Эксперимент 1

Подготовительный этап:

Создать скрипт mem.bash, реализующий 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).

```
 GNU nano 2.9.8                                  mem.bash

#!/bin/bash
arr=()
counter=0
elements=(1 2 3 4 5 6 7 8 9 10)
echo "" > report.log

while true
do
arr+=(${elements[@]})
let counter++
if [[ $counter == 100000 ]]
then
counter=0
echo "${#arr[@]}" >> report.log
fi
done
```

Первый этап:

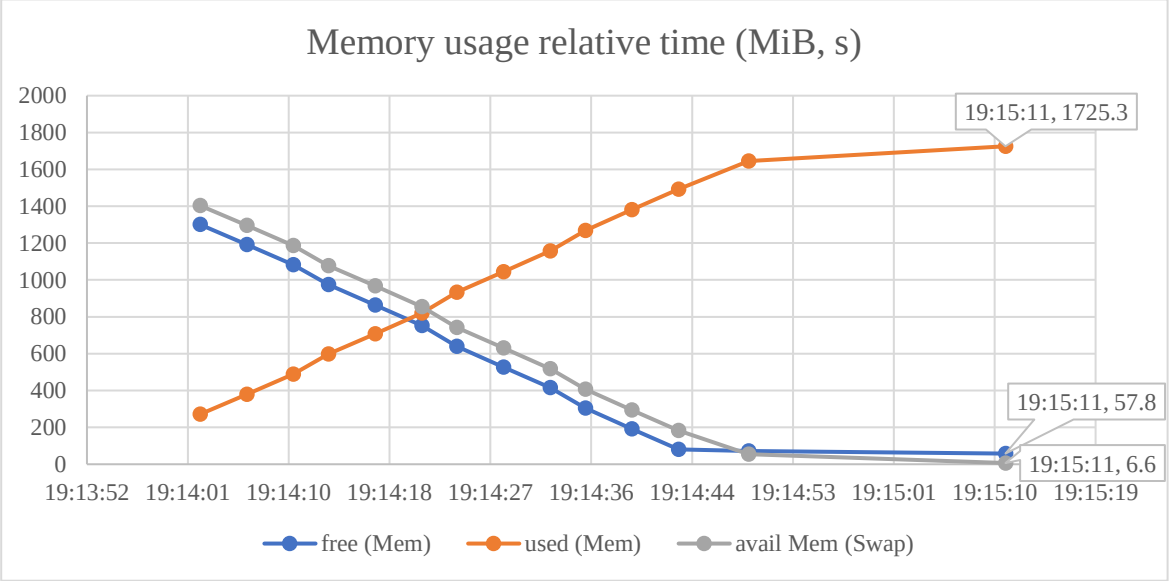
1. Запустить созданный скрипт mem.bash.
2. Дождать аварийной остановки процесса и вывода в консоль последних сообщений системного журнала.

```
[ 1170.498584] Out of memory: Killed process 1893 (mem.bash) total-vm:2608784kB, anon-rss:1600308kB
 file-rss:0kB, shmem-rss:0kB, UID:1000
[ 1170.625798] oom_reaper: reaped process 1893 (mem.bash), now anon-rss:0kB, file-rss:0kB, shmem-rs
:0kB
Killed
[user@localhost task1]$ _
```

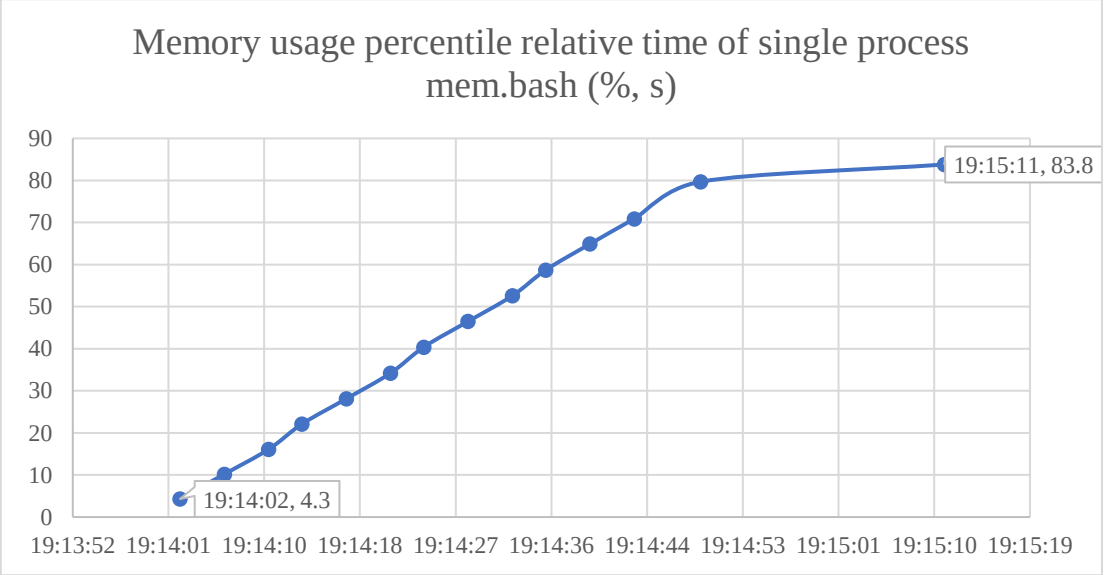
3. Зафиксировать значение в последней строке файла report.log

```
 GNU nano 2.9.8                                 report.log

1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
10000000
11000000
12000000
13000000
14000000
15000000
16000000
17000000
18000000
```

4. Изменения значения параметров памяти системы значения параметров в строке таблицы, соответствующей работающему скрипту; изменения в верхних пяти процессах во времени.



TIME	MiB Mem				MiB Swap			
	total	free	used	buff/cache	total	free	used	avail Mem
19:14:02	1827.1	1300.9	270.9	255.2	820	820	0	1404
19:14:06	1827.1	1192.1	379.7	255.2	820	820	0	1295.2
19:14:10	1827.1	1083.2	488.6	255.2	820	820	0	1186.4
19:14:13	1827.1	974	597.8	255.2	820	820	0	1077.2
19:14:17	1827.1	864	707.8	255.2	820	820	0	967.1
19:14:21	1827.1	752.3	819.5	255.2	820	820	0	855.4
19:14:24	1827.1	639.5	932.3	255.2	820	820	0	742.7
19:14:28	1827.1	527.3	1044.5	255.2	820	820	0	630.5
19:14:32	1827.1	415.4	1156.4	255.2	820	820	0	518.5
19:14:35	1827.1	303.8	1268	255.2	820	820	0	407
19:14:39	1827.1	191.4	1380.4	255.3	820	820	0	294.6
19:14:43	1827.1	80.1	1491.6	255.3	820	820	0	183.3
19:14:49	1827.1	72.2	1644.9	109.9	820	785.5	34.5	53.7
19:15:11	1827.1	57.8	1725.3	44	820	471.7	348.3	6.6



TIME	mem.bash process		
	VIRT	RES	%MEM
19:14:02	299180	79784	4.3
19:14:06	410324	190928	10.2
19:14:10	521600	302072	16.1
19:14:13	633140	413744	22.1
19:14:17	745604	526208	28.1
19:14:21	859784	640256	34.2
19:14:24	975020	755624	40.4
19:14:28	1089596	870200	46.5
19:14:32	1204040	984776	52.6
19:14:35	1318088	1.0g	58.7
19:14:39	1432928	1.2g	64.9
19:14:43	1546712	1.3g	70.9
19:14:49	1729928	1.4g	79.7
19:15:11	2117480	1.5g	83.8

5. Смотреть с помощью команды dmesg | grep "mem.bash" последние две записи о скрипте в системном журнале и зафиксировать их в отчете.

```
user@localhost task1]$ dmesg | grep "mem.bash"
 1170.497268] [ 1893]  1000  1893   652196   400077  4861952   196546             0 mem.bash
 1170.498584] Out of memory: Killed process 1893 (mem.bash) total-vm:2608784kB, anon-rss:1600308kB,
file-rss:0kB, shmem-rss:0kB, UID:1000
 1170.625798] oom_reaper: reaped process 1893 (mem.bash), now anon-rss:0kB, file-rss:0kB, shmem-rss
0kB
user@localhost task1]$ _
```

Вывод 1-го этапа:

В каждом процессе используется память. На графике мы могли видеть, что объем доступной и свободной памяти уменьшается обратно пропорционально количеству используемой памяти. В определенный момент объем памяти достигает предела, и процесс завершается. Скрипт завершился сбоем в точке цикла 18000000. На 1-м этапе лабораторной работы мы обнаружили, что скрипт начал аварийно завершаться, когда они превысили 80% памяти из таблицы top.

Второй этап

1. Создать копию скрипта, созданного на предыдущем этапе, в файл mem2.bash. Настроить её на запись в файл report2.log.

```
 GNU nano 2.9.8                    mem2.bash

#!/bin/bash
#declare -A arr
arr=()
counter=0
elements=(1 2 3 4 5)
rm report.log
echo "" > report.log

while [[ true ]]
do
arr+=(${elements[@]})
let counter++
if [[ $counter == 100000 ]]
then
counter=0
echo "${#arr[@]}" >> report2.log
fi
done
```

2. Зафиксируйте значения в последних строках файлов report.log и report2.log.

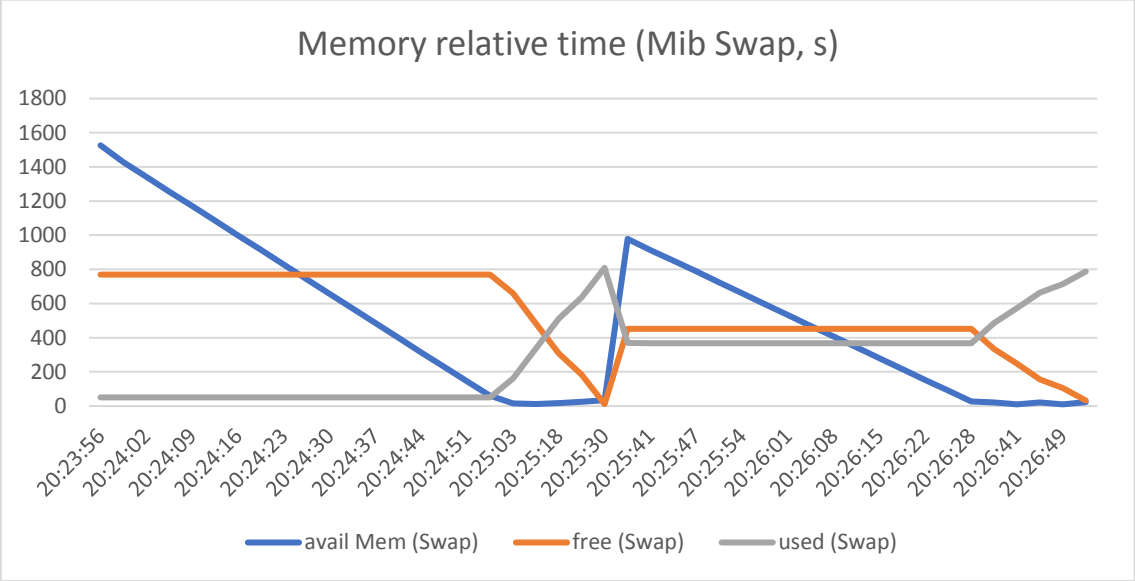
```
 GNU nano 2.9.8                    report2.log

16500000
17000000
17500000
18000000
18500000
19000000
19500000
20000000
20500000
21000000
21500000
22000000
22500000
23000000
23500000
24000000
24500000
25000000
25500000
26000000
26500000
27000000
27500000
28000000
28500000
29000000
29500000
30000000
30500000
```

```
 GNU nano 2.9.8                    report2.log

500000
1000000
1500000
2000000
2500000
3000000
3500000
4000000
4500000
5000000
5500000
6000000
6500000
7000000
7500000
8000000
8500000
9000000
9500000
10000000
10500000
11000000
11500000
12000000
12500000
13000000
13500000
14000000
14500000
15000000
15500000
16000000
```

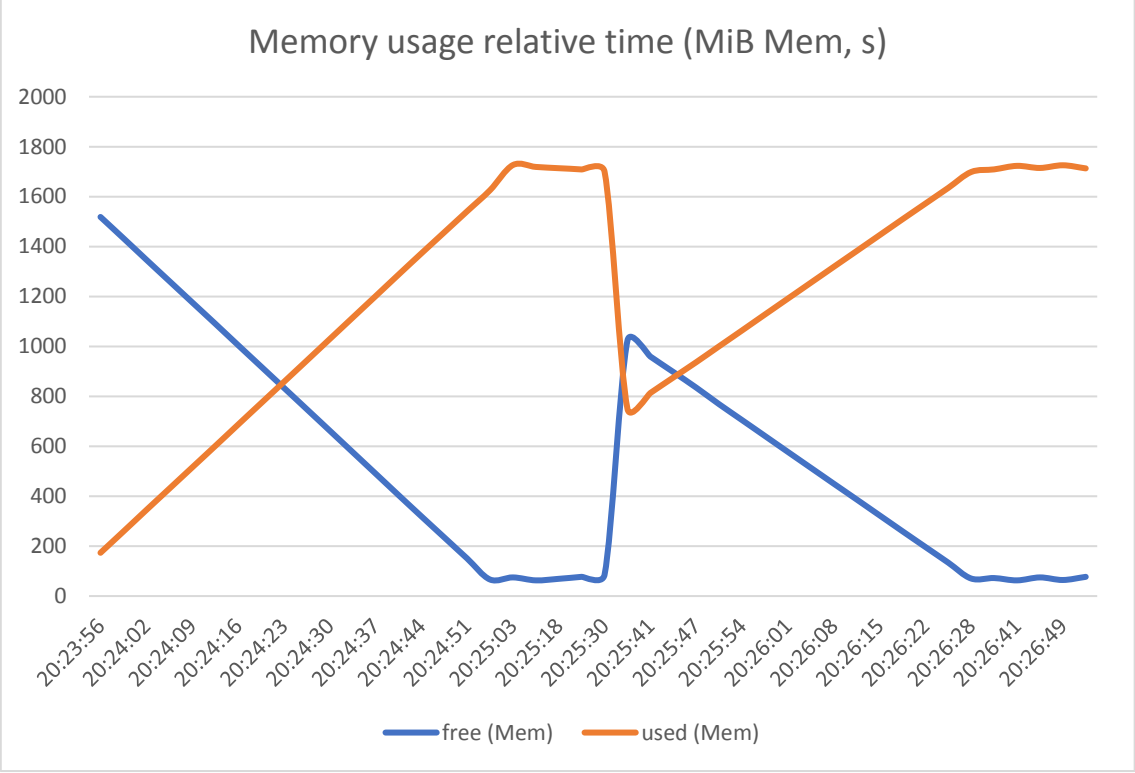
3. Изменения значения параметров памяти системы; значения параметров в строке таблицы, соответствующей работающему скрипту; изменения в верхних пяти процесса во времени в отчете

TIME	MiB Mem				MiB Swap			
	total	free	used	buff/cache	total	free	used	avail Mem
20:23:56	1827.1	1518.5	173.4	135.1	820	769	50.5	1526.3
20:23:59	1827.1	1433.4	258.6	135.1	820	769	50.5	1427.4
20:24:02	1827.1	1347.4	344.6	135.1	820	769	50.5	1341.3
20:24:06	1827.1	1262.1	429.8	135.1	820	769	50.5	1256.1
20:24:09	1827.1	1176.8	515.1	135.1	820	769	50.5	1170.8
20:24:13	1827.1	1091.7	600.3	135.1	820	769	50.5	1085.6
20:24:16	1827.1	1005.6	686.3	135.1	820	769	50.5	999.6
20:24:20	1827.1	920.1	771.9	135.1	820	769	50.5	914
20:24:23	1827.1	834.6	857.3	135.1	820	769	50.5	828.6
20:24:27	1827.1	748.9	943	135.1	820	769	50.5	742.9
20:24:30	1827.1	663.1	1028.9	135.1	820	769	50.5	657
20:24:33	1827.1	577	1114.9	135.1	820	769	50.5	571
20:24:37	1827.1	491.2	1200.7	135.1	820	769	50.5	485.2
20:24:40	1827.1	405.1	1286.9	135.1	820	769	50.5	399
20:24:44	1827.1	319.8	1372.2	135.1	820	769	50.5	313.7
20:24:47	1827.1	235.3	1456.6	135.1	820	769	50.5	229.3
20:24:51	1827.1	150.7	1541.2	135.1	820	769	50.5	144.7
20:24:54	1827.1	66.4	1625.6	135.1	820	769	50.5	60.3
20:25:03	1827.1	74.8	1726.5	25.7	820	659	160.7	14.7
20:25:11	1827.1	62.9	1718.9	45.3	820	484	336.5	12.6
20:25:18	1827.1	69.1	1713.8	44.2	820	307	512.7	18.2
20:25:23	1827.1	77.2	1708.8	41	820	184	635.9	24.7
20:25:30	1827.1	85.7	1699.6	41.8	820	11.2	808.8	33.6
20:25:37	1827.1	1023.8	749.8	53.5	820	451	368.9	977.4
20:25:41	1827.1	959.3	813.6	54.1	820	452	368.5	913.1
20:25:44	1827.1	897.6	875.4	54.1	820	452	368.5	851.4
20:25:47	1827.1	835	936.9	55.1	820	452	368.3	789.3
20:25:51	1827.1	768.4	1000.6	58	820	452	367.7	724.1
20:25:54	1827.1	705	1064	58	820	452	367.7	660.7
20:25:58	1827.1	641.2	1127.9	58	820	452	367.7	596.9
20:26:01	1827.1	577.5	1191.5	58	820	452	367.7	533.2
20:26:04	1827.1	513.9	1255	58.1	820	452	367.6	469.7
20:26:08	1827.1	450.4	1318.6	58.1	820	452	367.6	406.2
20:26:11	1827.1	387.1	1381.9	58.1	820	452	367.6	342.8
20:26:15	1827.1	323.5	1445.5	58.1	820	452	367.6	279.2
20:26:18	1827.1	259.8	1509.1	58.2	820	452	367.6	215.6
20:26:22	1827.1	196.8	1572	58.2	820	452	367.6	152.6
20:26:25	1827.1	133.9	1635	58.2	820	452	367.6	89.7
20:26:28	1827.1	70.3	1698.6	58.2	820	452	367.6	26.1
20:26:35	1827.1	72.2	1709.1	45.8	820	335	485.3	21.8
20:26:41	1827.1	62.9	1723.2	40.9	820	247	573.2	10.2
20:26:45	1827.1	74.8	1714.8	37.4	820	156	664.3	20.3
20:26:49	1827.1	64.5	1725.5	37.1	820	106	714.2	9.8
20:26:53	1827.1	77.1	1713.1	36.9	820	32.5	787.5	22.4

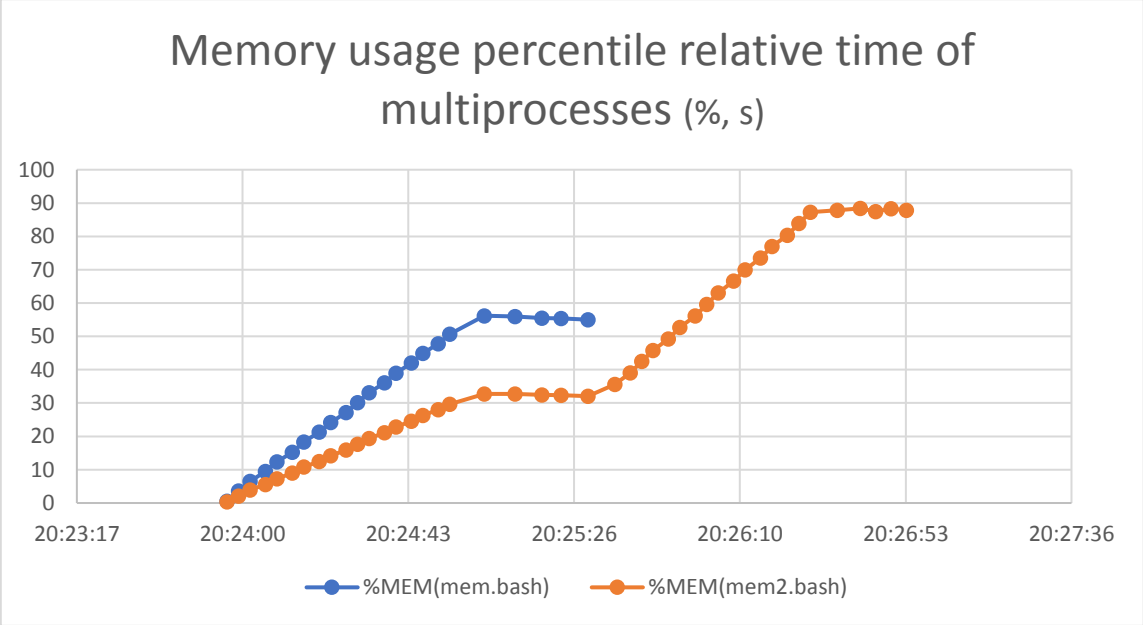
Граф 1 из таблицы MiB Mem



Граф 2 из таблицы MiB Swap



TIME	mem.bash process			mem.bash2 process		
	VIRT	RES	%MEM	VIRT	RES	%MEM
20:23:56	231200	11620	0.6	227240	7920	0.4
20:23:59	286112	66532	3.6	259316	39864	2.1
20:24:02	341552	121972	6.5	291788	72336	3.9
20:24:06	396728	177148	9.5	323732	104200	5.6
20:24:09	451640	232060	12.4	355940	136488	7.3
20:24:13	506420	286972	15.3	388148	168960	9
20:24:16	561860	342412	18.3	420620	201432	10.8
20:24:20	617168	397852	21.3	452828	233376	12.5
20:24:23	672476	453028	24.2	484722	265584	14.2
20:24:27	727652	508204	27.2	517244	297792	15.9
20:24:30	783092	563644	30.1	549584	330264	17.7
20:24:33	838664	619084	33.1	581924	363736	19.4
20:24:37	893972	674524	36.1	616264	394944	21.1
20:24:40	949544	730228	39	646736	427416	22.8
20:24:44	1004588	785140	42	678944	459624	24.6
20:24:47	1059104	839788	44.9	710624	491304	26.3
20:24:51	1113488	894172	47.8	742700	523248	28
20:24:54	1168004	948556	50.7	774512	555192	29.7
20:25:03	1310960	1g	56.2	855296	612192	32.7
20:25:11	1417220	1g	56	919052	611144	32.7
20:25:18	1523480	1g	55.5	980960	606164	32.4
20:25:23	1600436	1g	55.4	1026368	604864	32.3
20:25:30	1074716	1g	55	1087880	601196	32.1
20:25:37				1157312	666560	35.6
20:25:41				1222520	731768	39.1
20:25:44				1285748	794864	42.5
20:25:47				1388448	857432	45.8
20:25:51				1411676	920792	49.2
20:25:54				1476488	985472	52.7
20:25:58				1541696	1g	56.2
20:26:01				1606772	1.1g	59.6
20:26:04				1671452	1.1g	63.1
20:26:08				1736396	1.2g	66.6
20:26:11				1801208	1.2g	70
20:26:15				1866152	1.3g	73.5
20:26:18				1931096	1.4g	77
20:26:22				1995512	1.4g	80.4
20:26:25				2059796	1.5g	83.9
20:26:28				2124872	1.6g	87.3
20:26:35				2254496	1.6g	87.8
20:26:41				2354948	1.6g	88.4
20:26:45				2431244	1.6g	87.5
20:26:49				2497640	1.6g	88.3
20:26:53				2453508	1.6g	87.8



4. Посмотреть с помощью команды dmesg | grep "mem[2]*.bash" последние записи о скриптах в системном журнале и зафиксируйте их в отчете.

```
[user@localhost task2]$ dmesg | grep "mem[2]*.bash"
[ 1170.497268] [ 1893]  1000  1893   652196   400077  4861952   196546             0 mem.bash
[ 1170.498584] Out of memory: Killed process 1893 (mem.bash) total-vm:2608784kB, anon-rss:1600308kB,
 file-rss:0kB, shmem-rss:0kB, UID:1000
[ 1170.625798] oom_reaper: reaped process 1893 (mem.bash), now anon-rss:0kB, file-rss:0kB, shmem-rss
:0kB
[ 5339.909814] mem.bash invoked oom-killer: gfp_mask=0x6280ca(GFP_HIGHUSER_MOVABLE|__GFP_ZERO), node
mask=(null), order=0, oom_score_adj=0
[ 5339.911338] mem.bash cpuset=/ mems_allowed=0
[ 5339.911841] CPU: 0 PID: 2099 Comm: mem.bash Kdump: loaded Tainted: G                ---------r-
 - 4.18.0-193.el8.x86_64 #1
[ 5339.986142] [ 2099]  1000  2099   436805   265905  3133440   115414             0 mem.bash
[ 5339.986593] [ 2100]  1000  2100   278306   155492  1855488    67338             0 mem2.bash
[ 5339.987490] Out of memory: Killed process 2099 (mem.bash) total-vm:1747220kB, anon-rss:1063052kB,
 file-rss:568kB, shmem-rss:0kB, UID:1000
[ 5427.597404] [ 2100]  1000  2100   657443   418654  4898816   183219             0 mem2.bash
[ 5427.598709] Out of memory: Killed process 2100 (mem2.bash) total-vm:2629772kB, anon-rss:1674616kB
, file-rss:0kB, shmem-rss:0kB, UID:1000
```

Вывод 2-го этапа:

Для нескольких процессов мы могли видеть, что объем Swap памяти увеличивается, когда RAM достигает самой низкой точки на графике. Swap и свободная память меняются на каждом шагу. График ясно показывает эту демонстрацию, где мы могли видеть, что объем памяти Swap увеличивается, когда объем оперативной памяти уменьшается. Это может означать, что Swap помогает машине освободить дополнительную память. Из графика видно, что два скрипта завершились в разное время. Когда mem.bash закончился, доступная и свободная память значительно увеличилась. По прошествии определенного времени объем оперативной памяти достиг самой низкой точки, а использованная память достигла своего второго пика, который является концом 2-го процесса.

Эксперимент 2

Подготовительный этап: Создать копию скрипта mem.bash в файл newmem.bash. Изменить копию таким образом, чтобы она завершала работу, как только размер создаваемого массива превысит значение N, передаваемое в качестве параметра скрипту.

```
 GNU nano 2.9.8                                  newmem.bash

#!/bin/bash

arr=()
elements=(1 2 3 4 5 6 7 8 9 10)
N=$1
while true
do
arr+=(${elements[@]})
if [[ "${#arr[@]}" -ge $N ]]
then
exit 0
fi
done
```

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

1. N - 10 раз меньшую, чем размер массива, при котором происходила аварийная остановка процесса в первом этапе предыдущего эксперименти. N = 1 800 000, K = 10.

```
 GNU nano 2.9.8                                  startnew.sh

#!/bin/bash
N="$(cat report3.log | tail -n 1)"
N=$((N/10))
echo $N
K=10
for (( i=0; i < K; i++ ))
do
./newmem.bash $N &
echo "$((i+1))" >> start
sleep 1
done
```

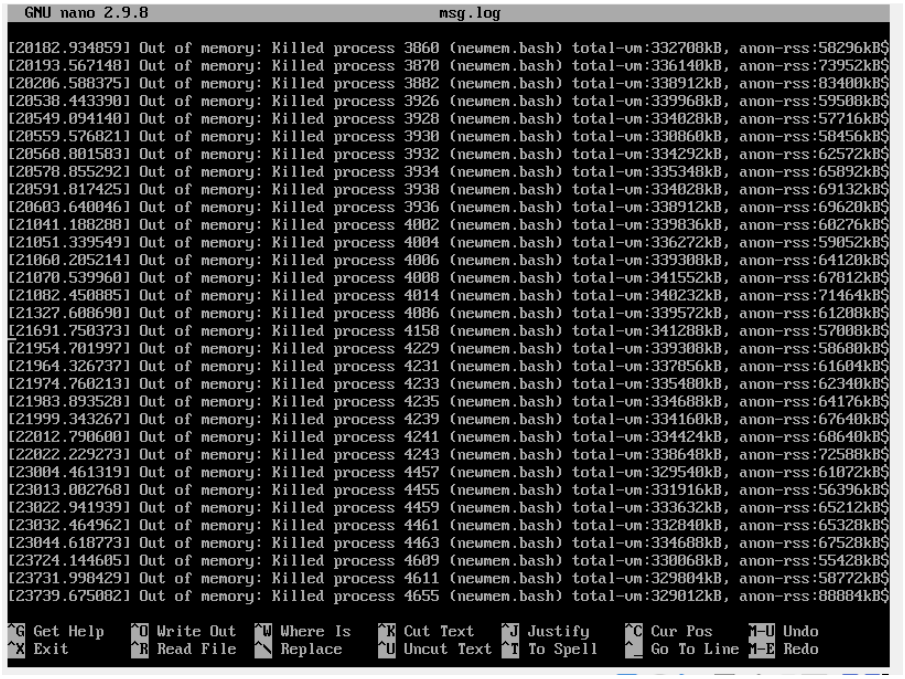
2. K = 30 N = 1 800 000

```
 GNU nano 2.9.8                                  startnew.sh

#!/bin/bash
N="$(cat report3.log | tail -n 1)"
N=$((N/10))
echo $N
K=30
for (( i=0; i < K; i++ ))
do
./newmem.bash $N &
echo "$((i+1))" >> start
sleep 1
done
```

3. K = 30. Указать в отчете сформулированные выводы по этому эксперименту и найденное значение N , чтобы при K=30 не происходило аварийных завершений процессов. K = 30, N = 1530507

```
 GNU nano 2.9.8                                  startnew2.sh

#!/bin/bash
K=30
N=1530507
for (( i=0; i < K; i++ ))
do
./newmem.bash $N &
echo "$((i+1))" >> start2
sleep 1
done
```

4. Посмотреть с помощью команды dmesg | grep "newmem.bash" последние записи о скриптах в системном журнале и зафиксируйте их в отчете.



Вывод:

1. При превышении 80% оперативной памяти процессы начинают аварийно завершаться.

2. Пространство подкачки в Linux используется, когда объем физической памяти (RAM) заполнен.

3. В нескольких процессах оперативная память используется пропорционально, и время окончания каждого процесса может варьироваться.

4. В зависимости от виртуальной машины ограничение оперативной памяти может варьироваться.