# Linux Concurrency Mechanisms

**Shuquan Li**

**April 22, 2014**

# ECT362 Homework #6

## Introduction:

This homework is to write a program in Linux. It is asked to have two processes share information using an anonymous pipe for communication. The information should have three charters and a number as stock symbols and value. It is also asked that the program will run for a user determined amount of time.

## Design and Analysis:

The program is designed as follows:

Read command line.
Run program and get the time that the user determines to run.
Show the stock symbols and value.
End the program when time's up.

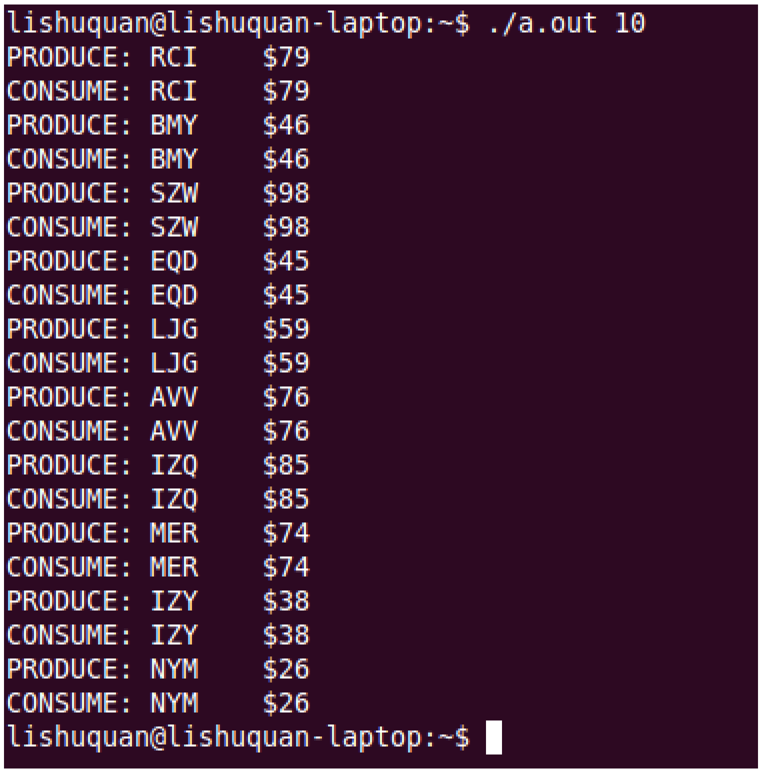
When the program starts, it will read the command line to get the name of the exe file and the time to run program. Then it will show the stock symbols and value on the screen until the time is up.

Since the stock has three characters and an integer as the symbols and value, a data structure is used to store them and make them easier to transmit in a pipe.

In order to make the stock information as a critical resource, a pipe is used to provide exclusion use of that information. Because it is blocking read and unblocking write, child process will be blocked if there is nothing in the pipe when it tries to read from the pipe. The write end of the pipe will be closed before reading, and the read end of the pipe will be closed before writing.

The time() function is used to get the current time. Make the number that the user types as a command line argument. Treat that number as a time_t data type and add it to current time in order to get the end time. Compare current time and end type; if current time is less than end time, keeping running the program; if current time is greater than end time then the program will end.

Figure 1 is the screen shot of cmd.exe when the program is running. To make it looks clear and the data match more easily, the sleep() function is used. The figure indicates that the program will run 10 seconds and list all the stock information both from the producer and the consumer. The consumer's information is the same as the producer's information before because the consumer's information is read what the producer writes in from the pipe.



```
lishuquan@lishuquan-laptop:~$ ./a.out 10
PRODUCE: RCI    $79
CONSUME: RCI    $79
PRODUCE: BMY    $46
CONSUME: BMY    $46
PRODUCE: SZW    $98
CONSUME: SZW    $98
PRODUCE: EQD    $45
CONSUME: EQD    $45
PRODUCE: LJG    $59
CONSUME: LJG    $59
PRODUCE: AVV    $76
CONSUME: AVV    $76
PRODUCE: IZQ    $85
CONSUME: IZQ    $85
PRODUCE: MER    $74
CONSUME: MER    $74
PRODUCE: IZY    $38
CONSUME: IZY    $38
PRODUCE: NYM    $26
CONSUME: NYM    $26
lishuquan@lishuquan-laptop:~$ 
```

Figure 1: The screen capture of cme.exe

**Conclusion**

It is desired to write a program to meet these requirements: run the program for the user determined amount of time, show the stock information from both producer and consumer process.

In doing this homework, the biggest problem is about the time() function. It's needed to know what the result of the time() function first. Then it's needed to find a way in order to make the number that the user types to be the end time which can compare with the current time.

**References**

*Generating random characters in C.* Kah – The Developer. nd. np. Web. Apr. 20, 2014

*function time.* cpulusplus.com. nd. np. Web. Apr. 20, 2014

*type time_t.* cpulusplus.com. nd. np. Web. Apr. 20, 2014

*Problem with time() using g++.* velocityreviews. nd. np. Web. Apr. 20, 2014

**Appendix**

```
/* Preprocessor directives */
#include <stdio.h> // printf
#include <iostream> // cin, cout, endl
```

```
#include <stdlib.h> // random number generation, atoi
#include <unistd.h> // pipe
#include <time.h> // time() function

using namespace::std; // using the standard namespace

/* Functions */
/****************************************************************************/
/* Name: main()                                                             */
/* Description: Main loop of execution, where the child threads are created,*/
/*              executed, and terminated                                    */
/****************************************************************************/
int main(int argc, char* argv[])
{
  int chPID; // process ID for the child process
  int pipeID[2];
  time_t run_time, current_time, end_time;

  struct stock
  {
       char symbol1;  // the first character of symbol
       char symbol2;  // the second character of symbol
       char symbol3;  // the third character of symbol
       int price;     // the stock price
  }stk;

  pipe(pipeID);

/* Read user arguments */
  if( argc != 2 )  // expect 2 string elements in argv[]
  {                //  else print correct usage
       printf("ERROR: Usage: <*.exe> <seconds to run program>\n");
       exit(0);
  }
  run_time = (time_t)atoi(argv[1]);
  current_time = time(0);
/* Calculate run time */
  end_time = current_time + run_time;

  srand(time(0)); //seeding

/* Create a process */
  chPID = fork();
  if(chPID < 0)  // failed to fork
  {
    printf("Failed to fork\n");
    exit(1);  // throw exception
  }
  else if(chPID == 0)  // Code executed by the child process
  {
    close(pipeID[1]); // child processes closes write
                      // side of pipe

       while(current_time < end_time )
       {
              read(pipeID[0], &stk.symbol1, sizeof(stk.symbol1));
              read(pipeID[0], &stk.symbol2, sizeof(stk.symbol2));
              read(pipeID[0], &stk.symbol3, sizeof(stk.symbol3));
```

```c
              read(pipeID[0], &stk.price, sizeof(stk.price));
              /* process the message in childBuf */
              printf("CONSUME: %c%c%c\t$%i\n",stk.symbol1,stk.symbol2,
                      stk.symbol3,stk.price);
              current_time = time(0);
       }

  }
  else  // Code executed by the parent process
  {
    close(pipeID[0]); // parent process closes read
                      // side of pipe

       while(current_time < end_time)
       {
              /* create a message to send to child process */
              stk.symbol1 = (char)((rand() % 26) + 65); // pick a number between 0 and 25
              stk.symbol2 = (char)((rand() % 26) + 65); // the ASCII code for "A" is 65,
              stk.symbol3 = (char)((rand() % 26) + 65); // so add 65 to the random number
                                                          // to get a lower case letter
                                                          // then convert it to a char type
              stk.price = rand() % 100 + 1; // pick a number between 1 to 100
              /* create a message to send to child process */
              write(pipeID[1], &stk.symbol1, sizeof(stk.symbol1));
              write(pipeID[1], &stk.symbol2, sizeof(stk.symbol2));
              write(pipeID[1], &stk.symbol3, sizeof(stk.symbol3));
              write(pipeID[1], &stk.price, sizeof(stk.price));
              printf("PRODUCE: %c%c%c\t$%i\n",stk.symbol1,stk.symbol2,
                      stk.symbol3,stk.price);
              current_time = time(0);

       }
 }

/* Code executed by both parent and child process */
  return 0;
}
```