Hacettepe University
Department of Computer Science and Engineering

BIL342 Programming Laboratory
Experiment IV

Subject:	Concurrent and Multithreaded Programming with Linux
Advisors:	Prof. Dr. Ali SAATÇİ
	R.A. Kerem ERZURUMLU
Submission Date:	26/04/2002
Deadline:	10/05/2002
Programming Language:	ANSI C

AIM

The aim of this experiment is to give students the ability to cope with the basic concepts of concurrent multithreaded programming on Linux. Within this context you are asked to write a C program with several threads which will manipulate a sample *account data file* concurrently.

PROBLEM DESCRIPTION

In this experiment, you will simulate operations on bank accounts. Your program should get the accounts data from ***accounts data file***. Atomic groups of operations (***transactions***) are to be applied on the accounts. Transactions will be read from ***transaction data file***. The sample account data file is shown in Figure-1.

```
Cost ID:Costumer Name:Account1:Account2
1:Kerem ERZURUMLU:1000:-250
2:Zerrin Ünal::200
```

Figure-1. Sample account data file

Each row of the account data file contains client information. Each client can have at most two accounts. The fields of a row are the first and second account's balances. These values can be less than zero.

In the transaction data file each row represent a transaction. A sample transaction data file is shown in Figure-2.

```
tr1 1:1 1:2 50
tr2 2:2 10
tr3 2:1 20
tr1 1:2 2:2 100
tr1 2:2 2:1 50
tr2 2:2 20
tr1 1:2 1:1 100
tr3 1:1 20
tr2 1:1 50
tr3 1:1 5
tr2 1:1 20
tr2 2:2 20
```

Figure-2. Sample transaction data file

In each row the first field represents the transaction type. According to this field, other fields gain different meanings. There are three types of transaction :

1. **tr1 :** Money transfer between two accounts. The line format for this type of transactions is as follows:

   `tr1 c1:a1 c2:a2 Q`

   where Q represents the quantity of money to be deducted from the account a1 of the client c1 and added to the account a2 of the client c2. In a formal representation this line can be defined as:

```
Transaction begin
     acc1 ← read (c1:a1)
     acc2 ← read (c2:a2)
     acc1 := acc1 – Q
     acc2 := acc2 + Q
     acc1 → write (c1:a1)
     acc2 → write (c2:a2)
Transaction end
```

   In Figure-2, the line "`tr1 1:1 1:2 50`" means: deduct 50 from the account 1 of the client 1 and and add it to the account 2 of the same client. At the end of this transaction, the balance of the first account will be normally 950 and the balance of the second -200.

2. **tr2 :** This transaction represents the deduction of a cost from an account. The line format for this type of transactions is as follows:

   `tr2 c1:a1 Q`

   where Q represents the quantity of money to be deducted from the account a1 of the client c1. In a formal representation this line can be defined as:

```
Transaction begin
     acc1 ← read (c1:a1)
     acc1 := acc1 – Q
     acc1 → write (c1:a1)
Transaction end
```

   In Figure 2, the second line "`tr2 2:2 10`" means the deduction of 10 from account 2 of the client 2. At the end of this transaction, the balance of account 2 will be 1190.

3. **tr3 :** This transaction represents the increase of an account by an interest rate. The line format for this type of transactions is as follows:

   `tr3 c1:a1 Q`

   where Q reperesent an interest rate which determines the amount of increase of the account a1 of the client c1. In a formal representation this line can be defined as:

```
Transaction begin
     acc1 ← read (c1:a1)
     acc1 := acc1 + acc1 * Q / 100
     acc1 → write (c1:a1)
Transaction end
```

In Figure 2, the third line "`tr3 2:1 20`" means the increase of 20 percent of the account 1 of the client 2. At the end of this transaction, the balance of the account will be 900.

Your program should include 4 threads. As there are 3 types of transactions there will be 3 different threads executing a particular type of transaction. These threads will get transactions from their own transaction queues and will execute them on the account matrix concurrently. The fourth thread will read the transactions file row by row and append each row to the appropiate transaction buffer (Figure-3).

To access the account matrix, your threads should, in the first place be programmed without any precaution (without inter-thread synchronization) while respecting the synchronization when accessing the transaction queues defined as pipes or fifo's.

Secondly, you are asked to program your threads in a fully synchronised manner using mutex or semaphore objects and to compare the results obtained in the two cases. (The use of mutex is the preferred way, because they are primarily designed for thread synchronization. The semaphores are generally used for process synchronization.)

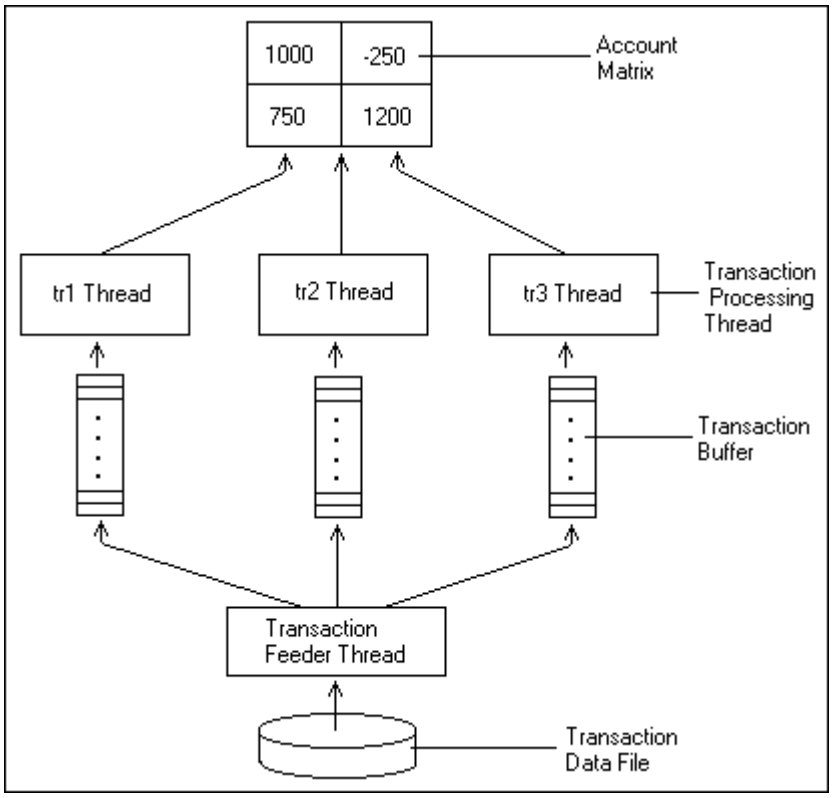

Figure-3. The execution schema of threads

USER INTERFACE

Your program will get two parameters from **command line**. The parameters will be the filenames of the account data file and the transaction data file respectively.

Every time a thread completes a transaction it should display a message on the screen. This message should include all the necessary information about the identity of the thread and the completed transaction.

When all the transactions within the transaction data file are completed, your program should display the final account matrix and you should check the result.

BIL342 Experiment IV

NOTES

1. Soft copy of this paper and other usefull documents is in ftp://ftp.cs.hacettepe.edu.tr/pub/dersler/bil342/2002/4 . Also there are some examples for thread and mutex programming.
2. You are asked to give your `Makefile` with your program.
3. Describe your communication flow-chart with detail.
4. You are asked to follow announcements made to "bil342 discussion list". If you are not subscribed yet, please subscribe to it by sending an e-mail to:

   `majordomo@cs.hacettepe.edu.tr`

   with a message body of

   "`subscribe bil342`".
5. Your report and program must be submitted at the same time.
6. Your report must include your source codes.
7. Reports written with MS-Word will not be accepted. Try HTML or plain text file.
8. Your works and reports will be posted with a floppy disc.
9. Office hours will be held on Friday morning's. You can also send e-mails to `kerem@linux.org.tr` for your additional questions.

Good Luck