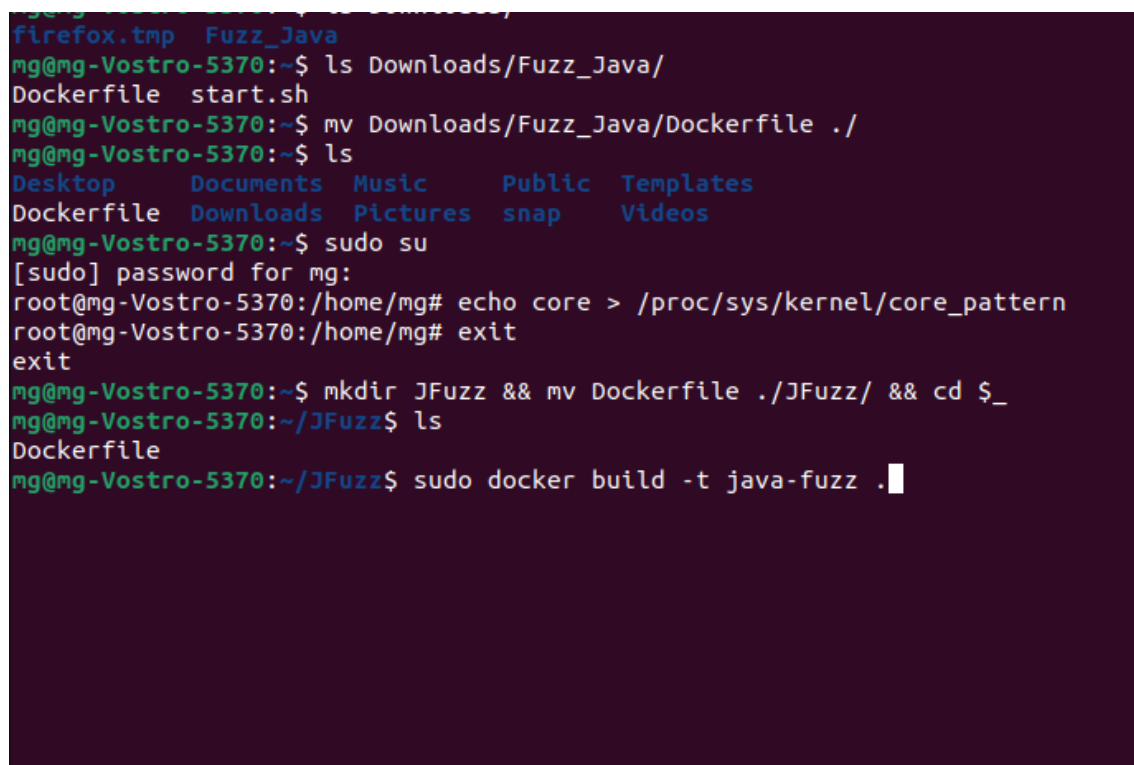


Для выполнения тестового задания использовалась вычислительная машина с ОС Ubuntu 22.04 Desktop LTS, HDD 256 ГБ, объем оперативной памяти 8 ГБ и число ядер процессора 4.

Для развертывания виртуальной среды был установлен пакет *docker.io*, ссылка на который содержится в пакетном менеджере *apt*. Для доступа к списку образов *images* имя пользователя *mg* было добавлено в файл */etc/group* и выполнена перезагрузка системы.

```
$ sudo apt install docker.io
$ sudo su
# vim /etc/group
# exit
$ sudo shutdown -r now
```

Был скачан и распакован архив с Dockerfile, затем вызвана команда сборки образа (рисунок 1).



```
firefox.tmp  Fuzz_Java
mg@mg-Vostro-5370:~$ ls Downloads/Fuzz_Java/
Dockerfile  start.sh
mg@mg-Vostro-5370:~$ mv Downloads/Fuzz_Java/Dockerfile ./
mg@mg-Vostro-5370:~$ ls
Desktop    Documents  Music      Public     Templates
Dockerfile Downloads  Pictures   snap       Videos
mg@mg-Vostro-5370:~$ sudo su
[sudo] password for mg:
root@mg-Vostro-5370:/home/mg# echo core > /proc/sys/kernel/core_pattern
root@mg-Vostro-5370:/home/mg# exit
exit
mg@mg-Vostro-5370:~$ mkdir JFuzz && mv Dockerfile ./JFuzz/ && cd $_
mg@mg-Vostro-5370:~/JFuzz$ ls
Dockerfile
mg@mg-Vostro-5370:~/JFuzz$ sudo docker build -t java-fuzz .
```

Рисунок 1 – сборка образа из Dockerfile

После сборки образа была запущена команда запуска контейнера в виртуальной консоли (рисунок 2).

```

mg@mg-Vostro-5370:~/JFuzz$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
<none>        <none>    edaedc74e745   14 seconds ago 2.74GB
ubuntu        20.04     3bc6e9f30f51   10 days ago   72.8MB
mg@mg-Vostro-5370:~/JFuzz$ sudo docker run -it <none>
bash: syntax error near unexpected token `newline'
mg@mg-Vostro-5370:~/JFuzz$ sudo docker run -it edaedc74e745
[sudo] password for mg:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

user@4eed18d0294c:~$
user@4eed18d0294c:~$
user@4eed18d0294c:~$ ls
afl  jacoco  jqf  PdfParse.java  pdf-sample.pdf  tika-1.28.1
user@4eed18d0294c:~$

```

Рисунок 2 – запуск контейнера

Домашний каталог пользователя *user* содержит *PdfParse.java* файл, который компилируется в процессе работы скрипта *./start.sh*.

```

user@4529bfb67dc0:~$ ./start.sh
Это среда для выполнения тестового задания по фаззингу проекта на Java.

На машине установлено и настроено
-----
1) Среда разработки и выполнения Open JDK, инструмент сборки Maven
2) Локальный репозиторий Java (/home/user/.m2/repository/) со всеми необходимыми
   компонентами для сборки и работы утилит
3) Фаззеры AFL (/home/user/afl) и JQF (/home/user/jqf) - команды их вызова досту
   пны без указания пути
4) Средство сбора покрытия кода JaCoCo установлено в /home/user/jacoco
5) Библиотека Apache Tika собрана в каталоге /home/user/tika-1.28.1 и основные е
   е компоненты установлены в локальный репозиторий
6) Пример файла формата PDF (/home/user/pdf-sample.pdf)
7) Пример программы для чтения файла PDF с помощью Apache Tika (/home/user/PdfPa
   rse.java)
8) Вспомогательные утилиты (Midnight Commander, Vim, wget, git и др.)

Далее?

```

Рисунок 3 – запуск скрипта start.sh

В задании выполнялось фаззинг-тестирование метода *parse* класса *PDFParser* библиотеки Apache Tika и оценка покрытия кода. Сначала была изучена сигнатура метода *parse* и написан класс *PDFParserTest.java* с методом-оберткой *testParse* и аннотацией JQF (Рисунок 4).

```

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.pdf.PDFParser;
import org.apache.tika.sax.BodyContentHandler;
import org.xml.sax.SAXException;

import edu.berkeley.cs.jqf.fuzz.Fuzz;
import edu.berkeley.cs.jqf.fuzz.JQF;
import org.junit.Assume;
import org.junit.runner.RunWith;

@RunWith(JQF.class)
public class PDFParserTest{
    @Fuzz
    public void testParser(InputStream input) throws SAXException, TikaException{
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        ParseContext pcontext = new ParseContext();
        PDFParser pdfparser = new PDFParser();

        try{
            pdfparser.parse(input, handler, metadata, pcontext);
        } catch (IOException e){
            Assume.assumeNoException(e);
        }
    }
}

```

Рисунок 4 – составленный класс PDFParserTest

Выполнена сборка со всеми необходимыми зависимостями и получен файл PDFParserTest.class (рисунок 5).

```

user@8edb710da8ed:~$
user@8edb710da8ed:~$ ls
afl jacoco jqf PdfParse.class PdfParse.java pdf-sample.pdf tika-1.28.1
user@8edb710da8ed:~$ vim PdfParse.java
user@8edb710da8ed:~$ vim PDFParserTest.java
user@8edb710da8ed:~$
user@8edb710da8ed:~$
user@8edb710da8ed:~$
user@8edb710da8ed:~$
user@8edb710da8ed:~$ javac -cp ".:$(jqf/scripts/classpath.sh):/home/user/.m2/repository/org/apache/tika/tika-core/1.28.1/tika-core-1.28.1-test-jar-with-dependencies.jar:/home/user/.m2/repository/org/apache/tika/tika-parsers/1.28.1/tika-parsers-1.28.1.jar" PDFParserTest.java
user@8edb710da8ed:~$ ls -l
итого 40
drwxr-xr-x 12 user user 4096 авг 12 20:53 afl
drwxr-xr-x  6 user user 4096 авг 12 21:10 jacoco
drwxr-xr-x 11 user user 4096 авг 12 20:53 jqf
-rw-r--r--  1 user user 1944 авг 13 00:29 PdfParse.class
-rw-r--r--  1 user user 1120 авг 12 21:11 PdfParse.java
-rw-r--r--  1 user user 1147 авг 13 00:38 PDFParserTest.class
-rw-r--r--  1 user user  952 авг 13 00:37 PDFParserTest.java
-rw-r--r--  1 user user 7945 дек 29 2014 pdf-sample.pdf
drwxr-xr-x 21 user user 4096 дек 31 1979 tika-1.28.1
user@8edb710da8ed:~$

```

Рисунок 5 – компиляция PDFParserTest.java

Для проведения тестирования был подготовлен начальный корпус входных файлов .pdf, в который вошли файлы *pdf-sample.pdf*, *afl/testcases/others/pdf/small.pdf*, а также небольшая коллекция из репозитория [2]. Все файлы были сгруппированы по каталогам *~/in1*, *~/in2*, *~/in3* и *~/in4* для каждого запущенного фаззинг-процесса. В конечном итоге входные данные собраны в единый каталог *testcases* и размещены в одноименной директории удаленного репозитория [1] на github.

С помощью команды *jqf-afl-fuzz* были запущены 4 параллельных фаззинг-процесса. Выходные данные для каждого процесса записывались в каталоги *out1/*, *out2/*, *out3/*, *out4/*, которые были собраны в директории *fuzzingres* [1]. Для фаззинга использовался словарь из коллекции словарей *afl/dictionaries/pdf.dict*.

Команда запуска процесса с id1 представлена на рисунке 6.

```
user@dd66c4524c07:~$ jqf-afl-fuzz -i ~/in_1/ -m none -x afl/dictionaries/pdf.dict -o ~/out_1 -S id1 -c ".: /home/user/.m2/repository/org/apache/tika/tika-core/1.28.1/tika-core-1.28.1-test-jar-with-dependencies.jar:/home/user/.m2/repository/org/apache/tika/tika-parsers/1.28.1/tika-parsers-1.28.1.jar:/home/user/.m2/repository/org/apache/pdfbox/pdfbox/2.0.25/pdfbox-2.0.25.jar:/home/user/.m2/repository/commons-logging/commons-logging/1.2/commons-logging-1.2.jar:/home/user/.m2/repository/org/apache/pdfbox/jempbox/1.8.16/jempbox-1.8.16.jar:/home/user/.m2/repository/org/apache/pdfbox/fontbox/2.0.25/fontbox-2.0.25.jar" PDFParserTest testParser
```

Рисунок 6 – команда запуска процесса с id1

Общее время фаззинга $T_f \sim 46$ мин. Окна состояний всех 4-х процессов на момент завершения представлены на рисунке 7. Аварийные входные данные собраны в каталоге *fuzzingres* [1].

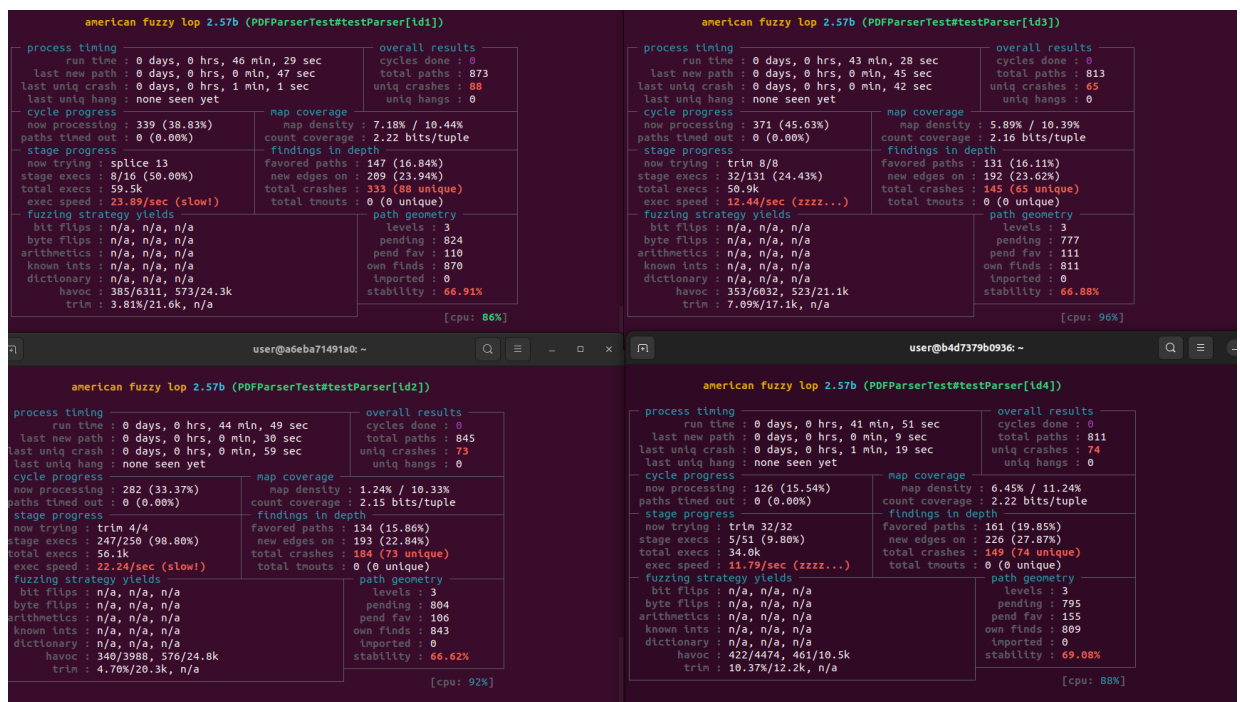


Рисунок 7 – окна состояний процессов на момент завершения фаззинга

Для сбора информации о покрытии использовались выходные данные, содержащие элементы очереди (каждому элементу соответствует уникальная трасса), которые выполнялись с помощью PdfParse. Перед этим было выполнено встраивание агента (рисунок 8).

```
user@46b2e1eb08b8:~$ java -javaagent:jacoco/lib/jacocoagent.jar=destfile=outfile,includes=PdfParse -cp ".: /home/user/.m2/repository/org/apache/tika/tika-core/1.28.1/tika-core-1.28.1-test-jar-with-dependencies.jar:/home/user/.m2/repository/org/apache/tika/tika-parsers/1.28.1/tika-parsers-1.28.1.jar:/home/user/.m2/repository/org/apache/pdfbox/pdfbox/2.0.25/pdfbox-2.0.25.jar:/home/user/.m2/repository/commons-logging/commons-logging/1.2/commons-logging-1.2.jar:/home/user/.m2/repository/org/apache/pdfbox/jempbox/1.8.16/jempbox-1.8.16.jar:/home/user/.m2/repository/org/apache/pdfbox/fontbox/2.0.25/fontbox-2.0.25.jar" PdfParse 2>/dev/null
```

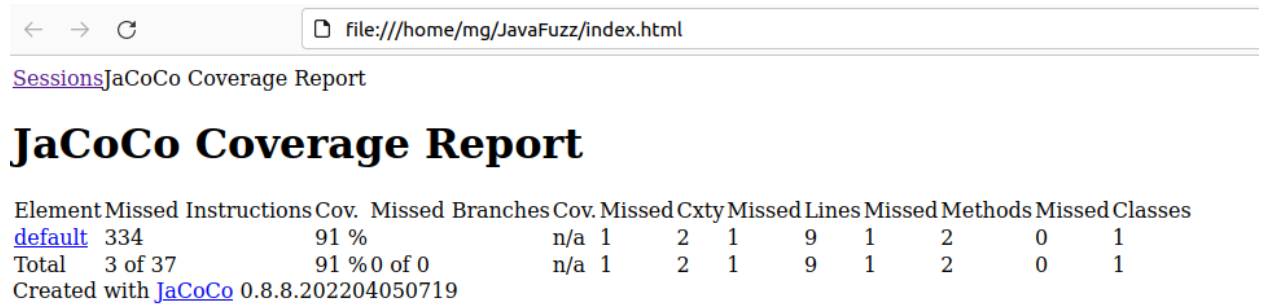
Рисунок 8 – встраивание агента с указанием выходного файла с jacoco.exec

Для формирования отчета в формате html использовалось jacococli.jar (рисунок 9).

```
user@46b2e1eb08b8:~$ java -jar jacoco/lib/jacococli.jar report outfile --classfiles PdfParse.class --html
```

Рисунок 9 – получение html отчета

В результате в текущей директории сформирован файл index.html, содержимое которого представлено на рисунке 10. Файл помещен в каталог html_report [1].



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
default	334	91 %	n/a	1	2	1	9	1
Total	3 of 37	91 % 0 of 0	n/a	1	2	1	9	1

Created with [JaCoCo](#) 0.8.8.202204050719

Рисунок 10 – отчет в html формате

Ссылки:

1. <https://github.com/MaxG175/JavaFuzz>
2. <https://github.com/willp-bl/lisbon-pdf-set>