access to ATM.
95              //Only after getting access to the ATM dad process can deposit money.
96
97              fseek(fp1,0L,0); //Dad will now deposit the money. And update the current balance.
98              bal2 += DepositAmount;
99              fprintf(fp1, "%d \n", bal2);
100             fclose(fp1);
101             printf("Dad writes new balance = %d \n", bal2);
102             printf("Dad will deposit %d more time\n",N-i); //Dad deposited the money.
103             printf("\n");
104             sleep(rand()%10+1); /* Dad will wait some time for requesting to see balance again.*/
105
106             // allowed updating balance
107             V(mutex);
108
```

Figure 1: Father depositing money.



```
// disabling overwriting of balance
P(mutex);

fp2 = fopen("balance.txt", "r+");//Son_1 reads the balance.
fscanf(fp2,"%d", &bal2);
printf("SON_1 reads balance. Available Balance: %d \n", bal2);
printf("SON_1 wants to withdraw money. "); //And if balance is greater than Withdraw amount, then son can withdraw money.
fseek(fp2,0L, 0);
bal2 -=WithdrawAmount;

// checking for negative balance or improper withdrawals along with stopping extra cpu use and attempts
if (bal2 <= 0) {
    printf("Not Enough Balance to withdraw amount $%d.\n", WithdrawAmount);
    fseek(fp3,0L, 0); //SON_1 will write the number of  attempt remaining in the attampt.txt file.
    N_Att -=1;
    fprintf(fp3, "%d\n", N_Att);
    fclose(fp2);
    fclose(fp3);
    printf("Number of attempts remaining:%d \n", N_Att);
}
else {
    fprintf(fp2,"%d\n", bal2);
    fclose(fp2);
    printf("SON_1 withdrew %d. New Balance: %d \n",WithdrawAmount, bal2);

    fseek(fp3,0L, 0); //SON_1 will write the number of  attempt remaining in the attampt.txt file.
    N_Att -=1;
    fprintf(fp3, "%d\n", N_Att);
    fclose(fp3);
    printf("Number of attempts remaining:%d \n", N_Att);
}

// allowed updating balance
V(mutex);
```

Figure 2: Son 1 withdrawing from balance

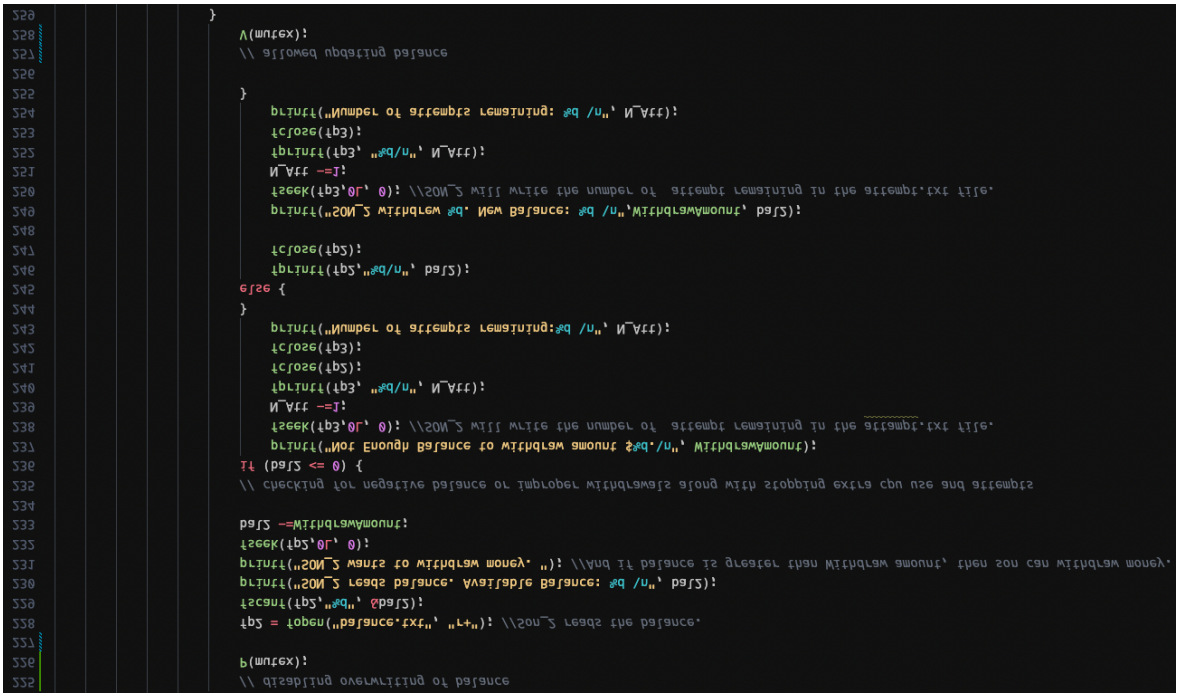


Figure 3: Son 2 withdrawing from balance