*1. Solving an IPC Problem using Semaphores*

**Example**

Open the file [example.c](https://giantprosper-digitalabraham.codio.io/example.c). This program creates one process (i.e. Child and of course the original program is the Parent).  
Before creating the process, the program creates a shared integer variable (counterptrcounterptr) using a memory mapped file to  
share the integer with the Parent and Child Processes. The memory mapped file only contains an integer that is incremented  
by both the parent and child processes. Second, the program create a semaphore mutexmutex. The semaphore is shared by the parent and child processes.

Run the program, and notice how the Parent and Child process are not in their critical sections at the same time. The semaphore operations  
sem\_wait (i.e. DOWN), and sem\_post(i.e. UP) on the semaphore mutexmutex enforce Mutual Exclusion. Therefore, no two processes can increment the shared counter variable counterptrcounterptr

**Poor Student - Dear Old Dad Problem**

You are to modify the program in Lab 3 - Part 2 that creates one shared variable of type int to represent the BankAccount, and as many other variables you will need for your solution. Your program must create two processes, one parent process and one child process, and allow the parent to deposit (i.e. add) money to the BankAccount, and also allow the child processes to withdraw (i.e. subtract) money from the BankAccount using Semaphores to ensure no race conditions will occur(see OS textbook Chap 5).Both processes (Dear Old Dad, and Poor Student) are able to check their balance randomly. Create as many Semaphores needed for your solution. Parent and Child processes should loop indefinitely and follow the rules below each time through the loop.

**Parent (Dear Old Dad) Rules**

1. Sleep some random amount of time between 0 - 5 seconds
2. After waking up, call printf("Dear Old Dad: Attempting to Check Balance\n");
3. Generate a random number. If the number is even, then if the last localBalance is < 100, then try to Deposit Money, else call printf("Dear old Dad: Thinks Student has enough Cash ($%d)\n", localBalance);
4. If the number is odd, then just check the balance by calling printf("Dear Old Dad: Last Checking Balance = $%d\n", localBalance);

Deposit Money:

1. Copy contents of BankAccount into a local non-shared variable localBalance
2. Randomly generate a amount to give the Student between 0−0−100
3. If the random number is even: Deposit the amount into the BankAccount, then call printf("Dear old Dad: Deposits $%d / Balance = $%d\n", amount, localBalance);
4. Copy contents of non-shared local variable localBalance bank into the shared variable BankAccount
5. If the random number is odd: Then call printf("Dear old Dad: Doesn't have any money to give\n");

**Child (Poor Student) Rules**

1. Sleep some random amount of time between 0 - 5 seconds
2. After waking up, call printf("Poor Student: Attempting to Check Balance\n");
3. Generate a random number. If this number is even, attempt to Withdraw Money.
4. If the number is odd, then just check the last balance by calling printf("Poor Student: Last Checking Balance = $%d\n", localBalance);

Withdraw Money:

1. Copy contents of BankAccount into a local non-shared variable localBalance
2. Randomly generate another number (need) that the Student needs between 0−0−50, then call printf("Poor Student needs $%d\n", need);
3. If the need is <= the localBalance: Withdraw the amount from the BankAccount, then call printf("Poor Student: Withdraws $%d / Balance = $%d\n", need, localBalance);
4. If the need is > than the localBalance Then call printf("Poor Student: Not Enough Cash ($%d)\n", localBalance);
5. Copy contents of non-shared local variable localBalance bank into the shared variable BankAccount

To kill the program, open another terminal, and execute the ps -eaf command. Send a signal to the parent process. Run the reference implementation ref\_psdd to compare your output for testing purposes.

**Extra Credit 4pts - Add an extra Parent (Lovable Mom), and *N* additional children**

Your program must add additional process - Lovable Mom, and up to … *N* child processes (*Poor Student(N)*). These values will be entered from the command line when starting the process. For example: if your executable is called ref\_psdd, then starting just a Dear Old Dad process, and 3 Poor Students you would enter:

ref\_psdd 1 3

To start both Mom and Dad, and 10 Poor students you would enter:

ref\_psdd 2 10

Starting one Dear Old Dad, and one Poor Student like in the original problem above you would enter:

ref\_psdd 1 1

Please add code for the Lovable Mom to follow the rules below. Also, Dear Old Dad and each Poor Student (child processes) follow the rules that allow them to check their localbalance randomly, and perform deposits and withdraws as before. Also, add any needed shared variables and Semaphores to ensure no race conditions occur between Mom, Dad, and the Children.

**Parent (Lovable Mom) Rules**

1. Sleep some random amount of time between 0 - 10 seconds
2. After waking up, call printf("Loveable Mom: Attempting to Check Balance\n");
3. if the last localBalance is <= 100, then always Deposit Money

Deposit Money:

1. Copy contents of BankAccount into a local non-shared variable localBalance
2. Randomly generate an amount to give the Student between 0−0−125
3. Deposit the amount into the localBalance, then call printf("Lovable Mom: Deposits $%d / Balance = $%d\n", amount, localBalance);
4. Copy contents of non-shared local variable localBalance bank into the shared variable BankAccount