**Крупников Андрей**

**Группа 6133**

**Лабораторная работа № 4**

**Дисциплина «Большие данные»**

**Цель работы**

* запустить ZooKeeper,
* изучить директорию с установкой ZooKeeper,
* запустить интерактивную сессию ZooKeeper CLI и освоить её команды,
* научиться проводить мониторинг ZooKeeper,
* разработать приложение с барьерной синхронизацией, основанной на ZooKeeper,
* запустить и проверить работу приложения.

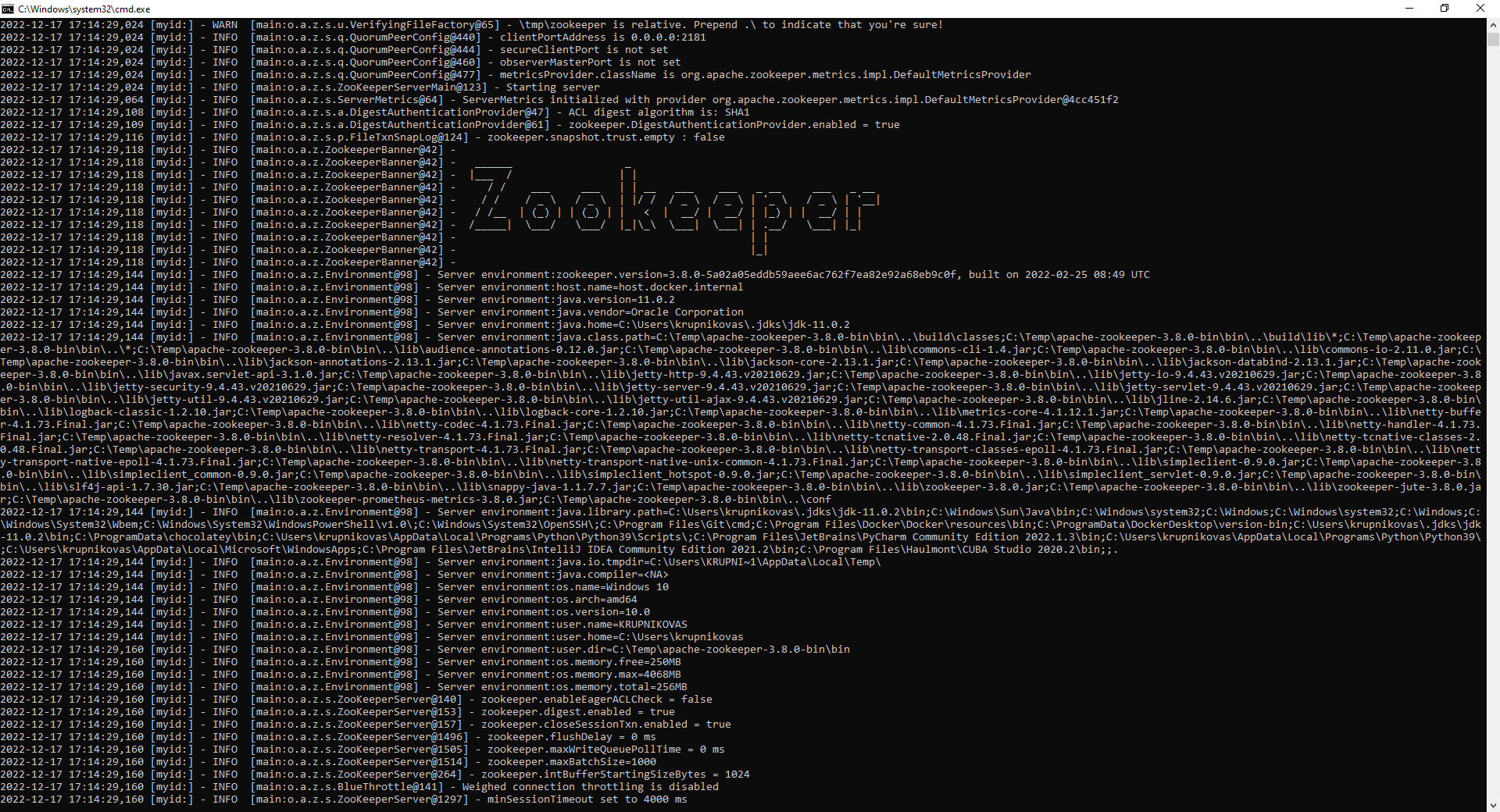
**Изучение ZooKeeper**

Запуск

Windows

В Windows запустим сервер двойным кликом по скрипту zkServer.cmd в папке ./bin/ или из терминала, набрав:

zkServer.cmd



**Изучение директории установки ZooKeeper**

В директории **ZooKeeper** находятся следующие папки:

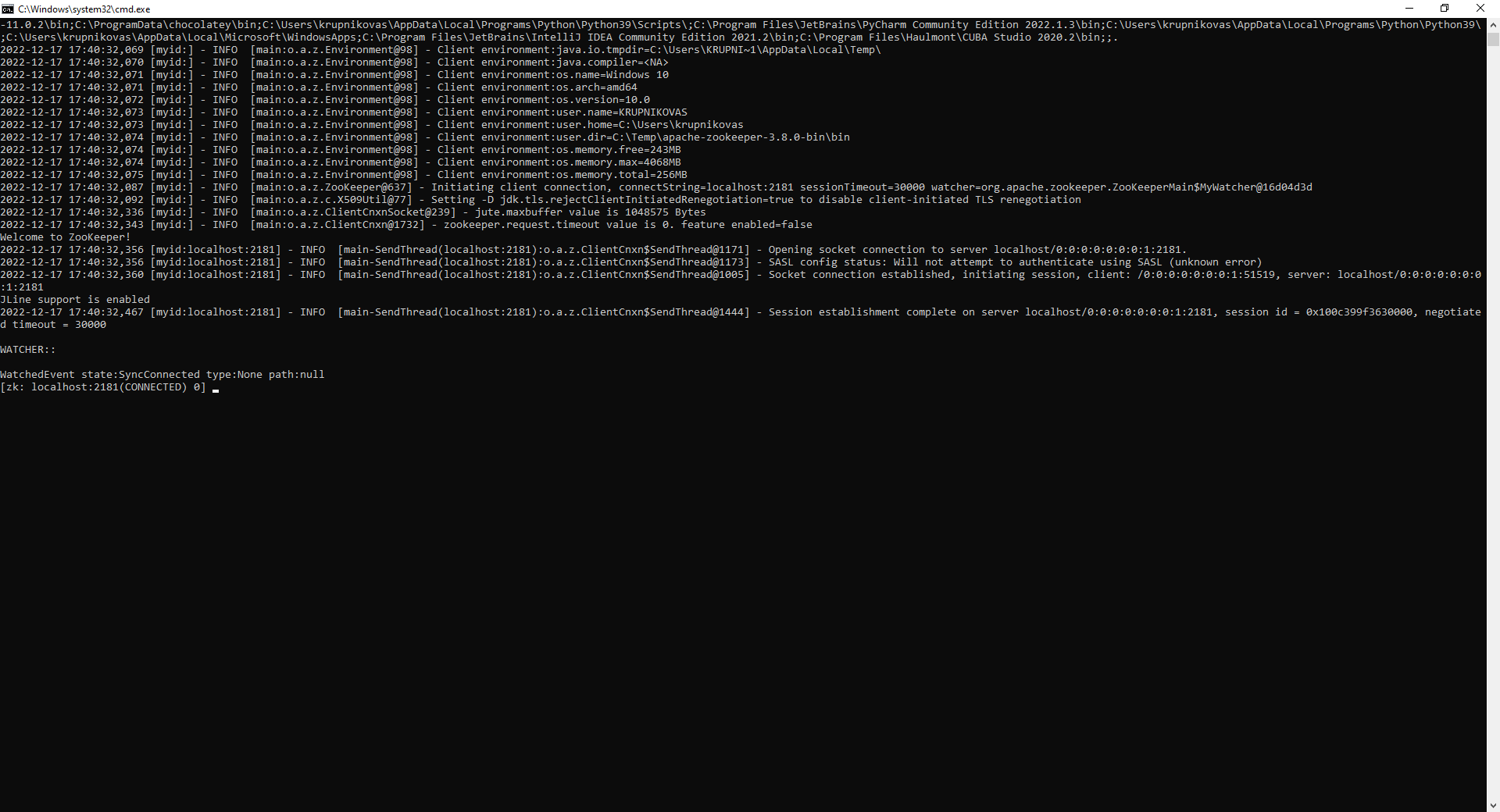
* bin с исполняемыми файлами для запуска, остановки и взаимодействия с ZooKeeper,
* conf с конфигурационными файлами,
* contrib с инструментами для интеграции ZooKeeper в другие системы: rest, fuse, perl и python библиотеки,
* dist-maven артефакты Maven,
* docs в которой хранится документация,
* recipes различные рецепты, помогающие решать задачи с использованием ZooKeeper (выбор лидера, блокировки, очереди),
* src с исходным кодом и тестовыми скриптами.

**Взаимодействие с ZooKeeper через командный интерфейс CLI**

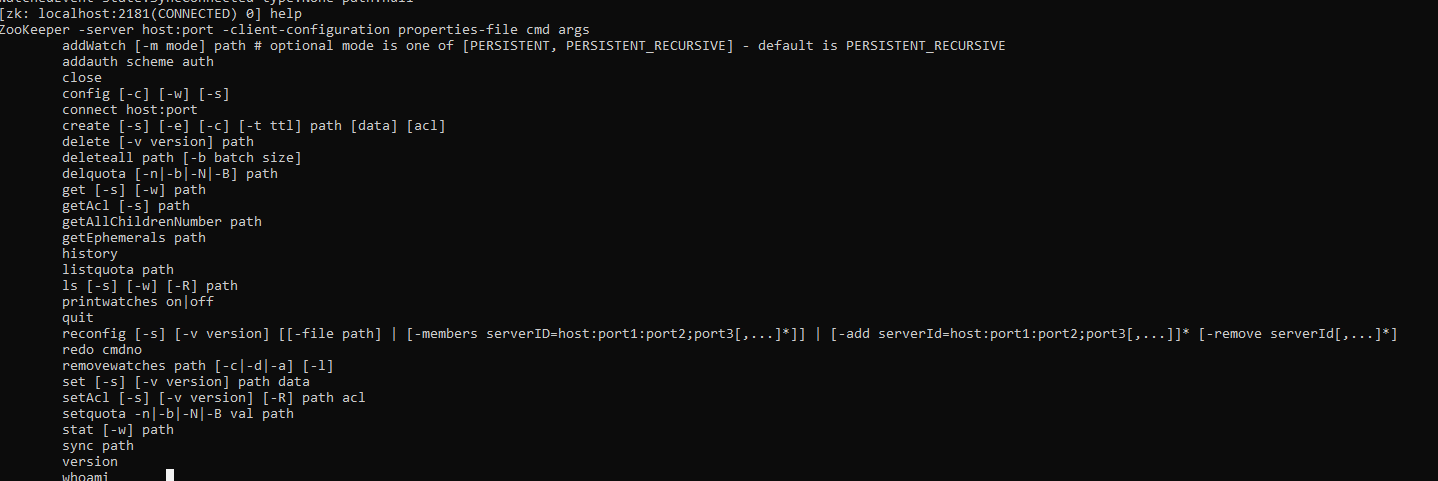
Одним из способов взаимодействия с ZooKeeper является консольный интерфейс ZooKeeper CLI.

Перейдем в папку bin директории установки ZooKeeper.

Для запуска интерактивной сессии ZooKeeper CLI используем скрипт.



Подключение установлено. Для вывода всех возможных команд наберем help.

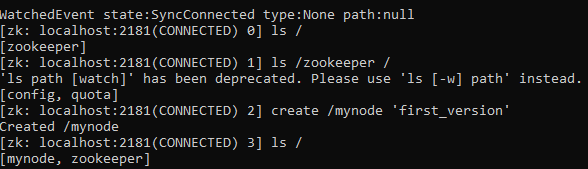


Выйти из консоли можно с помощью команды quit или отправив EOF символ сочетанием Ctrl+D.

Теперь в корне создадим свой узел /mynode с данными "first\_version" следующей командой:

create /mynode 'first\_version'

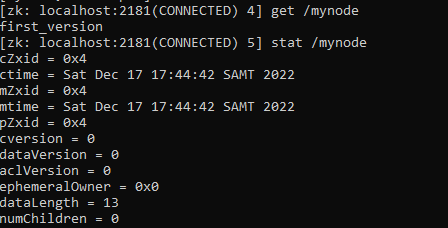
Проверем, что в корне появился новый узел, введя команду ls /.



Следующие команды возвращают данные и метаданные узла:

get /mynode

stat /mynode

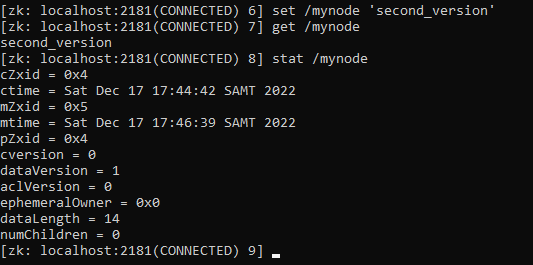


Изучим структуру, хранимую в узле

* 'first\_version' Хранимые данные
* cZxid Номер транзакции создания узла в системе
* ctime Время создания узла
* mZxid Номер транзакции модификации узла
* mtime Время модификации узла
* pZxid Номер транзакции модификации дочерних узлов
* cversion Количество изменений дочерних узлов
* dataVersion Количество изменений данных узла
* aclVersion Количество изменений прав доступа к данному узлу
* ephemeralOwner Идентификатор сессии владельца узла, если узел эфимерный. Иначе значение равно нулю.
* dataLength Размер данных
* numChildren Количество дочерних узлов

Изменим данные узла на "second\_version":

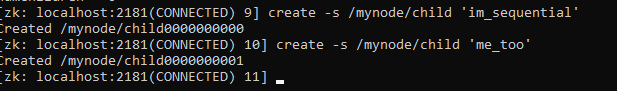
set /mynode 'second\_version'



В качестве результата мы получим обновлённые метаданные узла.

Изменилась дата модификации mtime и значение поля dataVersion стало больше на единицу, так как мы провели одно изменение. Также изменился размер данных.

Теперь создадим два нумерованных (sequential) узла в качестве дочерних mynode:



Передав дополнительно флаг -s, мы указали, что создаваемый узел нумерованный. Этот способ позволяет создавать узлы с уникальными именами, по которым можно узнать порядок поступления запросов на сервер.

Внутри CLI сессии, создадим узел mygroup с помощью команды:

create /mygroup 'top\_node'



Откроем две новых CLI консоли и в каждой создадим по дочернему узлу в mygroup:

Консоль 1 - grue.

create -e /mygroup/grue 'iam\_grue'



Консоль 2 - bleen.

create -e /mygroup/bleen 'iam\_bleen'



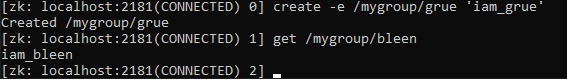
Эфимерный тип узла задаётся ключом -e.

Проверим в исходной консоли, что grue и bleen являются членами группы mygroup.



Представим теперь, что одному из клиентов нужна информация о другом клиенте (в качестве клиентов могут выступать узлы кластера). Этот сценарий эмулируется получением информации командой get, которую мы уже запускали ранее. Выберем консоль grue и обратимся к информации узла bleen.

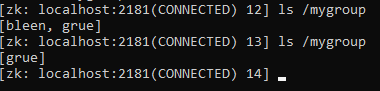
get /mygroup/bleen



Информацией, которая хранится в узле клиента может быть url адрес клиента, либо любая другая информация требуемая для работы распределённого приложения.

Теперь эмулируем аварийное отключение любого клиента. Нажмем сочетание клавиш Ctrl+D в одной из консолей, создавшей эфимерный узел.

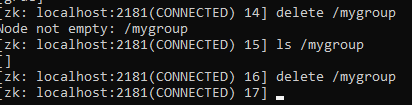
Проверим, что соответствующий узел пропал из mygroup. Изменение списка дочерних узлов может произойти не сразу — от 2 до 20 tickTime, значение которого можно посмотреть в zoo.cfg.



Таким образом клиенты могут получать информацию о появлении и отключении других клиентов.

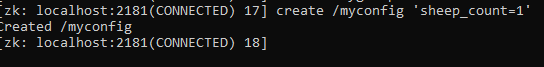
В заключении удалим узел /mygroup командой:

delete /mygroup



**Пример управления конфигурацией распределённого приложения**

Создадим в корне узел "myconfig" в задачу которого будет входить хранение конфигурации. В нашем случае узел будет хранить строку 'sheep\_count=1'.

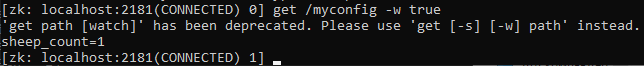


Во всех случаях, когда конфигурационная информация распределённого приложения будет изменяться, будем обновлять znode строкой с новым значением. Другим клиентам распределённого приложения достаточно проверять хранимые в этом узле данные.

Откроем новую консоль и подключимся к ZooKeeper. Данная консоль будет играть роль физического сервера, который ожидает получить оповещение в случае изменения конфигурационной информации, записанной в /myconfig znode.

Следующая команда устанавливает watch-триггер на узел:

get /myconfig -w true

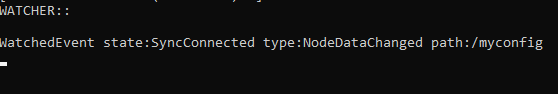


Вернемся к первому терминалу и измените значение myconfig:

set /myconfig 'sheep\_count=2'



Во втором терминале должно появиться оповещение об изменении данных!



Триггер сбрасывается после одного срабатывания, а значит его придётся 'взводить' каждый раз заново. Как правило, в приложении, в логике обработчика события присутствует такая процедура.

Удалим узел /myconfig. Проверим, что эта команда выполнилась.



**Задача о философах**

Philosopher 0 is going to eat

Philosopher 4 is going to eat

Philosopher 3 is going to eat

Philosopher 2 is going to eat

Philosopher 1 is going to eat

Philosopher 5 picked up the left fork

Philosopher 3 picked up the left fork

Philosopher 5 picked up the right fork

Philosopher 3 picked up the right fork

Philosopher 5 put the right fork

Philosopher 1 picked up the left fork

Philosopher 5 put the loft fork and finished eating

Philosopher 1 picked up the right fork

Philosopher 5 is thinking

Philosopher 1 put the right fork

Philosopher 2 picked up the left fork

Philosopher 1 put the loft fork and finished eating

Philosopher 1 is thinking

Philosopher 4 is eating

Philosopher 5 picked up the left fork

Philosopher 5 picked up the right fork

Philosopher 3 put the right fork

Philosopher 4 picked up the left fork

Philosopher 3 put the loft fork and finished eating

Philosopher 2 picked up the right fork

Philosopher 3 is thinking

Philosopher 2 is eating

Philosopher 0 is eating

Philosopher 3 picked up the left fork

Philosopher 2 put the right fork

Philosopher 2 put the loft fork and finished eating

Philosopher 2 is thinking

Philosopher 5 put the right fork

Philosopher 1 picked up the left fork

Philosopher 5 put the loft fork and finished eating

Philosopher 1 picked up the right fork

Philosopher 4 picked up the right fork

Philosopher 5 is thinking

Philosopher 4 put the right fork

Philosopher 4 put the loft fork and finished eating

Philosopher 3 picked up the right fork

Philosopher 4 is thinking

Philosopher 1 put the right fork

Philosopher 1 put the loft fork and finished eating

Philosopher 1 is thinking

Philosopher 3 put the right fork

Philosopher 3 put the loft fork and finished eating

Philosopher 3 is thinking

Philosopher 1 is eating

Philosopher 2 picked up the left fork

Philosopher 2 picked up the right fork

Philosopher 3 is eating

Philosopher 4 picked up the left fork

Philosopher 4 picked up the right fork

Philosopher 4 put the right fork

Philosopher 4 put the loft fork and finished eating

Philosopher 4 is thinking

Philosopher 2 put the right fork

Philosopher 2 put the loft fork and finished eating

Philosopher 2 is thinking

Process finished with exit code 0