Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Факультет Безопасности (ФБ)

Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

ПРОЦЕССЫ

Отчёт по практической работе № 4 по дисциплине  
«Системное программирование»

Студент группы 726

\_\_\_\_\_\_\_\_\_\_ Е.А. Антончик

\_\_\_\_\_\_\_\_\_\_

Принял

к.т.н., доцент кафедры КИБЭВС

\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ Я.Е. Мещеряков

оценка

\_\_\_\_\_\_\_\_\_\_

дата

1 Введение

Цель работы: Познакомиться с основными функциями POSIX API для работы с процессами, особенностями процессов в операционных системах Unix.

Ход работы:

1. Ознакомление с теоретическим материалом по работе с процессами

2. Разработка и написание программ на языках Python3 и С++

3. Отладка программ

4. Оформление отчёта

5. Сдача лабораторной работы

Задание:

1. Изучить теоретические сведения, самостоятельно изучить способы

работы с процессами в POSIX.

1. Разработать и написать программы на языке Python3 и С++, которые используют несколько процессов для решения задачи.

3. На moodle загрузить архив, содержащий: Отчет, все файлы проекта.

4. Загрузить все необходимые файлы в репозиторий на github.

4. Написать отчет и защитить у преподавателя.

2 Ход работы

В ходе работы были написаны 2 программы на языке Python3 и 1 программа на языке С++ в соответствии с вариантом 8, представленным ниже.

Задание для написания скрипта:

Написать две программы. Первая – вычисляет контрольную сумму файла. Вторая – вычисляет контрольную сумму всех файлов в директории, при этом обработка каждого отдельного файла осуществляется с помощью первой программы в отдельном процессе.

Результат работы программы, реализующей алгоритм высчитывания контрольной суммы файла представлен на рисунке 1.

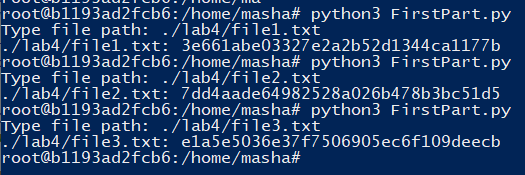


Рисунок 1 – Результат работы первой программы

Результат работы второй программы на языке программирования Python3 программы, реализующей вычисление контрольной суммы всех файлов в директории. Обработка каждого отдельного файла вычисляется с помощью первой программы в отдельном процессе.

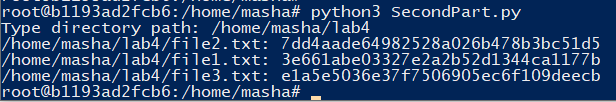


Рисунок 2 – Результат работы второй программы на Python3

Как можно заметить на рисунках 1 и 2 контрольные суммы файлов совпадают, из чего можно сделать вывод, что вторая программа работает правильно и действительно высчитывание контрольной суммы каждого отдельного файла осуществляется с помощью первой программы в отдельном процессе.

Результат работы программы на языке программирования C++ представлен на рисунке 3.

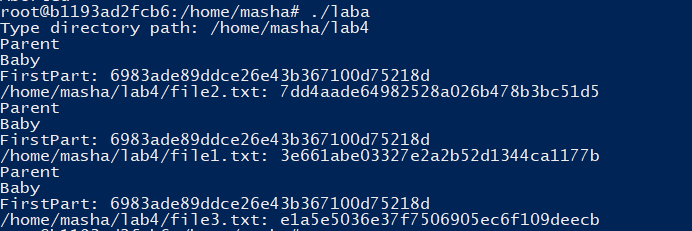


Рисунок 3 – Результат работы 2 программы на C++

Так же можно заметить, что контрольная сумма файлов директории совладает с контрольной суммой отдельных файлов, высчитанной ранее и представленной на рисунке 1

3 Заключение

В ходе работы были изучены с основные функции POSIX API для работы с процессами на языке Python и С++, особенностями процессов в операционных системах Unix.

Все необходимые файлы для воспроизведения работы данной практической работы были загружены в репозиторий: https://github.com/mariaigorevna/Process.

Приложение А

**import** hashlib

**def** hash\_file\_print**(**file\_path**):**

**with** open**(**file\_path**,** 'rb'**)** **as** f**:**

hash\_str **=** hashlib**.**md5**()**

**while** **True:**

data **=** f**.**read**(**8192**)**

**if** **not** data**:**

**break**

hash\_str**.**update**(**data**)**

**print(**file\_path **+** ": " **+** hash\_str**.**hexdigest**());**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

file\_path **=** input**(**"Type file path: "**)**

hash\_file\_print**(**file\_path**)**

Листинг А.1 – Исходный код первой Python программы

**import** os

**from** multiprocessing **import** Process

**from** FirstPart **import** hash\_file\_print

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

dir\_path **=** input**(**"Type directory path: "**)**

files\_arr **=** os**.**listdir**(**dir\_path**)**

processes **=** **[]**

j **=** 0

**for** file\_name **in** files\_arr**:**

processes**.**insert**(**j**,** Process**(**target**=**hash\_file\_print**,** args**=(**dir\_path**+**'/'**+**file\_name**,)))**

processes**[**j**].**start**()**

j **+=** 1

**for** i **in** range**(**j**):**

processes**[**i**].**join**()**

Листинг А.1 – Исходный код второй Python программы

#include <iostream>

#include <unistd.h>

#include <sys/wait.h>

#include <vector>

#include <filesystem>

Листинг А.2 – Исходный код второй C++ программы (Начало)

int main**()**

**{**

pid\_t pid**;**

std**::**string path**;**

std**::**vector**<**std**::**string**>** vecPaths**;**

std**::**cout **<<** "Type directory path: "**;**

std**::**cin **>>** path**;**

**for** **(**const auto **&**entry **:** std**::**filesystem**::**directory\_iterator**(**path**))**

**{**

vecPaths**.**push\_back**(**entry**.**path**());**

**}**

**if** **(**vecPaths**.**size**()** **==** 0**)**

**{**

std**::**cerr **<<** "Error: Empty directory\n"**;**

**return** 1**;**

**}**

**for** **(**const auto filePath **:** vecPaths**)**

**{**

**switch(**pid**=**fork**())**

**{**

**case** **-**1**:**

std**::**cerr **<<** "Process create error\n"**;**

exit**(**1**);**

**case** 0**:**

std**::**cout **<<** "Baby\n"**;**

execl**(**"FirstPart"**,** " FirstPart"**,** filePath**.**c\_str**(),** **NULL);**

exit**(**0**);**

**default:**

std**::**cout **<<** "Parent\n"**;**

wait**(**0**);**

**}**

**}**

**return** 0**;**

**}**

Листинг А.2 – Исходный код второй C++ программы (Конец)

FROM debian

RUN apt update

Листинг А.3 – Исходный код Dockerfile (Начало)

RUN apt install g++ python3 python -y

COPY ./FirstPart.py /home/masha/FirstPart.py

COPY ./SecondPart.cpp /home/masha/SecondPart.cpp

COPY ./FirstPart /home/masha/FirstPart

COPY ./SecondPart.py /home/masha/SecondPart.py

COPY ./lab4 /home/masha/lab4

ENTRYPOINT /bin/bash

Листинг А.3 – Исходный код Dockerfile (Конец)