

E0-253: Operating Systems Programming Assignment

Spring 2022

Background

Physical memory is often over-committed by operating systems and hypervisors wherein the total memory demand of applications exceeds the memory capacity available in the system. This assignment is related to two of the techniques that are used to facilitate memory over-commitment. The first technique (referred to as swapping) uses pre-configured disk space to swap out (infrequently accessed) pages from memory. Demand paging loads these pages back into memory when they are accessed by the applications. The second technique (referred to as memory ballooning) is commonly used in virtualized environments. Under ballooning, the guest OS running inside a virtual machine is responsible for releasing some of its memory upon hypervisor request. It is important to note that under ballooning, the virtual machine itself is responsible for saving its data before releasing pages back to the hypervisor.

Assignment summary

Linux maintains a set of swap files and transparently swaps-out process pages to one of the swap files as needed and swaps-in these pages when needed. In this assignment you have to implement a kernel mechanism that maintains these swap files on a per process basis. This per process swap file will be used to store the pages of the respective processes when they get swapped out. The naming format and other details about creation and deletion of this file are mentioned later in the document. You will also implement application level ballooning. At a high-level, we require you to build two components — one in kernel space and one in user space. The user space component will implement a page replacement policy of your choice while the kernel space component will implement the necessary mechanisms for ballooning and swapping.

In the kernel space component, you are responsible for checking the state of free memory in the system. When the amount of free memory falls below the specified threshold, you will send a signal SIGBALLOON to the application. Note that no such signal exists in the kernel today – you will create one for this assignment. In response to this signal, the application will issue one or more system calls asking for some of its pages to be swapped. The system call handler in the kernel will implement the actual mechanism to implement swapping of process pages to per process swap file.

In the user space component, you are free to implement any page replacement policy. Upon receiving the SIGBALLOON signal, your policy will first decide which page(s) it wants to swap out to disk. To choose an efficient page replacement policy, you can use the “idle page tracking” and “pagemap” infrastructure of Linux. It will help you in estimating the access patterns of your applications [3]. Note that the page replacement policy will be evaluated only for correctness and not for performance but it is highly encouraged that you follow through the resources provided.

Note: Note that only your main application will participate in ballooning via swapping. You need to disable swapping for all other applications in the system.

Getting started

You will build the kernel driver in Linux version 5.11.5. Download it as follows:

```
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.11.5.tar.xz
```

The application you will modify is available on github. Clone it on your system as follows:

```
$ git clone https://gitlab.com/shashankatgit/e0253-os-assignment-2022.git
```

Linux supports multiple page sizes. In this assignment, you will work only with base 4KB pages – disable Transparent Huge Pages manually or while compiling your kernel image. Your user space code must be implemented only in “main.c”. Do not update or add any other files. Follow the README.md in github repository for more instructions.

Assignment-0 (Due date: 15/02/2022)

Kernel space: Implement a system call that an application can use to register itself with the ballooning subsystem in the kernel. Your system call is expected to do the following:

1. Disable default swapping algorithm of the kernel – no pages from other applications can be swapped after the system call returns.
2. Set up a mechanism that will deliver the SIGBALLOON signal to the registered application. The signal must be delivered when the amount of free memory falls below 1GB.

User space: In the user space component:

1. Register the application with the kernel's ballooning component using the system call as discussed above. Also report the time taken by the system call to return. You can use hardware timestamp counters for precise measurement.
2. Set up a handler to receive the SIGBALLOON signal from the kernel. File "main.c" contains a variable named `nr_signals` – you are expected to increment it each time the signal is received in the application.

Deliverables: A single kernel patch that contains all your changes and "main.c" user space file.

Assignment-1 (Due date: 12/03/2022)

Linux has global swap file(s) where the kernel writes the memory pages according to its own page reclamation policy. You will modify this mechanism so that we have per-process swap files. Additionally, which pages go into the swap file would be decided by the registered userspace application.

Kernel space: Implement the following in kernel space:

1. Implement additional system call(s) that the user application will use to suggest the list of pages to be swapped out. There is no restriction on the design of system call interface – you can pass as many number or type of arguments as you wish.
2. Invocation of this system call(s) will trigger the mechanism to swap out the suggested list of pages to the process specific swap file. Note that you have to create this file from within the kernel in `/ballooning` directory when application registers itself for receiving SIGBALLOON signal and delete it when the application terminates. Name of this file should be `swapfile_####` where `####` is the process id.

User space: In the user space component:

1. Modify the signal handler to suggest a list of pages to be swapped out to the kernel via the new system call(s) introduced in the kernel space. The suggested pages should only be from the memory buffer allocated at the beginning of the program. After receiving the signal, the application has to respond with pages to swap within 10 seconds.
2. For suggesting the list of pages, you may need to implement a page selection policy. You will not be graded on the performance of the policy.
3. Note: Since you haven't yet handled demand paging for per process swap file, there can be crashes if the userspace program tries to access a swapped out memory region. For this, ensure that your userspace program doesn't read from the buffer.

Deliverables: A single kernel patch that contains all your changes and "main.c" user space file. The kernel patch must also include your changes from assignment-0. Note that you have to implement page selection policy in the application but we will not evaluate your assignment based on the performance of this policy.

Assignment-2 (Due date: 12/04/2022)

Kernel space:

1. Extend your kernel implementation from assignment-1 so that the demand paging of the swapped out pages works as expected.
2. Reads to the swapped out pages should work as expected. We will verify the contents of the buffer after swapping in against the contents before swapping out.

Deliverables: A single kernel patch that contains all your changes and “main.c” user space file. The kernel patch must also include your changes from assignment-0 and assignment-1.

Submission instructions

For each deliverable, you will submit an encrypted file. First, place all your files in a folder named as #####_e0253_*. The zip file must be named as #####_e0253_*.zip. Replace ##### with the last five digits of your SR number and * with the assignment number. For example, for SR number 15943 and assignment-0, the zipped file must be named as 15943_e0253_0.zip. The key for encrypting your zip file is provided at the end of this document – copy it to a file named e0253.key. Once you have placed all files in the folder as specified above, you can produce the encrypted file to submit as follows:

```
$ zip -r 15943_e0253_1.zip 15943_e0253_0/  
$ openssl smime -encrypt -binary -aes-256-cbc -in 15943_e0253_0.zip -out 15943_e0253_0.enc -outform  
DER e0253.key
```

Upload encrypted files to the respective assignment directory in the following link:

https://indianinstituteofscience-my.sharepoint.com/:f:/g/personal/vg_iisc-ac.in/EnLNNjy1W05H1DDqZ4NbAdYB9aL9wjcs60Z8w7KfBShHxg?e=j2pjSs

How to generate a patch using git:

First, stage all modified files for a commit using “git add”. Second, prepare a commit signed with your email id “git commit -s -m”. Third, fetch the commit id using “git log” (this will be the top-most commit id, say XXX). Finally, prepare a patch from your commit using “git format-patch”.

```
$ git add include/linux/abc.h  
$ git add mm/xyz.c  
$ git commit -s -m "os assignment-0"  
$ git log  
$ git format-patch XXX -1
```

Evaluation:

Evaluation will be based on the correctness of implementation. The execution of user application, page replacement policy, memory ballooning and the per-process swapping mechanisms should be working correctly for arbitrary programs unless exempted (as in case of assignment 1).

References

- [1] Understanding the Linux Kernel (3rd edition) by Daniel P. Bovet and Marco Cesati (Chapter 3 and 9).
- [2] Understanding the Linux Virtual Memory Manager by Mel Gorman (Chapter 4, 6, 10, 11 and Appendix J, K in code commentary).
- [3] Idle Page Tracking. https://www.kernel.org/doc/html/latest/admin-guide/mm/idle_page_tracking.html
- [4] Pagemap, from the userspace perspective. <https://www.kernel.org/doc/Documentation/vm/pagemap.txt>
- [5] Look at the following files in Linux source to get started:
include/linux/mm.types.h

```
include/linux/mm.h
mm/swap.c
```

```
-----BEGIN CERTIFICATE-----
MIIFazCCA10gAwIBAgIUcE9MKpoSWGUM8oWPuYMzI5sP+TUwDQYJKoZIhvcNAQEL
BQAwRTELMAkGA1UEBhMCVUxEzARBgNVBAGMC1NvbWUtU3RhdGUxITAfBgNVBAoM
GEludGVybmVOIFdpZGdpdHMgUHR5IEExOZDAeFw0yMjAyMTQxNDQ4NTRaFw0yMzAy
MTQxNDQ4NTRaMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb211LVNOYXRIMSEw
HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwggIiMAOGCSqGSIb3DQEB
AQUAA4ICDwAwggIKAoICAQDP7GkhD6SLRM79B98df2x92W9CJxRlLx0V5w86w7zU
UFnCcQJMF9rc3GEpHjS0lQzr43YRbXcpzYKYLScLvmjmaMoUj29m6PLvf/9tqf06Z
QtE3H2lEbXr5BT4XfPYPBEBXsz4oPTlYJ23n/saOPRFdrrThxQDy9dd+WEa19B0l
geIAfUE86YMHps/TBAa77m6UZM9UjJc3UX5LbzZ0trmnTfPq0mEYB+JaeduwY4f9
V64r45D/neVeEY+Tk7Cc1FG48oTG6Wy+DnNEkxTaRmez0pAFShJwSGj1mZc+NqhQ
ni7hTpolDjAwGhB3y5W067U30Tl+kACuEq4tTkXxvY8iWRyYDjk5moylIB/hR+sA
vu2oDEYoubE4J6TMNi3/bNZqEZYjYjmr2UYscNpF6vrETkHbX3w6gApYtQLbpI6
kDJwtu4qMHeu0afCJ+huUfO75boa0aVKa/WAm2D4DhWDhvhv2KTUDSoU+1r7k0SwD
ZDzox6mWrMqphoy3hCzcVKSz6RbP5c276LJYk5gRXLWkvPD0fCOhGZ/vy80a3b6V
Lmo6dLdZVg4albxPcxuk+MkDEKtjbike8XvMyoWgaFiwIBa/J1DmwDkQLG0EvpbY
sVUScJK9WB1jK4CNgt+odVt+58UE65wsPvxBttPffjYwZOX0sWayKMAG3/MpZUjQ
dwIDAQABo1MwUTAdBgNVHQ4EFgQU6A515G9M3dxe9tW9+wbTd0Z6GN4wHwYDVROj
BBgwFoAU6A515G9M3dxe9tW9+wbTd0Z6GN4wDwYDVROTAQH/BAUwAwEB/zANBgkq
hkiG9w0BAQsFAAOCAGEAXS/myHdZl8v43lHtjCJU3o2myVEsz68E0re0pefPbbqt
08+J+isQ6xhJDz8solaJueU/DhAzLTQ8bdD3EW9KiZ3kUSSkbem7fxxTndSmVWcf
G5ZcJS9RW5IASrb0Mpvswbq9lbCN3iLDF+pcg7wWanFIt0CF+URB8TnkTTw9Ualm
4AvK3LSs/JDRoTq+sPDsLZHDRbe84ykJEfBGnIDHTlIQqs0zutBN78qY3NmSYkCk
QW0Ha/ubWIRJffdfi7k88Qo2bIgNWpz1tdddFVDTVWCt7hTLU041j0JGIqcaog0t
X7Eb7SP25EQjFNh7gfMNXXHAXNDk8UwRUiZBMrmetyVgyJJ+hvYKMyLEc4CdjBQl
jOXofT4/ZGBco86Z1EABqC/DsJ08bx7QarUUFfdx3T21ZydITt1WhcyMCDAdFfXL
87qtyh6yjGYEBDVYNANAYeCQ9vHTBMZQeEysgWnZMxz7DgMmx8G1ik5olOuYH+dW
ryjycn1TTTI+EUtmgWc5V2wnW2aJnarhLq7TYxot3uoEQ3EFp+achGWdPdr10N+Y
Ft+0ra7YXbYIkVUH7i0d08o+WPzQgoxDyEnVnSjR9yq9y8+ubYMoycqEuFfBjsUJ
+znPkkM/PAQ5AJUEvI/4I8q8jFc4FoD/GBbQskn3Zt9kxrK/oM+gYw0Pxs8pnmQ=
-----END CERTIFICATE-----
```