МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа №5

По дисциплине «Операционные системы»

Управление памятью в ОС Linux

Выполнила студентка группы: №M3210
Горбоконенко Лидия Алексеевна

Проверил:
Осипов Святослав Владимирович

САНКТ-ПЕТЕРБУРГ

2020

Задание на лабораторную работу

- Проведите два виртуальных эксперимента в соответствии с требованиями и проанализируйте их результаты. В
- указаниях ниже описано, какие данные необходимо фиксировать в процессе проведения экспериментов.

Требования к проведению экспериментов и содержанию отчета

Зафиксируйте в отчете данные о текущей конфигурации операционной системы в аспекте управления памятью:

- Общий объем оперативной памяти.
- Объем раздела подкачки.
- Размер страницы виртуальной памяти.
- Объем свободной физической памяти в ненагруженной системе.
- Объем свободного пространства в разделе подкачки в ненагруженной системе.

Параметры	Значения
Общий объем оперативной памяти	1870900 kB
Объем раздела подкачки	839676 kB
Размер страницы виртуальной памяти	4096 kB
Объем свободной физической памяти в ненагруженной системе	1464408 kB
Объем свободного пространства в разделе подкачки в ненагруженной системе	839676 kB

Эксперимент No1

Подготовительный этап:

Создайте скрипт mem.bash, реализующий следующий сценарий. Скрипт выполняет бесконечный цикл. Перед началом выполнения цикла создается пустой массив и счетчик шагов, инициализированный нулем. На каждом шаге цикла в конец массива добавляется последовательность из 10 элементов, например, (1 2 3 4 5 6 7 8 9 10). Каждый 100000-ый шаг в файл report.log добавляется строка с текущим значением размера массива (перед запуском скрипта, файл обнуляется).

```
#!/bin/bash

>report.log

counter=0
arr=()

while true; do
        if (( (( counter % 100000 )) == 0 )); then
                echo ${#arr[*]} >> report.log
        else
                arr+=(1 2 3 4 5 6 7 8 9 10)
        fi
        counter=$(( counter + 1 ))
done
```

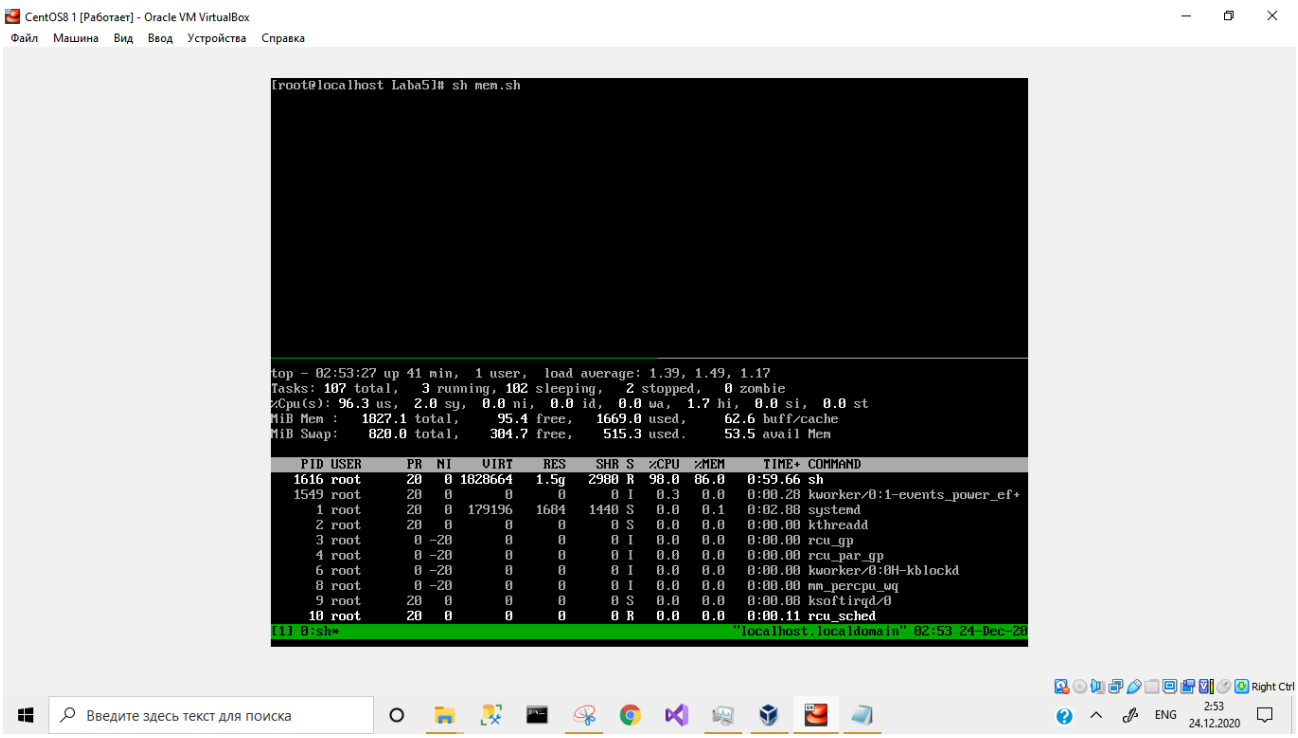
Первый этап:

Задача – оценить изменения параметров, выводимых утилитой top в процессе работы созданного скрипта.

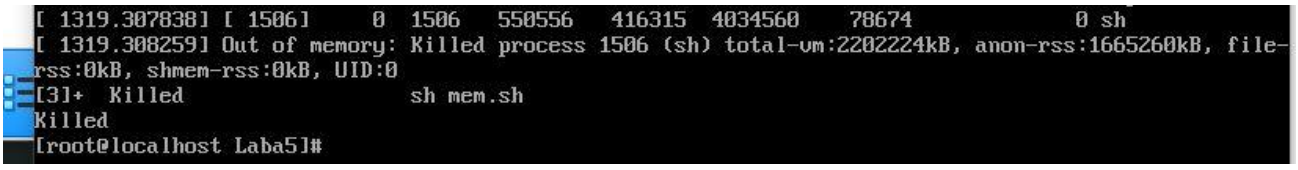
Ход эксперимента:

Запустите созданный скрипт mem.bash. Дождитесь аварийной остановки процесса и вывода в консоль последних сообщений системного журнала. Зафиксируйте в отчете последнюю запись журнала – значения параметров, с которыми произошла аварийная остановка процесса. Также зафиксируйте значение в последней строке файла report.log.

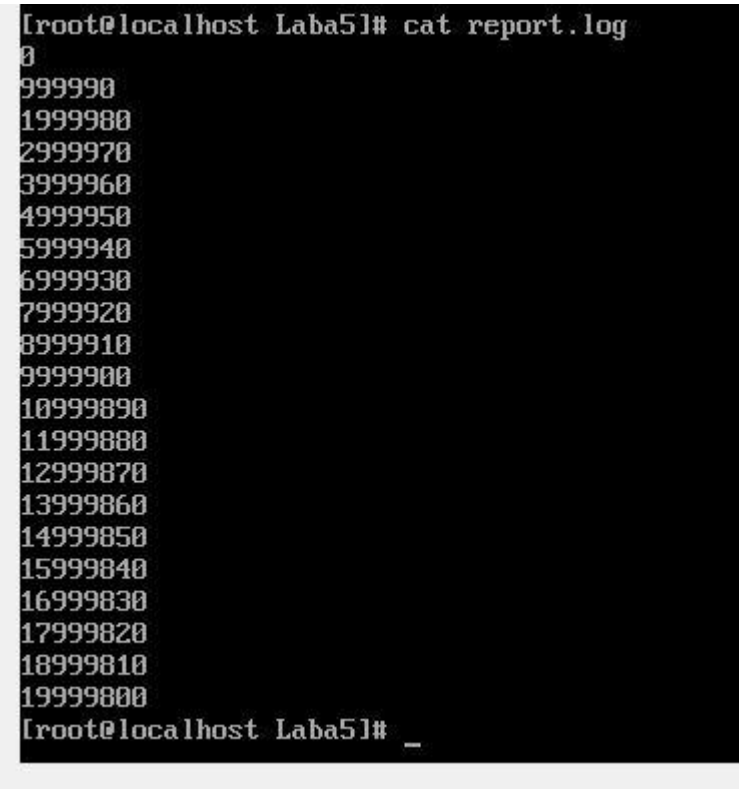
Значения прямо перед остановкой:



Сообщение об аварийном завершении процесса:



Содержание файла report.log:



Подготовьте две консоли. В первой запустите утилиту top. Во второй запустите скрипт и переключитесь на первую консоль. Убедитесь, что в top появился запущенный скрипт. Наблюдайте за следующими значениями (и фиксируйте их изменения во времени в отчете):

значения параметров памяти системы (верхние две строки над основной таблицей);

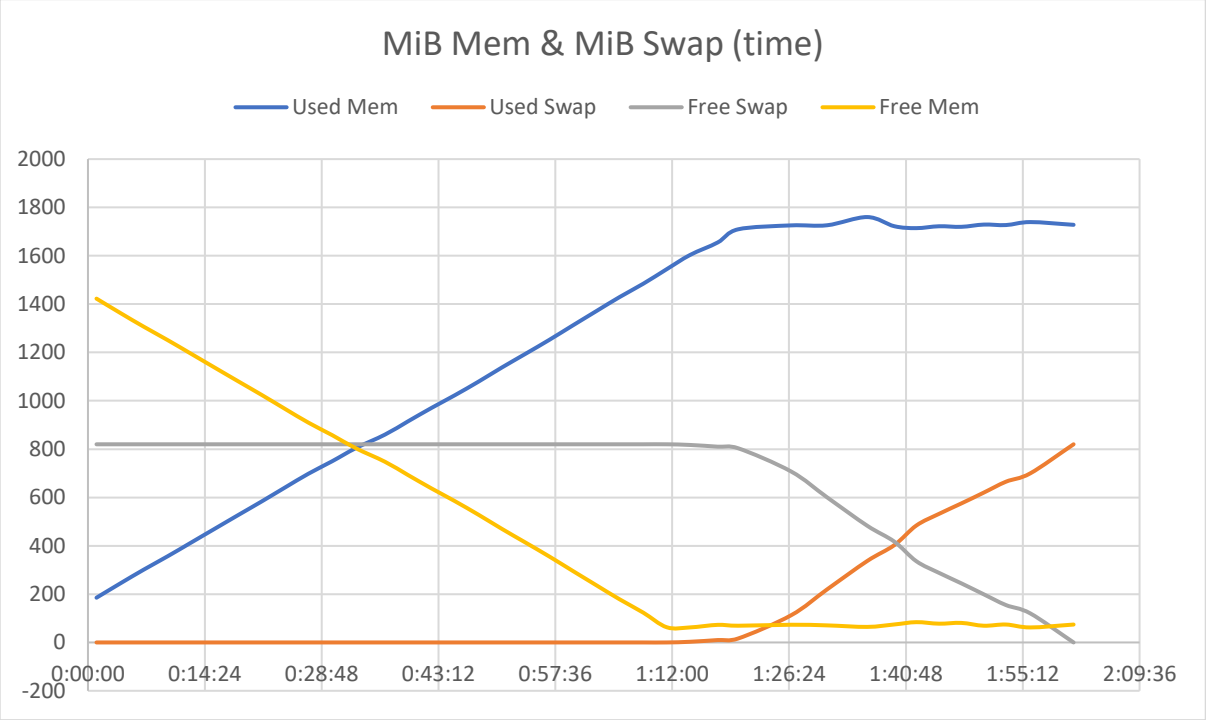
значения параметров в строке таблицы, соответствующей работающему скрипту;

изменения в верхних пяти процессах (как меняется состав и позиции этих процессов).

Проводите наблюдения и фиксируйте их в отчете до аварийной остановки процесса скрипта и его исчезновения из перечня процессов в top.

	MiB Mem				MiB Swap			
Time	total	free	used	buff/cache	total	free	used	avail Mem
0:01:03		1422,6	185,2	219,3		820	0	1490,6
0:03:58		1363,7	244	219,3		820	0	1431,7
0:06:53		1306,3	301,5	219,3		820	0	1374,3
0:09:49		1251,2	356,5	219,3		820	0	1319,2
0:12:40		1195,4	412,4	219,3		820	0	1263,3
0:15:35		1138	469,7	219,3		820	0	1206
0:18:30		1080,8	526,9	219,3		820	0	1148,8
0:21:25		1024,1	583,6	219,3		820	0	1092,1
0:24:18		966,2	641,5	219,3		820	0	1034,2
0:27:11		909,2	698,5	219,3		820	0	977,2
0:30:06		857,8	749,9	219,3		820	0	925,8
0:33:01		803,5	804,3	219,3		820	0	871,5
0:36:36		747,7	860	219,3		820	0	815,7
0:39:25		693,3	914,4	219,3		820	0	761,3
0:42:20		637,4	970,4	219,3		820	0	705,3
0:45:14		584,4	1023,3	219,3		820	0	652,4
0:48:08		528,4	1079,3	219,3		820	0	596,4
0:51:02		469,5	1138,2	219,3		820	0	537,5
0:53:55		413,5	1194,2	219,3		820	0	481,5
0:56:50	1827,1	356,1	1251,6	219,3	820	820	0	424,1
0:59:44		296,3	1311,4	219,3		820	0	364,3
1:02:40		236,4	1371,3	219,3		820	0	304,4
1:05:31		178,6	1429,2	219,3		820	0	246,5
1:08:26		123,4	1484,3	219,3		820	0	191,4
1:11:21		62,7	1545	219,3		820	0	130,7
1:14:12		62,3	1603,3	161,5		817,2	2,8	75,5
1:17:37		72,9	1655,2	98,9		810	10	48,2
1:20:07		69,2	1709,4	48,5		803,7	16,3	20,2
1:26:39		73,3	1725,7	28,1		707,5	112,5	14,4
1:31:06		70,7	1726,3	30,1		600,2	219,8	12,8
1:36:02		64,4	1760,2	32,4		483	337	7,7
1:39:23		74,5	1722,2	30,3		417	403	16,7
1:42:05		84	1714,4	28,7		336,2	483,8	25,4
1:44:53		77,7	1722,1	27,3		287,9	532,1	18,4
1:47:38		80,9	1719,8	26,3		244,7	575,2	21,1
1:50:24		69,2	1729,1	28,8		199,2	620	10,6
1:53:08		74,2	1727,1	25,8		154,7	665,2	14,2
1:56:08		62,2	1739,4	25,5		120,2	699,8	2
2:01:28		74,1	1728,1	24,9		0	820	13,6

Time	VIRT	RES	%Mem
0:01:03	233972	14492	0,8
0:03:58	294164	74684	4
0:06:53	352,772	133292	7,1
0:09:49	409136	189524	10,1
0:12:40	466292	256812	13,2
0:15:35	524768	305156	16,3
0:18:30	583244	363764	19,4
0:21:25	641192	421844	22,5
0:24:18	700328	480980	25,7
0:27:11	758540	539060	28,8
0:30:06	811076	591596	31,6
0:33:01	866648	647300	34,6
0:36:36	923672	704324	37,6
0:39:25	979376	759764	40,6
0:42:20	1036532	817052	43,7
0:45:14	1090652	871172	46,6
0:48:08	1147808	928460	49,6
0:51:02	1200000	908652	52,8
0:53:55	1265288	1,0g	55,9
0:56:50	1324028	1,1g	59
0:59:44	1385012	1,1g	62,3
1:02:40	1446260	1,2g	65,6
1:05:31	1505396	1,2g	68,7
1:08:26	1561760	1,3g	71,8
1:11:21	1623800	1,3g	75,1
1:14:12	1682936	1,4g	78,2
1:17:37	1739960	1,4g	81,3
1:20:07	1800152	1,5g	84,4
1:26:39	1924100	1,6g	89,7
1:31:06	2019932	1,6g	89,2
1:36:02	2123156	1,6g	88,3
1:39:23	2182424	1,6g	87,8
1:42:05	2240504	1,5g	86,5
1:44:53	2297924	1,6g	86,9
1:47:38	2356268	1,6g	87,7
1:50:24	2414744	1,6g	88,3
1:53:08	2468864	1,6g	88,8
1:56:08	2519288	1,6g	89,6
2:01:28	2630168	1,6g	88,9



Посмотрите с помощью команды dmesg | grep "mem.bash" последние две записи о скрипте в системном журнале и зафиксируйте их в отчете. Также зафиксируйте значение в последней строке файла report.log.

```
[root@localhost Laba5]# cat report.log
0
999990
1999980
2999970
3999960
4999950
5999940
6999930
7999920
8999910
9999900
10999890
11999880
12999870
13999860
14999850
15999840
16999830
17999820
18999810
19999800
20999790
```

```
[ 2504.452606] [ 1613]     0  1613    68535       60   155648      126             0 top
[ 2504.452996] [ 1616]     0  1616   541877   410408  3969024    76184             0 sh
[ 2504.453383] Out of memory: Killed process 1616 (sh) total-vm:2167508kB, anon-rss:1640288kB, file-
rss:1344kB, shmem-rss:0kB, UID:0
[ 2504.610482] oom_reaper: reaped process 1616 (sh), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB
[root@localhost Laba5]#
```

Второй этап:

Задача – оценить изменения параметров, выводимых утилитой top в процессе работы нескольких экземпляров созданного скрипта.

Ход эксперимента:

Создайте копию скрипта, созданного на предыдущем этапе, в файл mem2.bash. Настройте её на запись в файл report2.log. Создайте скрипт, который запустит немедленно друг за другом оба скрипта в фоновом режиме.

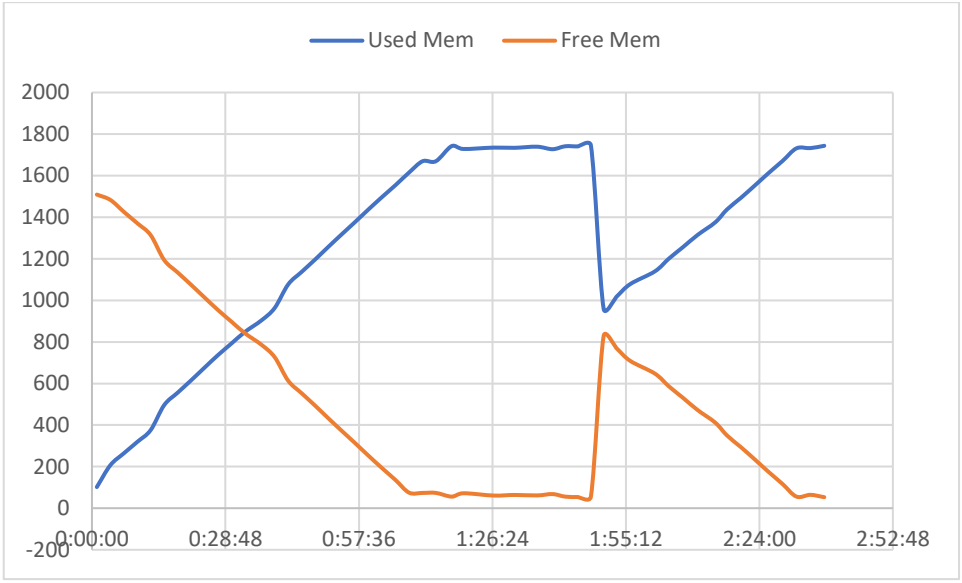
```
#!/bin/bash

sh mem.sh&pid1=$!
sh mem2.sh&pid2=$!

echo "mem.sh pid: "$pid1""
echo "mem2.sh pid: "$pid2""
```

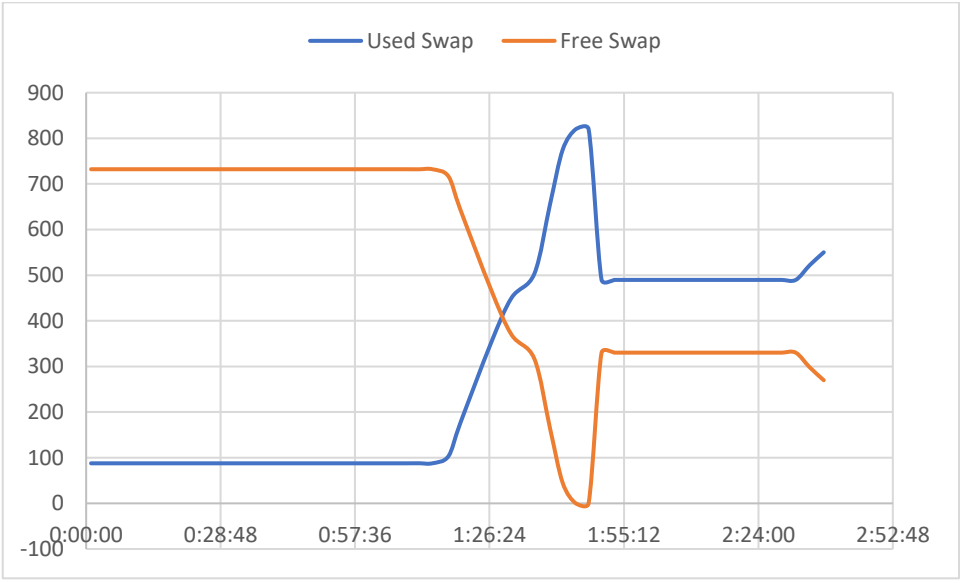
Подготовьте две консоли. В первой запустите утилиту top. Во второй запустите созданный перед этим скрипт и переключитесь на первую консоль. Убедитесь, что в top появились mem.bash и mem2.bash. Наблюдайте за следующими значениями (и фиксируйте их изменения во времени в отчете):

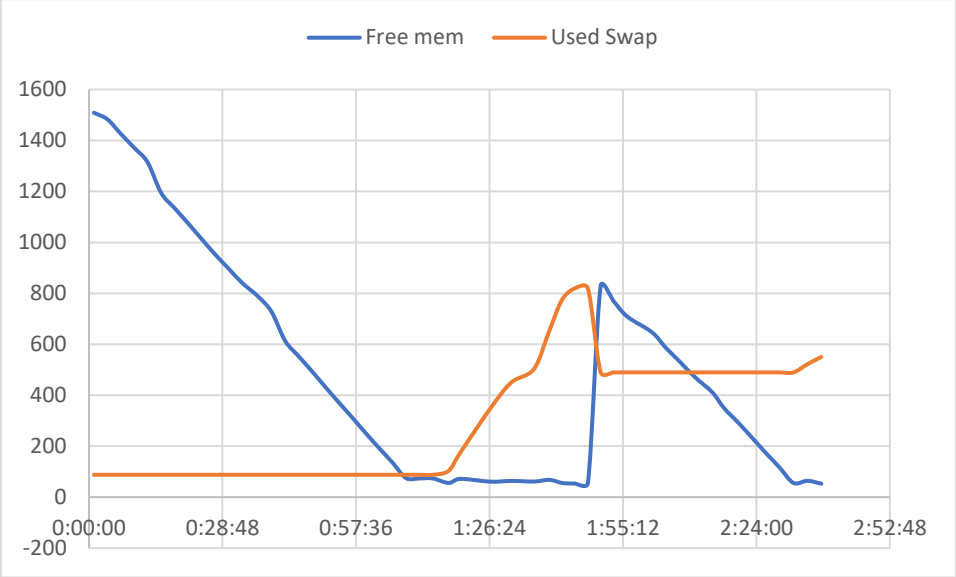
- значения параметров памяти системы (верхние две строки над основной таблицей);
- значения параметров в строке таблицы, соответствующей работающему скрипту;
- изменения в верхних пяти процессах (как меняется состав и позиции этих процессов).

	MiB Mem				MiB Swap					
Time	total	free	used	buff/cache	total	free	used	avail Mem	%Mem1	%Mem2
0:01:03		1508,9	101,8	137		732,2	87,8	1585	0	0
0:03:58		1483,2	206,8	137,1		732,2	87,8	1479,3	3	3
0:06:53		1425,7	264,3	137,1		732,2	87,8	1421,8	4,6	4,6
0:09:49		1370,3	319,7	137,1		732,2	87,8	1366,4	6,1	6,1
0:12:40		1313,7	376	137,1		732,2	87,8	1309,8	7,7	7,7
0:15:35	1827,1	1193,9	496,1	137,1	820	732,2	87,8	1190	7,7	7,7
0:18:30		1133,9	556,1	137,1		732,2	87,8	1130	10,9	10,9
0:21:25		1074,3	615,7	137,1		732,2	87,8	1870,4	12,6	12,6
0:24:18		1013,4	676,6	137,1		732,2	87,8	1009,5	14,2	14,2
0:27:11		953,5	736,5	137,1		732,2	87,8	949,6	15,9	15,9
0:30:06		897,5	792,5	137,1		732,2	87,8	893,6	17,5	17,5

0:33:01		841,6	848,4	137,1		732,2	87,8	837,7	19	19
0:36:36		785,7	904,3	137,1		732,2	87,8	781,8	20,6	20,5
0:39:25		726,7	963,3	137,1		732,2	87,8	722,8	22,1	22,1
0:42:20		613,3	1076,7	137,1		732,2	87,8	609,4	23,7	23,7
0:45:14		553,1	1136,9	137,1		732,2	87,8	549,2	23,7	23,7
0:48:08		493,7	1196,3	137,1		732,2	87,8	489,8	26,8	26,8
0:51:02		431,7	1258,3	137,1		732,2	87,8	427,8	28,4	28,4
0:53:55		371,6	1318,4	137,1		732,2	87,8	367,7	30,1	30,1
0:56:50		311,8	1378,2	137,1		732,2	87,8	307,9	31,8	31,8
0:59:44		250,9	1439,1	137,1		732,2	87,8	247	33,4	33,4
1:02:40		191,2	1498,8	137,1		732,2	87,8	187,3	35	35
1:05:31		135,1	1554,9	137,1		732,2	87,8	131,2	36,7	36,7
1:08:26		74,6	1615,4	137,1		732,2	87,8	70,7	38,3	38,3
1:11:21		73,6	1669,5	84		732,1	87,9	42,3	39,9	39,8
1:14:12		73,6	1669,5	84		732,1	87,9	42,3	41,5	41,5
1:17:37		55,7	1742,2	29,1		716,5	103,5	15,2	43	42,9
1:20:07		71,9	1728,4	26,8		645	175	12,3	43	42,9
1:26:39		60,8	1734,9	31,4		470,5	349,5	3,5	45	45
1:31:06		63,8	1733,9	29,3		370	450	5,5	42,3	47,4
1:36:02		61,5	1739,2	26,3		316,2	503,8	1,7	41,9	47,2
1:39:23		67,9	1727,2	31,9		163,7	656,3	10,9	42	45,5
1:42:05		55,7	1741,3	30		44,7	775,2	15,7	41,8	45,3
1:44:53		53,7	1740,8	32,5		0	820	14,9	46,1	42,5
1:47:38		54,6	1746,3	26,1		0	820	12,6		42,8
1:50:24		828,8	958,9	39,3		330	490	775,4		47
1:53:08		770,4	1016,7	40		330,3	489,7	717,1		50,2
1:56:08		708,8	1078,3	40		330,3	489,7	655,6		53,6
2:01:28		647,3	1139,7	40		330,3	489,7	594,1		56,9
2:04:28		586,2	1200,8	40		330,3	489,7	533		60,3
2:07:30		531,2	1255,9	40		330,3	489,7	477,9		63,3
2:10:40		472,3	1314,7	40		330,3	489,7	419,1		66,5
2:14:30		411,2	1375,9	40		330,3	489,7	358		69,8
2:17:00		350	1437,1	40		330,3	489,7	296,8		73,2
2:20:00		294	1493	40		330,3	489,7	240		76,2
2:23:00		234,6	1552,5	40		330,3	489,7	181,3		79,5
2:26:00		174,1	1613	40		330,3	489,7	120,9		82,8
2:29:00		116,3	1670,8	40		330,3	489,7	63		85,9
2:32:00		55,7	1731,4	40		330,3	489,7	2,5		89,2
2:35:00		64,3	1732,5	30,3		298	522	6,5		89,5
2:38:00		52,9	1743,6	30,5		269,8	550,2	13,2		89,5


Used Mem
Free Mem
2000
1800
1600
1400
1200
1000
800
600
400
200
0
-200
0:00:00
0:28:48
0:57:36
1:26:24
1:55:12
2:24:00
2:52:48


Used Swap
Free Swap
900
800
700
600
500
400
300
200
100
0
-100
0:00:00
0:28:48
0:57:36
1:26:24
1:55:12
2:24:00
2:52:48



Проводите наблюдения и фиксируйте их в отчете до аварийной остановки последнего из двух скриптов и их исчезновения из перечня процессов в top. Посмотрите с помощью команды dmesg | grep "mem[2]*.bash" последние записи о скриптах в системном журнале и зафиксируйте их в отчете. Также зафиксируйте значения в последних строках файлов report.log и report2.log.

```
[root@localhost Laba5]# cat report.log
0
999990
1999980
2999970
3999960
4999950
5999940
6999930
7999920
8999910
9999900
10999890
11999880
12999870
13999860
14999850
15999840
16999830
17999820
18999810
19999800
20999790
21999780
22999770
23999760
24999750
25999740
26999730
27999720
28999710
29999700
```

```
[root@localhost Laba5]# cat report2.log
0
999990
1999980
2999970
3999960
4999950
5999940
6999930
7999920
8999910
9999900
10999890
11999880
12999870
13999860
14999850
[root@localhost Laba5]# _
```

Обработка результатов:

По графику видно, что память из подкачки (swap) берется только тогда, когда объем свободной оперативной памяти (RAM) критично мал. T e swap можно назвать «экстренной памятью».

В случае, когда мы запустили 2 цикла одновременно, один из них был убит OOM Killer'ом раньше, чем второй из-за того, что память просто на просто закончилась. После того, как один из циклов аварийно завершился, использованная под него память снова оказалась доступной и поэтому второй скрипт проработал несколько дольше.

Эксперимент No2

Подготовительный этап:

Создайте копию скрипта mem.bash в файл newmem.bash. Измените копию таким образом, чтобы она завершала работу, как только размер создаваемого массива превысит значение N, передаваемое в качестве параметра скрипту. Уберите запись данных в файл.

```
#!/bin/bash

N=$(( $(cat report.log | tail -n 1) / 10 ))

K=10

while (( $K > 0 )); do
        sh newmem.sh $N
        K=$(( K - 1 ))
        sleep 1
done
```

Основной этап:

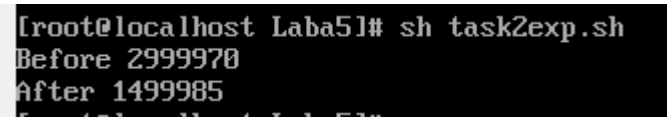
Задача – определить граничные значения потребления памяти, обеспечивающие безаварийную работу для регулярных процессов, запускающихся с заданной интенсивностью.

Ход эксперимента:

Создайте скрипт, который будет запускать newmem.bash каждую секунду, используя один и тот же параметр N так, что всего будет осуществлено K запусков. Возьмите в качестве значения N, величину, в 10 раз меньшую, чем размер массива, при котором происходила аварийная остановка процесса в первом этапе предыдущего эксперимента. Возьмите в качестве K значение 10. Убедитесь, что все K запусков

успешно завершились, и в системном журнале нет записей об аварийной остановке newmem.bash.

Измените значение K на 30 и снова запустите скрипт. Объясните, почему ряд процессов завершился аварийно. Подберите такое максимальное значение N, чтобы при K=30 не происходило аварийных завершений процессов. Укажите в отчете сформулированные выводы по этому эксперименту и найденное значение N.



Для вычисления N, значения последовательно менялись путем изменения знаменателя дроби в скрипте.

Вывод:

1. При использовании оперативной памяти > 88% (при должных параметрах OOM Killer) процесс аварийно завершается.
2. Swap используется для расширения RAM в критических ситуациях.
3. При запуске нескольких процессов, одинаково требовательных к ресурсам, эти самые ресурсы делятся пополам. (поэтому в 1 эксперименте один из скриптов упал на 46%)