Zram in Slackware ARM and ARM64 Architectures

The Slackware ARM and ARM64 architectures bring us Slackers the possibility of using much more powerful single board computer hardware and Slackware as a native operating system. Currently this includes existing hardware for ARM, and ARM64 hardware such as the RockPro64 and the Pinebook Pro for ARM64. In addition, the zram kernel module can be applied to optimize the performance of many single board computers, some with existing support in Slackware ARM architecture. SBCs such as the Orange Pi, and Bannana Pi, and Raspberry Pi. The kernel module 'zram' is included in a stock installation of Slackware.

What is zram? Lets first begin by defining it: In its simplest explanation, it is a compressed RAM block device. There are three real world examples listed. I find one to be most useful in this instance. Zram can be used as a temporary file system, as partition or location in RAM for log files, or to extend the amount of RAM available to a computer with low RAM resources as a swap space.

Swap space is what I will focus on explaining, and applying, to one of my own systems for the first time. It is very common for single board computers to run out of memory and start swapping. Using zram we can create a swap space that is stored in memory, is compressed in RAM, and is executed from memory. This will extend the life of your SD Card, solid state disk, or USB Stick. It will especially make your device far more responsive. There is no *cost* for using zram if you have RAM that is sitting empty and available.

Fix Me! - Work in progress

1. Article can be expanded to implement /tmp in zram on systems.
2. Article can be expanded to store log files in memory to preserve the SD card and other storage devices.

Getting Started: Requirements

Assure that your kernel has zram support built into it. You can check this by looking at the kernel configuration file either in text mode or in the ncurses interface.

```
cd /usr/src/linux
zcat /proc/config.gz > .config
make menuconfig
```

Navigate to and edit the following options in the kernel configuration:

```
Memory Management options  --->
    <M> Memory allocator for compressed pages
    [*]   Export zsmalloc statistics
Device Drivers  --->
    [*] Block devices --->
        <M> Compressed RAM block device support
        [*]     Write back incompressible or idle page to backing device
        [*]     Track zRam block status
```

Slackware will use the default compression algorithm from kernel.org. All of the other choices are already either a kernel module or built-in to the kernel. For now the **lzo-rle algorithm** will remain the default for Slackware zram block devices.

Support will be built into the generic Slackware Aarch64 kernel soon. You can safely ignore these steps once those changes have been implemented in the configuration.

Slackware ARM32 will require a kernel rebuild from the source provided by the kernel-source package within the distribution

Next Steps: Gathering information

Let us take a look at the zram kernel module on an existing Slackware ARM system. Slackware x86 and x86_64 machines have this module available as well. As an example I will use my Raspberry Pi 4.

As root, run:

```
root@fourb:~# modinfo zram
filename:       /lib/modules/5.10.20-v7l+/kernel/drivers/block/zram/zram.ko
description:    Compressed RAM Block Device
author:         Nitin Gupta <ngupta@vflare.org>
license:        Dual BSD/GPL
srcversion:     36D4C92C73413B20D799472
depends:        zsmalloc
intree:         Y
name:           zram
vermagic:       5.10.20-v7l+ SMP mod_unload modversions ARMv7 p2v8
parm:           num_devices:Number of pre-created zram devices (uint)
```

Now let us take look to see how much RAM the system has registered. I've removed the swap file for this exercise.

As root again, run:

```
root@fourb:~# free -m
               total        used        free      shared  buff/cache
available
Mem:            3827          35        3553           0         238
3748
Swap:              0           0           0
```

Additionally, here is how many processor cores are available:

```
root@fourb:~# lscpu
Architecture:        armv7l
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  1
Core(s) per socket:  4
Socket(s):           1
Vendor ID:           ARM
Model:               3
Model name:          Cortex-A72
Stepping:            r0p3
CPU max MHz:         1500.0000
CPU min MHz:         600.0000
BogoMIPS:            180.00
Flags:               half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva
idivt vfpd32 lpae evtstrm crc32
```

So the system has 4 cores, 4 threads, and 4GB of RAM. This should be plenty of power to run a basic graphical user interface, such as Xfce. KDE Plasma 5 should work too, but I have not tested it yet on this particular system. You may be asking yourself right now: What happens when I open Firefox? Calligra to write a paper? Or even compile a few SlackBuilds of marginal resource use?

That is where zram comes in. Let me show you how to set it up.

Next Steps: Configuration

Enable the kernel module to load from your initial ramdisk (/boot/initrd-armv8). This can be done by creating **/boot/local/load_kernel_modules.post** like so:

```
#!/bin/bash

modprobe -v zram num_devices=4
```

Then add that file to your ramdisk by executing:

```
os-initrd-mgr
```

Do not exceed the number of processor cores. The block devices can be used for a number of tasks. You do not want to burn out your SBC, so use sane values

Add a udev rule to create the block devices on boot. As root create /etc/udev/rules.d/10-zram.rules so it contains the following:

```
KERNEL=="zram[0-3]", SUBSYSTEM=="block", DRIVER=="", ACTION=="add",
ATTR{disksize}=="0", ATTR{disksize}="1024M", RUN+="/sbin/mkswap
$env{DEVNAME}"
```

Be aware of the first setting, **KERNEL==“zram[0-3]“** and the second setting **ATTR{disksize}=“1024M”**. This will create 4 block devices: /dev/zram0, /dev/zram1, /dev/zram2, /dev/zram3. The disk size attribute will set each swap to 1GB. Please be certain you have adjusted the number of block devices and their size to match your system. On a Raspberry Pi 4, this is just right!

On other systems you may have more or less processor cores to devote to zram block devices. The same goes for RAM. You can adjust the settings and see what works best for you

Edit /etc/fstab and add in your zram block devices to the end of the file. It should include this:

```
/dev/zram0  swap             swap        defaults,pri=50        0   0
/dev/zram1  swap             swap        defaults,pri=50        0   0
/dev/zram2  swap             swap        defaults,pri=50        0   0
/dev/zram3  swap             swap        defaults,pri=50        0   0
```

Confirm that the udev rules match zram[0-3]. Otherwise you will run into issues after you reboot your system

Save all your work and restart your system.

Final Step: Check Your Work

Take a look at your swap and block devices after you log in again as root:

```
root@fourb:~# cat /proc/swaps
Filename                                Type            Size            Used
Priority
/dev/zram0                              partition       1048572
61444           50
/dev/zram1                              partition       1048572
60712           50
/dev/zram2                              partition       1048572
60228           50
/dev/zram3                              partition       1048572
60816           50
```

You will notice the swap block devices are added and active.

```
root@fourb:~# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda           8:0    1 14.3G  0 disk
└─sda1        8:1    1 14.3G  0 part
mmcblk0     179:0    0 14.5G  0 disk
```

```
├─mmcblk0p1 179:1    0    1G  0 part /boot
└─mmcblk0p2 179:2    0 13.5G  0 part /
zram0       253:0    0 1024M  0 disk [SWAP]
zram1       253:1    0 1024M  0 disk [SWAP]
zram2       253:2    0 1024M  0 disk [SWAP]
zram3       253:3    0 1024M  0 disk [SWAP]
```

How much swap do you have now?

```
root@fourb:~# free -m
                total        used         free      shared  buff/cache
available
Mem:            3867         711         1313          72         1843
2995
Swap:           4095         235         3860
```

Compare to see what zramctl lists:

```
root@fourb:~# zramctl
NAME       ALGORITHM DISKSIZE  DATA COMPR TOTAL STREAMS MOUNTPOINT
/dev/zram3 lzo-rle         1G  1.1M  1.1M  1.1M       4 [SWAP]
/dev/zram2 lzo-rle         1G  1.2M  1.2M  1.2M       4 [SWAP]
/dev/zram1 lzo-rle         1G  1.6M  1.5M  1.6M       4 [SWAP]
/dev/zram0 lzo-rle         1G  1.1M  1.1M  1.1M       4 [SWAP]
```

Wrapping Up

So, you have four 1GB swap partitions stored in memory. Each is compressed using the lzo-rle compression algorithm, and takes up VERY little space in memory. As you use your system, the zram module will decompress each 1GB swap dynamically, and use, load, unload, and delete data from swap. The priority is set very high for these swap partitions. You can enable your “on disk” swap if you wish just as a back up by editing /etc/fstab. Use the swapon -a command. That is all for now.

Any questions can be directed towards me and I will address any concerns on the LinuxQuestions.org Slackware ARM forums.

Further Reading on Specific Hardware

- You may wish to adjust your swappiness, the cache pressure, the rate background processes write to disk, and lower the rate synchronous I/O occurs. All four of those settings will tell your system to use swap agresively , and initiate better use of zswap (zram in swap space). Below are those settings applied **to my RockPro64 with 4GB of RAM**. Interested persons can find a better description, specifically discussing the **Pinebook Pro and zram** here.
- An additional article about zram and zswap on the Raspberry Pi 4 has been published by Mr. Hayden James, here.

As root, add to and comment out other swap related syscalls /etc/sysctl.d/swap.conf:

```
vm.vfs_cache_pressure=500
vm.swappiness=100
vm.dirty_background_ratio=1
vm.dirty_ratio=50
```

References

Kernel 5.19.x documentation for zram

Zram in Slackware article on Lotar's Wiki

Slackware zram initialization script

Originally written by User mralk3

howtos, arm, arm64, zram, kernel, memory, swap, user mralk3