ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Доц., к.т.н.		В.В.Балберин
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Процессы. Системные вызовы

по курсу: Операционные системы семейства GNU/Linux

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4745М

	П.А. Константинов
подпись, дата	инициалы, фамилия

Санкт-Петербург 2018

Цели

1. Приобретение навыков по управлению процессами.
2. Изучение системного вызова fork().
3. Приобретение навыков написания и трансляции системного ПО на языке C.
4. Изучение утилиты make и её использование для трансляции программ на языке C.
5. Использование удалённых серверов для трансляции приложений.
6. Изучение Docker и базовое взаимодействие с ним
7. Изучение способов представления информации помимо стандартных

Выполнение работы

Для начала было необходимо выбрать основную платформу для Docker, 2 были предложены преподавателем – Alpine и RancherOS, и стандартная Ubuntu. В ходе апробации данных систем выяснилось, что, несмотря на своё удобство и компактность, предложенные системы не удовлетворяют требованиям, в силу своей же компактности, так как на обеих системах в стандартном пакете gcc не присутствуют статические библиотеки, а, следовательно, была выбрана старая добрая Ubuntu 16.04

```
sudo docker run -it ubuntu:latest
```

Обновление установщика:

```
apt-get update

    apt-get upgrade
```

Затем было необходимо поставить базовые программы:

```
apt-get install nano

    apt-get install gcc
```

```
apt-get install openssh
```

Создаем каталог work и переходим в него:

```
mkdir work

cd work
```

Копируем файлы с удаленного сервера:

```
scp -r -P 6666 student@openit.guap.ru:/container/ABC-Linux/lab4/* work/
```

Далее мы начинаем работать уже с полученными файлами В большинстве файлов были допущены лишь незначительные ошибки типа пробелов вместо табуляции, которые, однако не позволяли корректно запустить программу но в файле lab4.c была допущена критическая логическая ошибка:

```
    pid = getpid();
    ppid = getppid();

    if(fork() == -1)
    {
/* ошибка */
    }
    else if (pid == 0)
```

Вот как неправильно

(слово ошибка намекает)

Она представляет собой моментальное сравнение системного вызова fork() с числом вместо сравнения переменной, которой присвоено значение этого вызова Эту ошибку можно и нужно исправить следующим образом:

```
pid_t pid, ppid;
int a =  0;
int b =  100;
pid_t frez = fork();


if (frez < 0)
{
        return 1;
}
else if (frez == 0)
{
```

А вот так правильно

После исправления данных ошибок программа стала корректно работать во всем и выдавать правильные значения (pid, ppid) для дочернего и родительского процессов... кроме ppid дочернего процесса - он был как у родительского и равен 1, что, естественно, неправильно. Данная проблема была проанализирована, и были выведены следующие выводы:

1. pid = 1 означает процесс init, 1й процесс системы
2. раз ppid = 1, значит, у данного процесса нет родительского
3. раз у данного процесса нет родительского, значит, родительский процесс завершается раньше дочернего
4. раз родительский процесс завершается раньше дочернего, значит его необходимо задержать

Было принято решение в родительском процессе использовать команду sleep():

```
{
        sleep(3);
        b = DO_B(b);
        pid = getpid();
        ppid = getppid();
        printf("parent: My pid = %d, my ppid = %d,result a = 
}
rn 0;
```

Решение проблемы ppid = 1

Итак, программа работает, осталось лишь собрать ее из нашего Makefile:

```
lab4: lab4.o pr_a.o pr_b.o lab4.h
	gcc lab4.o pr_a.o pr_b.o -o lab4 -lm

r_a.o: pr_a.c
	gcc -c pr_a.c

r_b.o: pr_b.c
	gcc -c pr_b.c

lab4.o: lab4.c lab4.h
	gcc -c lab4.c

clean:
	rm -f lab4 lab4.o pr_a.o pr_b.o

install:
	cp lab4 bin/lab4

uninstall:
	rm -f bin/lab4
```

Внешний вид Makefile

Мы делаем команды

```
make
```

```
make install
```

```
make clean
```

И запускаем нашу программу из папки bin:

```
root@80272bd3cf77:/labs/work# ./bin/lab4
child: My pid = 199, my ppid = 198,result a = 1,result b = 100
parent: My pid = 198, my ppid = 1,result a = 0,result b = 99
root@80272bd3cf77:/labs/work#
```

Вывод программы

Ну вот и всё, осталось лишь сохранить результаты, для этого переходим в другой терминал и сначала ищем наш рабочий контейнер:

```
docker ps
```

В колонке ContainerID имя нашего контейнера, вводим его в команде

```
docker commit *имя нашего контейнера* *имя каталога куда*
```

Для меня данная команда выглядит так:

```
docker ps
CONTAINER ID        IMAGE                          COMMAND             CREATED             STATUS
          PORTS               NAMES
b92aee8a92b7        konstantinov/abc-linux-lab4    "/bin/sh"           9 days ago          Up 9 days
                              compassionate_jennings
80272bd3cf77        17da3e5ee6bf                   "/bin/bash"         2 weeks ago         Up 2 weeks
                              gracious_lamarr
pavel@pavelVB:~$ docker commit 80272bd3cf77 konstantinov/abc-linux-4-1
```

Сохранение Docker-контейнера

Затем мы заходим в dockerhub под нашим логином и паролем:

```
docker login
```

После этого мы здесь же тегируем и push'им наш контейнер

Загрузить данный контейнер после данных операций можно по команде:

```
docker run konstantinov/abc-linux-4-1
```

Выводы

Были исследованы процессы и системный вызов fork(), был рассмотрен процесс создания каталогов и программы, изучено взаимодействие с Docker, а также данный отчет был сделан при помощи средств удаленного документирования.