Лабораторная работа №2
«Распределенная банковская система»

Введение

Банковская система является классическим примером распределенной системы. Счета клиента могут находиться в разных филиалах банка. При этом обслуживание счетов филиалы осуществляют самостоятельно независимо друг от друга. Однако некоторые операции требуют взаимодействия филиалов между собой, например, при переводе денег между счетами, находящимися в разных филиалах, или подсчете полной суммы денег.

Рассмотрим задачу подсчета полной суммы денег, находящихся на счетах в разных филиалах банка. Предположим, что банковская система не позволяет вносить дополнительные денежные средства на счет филиала и снимать их наличными, а лишь осуществляет перевод различных сумм из одного филиала банка в другой с помощью сообщений. Чтобы определить полную сумму банк должен иметь сведения о количестве денег в каждом филиале. При подсчете возникают следующие проблемы:

- в момент подсчета в системе могут находиться незавершенные операции перевода: деньги уже сняты с одного счета, но еще не зачислены на другой;
- часы в разных филиалах идут с некоторым относительным сдвигом, т.е. имеет место небольшая рассинхронизация часов.

В данной лабораторной работе необходимо реализовать простую банковскую систему, использующую физическое время для подсчета полной суммы денег, находящихся на счетах в разных филиалах банка. В последующих лабораторных работах будет необходимо модифицировать работу банковской системы для использования логических часов.

Исходные данные

- Число N процессов, составляющих распределенную банковскую систему;
- Начальные балансы S для каждого из счетов (один процесс — один счет),

где N и S — целые числа. При автоматической проверке $N \in [2; 10]$, $S \in [1; 99]$.

Постановка задачи

За основу банковской системы необходимо взять модель распределенной системы, описанную в условии к первой лабораторной работе. Разделяют два типа процессов, составляющих банковскую систему:

- процессы, принимающие запросы от клиентов (тип «К»);
- процессы, отвечающие за обслуживание счетов (тип «С»).

В данной лабораторной работе используется один процесс типа «К» и $N - 1$ процессов типа «С». Каждый процесс имеет собственные физические часы. Процессы обмениваются сообщениями в асинхронном режиме.

Процесс «К» на основе запросов клиента формирует запросы к соответствующим процессам «С». Поддерживаются следующие операции:

- перевод денег между счетами;
- получение информации об истории изменения баланса.

При переводе денег процесс «К» отправляет сообщение процессу «C_{src}», с которого осуществляется списание. Процесс «C_{src}» после выполнения требуемых операций пересылает исходное сообщение процессу «C_{dst}». Процесс «C_{dst}» после выполнения всех необходимых действий отправляет процессу «К» подтверждение о выполнении операции.

Предполагается, что используются корректные запросы на перевод и каналы надежные, поэтому другие подтверждения о выполнении операций не требуются.

Задание

Используя топологию распределенной системы и библиотеку межпроцессного взаимодействия (IPC) из первой лабораторной работы, необходимо реализовать банковскую систему, описанную выше.

Во время выполнения программы осуществляются переводы денег между счетами. При завершении на экран выводится таблица с информацией о балансе каждого счета и полной сумме денег, находящихся на всех счетах, в каждый момент времени $t \in 0, 1, \ldots, T$. Последняя отметка времени T определяется на момент завершения каждого из процессов «С» (при получении соответствующего сообщения от процесса «К», см. далее), $T \leq MAX\_T$.

При запуске программы, как и раньше, указывается число дочерних процессов в полносвязной топологии. Последние X параметров задают начальные балансы для каждого из счетов. Например, следующая команда:

./pa2 -p 3 10 20 30

означает, что в банковской системе три счета (точнее три филиала, каждый из которых обслуживает всего один счет) с идентификаторами 1, 2, 3 и с начальными балансами $10, $20, $30 соответственно.

Для отслеживания времени все процессы используют физические часы, реализованные функцией *get_physical_time()*, входящей в библиотеку *libruntime.so*, предоставляемую преподавателем. Для формирования временных отметок *t* процессы должны использовать исключительно функцию *get_physical_time()*.

Процесс «К» реализуется на основе родительского процесса, процессы «С» — на основе дочерних процессов из лабораторной работы №1. После того, как получены сообщения *STARTED* от всех процессов «С», процесс «К» должен вызвать функцию *bank_robbery()*, которая выполняет ряд переводов денег между произвольными процессами «С» посредством вызовов функции *transfer()*. Перевод описывает структура *TransferOrder*, передаваемая сообщением типа *TRANSFER*. При выполнении перевода процесс «К» отправляет сообщение *TRANSFER* процессу «C_{src}», после чего переходит в режим ожидания подтверждения (пустое сообщение типа *ACK*) процессом «C_{dst}» получения перевода. Переводы могут быть инициированы только процессом «К».

После выполнения функции *bank_robbery()* процесс «К» отправляет сообщение *STOP* всем процессам «С» и дожидается получения сообщения *DONE* от всех дочерних процессов. После этого процесс «К» должен получить от каждого процесса «С» сообщение *BALANCE_HISTORY*, содержащее структуру *BalanceHistory*. Структуры *BalanceHistory* от всех процессов «С» агрегируются в структуру *AllHistory*, которая должна быть использована в качестве аргумента функции *print_history()* перед завершением родительского процесса. Функция *print_history()* реализуется библиотекой, поставляемой вместе с заданием, пример возможной реализации можно посмотреть здесь.

«Полезной» работой процесса «С» является ожидание и обработка сообщений двух типов: *TRANSFER* и *STOP*. При получении сообщения *TRANSFER* процесс «C_{src}» после выполнения всех необходимых операций пересылает это сообщение процессу «C_{dst}». Процесс «C_{dst}» обрабатывает сообщение и отправляет сообщение *ACK* процессу «К». При этом каждый процесс «С» должен хранить в структуре типа *BalanceHistory* информацию о состоянии своего баланса *BalanceState* в каждый момент времени *t*. Обратите внимание, что если в момент времени *t*=1 баланс процесса равен $10, а в t=3 ему пришел перевод в $5, то в t=2 следует указать баланс $10. Семантика структур *BalanceHistory* и *BalanceState* подробно описана в заголовочном файле *banking.h*. Значение поля *s_balance_pending_in* структуры *BalanceState* в данной работе следует всегда устанавливать равным 0.

При получении сообщения *STOP* процесс «C» переходит к выполнению третьей фазы. Во время выполнения третьей фазы процесс «C» может получать сообщения «*TRANSFER*» от других процессов «C», при этом гарантируется, что после отправки сообщения *STOP* процесс «K» не выполняет новых переводов. После получения сообщений *DONE* от всех остальных дочерних процессов и перед завершением процесс «C» отправляет процессу «K» сообщение *BALANCE_HISTORY*, содержащее структуру *BalanceHistory*.

В дополнение к событиям из предыдущей лабораторной работы (строки форматирования изменены) для процессов типа «C» добавлено два новых события:

- процесс отправил перевод на другой счет (*log_transfer_out_fmt*);
- процесс получил перевод с другого счета (*log_transfer_in_fmt*).

Для реализации асинхронного обмена сообщениями необходимо использовать неблокирующие функции *read()* и *write()*. Как и в предыдущей работе, в реализации запрещается использовать многопоточность: один процесс — один поток. Кроме того, нельзя использовать разделяемую память, примитивы синхронизации (семафоры и т.п.), функции *select()* и *poll()*. Логирование (*events.log*, *pipes.log* и терминал), как и в предыдущем задании.

Требования к реализации и среда выполнения

Реализацию необходимо выполнить на языке программирования Си с использованием предоставленных заголовочных файлов и библиотеки из архива *pa2345_starter_code.tar.gz*. Архив содержит следующие файлы:

ipc.h	объявления структур данных и функций для организации межпроцессного взаимодействия. Объявленные функции необходимо реализовать;
banking.h	объявления структур данных, констант и функций, связанных с банковскими операциями. Часть функций реализовано преподавателем;
pa2345.h	форматы строк для логирования;
bank_robbery.c	набор вызовов *transfer()* для тестирования реализации;
libruntime.so	библиотека, реализующая вспомогательные функции, в частности *get_physical_time()* и *print_history()*. Инструкции по использованию см. ниже;

Заголовочные файлы содержат большое число важных комментариев, пояснений и рекомендаций. Запрещается их модификация: при автоматической проверке они заменяются на оригинальные.

Файл *bank_robbery.c* содержит функцию *bank_robbery()*. Версия *bank_robbery.c* из архива отличается от той, что используется при проверке задания.

Для использования *libruntime.so* необходимо определить следующие переменные программного окружения:

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/path/to/pa2/dir";
### пустая строка
LD_PRELOAD=/full/path/to/libruntime.so ./pa2 -p 2 10 20
```

Использование неблокирующих *read()* и *write()* может потребовать дополнительных изменений в реализации IPC. Новая версия библиотеки IPC должна быть совместима с версией для предыдущей лабораторной работы.

Работа присылается в виде архива с именем *pa2.tar.gz*, содержащим каталог *pa2*. Все файлы с исходным кодом и заголовки должны находиться в корне этого каталога. Среда выполнения — Linux (Ubuntu 14.04, clang-3.5). При автоматической проверке используется следующая команда: *clang -std=c99 -Wall -pedantic *.c –L. -lruntime*. При наличии варнингов работа не принимается. При успешном выполнении запущенные процессы не должны использовать *stderr*, код завершения программы должен быть равен 0.