Gaurav Kuwar
CSC 332 LAB (Spring 2022)
May 5, 2022
Lab 5: Process Synchronization

Task 2 - Report

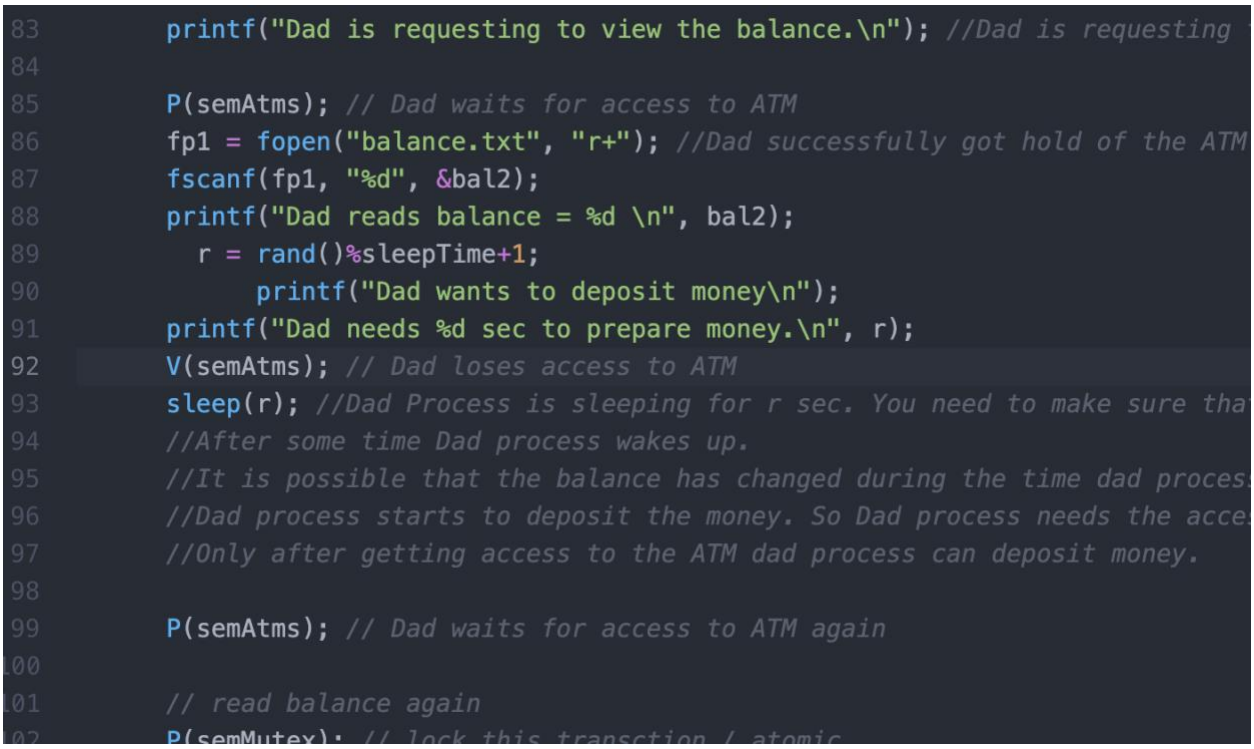
I used 3 semaphores to solve the issues.

```
int semAtms = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semMutex = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
int semNoBalance = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);

sem_create(semMutex, 1); // used to lock a transction, to prevent race conditions
sem_create(semAtms, 2); // This makes sure only 2 process are running at once
                        // Since there are only 2 ATM machines

// this semaphore is used to check if there is enough balance to be withdrawn
if (initBalance >= WithdrawAmount)
  sem_create(semNoBalance, 1);
else
  sem_create(semNoBalance, 0);
```

semMutex – is used to lock a transaction and prevent race condition by making the portion of code modifying balance.txt and/or attemps.txt one transction.

semAtms – used to make sure 2 processes are running at once, since there are only 2 ATM machines in the bank.

semNoBalance – main purpose is to prevent negative balance. This is used to check if there is enough balance to be withdrawn by a son process.



```
83        printf("Dad is requesting to view the balance.\n"); //Dad is requesting
84
85        P(semAtms); // Dad waits for access to ATM
86        fp1 = fopen("balance.txt", "r+"); //Dad successfully got hold of the ATM
87        fscanf(fp1, "%d", &bal2);
88        printf("Dad reads balance = %d \n", bal2);
89          r = rand()%sleepTime+1;
90              printf("Dad wants to deposit money\n");
91        printf("Dad needs %d sec to prepare money.\n", r);
92        V(semAtms); // Dad loses access to ATM
93        sleep(r); //Dad Process is sleeping for r sec. You need to make sure tha
94        //After some time Dad process wakes up.
95        //It is possible that the balance has changed during the time dad proces
96        //Dad process starts to deposit the money. So Dad process needs the acce
97        //Only after getting access to the ATM dad process can deposit money.
98
99        P(semAtms); // Dad waits for access to ATM again
100
101       // read balance again
102       P(semMutex); // lock this transction / atomic
```

This section of code is the first part of the dad process, where the dad process simply reads the balance from balance.txt. I use the P(semAtms) to wait for dad procees to get access to the ATM. But, then the dad process sleeps, and at this time since other processes need access to the ATM, so I use V(semAtms) to stop dad process from access to ATM. After the sleep dad process waits again for access to the ATM.

```
99          P(semAtms); // Dad waits for access to ATM again
100
101         // read balance again
102         P(semMutex); // lock this transction / atomic
103         fp1 = fopen("balance.txt", "r+"); //Dad successfully got hold of the ATM.
104         fscanf(fp1, "%d", &bal2);
105
106         fseek(fp1,0L,0); //Dad will now deposit the money. And update the current balance
107         bal3 = bal2 + DepositAmount; // bal2 is saved as previous balance
108         fprintf(fp1, "%d \n", bal3);
109         fclose(fp1);
110         printf("Dad writes new balance = %d \n", bal3);
111
112         // for 1 son process waiting for enough balance
113         if (bal2 < WithdrawAmount && bal3 >= WithdrawAmount)
114           V(semNoBalance);
115         // for both son process waiting for enough balance
116         if (bal2 < (2*WithdrawAmount) && bal3 >= (2*WithdrawAmount))
117           V(semNoBalance);
118
119         V(semMutex); // unlock
120
121         printf("Dad will deposit %d more time\n",N-i); //Dad depostited the money.
122         V(semAtms); // Dad loses access to ATM
123
124         sleep(rand()%(sleepTime*2)+1);  /* Dad will wait some time for requesting to see
```

This is the second part of the dad process. Here, the dad process gets access to the ATM again. Then, I used P(semMutex) so the atomic transaction can begin. I read the balance again, since it could have changed in the time dad process was asleep. I add the deposit amount to balance, but I have a bal3 integer, so I can store the balance before deposit, in bal2. I then use 2 if statements, which use V(semNoBalance) to signal the son processes waiting for enough balance. The second if statement is for situations where both sons are waiting for enough balance. I then use V(semMutex) to unlock the transaction and V(semAtms) so that the Dad process can lose access to ATM.

```
154         P(semAtms);
155         fp3 = fopen("attempt.txt" , "r+"); //Son_1 successfully got hold of the ATM.
156         fscanf(fp3, "%d", &N_Att); // Son_1 Checks if he has more than 0 attempt remaining.
157         printf("Attempt remaining: %d.\n", N_Att);
158         if(N_Att == 0)
159         {
160           fclose(fp3);
161           flag = TRUE;
162         }
163         else
164         {
165
166           fp2 = fopen("balance.txt", "r+");//Son_1 reads the balance.
167           fscanf(fp2,"%d", &bal2);
168           printf("SON_1 reads balance. Available Balance: %d \n", bal2);
169           printf("SON_1 wants to withdraw money. \n"); //And if balance is greater than Withdraw amount
170
```

```
168         printf("SON_1 reads balance. Available Balance: %d \n", bal2);
169         printf("SON_1 wants to withdraw money. \n"); //And if balance is greater than Withdraw amo
170
171         // when there is not enough balance / to prevent negative balance
172         if (bal2 < WithdrawAmount) {
173           V(semAtms); // son loses access to ATM
174           P(semNoBalance); // waits for dad to put balance
175           P(semAtms); // son gains access to ATM again
176         }
177
178         P(semMutex); // lock this transction / atomic
179         // read balance again since it could have changed
180         fp2 = fopen("balance.txt", "r+");
181         fscanf(fp2,"%d", &bal2);
182
183         fseek(fp2,0L, 0);
184         bal2 -=WithdrawAmount;
185         fprintf(fp2,"%d\n", bal2);
186         fclose(fp2);
187         printf("SON_1 withdrawed %d. New Balance: %d \n",WithdrawAmount, bal2);
188
189         // read attempts again since it could have changed
190         fp3 = fopen("attempt.txt" , "r+");
191         fscanf(fp3, "%d", &N_Att);
192
193         fseek(fp3,0L, 0); //SON_1 will write the number of  attempt remaining in the attampt.txt
194         N_Att -=1;
195         fprintf(fp3, "%d\n", N_Att);
196         fclose(fp3);
197         printf("Number of attempts remaining:%d \n", N_Att);
198         V(semMutex); // unlock
```

This is a section of the son 1 process code (but the same idea applies to son 2 process as well). First, the son process starts with P(semAtms) which gives the process access to ATM, and the iteration of the code exits with V(semAtms) before sleeping just like with the dad process. The if statement on line 172 is used to prevent negative balance. It checks if the bal2 is less than the withdrawamount, and if it is it will stop access to ATM, then it will call P(semNoBalance), at this point, since there isn't enough balance semNoBalance would be 0, so the son process will now wait for the dad process to add enough balance and call V(semNoBalance), as describe in the previous code. Once, there is enough balance, the process can continue, it gains access back to ATM (waits first). Then it waits for semMutex, to start its transaction.

In this transaction, balance is read again since it could have changed, and the withdraw actions is performed. I also read attempt.txt again, since it could have changed too. Finally, unlock the transction with V(semMutex). semMutex semaphore is used to prevent race conditions, since balance.txt and attempt.txt are both being modified in this transaction.