Выполнила: Белоусова Е., ИП-911

Задача

Цель: знакомство с процессами Linux.

Упражнение 1. Протестируйте программы, рассмотренные на Лекции 2.

Упражнение 2. Создайте процесс с помощью вызова fork, с помощью команд ps и grep получите информацию о созданных вами родительском и дочернем процессах. Используя команду kill убейте родительский процесс, продолжил ли выполняться дочерний процесс?

Упражнение 3. Создайте дерево процессов с помощью вызова fork. С помощью команды pstree найдите поддерево созданных процессов. В каталоге /proc виртуальной файловой системы найдите папки с именами, совпадающими с идентификаторами созданных процессов, и просмотрите содержимое папок task/children.

Описание работы программы

Упражнение 1:

Протестируем программы из лекции 2.

Создание процесса:

```
sonya@sonya-S551LB:~/OS$ gcc -o 2 2.c
sonya@sonya-S551LB:~/OS$ ./2
Before RECREATION 4267
I'm not yet dead! My ID is 4267
I'm not yet dead! My ID is 4267
I'm not yet dead! My ID is 4267
I'm not yet dead! My ID is 4267
I'm not yet dead! My ID is 4267
Who I am? My ID is 4268
Who I am? My ID is 4268
Who I am? My ID is 4268
Who I am? My ID is 4268
Who I am? My ID is 4268
sonya@sonya-S551LB:~/OS$
```

При создании процесса с помощью fork() копируется адресное пространство, но отображение на физическую память различно. Если вызов fork() произошел успешно, то вернется 0, иначе возвращается код родительского процесса, либо -1, если произошла ошибка при порождении процесса.

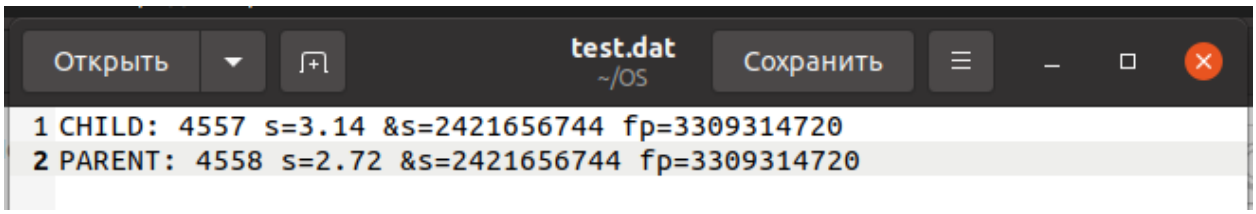
2.c | 2_1.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    pid_t child_pid, parent_pid;
    double s=0.0;
    child_pid=fork();
    if(child_pid!=0)
    {
        s+=3.14;
        fprintf(stdout, "CHILD: %i s=%g &s=%u\n", (int) getpid(),s,&s);
    }
    else
    {
        s+=2.72;
        fprintf(stdout, "PARENT: %i s=%g &s=%u\n", (int) getpid(),s, &s);
    }
    return 0;
}
```

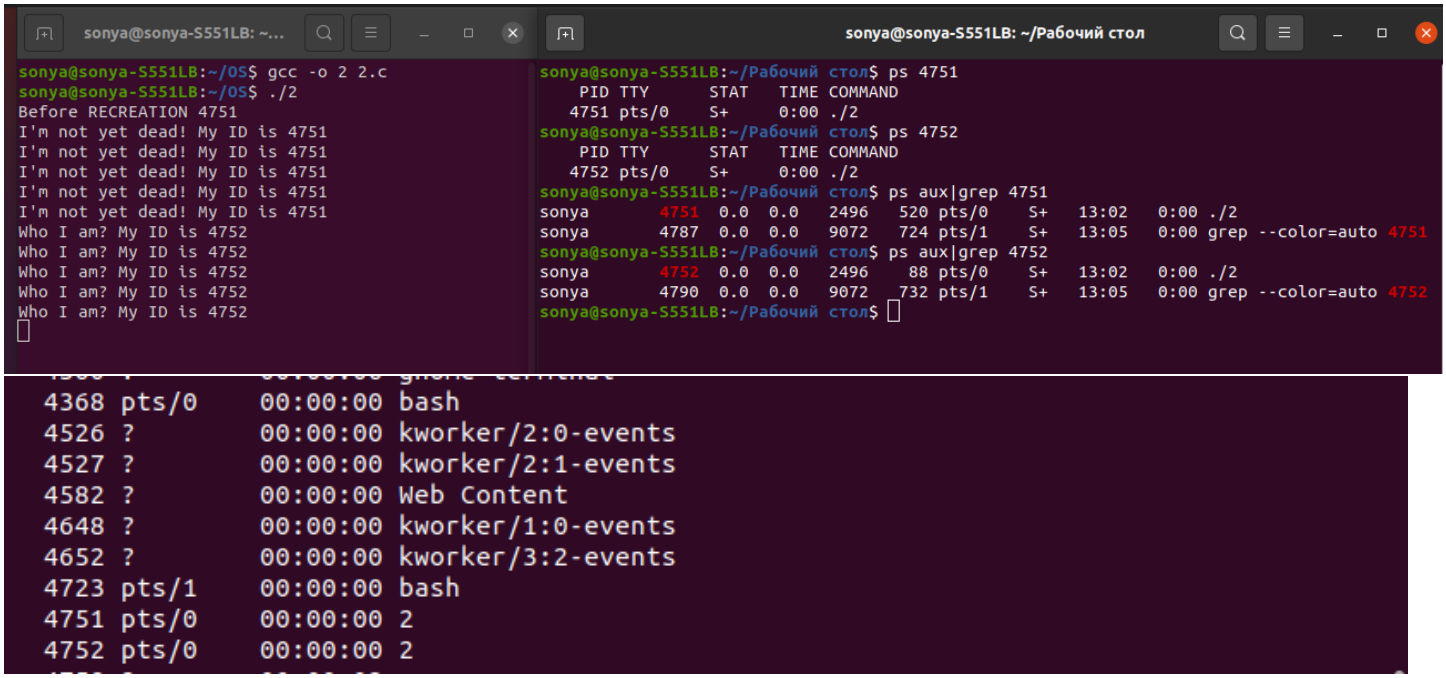
```
sonya@sonya-S551LB:~/OS$ ./2_1
CHILD: 4399 s=3.14 &s=2987795616
PARENT: 4400 s=2.72 &s=2987795616
sonya@sonya-S551LB:~/OS$
```

Дескрипторы файлов при копировании сохраняются.

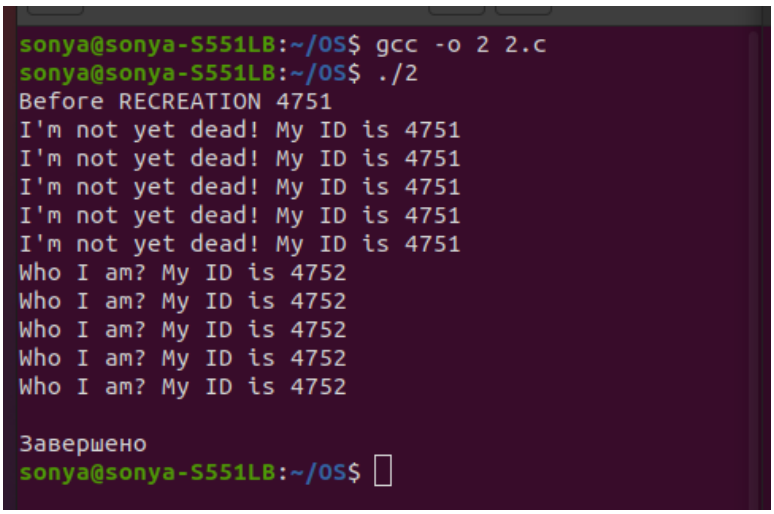


Упражнение 2:

Создадим процесс с помощью fork(). Получим информацию о родительском и дочернем процессах с помощью команд ps, grep, пока программа с помощью getchar() будет дожидаться нажатия клавиши.



Нажмем любую клавишу, после чего убьем родительский процесс. Проверим, выполняется ли дочерний процесс.



```
4368 pts/0    00:00:00 bash
4526 ?        00:00:00 kworker/2:0-events
4527 ?        00:00:00 kworker/2:1-events
4582 ?        00:00:00 Web Content
4648 ?        00:00:00 kworker/1:0-events
4652 ?        00:00:00 kworker/3:2-events
4723 pts/1    00:00:00 bash
4752 pts/0    00:00:00 2
4758 ?        00:00:02 eog
```

```
4898 pts/1    00:00:00 ps
sonya@sonya-S551LB:~/Рабочий стол$ ps 4752
    PID TTY      STAT   TIME COMMAND
   4752 pts/0    S      0:00 ./2
sonya@sonya-S551LB:~/Рабочий стол$ ps 4751
    PID TTY      STAT   TIME COMMAND
sonya@sonya-S551LB:~/Рабочий стол$
```

На приведенных скриншотах видно, что родительский процесс перестал существовать, а дочерний продолжает выполняться.

Упражнение 3:

Создадим дерево процессов с помощью вызовов fork(). Используем для этого тот факт, что после fork() код выполняется дважды, как в процессе потомке, так и в процессе-родителе.

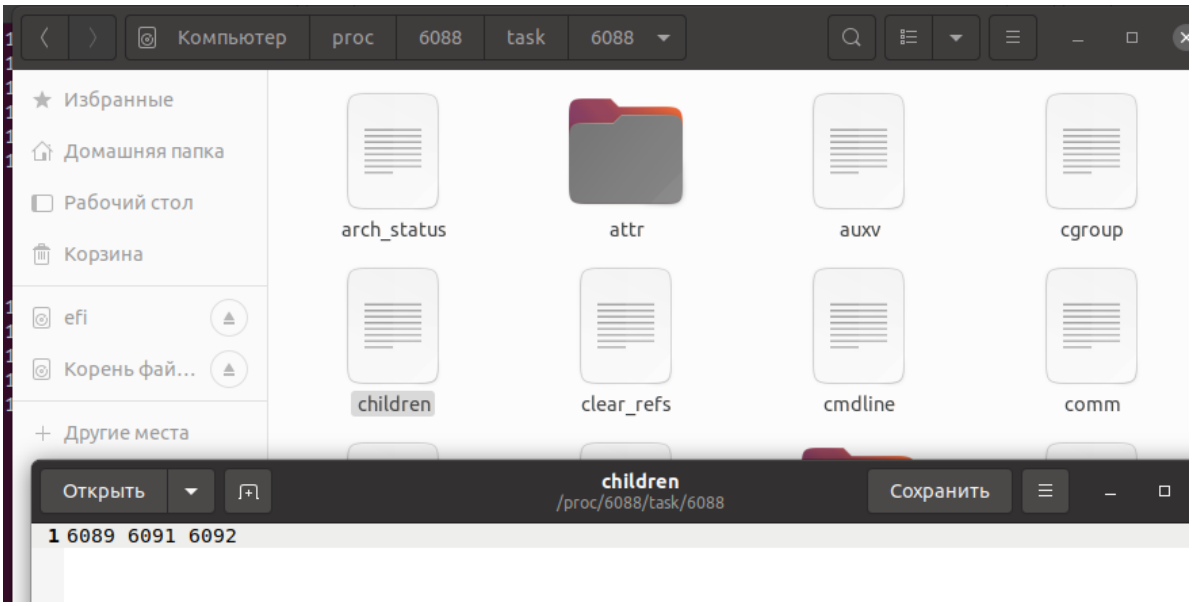
```
sonya@sonya-S551LB:~/OS$ gcc -o 2 2.c
sonya@sonya-S551LB:~/OS$ ./2
Before RECREATION 6088
6089 spawned 6090
6088 spawned 6091
6088 spawned 6092
6089 spawned 6093
6091 spawned 6094
6090 spawned 6095
```

С помощью команды pstree найдем поддерево созданных процессов.

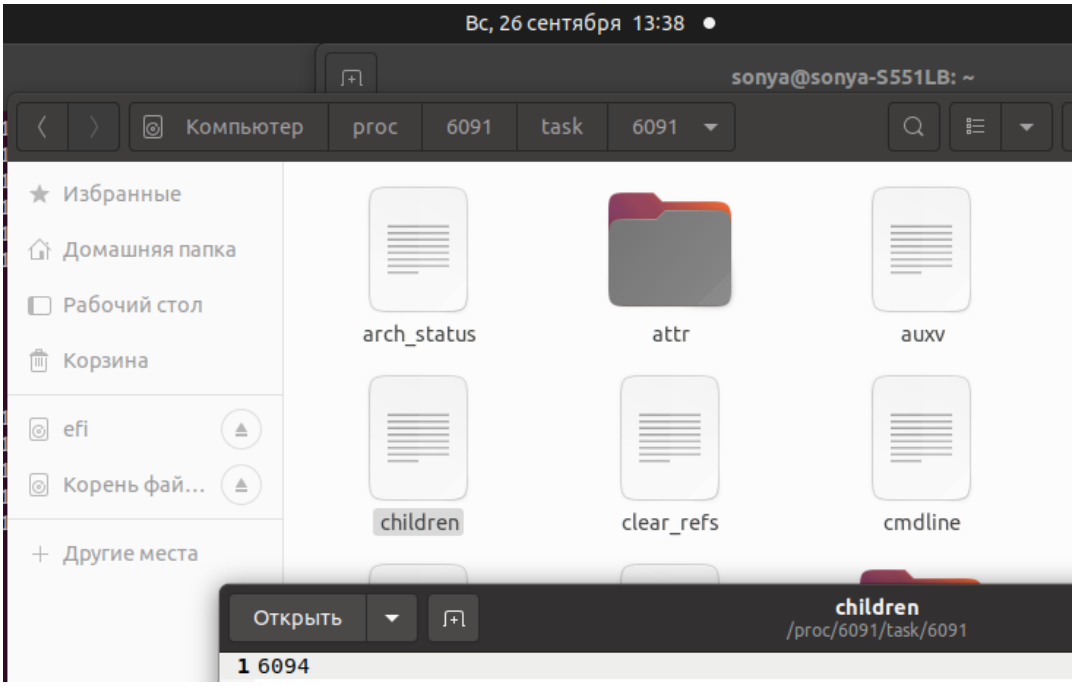
```
4360 ?        00:00:21  \_ gnome-terminal-
4368 pts/0    00:00:00  |   \_ bash
6088 pts/0    00:00:00  |   |   \_ 2
6089 pts/0    00:00:00  |   |       \_ 2
6090 pts/0    00:00:00  |   |       |   \_ 2
6095 pts/0    00:00:00  |   |       |   |   \_ 2
6093 pts/0    00:00:00  |   |       |   \_ 2
6091 pts/0    00:00:00  |   |       \_ 2
6094 pts/0    00:00:00  |   |       |   \_ 2
6092 pts/0    00:00:00  |   |       \_ 2
4723 pts/1    00:00:00  |   \_ bash
6329 pts/1    00:00:00  |       \_ ps
6330 pts/1    00:00:00  |       \_ less
```

```
sonya@sonya-S551LB:~/Рабочий стол$ pstree 6091
2──2
sonya@sonya-S551LB:~/Рабочий стол$ pstree 6088
2─┬─2─┬─2──2
  │   └─2
  ├─2──2
  └─2
sonya@sonya-S551LB:~/Рабочий стол$ pstree 6089
2─┬─2──2
  └─2
```

В каталоге /proc виртуальной файловой системы найдем папки с именами 6088 (корень поддерева) и просмотрим содержимое папок task/children.



Проделаем это же для дочернего процесса 6091.



Листинг

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void oldman(); void recreation();

int main()
{
	pid_t child_pid, parent_pid;
	int i=0;
	fprintf(stdout, "Before RECREATION %i\n",
	parent_pid=(int) getpid());
	child_pid=fork();
	while(i++<5)
	{
		if(child_pid!=0)
		{
			oldman();
		}
		else
		{
			recreation();
		}
	}
	getchar();
	kill(parent_pid, SIGTERM);
	return 0;
}


void oldman()
{
	fprintf(stdout, "I'm not yet dead! My ID is %i\n", (int) getpid());
}
void recreation()
{
	fprintf(stdout, "Who I am? My ID is %i\n", (int) getpid());
}

//№3

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

void createChildProcess()
{
	pid_t child_pid = fork();
```

```
	if (child_pid != 0)
	{
		fprintf(stdout, "%u spawned %u\n", getpid(), child_pid);
	}
	else
	{
		return;
	}
}

int main()
{
	pid_t child_pid, parent_pid;
	int i = 0;
	fprintf(stdout, "Before RECREATION %i\n", parent_pid = (int) getpid());
	child_pid = fork();
	createChildProcess();
	createChildProcess();
	getchar();
	//kill(parent_pid, SIGTERM);
	return 0;
}
```