When the process’ aren’t always run in the same order there is a race condition involved. In order to fix this, we must use semaphores and place P(sem) and V(sem) commands around the critical sections of the code to allow for only 1 user to be accessing the data (balance) at once. My solution prevents the race condition by only allowing either the dad to be putting money into the account, son 1 to take money, or son 2 to take money, and never all three or two at once. This prevents the family me members from looking into the account at the same time and taking out money (son’s case) or inserting money (dad case). This in turn caused a longer run time, but there was no overlapping of processes. In order to time the processes themselves I also used semaphores. The timing was not done in a sense of how long it takes, but how many times it was ready to execute when another process was currently executing. Timings for various values of N can be seen below in table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Value of N | Time for Dad | Time for Son 1 | Time for Son 2 |
| 5 | 15 | 2 | 3 |
| 6 | 13 | 3 | 2 |
| 7 | 11 | 3 | 2 |
| 9 | 7 | 3 | 2 |
| 10 | 5 | 3 | 2 |

Table 1: Shows wait timings for dad, son 1, and son 2 for various values of N.

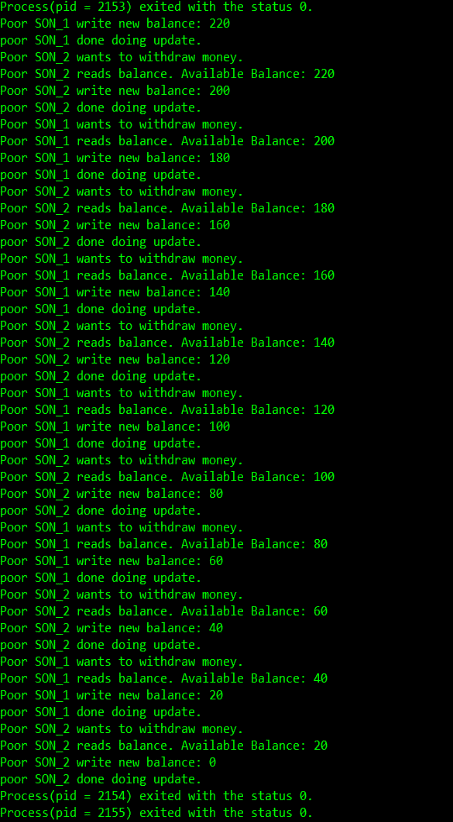
 From the table we can see that the timings go down the more we increase N for the dad, that is because N is the amount of times the dad will be putting money into the account so doing it more often means less time waiting. Below you can see the output before I used semaphores and then after adding semaphores. You will notice that image 1 has child 1 and child 2 taking from the fund at the same time and therefore find the value the same, which does not make any sense. The second image shows this fixed with one son taking after the other and no overlapping values.

Image 1 on the left shows a race condition without semaphores. While Image 2 on the right shows the race condition resolved by using semaphores.