


**Министерство науки и высшего образования Российской Федерации**  
федеральное государственное автономное образовательное учреждение  
высшего образования  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

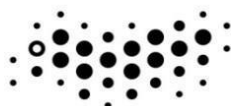
## **Отчёт**

По лабораторной работе №5. Управление памятью в ОС Linux  
по дисциплине «Операционные системы»

Автор: Андреев Артём Русланович

Группа: М3200

Подпись: \_\_\_\_\_  




**УНИВЕРСИТЕТ ИТМО**

Санкт-Петербург 2021

### Исходное состояние:

- Общий объём оперативной памяти: 1024 Мб
- Общий объём раздела подкачки: 820 Мб
- Размер страницы виртуальной памяти: 4 Кб
- Объём свободной физической памяти в ненагруженной системе: 637,8 Мб
- Объём свободного пространства в разделе подкачки в ненагруженной системе: 764,4 Мб

### Эксперимент #1:

Скрипт mem.bash

```
GNU nano 2.9.8 mem.bash

#!/bin/bash

declare -a array

counter=0
echo "" > report.log

while true;
do
    array+=(1 2 3 4 5 6 7 8 9 10)
    let counter++
    if [[ $counter -eq 1000000 ]]
    then
        counter=0
        echo "${#array[@]}" >> report.log
    fi
done
```

Последнее значение в файле report.log: 18000000

В самом начале после запуска mem.bash процесс mem.bash попал на первую строчку top и остался там до своей смерти.

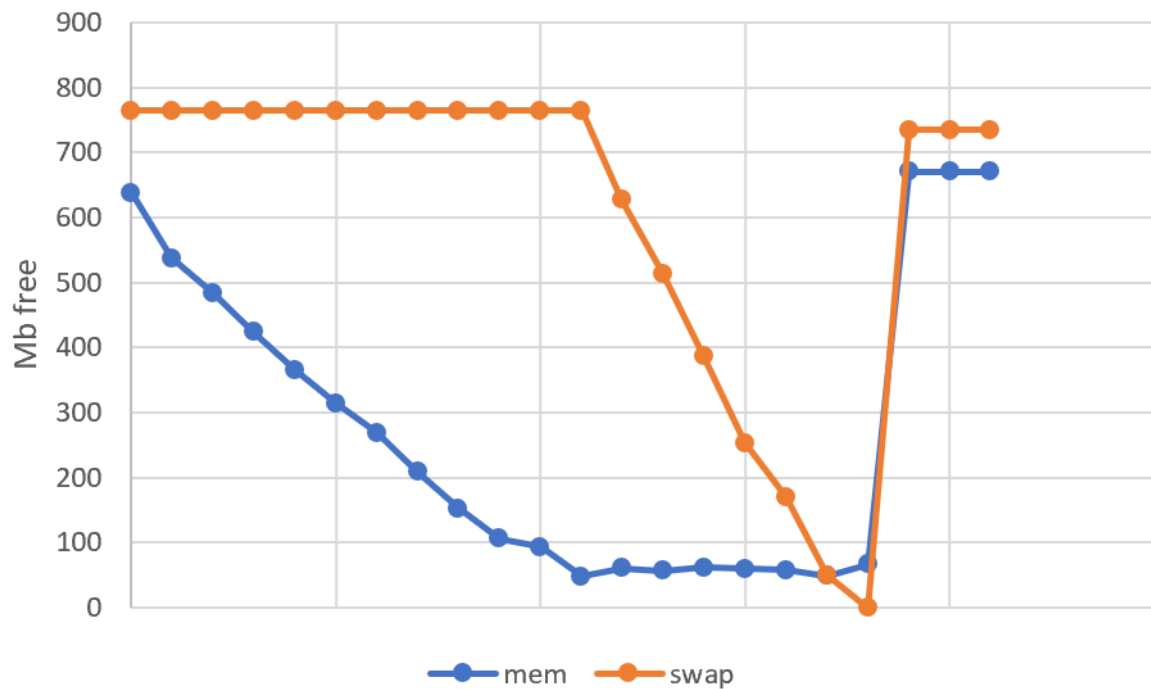
В первой пятёрке процессов в top также оказался top\_parser – скрипт, который записывает показатели top во времени и фиксирует значения в файл.

Спустя некоторое время, когда осталось мало свободной оперативной памяти, появляется процесс kswarpd0, процесс-демон, отвечающий за управление разделом подкачки.

```
GNU nano 2.9.8                                top_parser
#!/bin/bash
echo "" > top5_data
echo "" > membash_data
echo "" > mem_data
echo "" > swap_data

while true;
do
    sec=$(date +%s)
    top -bn1 > .tmp
    cat .tmp | sed -n '4p' | awk -v s="$sec" '{print s,$6}' >> mem_data
    cat .tmp | sed -n '5p' | awk -v s="$sec" '{print s,$5}' >> swap_data
    cat .tmp | head -12 | tail -5 >> top5_data
    echo "-----" >> top5_data
    cat .tmp | grep "mem[2]*.bash" >> membash_data
done
```

По показателям свободной оперативной памяти и файлом подкачки во времени, был построен график:



Количество свободной оперативной памяти уменьшалось линейно. В момент, когда свободное места в оперативной памяти стало слишком мало, линейно начал задействоваться раздел подкачки. В момент, когда раздел подкачки оказался полностью забитым, процесс mem.bash был убит, а раздел подкачки очищен.

Последние две записи в системном журнале (dmesg) о процессе mem.bash:

```
[ 234.465983] Out of memory: Killed process 1491 (mem.bash) total-vm:1645712kB, anon-rss:671416kB,
file-rss:0kB, shmem-rss:0kB, UID:1003
[ 234.568026] oom_reaper: reaped process 1491 (mem.bash), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB
```

#### Скрипт mem2.bash

```
GNU nano 2.9.8 mem2.bash

#!/bin/bash

declare -a array

counter=0
echo "" > report2.log

while true;
do
    array+=(1 2 3 4 5 6 7 8 9 10)
    let counter++
    if [[ $counter -eq 1000000 ]]
    then
        counter=0
        echo "${#array[@]}" >> report2.log
    fi
done
```

В самом начале после запуска mem.bash и mem2.bash оба процесса попали на первые две строчки top и оставались там до своей смерти.

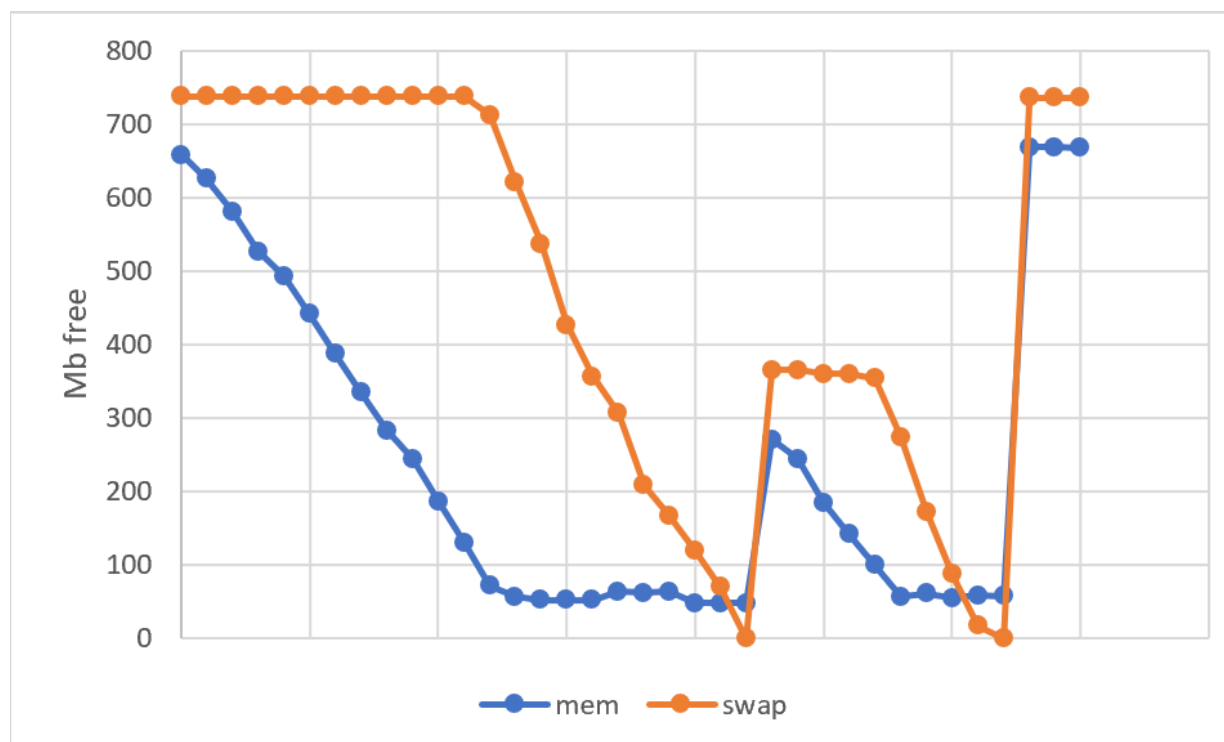
Сначала был завершён процесс mem2.bash, позже через некоторое время завершился mem.bash.

Последнее значение в файле report.log: 18000000

Последнее значение в файле report2.log: 9000000

В первой пятёрке процессов в top также оказался top\_parser. Спустя некоторое время, когда осталось мало свободной оперативной памяти, появляется процесс kswarpd0, процесс-демон, отвечающий за управление разделом подкачки.

По показателям свободной оперативной памяти и файлом подкачки во времени, был построен график:



Количество свободной оперативной памяти уменьшалось линейно, как в случае с одним процессом. В момент, когда свободное места в оперативной памяти стало слишком мало, линейно начал задействоваться раздел подкачки. В момент, когда раздел подкачки оказался полностью забитым, процесс mem2.bash был убит, в результате чего освободилось примерно половина оперативной памяти и половина раздела подкачки. Для процесса mem.bash всё повторилось как со случаем для одного процесса.

Последние записи в системном журнале (dmesg) о процессах mem.bash и mem2.bash:

```
[ 2410.685692] Out of memory: Killed process 1479 (mem2.bash) total-vm:941360kB, anon-rss:339532kB,
file-rss:0kB, shmem-rss:0kB, UID:1003
[ 2410.715332] oom_reaper: reaped process 1479 (mem2.bash), now anon-rss:0kB, file-rss:0kB, shmem-rs
s:0kB
[ 2440.022823] [ 1478] 1003 1478 413309 168413 2949120 189334 0 mem.bash
[ 2440.023690] Out of memory: Killed process 1478 (mem.bash) total-vm:1653236kB, anon-rss:673648kB,
file-rss:4kB, shmem-rss:0kB, UID:1003
[ 2440.113612] oom_reaper: reaped process 1478 (mem.bash), now anon-rss:0kB, file-rss:0kB, shmem-rss
:0kB
```

## Эксперимент #2:

### Скрипт newmem.bash

```
GNU nano 2.9.8 newmem.bash

#!/bin/bash

declare -a array

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

### Скрипт run\_k для запуска k процессов newmem.bash каждую секунду, выделяющих до n элементов для своего массива

```
CentOS8 [Работает] - Oracle VM VirtualBox
GNU nano 2.9.8 run_k

#!/bin/bash

counter=1

while [[ $counter -le $1 ]]
do
    ./newmem.bash $2 &
    sleep 1
    let counter++
done
```

При запуске run\_k с параметрами: k=10 и n=1800000 (в 10 раз меньше, чем максимально полученное значение для mem.bash в предыдущем эксперименте) все процессы были завершены успешно.

При запуске run\_k с параметрами: k=30 и n=1800000 некоторые процессы были убиты из-за нехватки памяти.

При запуске run\_k с параметрами: k=30 и n=1700000 все процессы были завершены успешно.

При запуске run\_k с параметрами: k=30 и n=1750000 некоторые процессы были убиты из-за нехватки памяти.

При запуске run\_k с параметрами: k=30 и n=1740000 все процессы были завершены успешно.

Следовательно, для k=30 оптимальное n находится между 1740000 и 1750000.

Значение 1740000 достаточно близко к 1800000, хотя процессов стало в 3 раза больше. Это объясняется тем, что к моменту запуска очередного процесса newmem.bash, некоторые процессы уже успевают неаварийно завершить своё исполнение, заполнив свой массив n элементами.