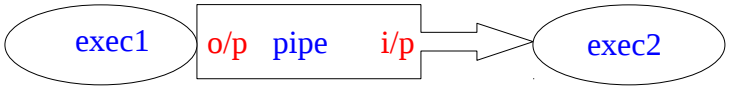
Understanding the pipe operator of shell

The pipe operator of the shell is used to connect two or more shell commands through a pipe. Essentially the output produced by the command at the left end of a pipe is buffered as input to the command at the right end of the pipe. If exec1 and exec2 are two executable files, then they are connected through a pipe using the pipe operator | as shown below.

$ ./exec1 | ./exec2



What is not at all obvious is that both the commands connected through a pipe are run concurrently. The behaviour of commands connected through a a pipe is examined through a series of experiments.

The following C++ program is used in this document for illustration :

```
/* filename : prime.C
     i/p : a range [ low, high]
     o/p : all prime numbers in the range
     method :
       a) Check for each number n, whether any number i, 2 <= i <= floor(sqrt(n)),  is a factor of n
       b) if no such i exists, n is prime and composite otherwise
 */

   #include <iostream>
   using namespace std;

   int main()
   {
     int low, high;
     int primecount = 0, factcount;
     cout << " give 2 integers as range low and high  ";
     cin >> low >> high;
     for ( int num = low; num <= high; num++)
     {  factcount = 0;
       for ( int i = 2; i*i <= num ; i++ )
         if ( num % i == 0 ) factcount++; // efficient ???
       if (factcount == 0)
       {
         primecount ++;
         cout << " Prime No. " << primecount << " is : " << num << endl;
       }
      }
      cout << " There are " << primecount << " primes in the range ["
           << low << " , " << high << "]" << endl;
      return 0;
   }
```

Experiment 1 :

Executable file, prime, is produced from prime.C which finds and reports all primes in the range from 3000000 to 5999999.

```
$ g++ prime.C -o prime
```

Execute the command : **$ time ./prime**

The output is saved in the file "prout". The display on the screen is given below.

Run1 : Result of execution of prime for first time

```
   $ time ./prime
 Process Id : prime 1 2636

real    0m22.487s
user    0m22.253s
sys     0m0.184s
```

Run 2: Result of execution of prime for second time

```
~/Desktop/cs101_2013/2013/lec18/shell$ time ./prime
 Process Id : prime 1 2711

real    0m22.472s
user    0m22.221s
sys     0m0.204s
```

Experiment 2 :

Three executables, prime1, prime2 and prime3, are produced from three files prime1.C, prime2.C and prime3.C which determine primes in the ranges [3000000, 3999999], [4000000, 4999999] and [5000000, 5999999] respectively. Simply stated that workload of prime.C is now distributed to three prime programs present in prime1.C, prime2.C and prime3.C as shown below.

```
$ s++ prime1.C -o prime1
$ s++ prime2.C -o prime2
$ s++ prime3.C -o prime3
```

Execute the following command

$ time (./prime1 | ./prime2 | ./prime3)

The output of the three programs are saved in the file, say "**out**". Each executable opens the same file "**out**" in append mode and writes its output to it. The display on the screen is given below.

Run1 : Result of execution of three prime processes connected through pipe for first time

```
~/Desktop/cs101_2013/2013/lec18/shell$ time (./prime1 | ./prime2 | ./prime3)
 Process Id : prime 2 2677
 Process Id : prime 1  Process Id : prime 3 2676     2678

real    0m11.587s
user    0m29.202s
sys     0m0.276s
```

Run2 : Result of execution of three prime processes connected through pipe for second time

```
~/Desktop/cs101_2013/2013/lec18/shell$ time (./prime1 | ./prime2 | ./prime3)
 Process Id : prime 3 2704
 Process Id : prime 2  Process Id : prime 1 2702   2703

real    0m11.078s
user    0m29.802s
sys     0m0.260s
```

Explanation :

1. The three executables, prime1, prime2 and prime3 lead to the creation of three separate processes in Experiment 2. Single process is created from the executable prime in Experiment 1.

2. Process id is an unique identifier, which is a non-negative integer, assigned by the Operating System (OS) to a process at the time of creation. For example, the process ids attached to the 3 processes in the two data sets are given below. To know the process identifier, while it is executing, we make the system call, getpid(). This function, being part of the kernel, is allowed to access the OS data structure, in particular the process id assigned to the executing process. Function getpid() returns this as an integer value. This function is declared in the header file, <unistd.h>

```
 int proc_id;
proc_id = getpid(); // needs the header file <unistd.h>
```

Process	Process id : Run 1	Process id : Run 2
prime1	2676	2702
prime2	2677	2703
prime3	2678	2704
prime	2636	2711

3. Examine the file "**prout**"; a few snapshots of this file is extracted below for your reference. There are 196034 lines in the file. Each of the first 196033 lines show a prime number being displayed, while the last line gives a summary. Though obvious from the sequence in each line, the output displayed is in the same order as the primes are detected in the for-loop

```
prime :  Prime No. 1 is : 3000017
prime :  Prime No. 2 is : 3000029
prime :  Prime No. 3 is : 3000047
prime :  Prime No. 4 is : 3000061
prime :  Prime No. 5 is : 3000073
prime :  Prime No. 6 is : 3000077
prime :  Prime No. 7 is : 3000089
prime :  Prime No. 8 is : 3000103
prime :  Prime No. 9 is : 3000131
prime :  Prime No. 10 is : 3000133

.....

prime :  Prime No. 196030 is : 5999927
prime :  Prime No. 196031 is : 5999933
prime :  Prime No. 196032 is : 5999947
prime :  Prime No. 196033 is : 5999993

prime :  There are 196033 primes in the range [3000000 , 5999999]
```

4. Examine the file "**out**", which is the output of Experiment 2.

```
prime 3:  Prime No. 1 is : 5000011
prime 1 :  Prime No. 1 is : 3000017
prime 1 :  Prime No. 2 is : 3000029
prime 2 :  Prime No. 1 is : 4000037
prime 2 :  Prime No. 2 is : 4000039
prime 1 :  Prime No. 3 is : 3000047
prime 2 :  Prime No. 3 is : 4000043
prime 3:  Prime No. 2 is : 5000077
prime 3:  Prime No. 3 is : 5000081
prime 3:  Prime No. 4 is : 5000087
prime 1 :  Prime No. 4 is : 3000061
prime 3:  Prime No. 5 is : 5000101
prime 2 :  Prime No. 4 is : 4000063
prime 1 :  Prime No. 5 is : 3000073
prime 2 :  Prime No. 5 is : 4000067
prime 1 :  Prime No. 6 is : 3000077
prime 3:  Prime No. 6 is : 5000111
prime 3:  Prime No. 7 is : 5000113
prime 1 :  Prime No. 7 is : 3000089
prime 2 :  Prime No. 6 is : 4000079
prime 2 :  Prime No. 7 is : 4000081
```

Observe the pattern of lines in the first 21 lines of output to get a feel for the scheduling order of the processes.

prime3[1], prime1[2], prime2[2], prime1[1], prime2[1], prime3[3], prime1[1], prime3[1], prime2[1], prime1[1], prime2[1]. Prime1[1], prime3[2], prime1[1], prime2[2], ...

The notation primex[n] denotes that the executable primex produced n number of lines of output. Examining the output leads to the following observations.

1. The order in which the three processes are scheduled is not constant, {3, 1, 2, 1, 2, 3, 1, 3, 2,...} as seen in the output file.

2. The number of primes detected in one time slice, indicated by the number of lines of output during the time period, is also not the same for the processes, as shown by :

Time slice	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Proc[lines]	p3[1]	p1[2]	p2[2]	p1[1]	p2[1]	p3[3]	p1[1]	p3[1]	p2[1]	p1[1]

Even for the first ten time slices, the variation in number of primes detected for p1 and p2 is from 1 to 2, while for p3 it is from 1 to 3.

3. The following three lines of output gives information about the total number of primes found by the three prime processes.

Line No. 166777 : prime 3 : There are 64336 primes in the range [5000000 , 5999999]
Line No. 188803 : prime 1 : There are 66330 primes in the range [3000000 , 3999999]
Line No. 196036 : prime 2 : There are 65367 primes in the range [4000000 , 4999999]

Examining the entire output, we obtain the following information shown below.

	Lines 1 - 166777 of output	Lines 166778 - 188803 of output	Lines 166778 - 188803 of output	Total
prime1	54705	11626	-	66331
prime2	47735	10400	7233	65368
prime3	64337	-	-	64337
Total	166777	22026	7233	196036

The share of each process in the first 166777 lines of output reveals that share of prime 3 is around 38.5% while that of prime1 and prime2 are 33% and 29% respectively. After a certain time duration prime3 terminates (corresponding to line 166777) and the other two prime processes are scheduled thereafter. After some more time slices (corresponding to line 188803), prime1 terminates and for the remaining time slices only the process prime2 is scheduled till completion. It has been assumed for the sake of illustration that there are only 3 processes in the system, but in reality the 3 prime processes will compete with other processes of the system.

If you repeat the same experiment again, the output is quite likely to be different than what is shown above. The non-repeatability of the order of the output data shows the dynmic aspects of process scheduling which depends on many system parameters including its load.

****** end of document *******