

Отчёт по лабораторной работе №6

Управление процессами

Ришард Когенгар

Содержание

1 Цель работы	5
2 Ход выполнения	6
2.1 Управление заданиями	6
2.2 Управление процессами	10
2.3 Самостоятельная работа	12
2.3.1 Задание 1	12
2.3.2 Задание 2	13
2.4 Вывод	19
3 Контрольные вопросы	20

Список иллюстраций

2.1 Просмотр списка заданий jobs	7
2.2 Завершение всех заданий и пустой вывод jobs	8
2.3 Отображение процесса dd в top	9
2.4 Отсутствие процесса dd после завершения	10
2.5 Просмотр процессов dd через ps aux	11
2.6 Завершение родительского процесса и остановка dd	12
2.7 Запуск трёх процессов dd в фоновом режиме	12
2.8 Запуск yes в фоне с подавлением вывода	14
2.9 Запуск yes без подавления вывода	15
2.10 Перевод фонового процесса на передний план и остановка	16
2.11 Отображение процессов yes в утилите top	17
2.12 Запуск дополнительных процессов yes	18

Список таблиц

1 Цель работы

Получить навыки управления процессами операционной системы.

2 Ход выполнения

2.1 Управление заданиями

1. Для выполнения операций с заданиями получены полномочия администратора с помощью команды **su**.

После входа в root-оболочку были запущены следующие команды:

- `sleep 3600 &` – запуск таймера на 3600 секунд в фоновом режиме;
- `dd if=/dev/zero of=/dev/null &` – запуск процесса генерации нагрузки в фоне;
- `sleep 7200` – запуск таймера на 7200 секунд на переднем плане.

2. Поскольку команда `sleep 7200` была запущена без символа &, управление оболочкой было заблокировано данным процессом.

Для остановки его выполнения использовано сочетание клавиш **Ctrl + Z**, в результате чего задание было переведено в состояние **Stopped**.

3. Для просмотра списка активных заданий текущей оболочки выполнена команда **jobs**.

В выводе отображены три задания:

- задание №1 – **Running** (`sleep 3600`);
- задание №2 – **Running** (`dd if=/dev/zero of=/dev/null`);
- задание №3 – **Stopped** (`sleep 7200`).

```
rishard@rishardkogengar:~$ su
Password:
root@rishardkogengar:/home/rishard# sleep 3600 &
[1] 4217
root@rishardkogengar:/home/rishard# dd if=/dev/zero of=/dev/null &
[2] 4255
root@rishardkogengar:/home/rishard# sleep 7200
^Z
[3]+  Stopped                  sleep 7200
root@rishardkogengar:/home/rishard# jobs
[1]  Running                  sleep 3600 &
[2]-  Running                  dd if=/dev/zero of=/dev/null &
[3]+  Stopped                  sleep 7200
root@rishardkogengar:/home/rishard# bg 3
[3]+ sleep 7200 &
root@rishardkogengar:/home/rishard# jobs
[1]  Running                  sleep 3600 &
[2]-  Running                  dd if=/dev/zero of=/dev/null &
[3]+  Running                  sleep 7200 &
root@rishardkogengar:/home/rishard# █
```

Рис. 2.1: Просмотр списка заданий **jobs**

4. Для продолжения выполнения задания №3 в фоновом режиме использована команда **bg 3**.

После этого повторный вывод команды **jobs** показал, что задание №3 перешло в состояние **Running**.

5. Для переноса задания №1 на передний план выполнена команда **fg 1**.

После возврата управления процесс был остановлен пользователем с помощью **Ctrl + C**.

6. Аналогичным образом были остановлены задания №2 и №3: каждое из них переводилось на передний план командой **fg**, после чего завершалось сочетанием **Ctrl + C**.

Повторный вывод команды **jobs** подтвердил отсутствие активных заданий.

```
root@rishardkogengar:/home/rishard# fg 1
sleep 3600
^C
root@rishardkogengar:/home/rishard# jobs
[2]-  Running                  dd if=/dev/zero of=/dev/null &
[3]+  Running                  sleep 7200 &
root@rishardkogengar:/home/rishard# fg 2
dd if=/dev/zero of=/dev/null
^C202407824+0 records in
202407823+0 records out
103632805376 bytes (104 GB, 97 GiB) copied, 75.6076 s, 1.4 GB/s

root@rishardkogengar:/home/rishard# fg 3
sleep 7200
^C
root@rishardkogengar:/home/rishard# jobs
root@rishardkogengar:/home/rishard# █
```

Рис. 2.2: Завершение всех заданий и пустой вывод jobs

7. Во втором терминале под учётной записью пользователя был запущен процесс `dd if=/dev/zero of=/dev/null &`.
После этого второй терминал был закрыт командой **exit**.
8. В первом терминале под учётной записью пользователя была запущена утилита мониторинга процессов **top**.
В списке процессов был обнаружен процесс `dd`, продолжающий выполнение несмотря на закрытие терминала, из которого он был запущен.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4590	rishard	20	0	226848	1920	1796	R	100.0	0.1	0:09.06	dd
1	root	20	0	50188	41896	10408	S	0.0	1.1	0:01.18	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pool_workqueue_release
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-rcu_gp
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-sync_wq
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-slub_flushwq
7	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-netns
8	root	20	0	0	0	0	I	0.0	0.0	0:00.02	kworker/0:0-rcu_gp
9	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/0:1-ata_sff
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-events_highpri
11	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/u16:0-events_unbound
12	root	20	0	0	0	0	I	0.0	0.0	0:00.03	kworker/u16:1-netns
13	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-mm_percpu_wq
14	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_kthread
15	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_rude_kthread
16	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_trace_kthread
17	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
18	root	20	0	0	0	0	I	0.0	0.0	0:00.04	rcu_preempt
19	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_exp_par_gp_kthread_worker/0
20	root	20	0	0	0	0	S	0.0	0.0	0:00.01	rcu_exp_gp_kthread_worker

Рис. 2.3: Отображение процесса dd в top

- После повторного запуска **top** процесс **dd** был завершён с помощью команды **k** (kill) с указанием его PID.

После выхода из **top** подтверждено, что процесс **dd** больше не выполняется.

top - 14:25:14 up 7 min, 4 users, load average: 0.61, 0.34, 0.13												
Tasks: 272 total, 1 running, 271 sleeping, 0 stopped, 0 zombie												
%Cpu(s): 0.9 us, 0.9 sy, 0.0 ni, 97.6 id, 0.0 wa, 0.4 hi, 0.1 si, 0.0 st												
MiB Mem : 3652.9 total, 2128.5 free, 1205.5 used, 549.2 buff/cache												
MiB Swap: 4040.0 total, 4040.0 free, 0.0 used. 2447.4 avail Mem												
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	
2710	rishard	20	0	4989800	308292	123884	S	5.0	8.2	0:04.08	gnome-shell	
4001	rishard	20	0	3042720	269068	102500	S	2.7	7.2	0:02.44	ptyxis	
70	root	20	0	0	0	0	I	0.7	0.0	0:00.32	kworker/u18:1-events_unbound	
1	root	20	0	50188	41896	10408	S	0.3	1.1	0:01.20	systemd	
44	root	20	0	0	0	0	I	0.3	0.0	0:00.07	kworker/u19:0-events_unbound	
1101	root	20	0	530156	7396	6504	S	0.3	0.2	0:00.05	accounts-daemon	
3032	rishard	20	0	615360	10640	8952	S	0.3	0.3	0:00.03	goa-identity-se	
3401	rishard	20	0	370016	2316	1968	S	0.3	0.1	0:00.02	VBoxClient	
4650	rishard	20	0	231604	5452	3280	R	0.3	0.1	0:00.03	top	
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd	
3	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pool_workqueue_release	
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-rcu_gp	
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-sync_wq	
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-slab_flushwq	
7	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-netns	
8	root	20	0	0	0	0	I	0.0	0.0	0:00.03	kworker/0:0-mm_percpu_wq	
9	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/0:1-ata_sff	
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-events_highpri	
11	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/u16:0-events_unbound	
12	root	20	0	0	0	0	I	0.0	0.0	0:00.03	kworker/u16:1-netns	
13	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-mm_percpu_wq	
14	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_kthread	

Рис. 2.4: Отсутствие процесса dd после завершения

2.2 Управление процессами

- Получены полномочия администратора и в одной root-оболочке последовательно запущены три процесса dd if=/dev/zero of=/dev/null в фоновом режиме.
- Для отображения всех процессов, содержащих строку dd, выполнена команда **ps aux | grep dd**.

В выводе отображены запущенные процессы dd с их идентификаторами PID.

```

root@rishardkogengar:/home/rishard# dd if=/dev/zero of=/dev/null &
[1] 4969
root@rishardkogengar:/home/rishard# dd if=/dev/zero of=/dev/null &
[2] 4971
root@rishardkogengar:/home/rishard# dd if=/dev/zero of=/dev/null &
[3] 4973
root@rishardkogengar:/home/rishard# ps aux | grep dd
root      2  0.0  0.0    0   0 ?        S   14:17  0:00 [kthreadd]
root     92  0.0  0.0    0   0 ?        I<  14:17  0:00 [kworker/R-ipv6_addrconf]
root    1361  0.0  0.0 512956 3444 ?       Sl  14:17  0:00 /usr/sbin/VBoxService --pidfile /var/run/vboxadd-service.sh
rishard  3094  0.0  0.6 1037324 26136 ?       Ssl 14:18  0:00 /usr/libexec/evolution-addressbook-factory
root    4969 99.3  0.0 226848 1860 pts/0    R   14:26  0:12 dd if=/dev/zero of=/dev/null
root    4971 99.0  0.0 226848 2004 pts/0    R   14:26  0:10 dd if=/dev/zero of=/dev/null
root    4973 99.2  0.0 226848 1984 pts/0    R   14:26  0:10 dd if=/dev/zero of=/dev/null
root    4999  0.0  0.0 227688 2172 pts/0    S+  14:26  0:00 grep --color=auto dd
root@rishardkogengar:/home/rishard# renice -n 5 4969
4969 (process ID) old priority 0, new priority 5
root@rishardkogengar:/home/rishard# █

```

Рис. 2.5: Просмотр процессов dd через ps aux

3. Для изменения приоритета одного из процессов dd использована команда **renice** с указанием PID выбранного процесса.

В результате приоритет процесса был изменён с значения 0 на значение 5, что подтверждено выводом команды.

4. Для просмотра иерархии процессов выполнена команда **ps fax | grep -B5 dd**.

Это позволило отобразить дерево процессов и определить PID родительской root-оболочки, из которой были запущены все процессы dd.

5. Для завершения всех процессов dd одновременно выполнено принудительное завершение родительской оболочки с помощью сигнала **SIGKILL**. После выполнения команды оболочка была закрыта, а все дочерние процессы dd автоматически остановлены, что подтверждает зависимость дочерних процессов от родительского.

```

-- 
4001 ? Rsl 0:03 \_ /usr/bin/ptyxis --application-service
4009 ? Ssl 0:00 | \_ /usr/libexec/ptyxis-agent --socket-fd=3 --rlimit-nofile=1024
4075 pts/0 Ss 0:00 | \_ /usr/bin/bash
4124 pts/0 S 0:00 | \_ su
4156 pts/0 S 0:00 | \_ bash
4969 pts/0 RN 0:58 | \_ dd if=/dev/zero of=/dev/null
4971 pts/0 R 0:57 | \_ dd if=/dev/zero of=/dev/null
4973 pts/0 R 0:57 | \_ dd if=/dev/zero of=/dev/null
5109 pts/0 R+ 0:00 | \_ ps fax
5110 pts/0 S+ 0:00 | \_ grep --color=auto -B5 dd
root@rishardkogengar:/home/rishard# kill -9 4156
Killed

```

Рис. 2.6: Завершение родительского процесса и остановка dd

2.3 Самостоятельная работа

2.3.1 Задание 1

1. В root-оболочке трижды запущена команда **dd if=/dev/zero of=/dev/null** в фоновом режиме.

В результате были созданы три параллельно выполняющихся процесса, каждому из которых был присвоен собственный идентификатор PID.

```

root@rishardkogengar:/home/rishard#
root@rishardkogengar:/home/rishard# dd if=/dev/zero of=/dev/null &
[1] 5485
root@rishardkogengar:/home/rishard# dd if=/dev/zero of=/dev/null &
[2] 5497
root@rishardkogengar:/home/rishard# dd if=/dev/zero of=/dev/null &
[3] 5500
root@rishardkogengar:/home/rishard# renice -n 5 5485
5485 (process ID) old priority 0, new priority 5
root@rishardkogengar:/home/rishard# renice -n 15 5485
5485 (process ID) old priority 5, new priority 15
root@rishardkogengar:/home/rishard# killall dd
[1]  Terminated      dd if=/dev/zero of=/dev/null
[2]- Terminated      dd if=/dev/zero of=/dev/null
[3]+ Terminated      dd if=/dev/zero of=/dev/null
root@rishardkogengar:/home/rishard# █

```

Рис. 2.7: Запуск трёх процессов dd в фоновом режиме

2. Для одного из запущенных процессов dd был изменён приоритет с использованием утилиты **renice** и значения приоритета **-5**.

Команда выполнилась успешно, что подтверждается сообщением об изменении приоритета с исходного значения на новое.

3. Для того же процесса приоритет был изменён повторно, на этот раз с использованием значения **-15**.

В результате процесс получил более высокий приоритет по сравнению с предыдущим состоянием.

Разница между значениями **-5** и **-15** заключается в том, что чем меньше числовое значение приоритета (ближе к **-20**), тем больше процессорного времени ядро выделяет данному процессу.

4. Для завершения всех ранее запущенных процессов dd использована команда **killall dd**.

В результате выполнения команды все процессы были успешно завершены, что подтверждается сообщениями о завершении заданий.

2.3.2 Задание 2

1. Программа **yes** была запущена в фоновом режиме с подавлением потока вывода путём перенаправления вывода в **/dev/null**.

Процесс начал выполняться в фоне, о чём свидетельствует его статус **Running**.

```
root@rishardkogengar:/home/rishard# yes > /dev/null &
[1] 5977
root@rishardkogengar:/home/rishard# yes > /dev/null
^Z
[2]+  Stopped                  yes > /dev/null
root@rishardkogengar:/home/rishard# yes > /dev/null
^C
root@rishardkogengar:/home/rishard# jobs
[1]-  Running                  yes > /dev/null &
[2]+  Stopped                  yes > /dev/null
root@rishardkogengar:/home/rishard# █
```

Рис. 2.8: Запуск yes в фоне с подавлением вывода

2. Далее программа **yes** была запущена на переднем плане с подавлением потока вывода.

Выполнение программы было приостановлено с помощью сочетания клавиш **Ctrl + Z**, после чего она была повторно запущена с теми же параметрами и завершена пользователем.

3. После этого программа **yes** была запущена на переднем плане без подавления потока вывода.

Процесс был приостановлен, затем повторно запущен с теми же параметрами и завершён пользователем.

```
root@rishardkogengar:/home/rishard# ./yes > /dev/null &
root@rishardkogengar:/home/rishard# jobs
[1]-  Running                  yes > /dev/null &
[2]+  Stopped                  yes > /dev/null
root@rishardkogengar:/home/rishard# fg 1
yes > /dev/null
^C
root@rishardkogengar:/home/rishard# jobs
[2]+  Stopped                  yes > /dev/null
root@rishardkogengar:/home/rishard# bg 2
[2]+ yes > /dev/null &
root@rishardkogengar:/home/rishard# jobs
[2]-  Running                  yes > /dev/null &
root@rishardkogengar:/home/rishard# nohup yes > /dev/null &
[3] 6440
root@rishardkogengar:/home/rishard# nohup: ignoring input and redirecting stderr to stdout
root@rishardkogengar:/home/rishard# jobs
[2]-  Running                  yes > /dev/null &
[3]+  Running                  nohup yes > /dev/null &
root@rishardkogengar:/home/rishard# █
```

Рис. 2.9: Запуск yes без подавления вывода

4. Состояния всех заданий были проверены с помощью команды **jobs**. В выводе отображались задания в состояниях **Running** и **Stopped**, в зависимости от предыдущих действий.
5. Процесс, выполнявшийся в фоновом режиме, был переведён на передний план и затем остановлен пользователем с помощью **Ctrl + C**.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5992	root	20	0	226820	1900	1780	R	99.3	0.1	2:22.05	yes
6440	root	20	0	226820	1928	1808	R	99.0	0.1	0:43.34	yes
2710	rishard	20	0	5058924	308816	124168	S	3.0	8.3	0:08.08	gnome-shell
6536	rishard	20	0	3011612	216864	98332	S	2.0	5.8	0:00.35	ptyxis
4812	root	20	0	0	0	0	I	1.0	0.0	0:00.18	kworker/u18:4-events_unbound
18	root	20	0	0	0	0	R	0.3	0.0	0:00.12	rcu_preempt
1	root	20	0	50188	41896	10408	S	0.0	1.1	0:01.78	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pool_workqueue_release
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-rcu_gp
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-sync_wq
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-slub_flushwq
7	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-netns
8	root	20	0	0	0	0	I	0.0	0.0	0:00.09	kworker/0:0-events
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-events_highpri
11	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/u16:0-events_unbound
12	root	20	0	0	0	0	I	0.0	0.0	0:00.07	kworker/u16:1-netns
13	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-mm_percpu_wq
14	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_kthread
15	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_rude_kthread
16	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_trace_kthread
17	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0
19	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_exp_par_gp_kthread_worker/0

Рис. 2.10: Перевод фонового процесса на передний план и остановка

6. Один из процессов yes с подавлением потока вывода был переведён в фоновый режим.

После этого его состояние изменилось на **Running**, что было подтверждено повторным вызовом команды **jobs**.

7. Процесс yes был запущен в фоновом режиме с использованием механизма, позволяющего ему продолжать выполнение после отключения от терминала.

В списке заданий данный процесс отображался как выполняющийся независимо от текущей сессии.

8. Окно терминала было закрыто, после чего консоль была запущена повторно.

Проверка показала, что процесс yes продолжил свою работу, что подтверждает корректность его запуска.

9. Для получения информации о текущих процессах в системе использована утилита **top**.

В списке процессов были обнаружены активные процессы **yes**, потребляющие значительную долю процессорного времени.

```
root@rishardkogengar:/home/rishard# yes > /dev/null &
[1] 6852
root@rishardkogengar:/home/rishard# yes > /dev/null &
[2] 6854
root@rishardkogengar:/home/rishard# yes > /dev/null &
[3] 6856
root@rishardkogengar:/home/rishard# kill 6854
[2]- Terminated          yes > /dev/null
root@rishardkogengar:/home/rishard# fg 1
yes > /dev/null
^C
root@rishardkogengar:/home/rishard# kill -1 6856
[3]+ Hangup            yes > /dev/null
root@rishardkogengar:/home/rishard# kill -1 6440
root@rishardkogengar:/home/rishard# yes > /dev/null &
[1] 6969
root@rishardkogengar:/home/rishard# yes > /dev/null &
[2] 6971
root@rishardkogengar:/home/rishard# killall yes
[2]+ Terminated          yes > /dev/null
[1]+ Terminated          yes > /dev/null
root@rishardkogengar:/home/rishard# █
```

Рис. 2.11: Отображение процессов **yes** в утилите **top**

10. Дополнительно были запущены ещё три процесса **yes** в фоновом режиме с подавлением потока вывода.

```

root@rishardkogengar:/home/rishard# yes > /dev/null &
[1] 7187
root@rishardkogengar:/home/rishard# nice -n 5 yes > /dev/null &
[2] 7210
root@rishardkogengar:/home/rishard# ps -l | grep yes
4 R    0    7187    6748 99  80   0 - 56705 -      pts/0    00:00:17 yes
4 R    0    7210    6748 99  85   5 - 56705 -      pts/0    00:00:08 yes
root@rishardkogengar:/home/rishard# renice -n 5 7187
7187 (process ID) old priority 0, new priority 5
root@rishardkogengar:/home/rishard# ps -l | grep yes
4 R    0    7187    6748 99  85   5 - 56705 -      pts/0    00:00:44 yes
4 R    0    7210    6748 99  85   5 - 56705 -      pts/0    00:00:36 yes
root@rishardkogengar:/home/rishard# killall yes
[1]- Terminated          yes > /dev/null
[2]+ Terminated          nice -n 5 yes > /dev/null
root@rishardkogengar:/home/rishard#

```

Рис. 2.12: Запуск дополнительных процессов yes

11. Два процесса были завершены:

- один – с использованием его PID;
 - второй – с использованием идентификатора задания.
- В обоих случаях процессы были успешно остановлены.

12. Была выполнена попытка отправки сигнала **SIGHUP (1)** процессу, запущенному с использованием механизма сохранения выполнения после выхода из терминала, и обычному процессу.

В результате обычный процесс был завершён, тогда как процесс, запущенный с сохранением выполнения, продолжил работу.

13. После этого были запущены ещё несколько процессов **yes** в фоновом режиме с подавлением потока вывода.

Все они были одновременно завершены с помощью команды **killall**, что позволило остановить группу однотипных процессов одной командой.

14. В завершение одна программа **yes** была запущена в фоновом режиме с подавлением вывода, а вторая – с теми же параметрами, но с увеличенным

приоритетом с использованием утилиты **nice**.

Сравнение абсолютных и относительных приоритетов показало, что процесс с изменённым приоритетом получает меньше процессорного времени.

15. С помощью утилиты **renice** был изменён приоритет одного из запущенных процессов `yes`.

Повторная проверка приоритетов подтвердила корректность внесённых изменений и их влияние на планирование процессов.

2.4 Вывод

В ходе работы были изучены и отработаны основные механизмы управления заданиями и процессами в ОС Linux. Освоены приёмы запуска процессов в фоновом и переднем режимах, их приостановки, возобновления и завершения с использованием команд управления заданиями. Практически продемонстрировано изменение приоритетов процессов с помощью `nice` и `renice` и влияние этих изменений на распределение процессорного времени. Рассмотрены способы завершения отдельных процессов и групп процессов, а также поведение процессов при получении различных сигналов и при завершении родительской оболочки. Полученные результаты подтверждают корректную работу средств управления процессами и заданий в Linux.

3 Контрольные вопросы

1. Какая команда даёт обзор всех текущих заданий оболочки?

Команда `jobs` – отображает список всех заданий, запущенных в текущей оболочке, с указанием их номеров и состояний (Running, Stopped).

2. Как остановить текущее задание оболочки, чтобы продолжить его выполнение в фоновом режиме?

Необходимо приостановить выполнение текущего задания сочетанием клавиш `Ctrl + Z`, после чего выполнить команду `bg`, указав номер задания при необходимости.

3. Какую комбинацию клавиш можно использовать для отмены текущего задания оболочки?

Для отмены текущего задания используется сочетание клавиш `Ctrl + C`, которое отправляет процессу сигнал завершения.

4. Необходимо отменить одно из начатых заданий. Доступ к оболочке, в которой в данный момент работает пользователь, невозможен. Что можно сделать, чтобы отменить задание?

Можно определить идентификатор процесса (PID) с помощью команд `ps` или `top`, после чего завершить процесс командой `kill` или `kill -9`, используя найденный PID.

5. Какая команда используется для отображения отношений между родительскими и дочерними процессами?

Для отображения иерархии процессов используется команда `ps fax`, которая показывает дерево процессов и их взаимосвязи.

6. Какая команда позволит изменить приоритет процесса с идентификатором 1234 на более высокий?

Для повышения приоритета процесса используется команда

`renice -n -5 1234,`

где отрицательное значение означает более высокий приоритет.

7. В системе в настоящее время запущено 20 процессов dd. Как проще всего остановить их все сразу?

Проще всего использовать команду `killall dd`, которая завершит все процессы с именем dd.

8. Какая команда позволяет остановить команду с именем mycommand?

Для остановки процесса по имени используется команда `killall mycommand`.

9. Какая команда используется в top, чтобы убить процесс?

В утилите `top` для завершения процесса используется команда `k` с последующим вводом PID процесса и сигнала.

10. Как запустить команду с достаточно высоким приоритетом, не рискуя, что не хватит ресурсов для других процессов?

Для этого используется команда `nice` с умеренно повышенным приоритетом, например `nice -n -5`, что позволяет увеличить приоритет процесса, не нарушая баланс распределения ресурсов в системе.