МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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**

Выполнил студент группы №M3204

***Воронин Никита Николаевич***

Проверила

***Титова Анастасия Витальевна***

***САНКТ-ПЕТЕРБУРГ***

***2020***

**Рассматриваемые вопросы**

1. Использование утилиты top для мониторинга параметров памяти

2. Использование имитационных экспериментов для анализа работы механизмов управления памятью.

**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.

Рекомендуется написать «следящие» скрипты и собирать данные, например, из вывода утилиты top автоматически с заданной периодичностью, например, 1 раз в секунду. Можно проводить эксперименты и фиксировать требуемые параметры и в ручном режиме, но в этом случае рекомендуется замедлить эксперимент, например, уменьшив размер добавляемой к массиву последовательности с 10 до 5 элементов.

**Текущая конфигурация операционной системы**

Общий объем оперативной памяти = ***2048 МБ***

Объем раздела подкачки = ***820 МБ*** *(swapon –show)*

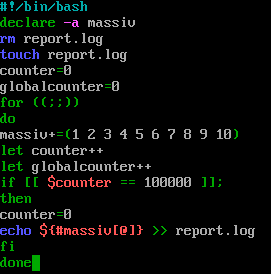
Размер страницы виртуальной памяти = ***4 КБ***

Объем свободной физической памяти в ненагруженной системе = ***1601176 КБ***

Объем свободного пространства в разделе подкачки в ненагруженной системе = ***758220 КБ***

**Эксперимент 1**

*mem.bash*

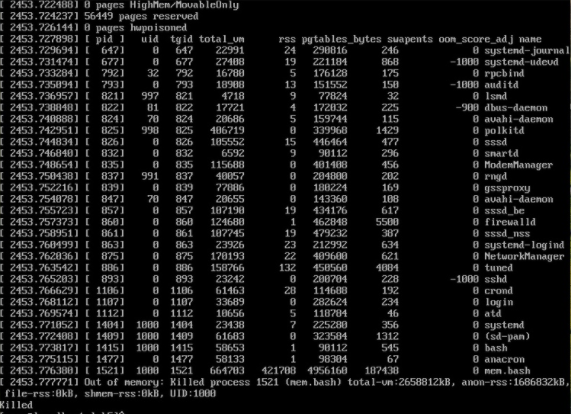


*Первый этап:*

**Первая часть:**

Последнее значение в report.log – **31000000**

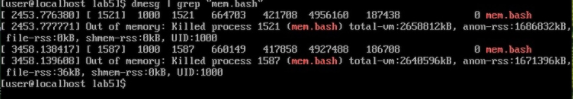
Журнал после аварийного завершения процесса:



**Вторая часть:**

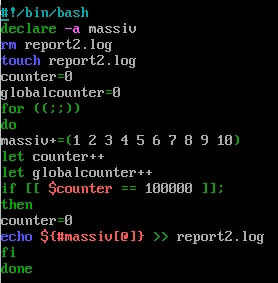
Последнее значение в report.log – **30000000**

Последние две строки журнала после аварийного завершения процесса:



*Второй этап:*

*mem2.bash*



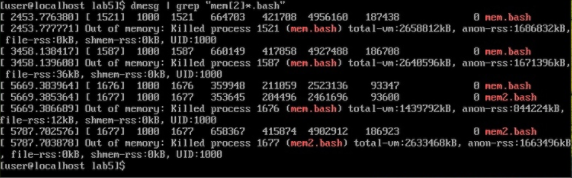
*exp1*



Последнее значение в report.log – **15000000**

Последнее значение в report2.log – **30000000**

Последние записи о скрипте в журнале:



**Графики по первому этапу первого эксперимента:**

**Графики по второму этапу первого эксперимента:**

**Изменения в верхних пяти процессах:**

Пока один из процессов не завершил аварийно свою работу, первые две позиции занимали процессы *mem.bash* и *mem2.bash.* Третью позицию большую часть времени занимал процесс *top.* Иногда выше него появлялись процессы khugepaged, tuned, kworker, kthreadd, kswapd0,rngd, ksoftirqd, rcu\_gp, systemmd. Процесс systemmd всегда был в пятерке верхних процессов.

**kworker** - это процесс-заполнитель для рабочих потоков ядра, которые выполняют большую часть фактической обработки для ядра, особенно в тех случаях, когда имеются прерывания, таймеры, операции ввода-вывода и т. д.

**Служба tuned** отслеживает использование системных компонентов и динамически изменяет настройки системы, исходя из полученной информации о занятости компонентов в разное время

**kthreadd** - это поток ядра, который используется, чтобы помочь другим создавать новые потоки ядра

**kswapd** занимается обслуживанием page cache — вытеснением старых страниц при нехватке памяти

**rngd** - процесс, проверяющий и получающий случайные данные от аппаратных устройств для ядра случайных устройств

**ksoftirqd** - это поток ядра для каждого процессора, который запускается, когда машина находится под большой нагрузкой мягкого прерывания.

**Read – Copy Update (RCU)** – техника синхронизации, предназначенная для «почти read-only», то есть редко изменяемых, структур данных

**Systemd** запускает сервисы описанные в его конфигурации. Конфигурация состоит из множества файлов, которые по-модному называют юнитами

**Наблюдения по эксперименту 1:**

В этом эксперименте нам важно увидеть нелинейность изменения параметров.

Первая часть:

Рассмотрим графики **MiBMEM used** и **MiBSwap used**. Можно заметить, что на первом из графиков происходит линейный процесс увеличения используемой памяти до определенного значения. При этом значении происходит переход на подкачку, пороговое изменение параметров и изменение наклона графика. Стоит отметить, что после изменения наклона, значения памяти не сильно отличаются друг от друга. Следовательно, можно сказать, что это почти линейный участок, но с другим наклоном.

На втором же графике мы видим обратную картину, сначала идет почти линейный участок, потом при том же значении времени, при котором менялся график **MiBMEM used,** меняется и **MiBSwap used**, но уже в линейный процесс.

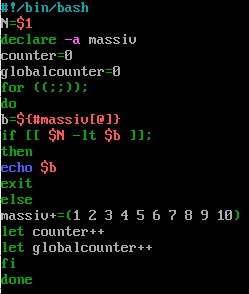
Вторая часть:

Во втором случае мы видим похожую картину, но только изменение наклона происходит трижды. Во второй раз аварийно выключается один из процессов, после чего происходит пороговое изменение параметров.

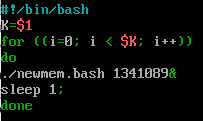
Время аварийного завершения процесса мы можем увидеть и на графике **MiBSwap used.**

**Эксперимент 2**

*newmem.bash*



*exp2*



Максимальное значение N при K = 30 – **1341098**

**Наблюдения во втором эксперименте:**

При **К = 10** и **N = 3000000** программа закончила свое выполнение без ошибок.

При **К = 30** и **N = 3000000** программа аварийно завершила свое выполнение.

**Вывод:** в ходе экспериментов мы убедились, что массив сначала заполняет физическую память, когда она заканчивается, начинается выделение памяти из подкачки. Когда память заканчивается и в разделе подкачки, процесс завершается аварийно. Так же мы заметили, что если уменьшить максимальные размеры массивов в N раз и увеличить кол-во одинаковых процессов в N раз, то процессы завершатся без аварий.