دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

پیاده سازی الگوریتم FIFO در حافظه ی نهان EnhanceIO

اعضای گروه

پرهام عسکرزاده

شایان شعبانزاده

امیررضا قاسمی

نام درس

سیستم‌های عامل

نیم‌سال اول ۱۴۰۱-۱۴۰۲

نام استاد درس

حسین اسدی

۱ مقدمه

بسیاری از سیستم های کامپیوتری امروزه از دیسک های سخت (hdd) استفاده میکنند. با پیشرفت فناوری و وارد شدن دیسک های SSD که از سرعت بالاتری برخوردار هستند و به همین نسبت قیمت بالاتری هم دارند روش های مختلفی برای بهبود عملکرد hdd ها از طریق استفاده از ssd ها به وجود آماده است. یکی از این روش ها استفاده از ssd به عنوان cache برای hdd ها میباشد. در این زمینه برنامه های مختلفی برای caching در سیستم عامل ها به وجود دارد. یکی از این برنامه های استفاده از EnhanceIO میباشد. EnhanceIO برنامه ای ساخت شرکت sTec میباشد که الهام گرفته از برنامه ی open source ، Flashcache شرکت Facebook میباشد. این برنامه برخلاف برنامه Flashcache از device mapper استفاده نمیکند یعنی میتوان با اضافه کردن آن بدون reformat کردن از آن استفاده کرد و کارایی سیستم رو با استفاده از caching بالا برد.

۲ اهداف پروژه

در این پروژه ما قصد داریم تا در وهله اول با EnhanceIO و ابزار FIO آشنا شویم. سپس با نصب FIO عملکرد آن را تست کنیم و در مرحله بعد کد های آن را بررسی کنیم و در نهایت الگوریتم FIFO را به آن اضافه کنیم.

۳ مفاهیم اولیه

۳-۱ آشنایی با EnhanceIO

۳-۱-۱ معرفی

Enhanceio یک نرم افزار ایجاد حافظه کش ssd است، این نرم افزار از حافظه ssd به عنوان دستگاه های کش برای درایو های هارد دیسک چرخان سنتی (hdd) استفاده می کند. این نرم افزار می تواند با هر دستگاه بلوکی از قبیل دیسک های فیزیکی ، پارتیشن دیسک جداگانه ، یک دستگاه RAID ، یک حجم SAN و ... کار کند. EnhanceIO از سه حالت کش read-only، write-through، back write و سه سیاست جایگزینی حافظه پنهان FIFO RANDOM، و LRU پشتیبانی می کند.

- حالت ذخیره read-only باعث می شود EnhanceIO فقط درخواست های نوشتن IO را مستقیماً هدایت کند. درخواست های IO Read به HDD صادر می شود و داده های خوانده شده از HDD روی SSD ذخیره می شود. درخواست‌های خواندن بعدی برای همان بلوک‌ها از SSD خارج می شود، بنابراین تأخیر آنها را به میزان قابل توجهی کاهش می دهد.
- حالت Write-through مانند حالت read-only است با این تفاوت که حالت Write-through باعث می شود EnhanceIO داده های برنامه را هم برای HDD و هم برای SSD بنویسد.این امر باعث می شود برای خواندن بعدی از همان داده ها استفاده شود زیرا آنها می توانند از SSD بدست آیند.
- حالت Write-back با نوشتن داده های درخواستی برنامه، تأخیر نوشتن را بهبود می بخشد به این داده ها ، داده های کثیف می گویند که در ssd قرار میگیرند و بعدا در هارد کپی میشوند ، خواندن در این حال شبیه به read-only است .

۳-۱-۲ ویژگی های ایجاد ، حذف و ویرایش کش

ابزار eio-cli برای ایجاد و حذف کش و ویرایش استفاده می شود خواص Manpage برای ابزار eio-cli موارد بیشتری را ارائه می دهد :

- پایدار کردن پیکربندی کش ضروری است که یک کش قبل از هر برنامه یا یک برنامه از سر گرفته شود فایل سیستم از حجم منبع در هنگام راه اندازی استفاده می کند. اگر کش فعال باشد پس از نوشتن حجم منبع، داده های قدیمی ممکن است در آن وجود داشته باشد که در نتیجه حافظه پنهان ممکن است باعث خرابی داده ها شود.
- اگر یک SSD در هنگام راه‌اندازی بالا نیامد، خوب است اجازه خواندن داده شود و دسترسی نوشتن به HDD فقط در زمانی که در مد Write-through داده شود. هنگامی که ssd در حالت read-only بود لازم است یک کش در دسترس دوباره ایجاد شود. اگر پیکربندی حافظه پنهان قبلی از سر گرفته شود، ممکن است باعث شود داده های قدیمی خوانده شوند.
- استفاده از حافظه پنهان Write-back کاملاً ضروری است که یک پیکربندی حافظه پنهان Write-back انجام دهید به ویژه که در مورد خرابی سیستم عامل یا قطع برق ، حافظه پنهان Write-back ممکن است حاوی بلوک های کثیفی باشد که هنوز روی HDD نوشته نشده است. باید توجه داشت که خواندن حجم منبع بدون فعال کردن حافظه نهان باعث می

شود که داده های نادرست خوانده شوند. حافظه پنهان write-back باید عملیات clean را انجام دهد تا بتوان داده های کثیف را از hdd شستشو داد.

۳-۱-۳ آشنایی با clean

clean را می توان با استفاده از syscl تنظیم کرد.

- آستانه بالای کثیف : حد بالایی در درصد کثیف بودن بلوک در کل کش.
- آستانه پایین کثیف : حد پایین در درصد کثیفی بلوک در کل کش.
- آستانه بالای تنظیم شده کثیف : حد بالایی در درصد کثیف بودن بلوک ها در یک مجموعه
- آستانه پایین مجموعه کثیف : حد پایین در درصد کثیف بودن بلوک ها در یک مجموعه
- آستانه پاکسازی خودکار: پاکسازی خودکار حافظه پنهان تنها در صورتی رخ می دهد که تعداد درخواست های ورودی/خروجی معوق از طرف HDD زیر آستانه است.
- بازه پاکسازی مبتنی بر زمان (دقیقه): این گزینه به شما امکان می دهد یک فاصله زمانی بین هر فرآیند پاکسازی را مشخص کنید.

  Clean زمانی فعال می شود که یکی از آستانه های بالا یا زمان تمیز باشد بدین صورت که آستانه برآورده می شود و زمانی که تمام آستانه های پایین تر برآورده می شوند، متوقف می شود.

حال که با ابزار Enhanceio آشنا شدیم نوبت به آشنایی به ابزار FIO می رسد. از آنجا که از این ابزار برای تست و مقایسه عملکرد Enhanceio در این پروژه استفاده میکنیم ، لازم است شرح مختصری از آن را ذکر کنیم و با مفهوم آن آشنا شویم.

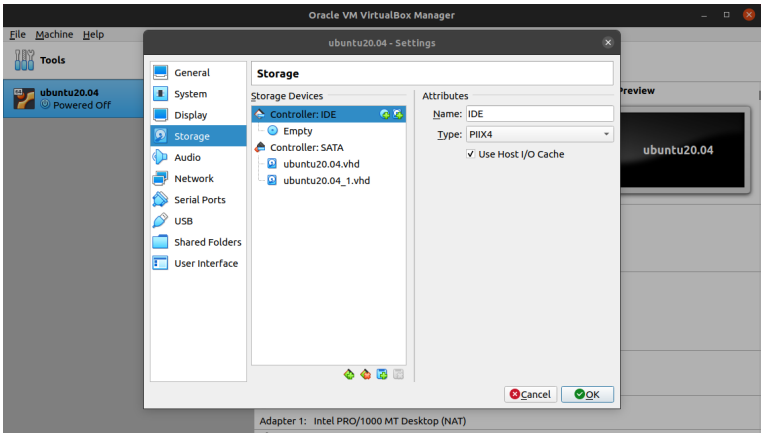
۳-۲ آشنایی با FIO

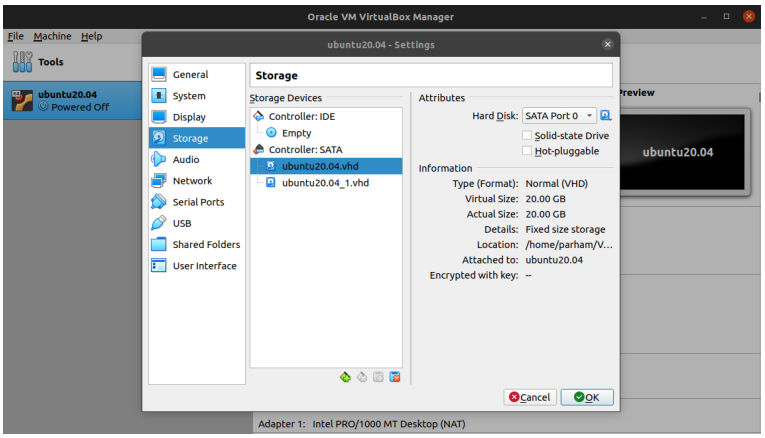
fio ابزاری است که تعدادی رشته یا فرآیند را ایجاد می کند که نوع خاصی از عملکرد I/O را همانطور که توسط کاربر مشخص شده است انجام دهد. استفاده معمولی از fio نوشتن یک فایل job مطابق با بار ورودی/خروجی است که با استفاده از آن می‌خواهیم عمل شبیه‌سازی را انجام دهیم.

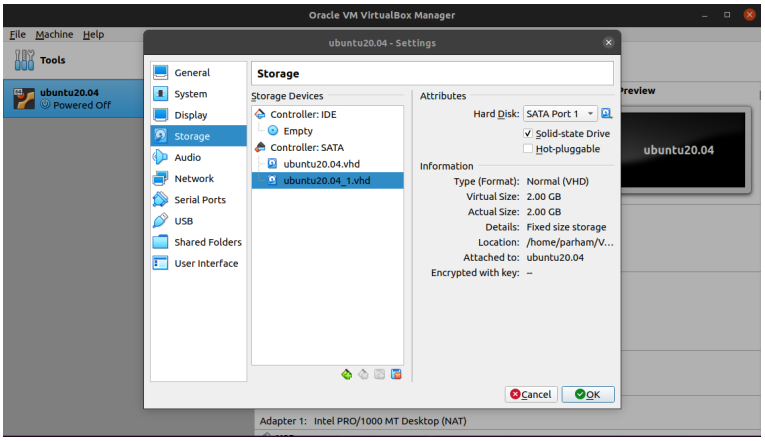
۴ فاز اول پروژه

۴-۱ نصب و راه اندازی سیستم عامل در ماشین مجازی

حال به سراغ مراحل پروژه میرویم در بخش اول نیاز است که ما ماشین مجازی راه اندازی کنیم که دارای دو پارتیشن باشد. بدین صورت که یکی از پارتیشن ها در قالب ssd و دیگری در قالب hdd باشد. باید به این نکته توجه کرد که اندازه hdd باید ۱۰ برابر اندازه ssd باشد . ما در این جا ssd را ۲ گیگابایت و hdd را ۲۰ گیگابایت در نظر میگیریم همچنین سیستم عامل مورد نظر را ubuntu server ۰۴.۲۰ در نظر گرفته ایم. مراحل نصب و راه اندازی ماشین طبق شرایط گفته شده به صورت تصویری در ادامه قابل مشاهده است.








```
parham@project14:~$ lsblk
NAME                      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0                       7:0    0 67.8M  1 loop /snap/lxd/22753
loop1                       7:1    0   47M  1 loop /snap/snapd/16292
loop2                       7:2    0   62M  1 loop /snap/core20/1611
sda                         8:0    0   10G  0 disk
├─sda1                      8:1    0    1M  0 part
├─sda2                      8:2    0  1.8G  0 part /boot
└─sda3                      8:3    0  8.3G  0 part
  └─ubuntu--vg-ubuntu--lv 253:0    0  8.3G  0 lvm  /
sdb                         8:16   0    2G  0 disk
sdc                         8:32   0   20G  0 disk
sr0                        11:0    1 1024M  0 rom
```

۲-۴ نصب و راه اندازی ابزار EnhanceIO

در مرحله بعدی نیاز است که ابزار Enhanceio را دریافت کرده و سپس آن را کامپایل و نصب کنیم. توجه داشته باشید که برای این کار حتما نیاز است که پکیج build-essential را در سیستم عامل خود داشته باشیم و همچنین ورژن python مورد نیاز ما در انجام این پروژه ۲python هست. مراحل نصب و راه اندازی Enhanceio به صورت تصویری در ادامه قابل مشاهده است.

```
parham@project14:~/EnhanceIO$ sudo update-alternatives --install /usr/bin/python python /usr/bin/py
hon2 1
update-alternatives: using /usr/bin/python2 to provide /usr/bin/python (python) in auto mode
parham@project14:~/EnhanceIO$ sudo update-alternatives --install /usr/bin/python python /usr/bin/py
hon3 2
update-alternatives: using /usr/bin/python3 to provide /usr/bin/python (python) in auto mode
parham@project14:~/EnhanceIO$ sudo update-alternatives --config python
There are 2 choices for the alternative python (providing /usr/bin/python).

  Selection    Path             Priority   Status
------------------------------------------------------------
* 0            /usr/bin/python3   2         auto mode
  1            /usr/bin/python2   1         manual mode
  2            /usr/bin/python3   2         manual mode

Press <enter> to keep the current choice[*], or type selection number: 1
update-alternatives: using /usr/bin/python2 to provide /usr/bin/python (python) in manual mode
parham@project14:~/EnhanceIO$ python --version
Python 2.7.18
```

```
parham@project14:~/EnhanceIO$ sudo apt-get install build-essential _
```

```
parham@project14:~$ git clone "https://github.com/lanconnected/EnhanceIO"
Cloning into 'EnhanceIO'...
remote: Enumerating objects: 1014, done.
remote: Total 1014 (delta 0), reused 0 (delta 0), pack-reused 1014
Receiving objects: 100% (1014/1014), 535.83 KiB | 627.00 KiB/s, done.
Resolving deltas: 100% (607/607), done.
parham@project14:~$ cd EnhanceIO/
parham@project14:~/EnhanceIO$ sudo ./Install-EIO _
```

```
Make and Make Install:

   To build and install the EnhanceIO modules run:

   # cd ./Driver/enhanceio
   # make && make install

Installing eio_cli:

   The best way to install the EnhanceIO utility is to use:

   # install -o root -g root -m 700 ./CLI/eio_cli /sbin/eio_cli

   The manual can be installed using:

   # install -o root -g root -m 644 ./CLI/eio_cli.8 /usr/share/man/man8/eio_cli.8

   This ensures the files are placed correctly with the correct owner, group, and permissions.

Loading Modules:

   Once the modules have been installed on the running kernel they can be loaded using:

   # modprode enhanceio
   # modprode enhanceio_rand
   # modprode enhanceio_fifo
   # modprode enhanceio_lru

   Once the modules have been loaded you will be able to use eio_cli to create/modify caches.


Notes
-----

Caches created with eio_cli will be persistent by default. See ./Ducuments/Persistence.txt.
```

۳-۴ نصب و راه اندازی ابزار FIO

```
parham@project14:~$ sudo apt-get install fio
```

۴-۴ انجام تست با FIO برای تست عملکرد ابزار

بعد از اتمام نصب نوبت به بررسی آن میرسد. برای اینکه شهود و درک بهتری نسبت به نتایج بدست آمده داشته باشیم ابتدا یک بار تست های (گفته شده در داکیومنت پروژه) زیر را به طور عادی و بدون فعال کردن کش اجرا میکنیم. تست های خواسته شده در تصویر زیر آمده است.

Rand Read Write 50 50

```
fio --filename=<Device_Path> --readwrite=randrw --name=randrw --blocksize=4k --rwmixread=50 --
rwmixwrite=50  --direct=  --size=30G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --
norandommap --randrepeat=0  --group_reporting
```

Rand Read Write 30 70

```
fio --filename=<Device_Path> --readwrite=randrw --name=randrw --blocksize=4k --rwmixread=30 --
rwmixwrite=70  --direct=  --size=30G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --
norandommap --randrepeat=0  --group_reporting
```

Rand Read Write 70 30

```
fio --filename=<Device_Path> --readwrite=randrw --name=randrw --blocksize=4k --rwmixread=70 --
rwmixwrite=30  --direct=  --size=30G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --
norandommap --randrepeat=0  --group_reporting
```

Sequential Write

```
fio --filename=<Device_Path> --readwrite=write --name=write --blocksize=128k --direct=1 --ioengine=libaio --
-size=3G --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepeat=0 --group_reporting
```

Sequential Read

```
fio --filename=<Device_Path> --readwrite=read --name=read --blocksize=128k --direct=1 --ioengine=libaio --
size=3G --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepeat=0 --group_reporting
```

نمونه تست های گرفته شده در تصاویر زیر آمده است.

```
parham@project14:~$ sudo fio --filename=/dev/sdc --readwrite=randrw --name=randrw --blocksize=4k --r
wmixread=70 --rwmixwrite=30 --direct=1 --size=5G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_
buffers --norandommap --randrepeat=0 --group_reporting
randrw: (g=0): rw=randrw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iod
epth=2
...
fio-3.16
Starting 4 processes
Jobs: 4 (f=4)
```

```
parham@project14:~$ sudo fio --filename=/dev/sdc --readwrite=write --name=write --blocksize=128k  --
direct=1 --size=3G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randre
peat=0 --group_reporting
```

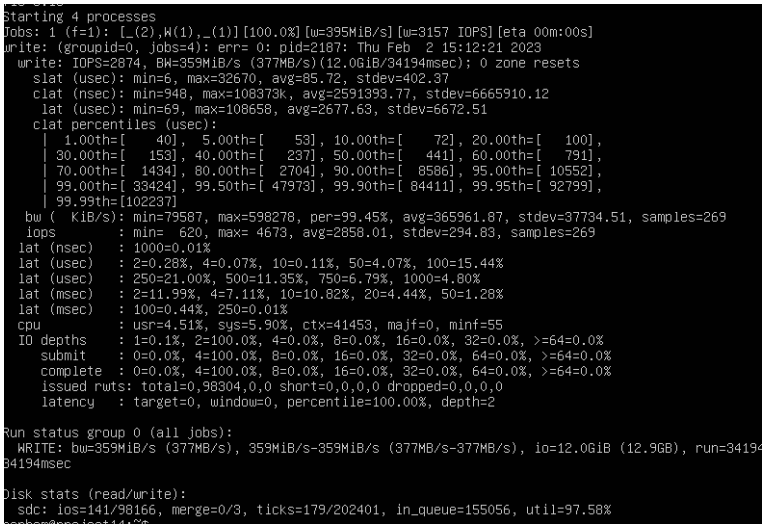
```
parham@project14:~$ sudo fio --filename=/dev/sdc --readwrite=read --name=read --blocksize=128k  --di
rect=1 --size=3G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepe
at=0 --group_reporting
```

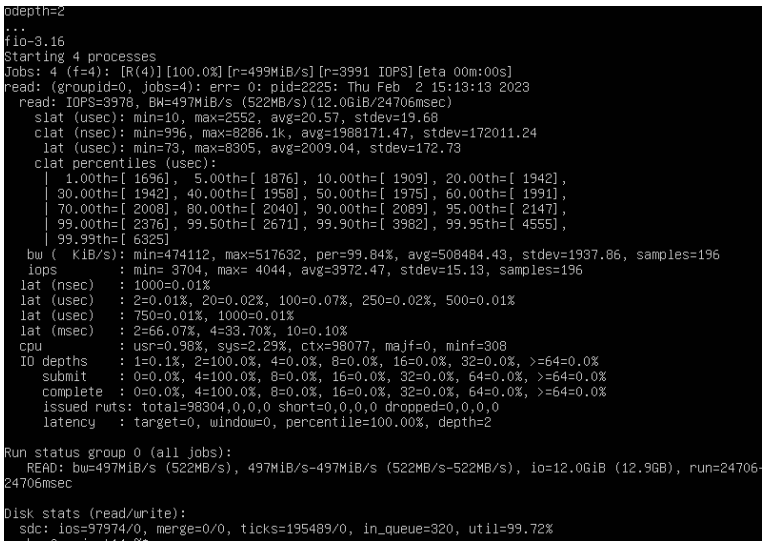
نتیجه تست های صورت گرفته بدون حضور کش به صورت تصویری در ادامه قابل مشاهده است.

```
     | 99.99th=[34341]
   bw (  KiB/s): min=23890, max=47160, per=99.58%, avg=40687.36, stdev=1113.82, samples=2048
   iops        : min= 5971, max=11790, avg=10170.82, stdev=278.44, samples=2048
  write: IOPS=10.2k, BW=39.9MiB/s (41.9MB/s)(10.0GiB/256535msec); 0 zone resets
    slat (usec): min=8, max=5835, avg=37.39, stdev=46.56
    clat (nsec): min=681, max=32778k, avg=134771.48, stdev=188001.72
     lat (usec): min=35, max=32794, avg=172.36, stdev=183.42
    clat percentiles (nsec):
     |  1.00th=[    884],  5.00th=[  12992], 10.00th=[  13888],
     | 20.00th=[  16768], 30.00th=[  40192], 40.00th=[  58624],
     | 50.00th=[  66048], 60.00th=[  82432], 70.00th=[ 122368],
     | 80.00th=[ 203776], 90.00th=[ 358400], 95.00th=[ 505856],
     | 99.00th=[ 856064], 99.50th=[1019904], 99.90th=[1466368],
     | 99.95th=[1712128], 99.99th=[2572288]
   bw (  KiB/s): min=24153, max=46576, per=99.58%, avg=40719.08, stdev=1096.52, samples=2048
   iops        : min= 6036, max=11644, avg=10178.70, stdev=274.12, samples=2048
  lat (nsec)   : 750=0.01%, 1000=1.07%
  lat (usec)   : 2=0.64%, 4=0.01%, 10=0.01%, 20=8.90%, 50=7.93%
  lat (usec)   : 100=14.63%, 250=13.66%, 500=35.63%, 750=12.20%, 1000=3.17%
  lat (msec)   : 2=1.46%, 4=0.06%, 10=0.56%, 20=0.03%, 50=0.04%
  lat (msec)   : 100=0.01%
  cpu          : usr=2.71%, sys=17.22%, ctx=2165004, majf=0, minf=59
  IO depths    : 1=0.1%, 2=100.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=2620443,2622437,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=2

Run status group 0 (all jobs):
   READ: bw=39.9MiB/s (41.8MB/s), 39.9MiB/s-39.9MiB/s (41.8MB/s-41.8MB/s), io=9.00GiB (10.7GB), run=
256535-256535msec
  WRITE: bw=39.9MiB/s (41.9MB/s), 39.9MiB/s-39.9MiB/s (41.9MB/s-41.9MB/s), io=10.0GiB (10.7GB), run=
256535-256535msec

Disk stats (read/write):
  sdc: ios=2619728/2621525, merge=0/0, ticks=1088078/188652, in_queue=306900, util=99.59%
```

```
     | 99.99th=[73925]
   bw (  KiB/s): min=11979, max=27910, per=98.56%, avg=23185.98, stdev=766.01, samples=2136
   iops        : min= 2993, max= 6977, avg=5795.36, stdev=191.47, samples=2136
  write: IOPS=13.7k, BW=53.6MiB/s (56.2MB/s)(14.0GiB/267386msec); 0 zone resets
    slat (usec): min=4, max=6330, avg=38.99, stdev=54.54
    clat (nsec): min=677, max=26363k, avg=158434.48, stdev=273188.06
     lat (usec): min=35, max=26375, avg=197.59, stdev=268.94
    clat percentiles (nsec):
     |  1.00th=[    964],  5.00th=[  13120], 10.00th=[  13632],
     | 20.00th=[  15168], 30.00th=[  36608], 40.00th=[  51456],
     | 50.00th=[  61696], 60.00th=[  69120], 70.00th=[  92672],
     | 80.00th=[ 207872], 90.00th=[ 468992], 95.00th=[ 724992],
     | 99.00th=[1286144], 99.50th=[1531904], 99.90th=[2146304],
     | 99.95th=[2441216], 99.99th=[3522560]
   bw (  KiB/s): min=28504, max=62617, per=98.56%, avg=54117.30, stdev=1763.58, samples=2136
   iops        : min= 7126, max=15654, avg=13528.21, stdev=440.87, samples=2136
  lat (nsec)   : 750=0.01%, 1000=0.78%
  lat (usec)   : 2=0.49%, 4=0.01%, 10=0.01%, 20=17.17%, 50=9.22%
  lat (usec)   : 100=22.51%, 250=9.52%, 500=19.55%, 750=10.18%, 1000=4.99%
  lat (msec)   : 2=4.42%, 4=0.26%, 10=0.79%, 20=0.04%, 50=0.05%
  lat (msec)   : 100=0.01%
  cpu          : usr=2.22%, sys=17.34%, ctx=1559370, majf=0, minf=63
  IO depths    : 1=0.1%, 2=100.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=1572508,3670372,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=2

Run status group 0 (all jobs):
   READ: bw=22.0MiB/s (24.1MB/s), 22.0MiB/s-22.0MiB/s (24.1MB/s-24.1MB/s), io=6143MiB (6441MB), run=
267386-267386msec
  WRITE: bw=53.6MiB/s (56.2MB/s), 53.6MiB/s-53.6MiB/s (56.2MB/s-56.2MB/s), io=14.0GiB (15.0GB), run=
267386-267386msec

Disk stats (read/write):
  sdc: ios=1572650/3670240, merge=0/0, ticks=875699/265693, in_queue=400192, util=97.83%
```

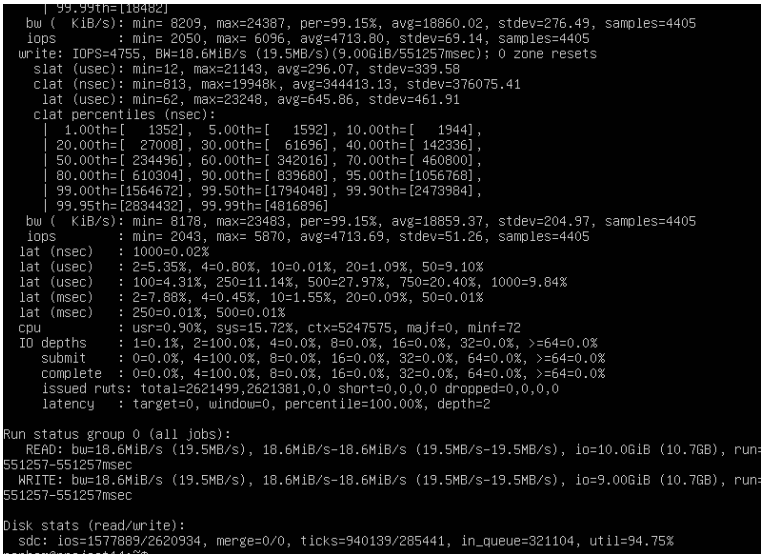
```
     | 99.99th=[29230]
   bw (  KiB/s): min=40241, max=67645, per=99.94%, avg=59799.92, stdev=1120.30, samples=1959
   iops        : min=10059, max=16911, avg=14949.38, stdev=280.09, samples=1959
  write: IOPS=6418, BW=25.1MiB/s (26.3MB/s)(6149MiB/245251msec); 0 zone resets
    slat (usec): min=8, max=6664, avg=34.50, stdev=40.33
    clat (nsec): min=701, max=8860.5k, avg=116747.13, stdev=133967.49
     lat (usec): min=36, max=8900, avg=151.52, stdev=129.30
    clat percentiles (nsec):
     |  1.00th=[    868],  5.00th=[  13248], 10.00th=[  14784],
     | 20.00th=[  37120], 30.00th=[  42240], 40.00th=[  61184],
     | 50.00th=[  71168], 60.00th=[  92672], 70.00th=[ 125440],
     | 80.00th=[ 175104], 90.00th=[ 280576], 95.00th=[ 374784],
     | 99.00th=[ 610304], 99.50th=[ 716800], 99.90th=[1056768],
     | 99.95th=[1286144], 99.99th=[1974272]
   bw (  KiB/s): min=17423, max=30112, per=99.94%, avg=25659.42, stdev=505.63, samples=1959
   iops        : min= 4355, max= 7528, avg=6414.03, stdev=126.40, samples=1959
  lat (nsec)   : 750=0.01%, 1000=0.74%
  lat (usec)   : 2=0.65%, 4=0.01%, 10=0.01%, 20=3.90%, 50=5.39%
  lat (usec)   : 100=8.41%, 250=17.10%, 500=51.28%, 750=10.17%, 1000=1.48%
  lat (msec)   : 2=0.47%, 4=0.03%, 10=0.34%, 20=0.02%, 50=0.03%
  lat (msec)   : 100=0.01%
  cpu          : usr=3.13%, sys=16.58%, ctx=2607536, majf=0, minf=63
  IO depths    : 1=0.1%, 2=100.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=3668651,1574229,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=2

Run status group 0 (all jobs):
   READ: bw=58.4MiB/s (61.3MB/s), 58.4MiB/s-58.4MiB/s (61.3MB/s-61.3MB/s), io=13.0GiB (15.0GB), run=
245251-245251msec
  WRITE: bw=25.1MiB/s (26.3MB/s), 25.1MiB/s-25.1MiB/s (26.3MB/s-26.3MB/s), io=6149MiB (6448MB), run=
245251-245251msec

Disk stats (read/write):
  sdc: ios=3668762/1574212, merge=0/0, ticks=1236319/114078, in_queue=186868, util=100.00%
```

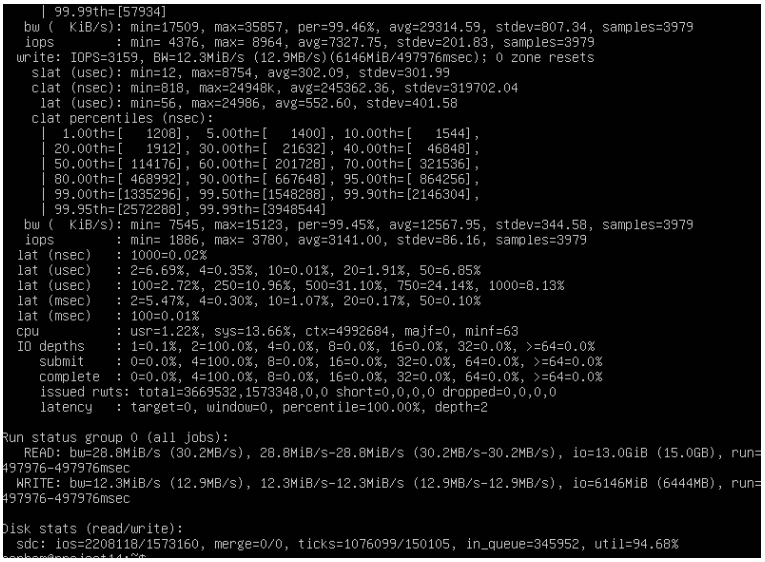


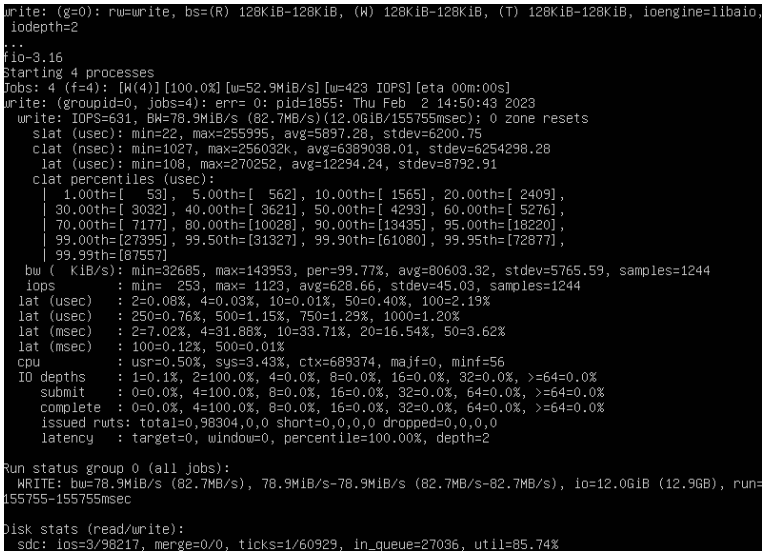


بعد از انجام این مرحله نوبت اینست که کش مورد نظر را ساخته و فعال کنیم مراحل ساخت و اطلاعات کش مورد نظر در تصویر زیر قابل مشاهده است.

حال که کش را ساختیم و آن را فعال کردیم بار دیگر تست های گفته شده را انجام میدهیم برخی از تست های صورت گرفته به صورت تصویری در ادامه قابل مشاهده است.







۵-۴ نتیجه گیری

از تصاویر واضح است که به طور محسوسی پهنای باند ما بیشتر شده است. در جدول زیر پهنای باند در حضور کش و بدون کش در پنج تست گفته شده ثبت شده است. همانطور که از مقایسه اعداد جدول نیز واضح است پهنای باند ما در حضور کش وضعیت بهتری را دارا می باشد

with-cache		without-cache	
read-bandwidth	write-bandwidth	read-bandwidth	write-bandwidth
۱۸.۶	۱۸.۶	۳۹.۹	۳۹.۹
۱۰.۹	۲۵.۵	۲۲	۵۳.۶
۲۸.۸	۱۲.۳	۵۸.۴	۲۵.۱
۷۸.۹		۳۵۹	
۱۳۳		۴۹۷	

۵ فاز دوم پروژه (بررسی کد و فلوچارت برنامه)

۵-۱ فایل بش :Install-EIO

برای نصب برنامه‌ی enhanceIO نیاز است تا فایل بش Install-EIO را اجرا کنید. این فایل برای ما چند کار مهم را انجام میدهد که در ادامه توضیح داده شده است.

- با اجرای دستور make فایل های موجود در پوشه‌ی راbuid drive میکند.
- با دستور install make آن ها را install میکند.
- بررسی ورژن Python
- نصب CLI
- لود کردن ماژول‌های ساخته شده با دستور .modprobe
- کلین کردن فایل های build شده.

۵-۲ بررسی فایل های CLI

بعد از اجرای دستور Install-EIO فایل های CLI نصب میشوند. فایل eio-cli که در این دایرکتوری وجود دارد و به زبان Python است نقش CLI برنامه را اجرا میکند یعنی برای اجرای برنامه ، ساخت کش و ... میتوان از این فایل استفاده کرد. که بعد نصب enhanceIO به سیستم اضافه میشود. این فایل درواقع با import کردن فایل های دارکتوری Drive و کال کردن متدهای آن enhanceIO را اجرا میکند. در قسمت main این برنامه ابتدا یک Parser ساخته میشود که وظیفه‌ی آن ساخت man برای این برنامه است. سپس به آرگمان اول برنامه نگاه میکند که میتواند حالت های زیر باشد.

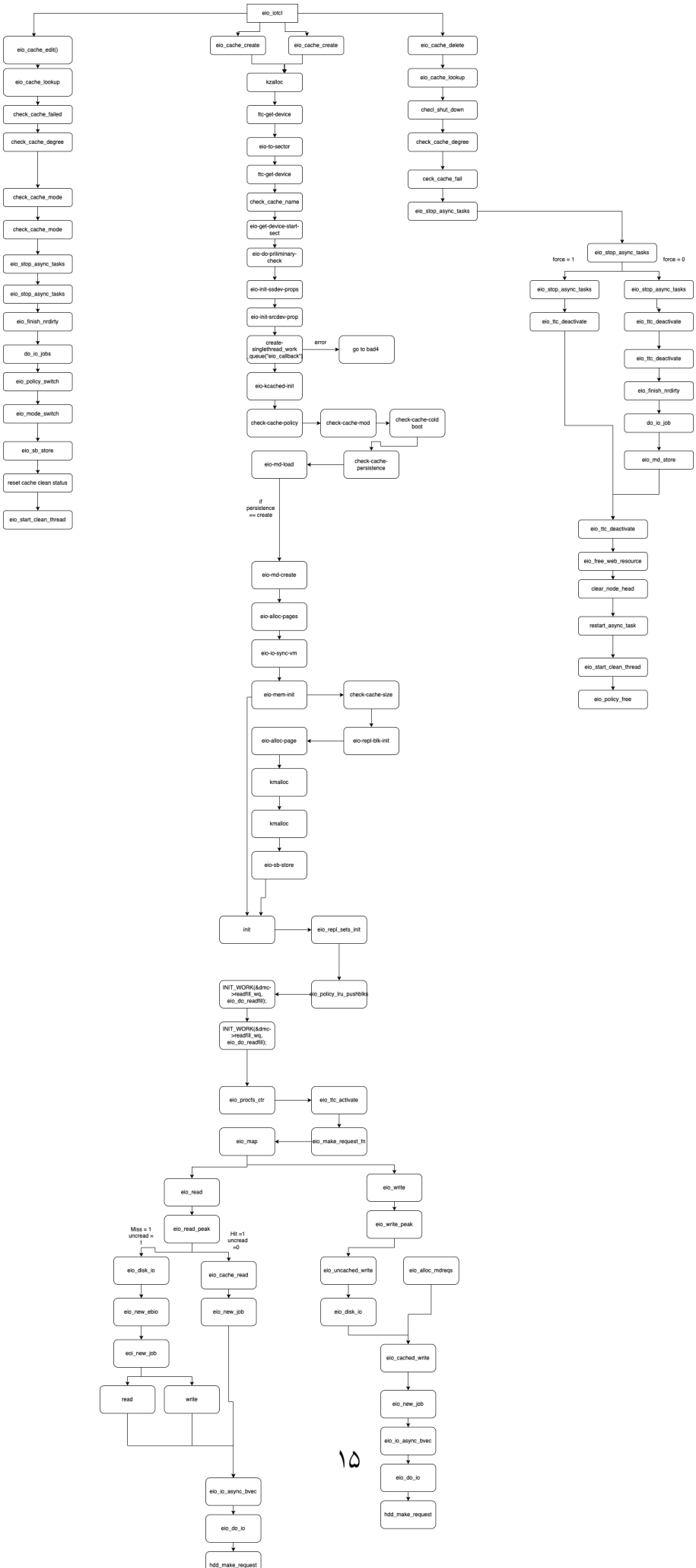
- create : این دستور برای ساخت cache استفاده میشود.
- info : این دستور برای نمایش کش های موجود استفاده میشود.
- delete : این دستور برای پاک کردن یک کش استفاده میشود.
- edit : این دستور برای edit کردن یک کش موجود استفاده میشود.

- clean : این دستور برای پاک کردن محتوای یک cache استفاده میشود.

درصورت وارد کردن یکی از دستورات create ، delete یا edit دستور do-eio-ioctl اجرا میشود که این دستور در فایل eio-ioctl.c در drive تعریف شده است. نحوه ی اجرای هرکدام از این دستورات را در فلوچارت برنامه تعریف شده است.

نقطه‌ی شروع برنامه پس از CLI از فایل eio-ioctl.c در drive میباشد. این فایل ها تماما هنگام اجرای فایل Install-EIO ساخته و نصب میشوند.

در فایل eio-ioctl.c تابع eio-ioctl یک case switch دارد که بسته به نوع دستور یک سری عملیات انجام میدهد. در ادامه فلوچارت مربوط به پروژه EnhanceIO رسم شده است که به ما کمک میکند درک و شهود بهتری از عملکرد برنامه داشته باشیم


eio_iotcl
eio_cache_edit()
eio_cache_lookup
check_cache_failed
check_cache_degree
check_cache_mode
check_cache_mode
eio_stop_async_tasks
eio_stop_async_tasks
eio_finish_nrdirty
do_io_jobs
eio_policy_switch
eio_mode_switch
eio_sb_store
reset cache clean status
eio_start_clean_thread
eio_cache_create
eio_cache_create
kzalloc
ttc-get-device
eio-to-sector
ttc-get-device
check_cache_name
eio-get-device-start-sect
eio-do-priliminary-check
eio-init-ssdev-props
eio-init-srcdev-prop
create-singlethread_work queue("eio_callback")
error
go to bad4
eio-kcached-init
check-cache-policy
check-cache-mod
check-cache-cold boot
check-cache-persistence
eio-md-load
if persistence == create
eio-md-create
eio-alloc-pages
eio-io-sync-vm
eio-mem-init
check-cache-size
eio-alloc-page
eio-repl-blk-init
kmalloc
kmalloc
eio-sb-store
init
eio_repl_sets_init
INIT_WORK(&dmc->readfill_wq, eio_do_readfill);
eio_policy_lru_pushblks
INIT_WORK(&dmc->readfill_wq, eio_do_readfill);
eio_procfs_ctr
eio_ttc_activate
eio_map
eio_make_request_fn
eio_read
eio_read_peak
Miss = 1 uncread = 1
Hit =1 uncread =0
eio_disk_io
eio_cache_read
eio_new_ebio
eio_new_job
eoi_new_job
read
write
eio_io_async_bvec
eio_do_io
hdd_make_request
eio_write
eio_write_peak
eio_uncached_write
eio_alloc_mdreqs
eio_disk_io
eio_cached_write
eio_new_job
eio_io_async_bvec
eio_do_io
hdd_make_request
eio_cache_delete
eio_cache_lookup
checl_shut_down
check_cache_degree
cack_cache_fail
eio_stop_async_tasks
eio_stop_async_tasks
force = 1
force = 0
eio_stop_async_tasks
eio_stop_async_tasks
eio_ttc_deactivate
eio_ttc_deactivate
eio_ttc_deactivate
eio_finish_nrdirty
do_io_job
eio_md_store
eio_ttc_deactivate
eio_free_web_resource
clear_node_head
restart_async_task
eio_start_clean_thread
eio_policy_free

۵-۳ معماری

۵-۳-۱ cache replacement

به طور کلی در caching دو نوع الگوریتم وجود دارد:

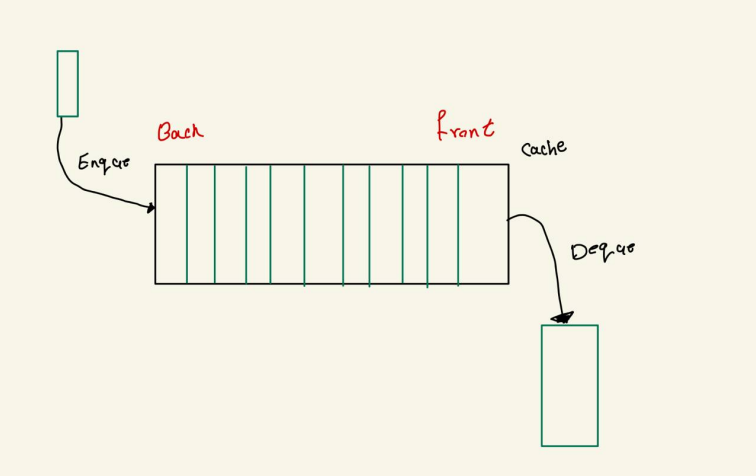
- promotion
- replacement or depromotion

الگوریتم های پروموشن به منظور پر کردن cache بکار میروند مانند الگوریتم های always و ... الگوریتم های replacement که در این پروژه مدنظر ما هستند به منظور خالی کردن cache به کار میروند که از انواع آن میتوانیم به الگوریتم های RANDOM،FIFO،LRU اشاره کرد که در ادامه آن ها را به طور مختصر توضیح میدهیم

۵-۳-۲ الگوریتم های replacement cache

- FIFO

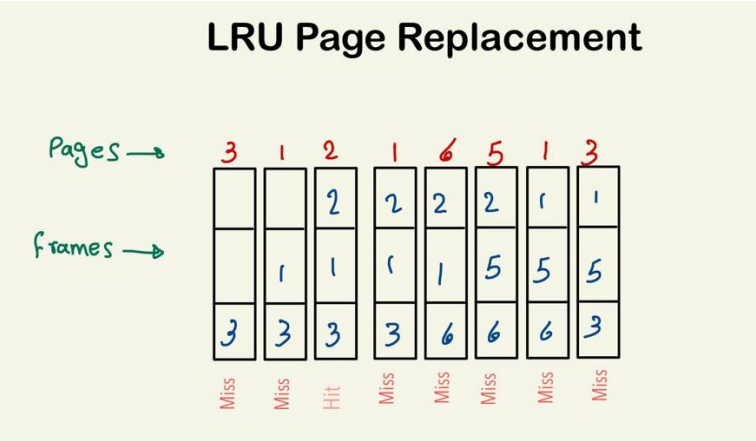
در این الگوریتم خالی شدن Cache به ترتیب ورود به cache صورت میگرد به این صورت که اولین آبجکت وارد شده به cache در صورت نیاز برای خالی کردن cache از cache خارج میشود.



- LRU

این الگوریتم یکی از الگوریتم های پیجینگ در سیستم عامل است. این الگوریتم گریدی اشاره

دارد به used recently least به این صورت که در عملیات پیجینگ جایگزین آخرین پیج استفاده شده میشود. چرا که با احتمال خوبی به آن آبجکت دیگر نیازی نخواهد بود.



- Random در این الگوریتم برای خالی کردن cache برا وارد کردن آبجکت جدید یکی از آبجکت ها به صورت رندوم از cache خارج میشود و آبجکت جدید جایگزین آن میشود. از مزایای این روش میتوان به نگه نداشتن هیچ گونه رفرنس و آسانی پیاده سازی اشاره کرد. این الگوریتم در پردازنده های ARM و i۸۶۰ استفاده شده است.

۶ فاز سوم پروژه (اضافه کردن الگوریتم fifo به ابزار EnhancIO)

کد مربوطه برای قسمت پیاده سازی fifo برای این ابزار در ادامه پیوست شده است.

۶-۱ بخش های مهم کد

دراین بخش به توضیح بخش ها و توابع مهم کد پیاده سازی شده میپردازیم

۶-۱-۱ تابع init set cache

همانطور که میدانیم در روش FIFO داده ساختار های Block مدنظر ما نیستند و داده ساختار های Set مد نظر هستند.
کار اصلی این متد بدین شکل است که یک set به اندازه سایز مدنظر ما ایجاد میکند و برای آن در حافظه فضای کافی را در نظر میگیرد.

```
int eio_fifo_cache_sets_init(struct eio_policy *p_ops)
{
    int i;
    sector_t order;
    struct cache_c *dmc = p_ops->sp_dmc;
    struct eio_fifo_cache_set *cache_sets;

    pr_info("Initializing fifo cache sets\n");
    order = (dmc->size >> dmc->consecutive_shift) *
        sizeof(struct eio_fifo_cache_set);

    dmc->sp_cache_set = vmalloc((size_t)order);
    if (dmc->sp_cache_set == NULL)
        return -ENOMEM;

    cache_sets = (struct eio_fifo_cache_set *)dmc->sp_cache_set;

    for (i = 0; i < (int)(dmc->size >> dmc->consecutive_shift); i++) {
        cache_sets[i].set_fifo_next = i * dmc->assoc;
        cache_sets[i].set_clean_next = i * dmc->assoc;
    }

    return 0;
}
```

۶-۱-۲ تابع reclaim cache

زمانی که cache پر میشود و ما نیاز داریم که آبجکت جدیدی را در این حافظه ذخیره کنیم بر اساس توضیحات الگوریتم fifo که در قسمت قبلی توضیح داده شد باید عمل کنیم و اولین خانه cache را باید به عنوان قربانی از این حافظه خارج کنیم تا جای کافی برای وارد کردن آبجکت جدید ایجاد شود.

تابع reclaim cache در واقع همین کار را برای ما انجام میدهد

```
void eio_fifo_find_reclaim_dbn(struct eio_policy *p_ops, index_t start_index,
               index_t *index)
{
    index_t end_index;
    int slots_searched = 0;
    index_t i;
    index_t set;
    struct eio_fifo_cache_set *cache_sets;
    struct cache_c *dmc = p_ops->sp_dmc;

    set = start_index / dmc->assoc;
    end_index = start_index + dmc->assoc;
    cache_sets = (struct eio_fifo_cache_set *)dmc->sp_cache_set;

    i = cache_sets[set].set_fifo_next;
    while (slots_searched < (int)dmc->assoc) {
        EIO_ASSERT(i >= start_index);
        EIO_ASSERT(i < end_index);
        if (EIO_CACHE_STATE_GET(dmc, i) == VALID) {
            *index = i;
            break;
        }
        slots_searched++;
        i++;
        if (i == end_index)
            i = start_index;
    }
    i++;
    if (i == end_index)
        i = start_index;
    cache_sets[set].set_fifo_next = i;
}
```

۳-۱-۶ تابع clean cache

این تابع از کد نقش خالی کردن cache را به عهده دارد به این صورت که برا تمام خانه های موجود در interate cache میکند و در صورتی که خانه ای پر بود آن را خالی میکند.

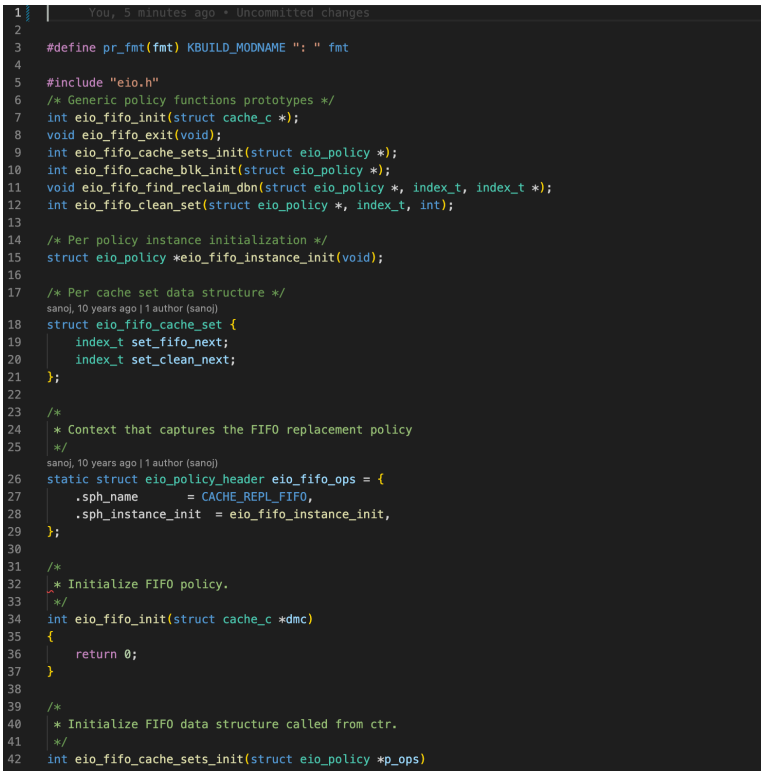
```
int eio_fifo_clean_set(struct eio_policy *p_ops, index_t set, int to_clean)
{
    index_t i;
    int scanned = 0, nr_writes = 0;
    index_t start_index;
    index_t end_index;
    struct eio_fifo_cache_set *cache_sets;
    struct cache_c *dmc;

    dmc = p_ops->sp_dmc;
    cache_sets = (struct eio_fifo_cache_set *)dmc->sp_cache_set;
    start_index = set * dmc->assoc;
    end_index = start_index + dmc->assoc;
    i = cache_sets[set].set_clean_next;

    while ((scanned < (int)dmc->assoc) && (nr_writes < to_clean)) {
        if ((EIO_CACHE_STATE_GET(dmc, i) & (DIRTY | BLOCK_IO_INPROG)) ==
            DIRTY) {
            EIO_CACHE_STATE_ON(dmc, i, DISKWRITEINPROG);
            nr_writes++;
        }
        scanned++;
        i++;
        if (i == end_index)
            i = start_index;
    }
    cache_sets[set].set_clean_next = i;

    return nr_writes;
}
```

۴-۱-۶ تابع init instance cache

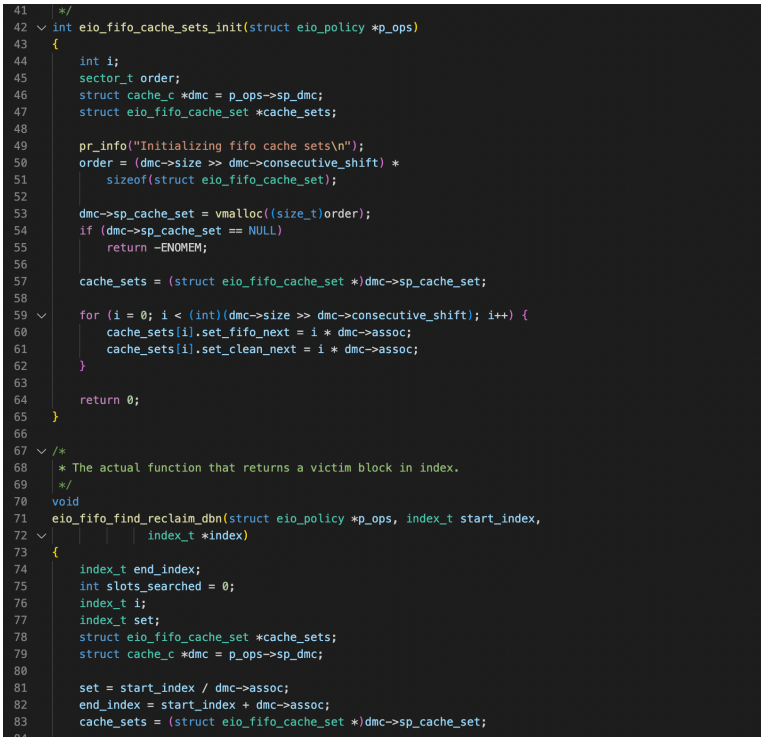
وظیفه این تابع این است که یک instance جدید از plociy بسازد و همچنین توابع راFIFO به آن مپ کند از دیگر وظایف این تابع این است که وظیفه دارد مقدار فضای لازم در حافظه برای CACHE را مشخص کند

```
struct eio_policy *eio_fifo_instance_init(void)
{
    struct eio_policy *new_instance;

    new_instance = vmalloc(sizeof(struct eio_policy));
    if (new_instance == NULL) {
        pr_err("ssdscache_fifo_instance_init: vmalloc failed");
        return NULL;
    }

    /* Initialize the FIFO specific functions and variables */
    new_instance->sp_name = CACHE_REPL_FIFO;
    new_instance->sp_policy.lru = NULL;
    new_instance->sp_repl_init = eio_fifo_init;
    new_instance->sp_repl_exit = eio_fifo_exit;
    new_instance->sp_repl_sets_init = eio_fifo_cache_sets_init;
    new_instance->sp_repl_blk_init = eio_fifo_cache_blk_init;
    new_instance->sp_find_reclaim_dbn = eio_fifo_find_reclaim_dbn;
    new_instance->sp_clean_set = eio_fifo_clean_set;
    new_instance->sp_dmc = NULL;

    try_module_get(THIS_MODULE);

    pr_info("eio_fifo_instance_init: created new instance of FIFO");

    return new_instance;
}
```

در ادامه بخشی از کد های الگوریتم مربوطه به صورت تصویری قابل مشاهده است.



```
1   You, 5 minutes ago • Uncommitted changes
2
3   #define pr_fmt(fmt) KBUILD_MODNAME ": " fmt
4
5   #include "eio.h"
6   /* Generic policy functions prototypes */
7   int eio_fifo_init(struct cache_c *);
8   void eio_fifo_exit(void);
9   int eio_fifo_cache_sets_init(struct eio_policy *);
10  int eio_fifo_cache_blk_init(struct eio_policy *);
11  void eio_fifo_find_reclaim_dbn(struct eio_policy *, index_t, index_t *);
12  int eio_fifo_clean_set(struct eio_policy *, index_t, int);
13
14  /* Per policy instance initialization */
15  struct eio_policy *eio_fifo_instance_init(void);
16
17  /* Per cache set data structure */
    sanoj, 10 years ago | 1 author (sanoj)
18  struct eio_fifo_cache_set {
19      index_t set_fifo_next;
20      index_t set_clean_next;
21  };
22
23  /*
24   * Context that captures the FIFO replacement policy
25   */
    sanoj, 10 years ago | 1 author (sanoj)
26  static struct eio_policy_header eio_fifo_ops = {
27      .sph_name       = CACHE_REPL_FIFO,
28      .sph_instance_init  = eio_fifo_instance_init,
29  };
30
31  /*
32   * Initialize FIFO policy.
33   */
34  int eio_fifo_init(struct cache_c *dmc)
35  {
36      return 0;
37  }
38
39  /*
40   * Initialize FIFO data structure called from ctr.
41   */
42  int eio_fifo_cache_sets_init(struct eio_policy *p_ops)
```



```
41   */
42  int eio_fifo_cache_sets_init(struct eio_policy *p_ops)
43  {
44      int i;
45      sector_t order;
46      struct cache_c *dmc = p_ops->sp_dmc;
47      struct eio_fifo_cache_set *cache_sets;
48
49      pr_info("Initializing fifo cache sets\n");
50      order = (dmc->size >> dmc->consecutive_shift) *
51          sizeof(struct eio_fifo_cache_set);
52
53      dmc->sp_cache_set = vmalloc((size_t)order);
54      if (dmc->sp_cache_set == NULL)
55          return -ENOMEM;
56
57      cache_sets = (struct eio_fifo_cache_set *)dmc->sp_cache_set;
58
59      for (i = 0; i < (int)(dmc->size >> dmc->consecutive_shift); i++) {
60          cache_sets[i].set_fifo_next = i * dmc->assoc;
61          cache_sets[i].set_clean_next = i * dmc->assoc;
62      }
63
64      return 0;
65  }
66
67  /*
68   * The actual function that returns a victim block in index.
69   */
70  void
71  eio_fifo_find_reclaim_dbn(struct eio_policy *p_ops, index_t start_index,
72                            index_t *index)
73  {
74      index_t end_index;
75      int slots_searched = 0;
76      index_t i;
77      index_t set;
78      struct eio_fifo_cache_set *cache_sets;
79      struct cache_c *dmc = p_ops->sp_dmc;
80
81      set = start_index / dmc->assoc;
82      end_index = start_index + dmc->assoc;
83      cache_sets = (struct eio_fifo_cache_set *)dmc->sp_cache_set;
```

```
84
85      i = cache_sets[set].set_fifo_next;
86      while (slots_searched < (int)dmc->assoc) {
87          EIO_ASSERT(i >= start_index);
88          EIO_ASSERT(i < end_index);
89          if (EIO_CACHE_STATE_GET(dmc, i) == VALID) {
90              *index = i;
91              break;
92          }
93          slots_searched++;
94          i++;
95          if (i == end_index)
96              i = start_index;
97      }
98      i++;
99      if (i == end_index)
100         i = start_index;
101     cache_sets[set].set_fifo_next = i;
102 }
103
104 /*
105  * Go through the entire set and clean.
106  */
107 int eio_fifo_clean_set(struct eio_policy *p_ops, index_t set, int to_clean)
108 {
109     index_t i;
110     int scanned = 0, nr_writes = 0;
111     index_t start_index;
112     index_t end_index;
113     struct eio_fifo_cache_set *cache_sets;
114     struct cache_c *dmc;
115
116     dmc = p_ops->sp_dmc;
117     cache_sets = (struct eio_fifo_cache_set *)dmc->sp_cache_set;
118     start_index = set * dmc->assoc;
119     end_index = start_index + dmc->assoc;
120     i = cache_sets[set].set_clean_next;
121
122     while ((scanned < (int)dmc->assoc) && (nr_writes < to_clean)) {
123         if ((EIO_CACHE_STATE_GET(dmc, i) & (DIRTY | BLOCK_IO_INPROG)) ==
124             DIRTY) {
125             EIO_CACHE_STATE_ON(dmc, i, DISKWRITEINPROG);
126             nr_writes++;
```

```
126                 nr_writes++;
127             }
128             scanned++;
129             i++;
130             if (i == end_index)
131                 i = start_index;
132         }       sanoj, 10 years ago • Work in Progress …
133         cache_sets[set].set_clean_next = i;
134
135         return nr_writes;
136     }
137
138     /*
139      * FIFO is per set, so do nothing on a per block init.
140      */
141     int eio_fifo_cache_blk_init(struct eio_policy *p_ops)
142     {
143         return 0;
144     }
145
146     /*
147      * Allocate a new instance of eio_policy per dmc
148      */
        sanoj, 10 years ago | 1 author (sanoj)
149     struct eio_policy *eio_fifo_instance_init(void)
150     {
151         struct eio_policy *new_instance;
152
153         new_instance = vmalloc(sizeof(struct eio_policy));
154         if (new_instance == NULL) {
155             pr_err("ssdscache_fifo_instance_init: vmalloc failed");
156             return NULL;
157         }
158
159         /* Initialize the FIFO specific functions and variables */
160         new_instance->sp_name = CACHE_REPL_FIFO;
161         new_instance->sp_policy.lru = NULL;
162         new_instance->sp_repl_init = eio_fifo_init;
163         new_instance->sp_repl_exit = eio_fifo_exit;
164         new_instance->sp_repl_sets_init = eio_fifo_cache_sets_init;
165         new_instance->sp_repl_blk_init = eio_fifo_cache_blk_init;
166         new_instance->sp_find_reclaim_dbn = eio_fifo_find_reclaim_dbn
167         new_instance->sp_clean_set = eio_fifo_clean_set;
168         new_instance->sp_dmc = NULL;
169
```

```
170     try_module_get(THIS_MODULE);
171
172     pr_info("eio_fifo_instance_init: created new instance of FIFO");
173
174     return new_instance;
175 }
176
177 /*
178  * Cleanup an instance of eio_policy (called from dtr).
179  */
180 void eio_fifo_exit(void)
181 {
182     module_put(THIS_MODULE);
183 }
184
185 static
186 int __init fifo_register(void)
187 {
188     int ret;
189
190     ret = eio_register_policy(&eio_fifo_ops);
191     if (ret != 0)
192         pr_info("eio_fifo already registered");
193
194     return ret;
195 }
196
197 static
198 void __exit fifo_unregister(void)
199 {
200     int ret;
201
202     ret = eio_unregister_policy(&eio_fifo_ops);
203     if (ret != 0)
204         pr_err("eio_fifo unregister failed");
205 }
206
207 module_init(fifo_register);
208 module_exit(fifo_unregister);
209
210 MODULE_LICENSE("GPL");
211 MODULE_DESCRIPTION("FIFO policy for EnhanceIO");
212 MODULE_AUTHOR("STEC, Inc. based on code by Facebook");
213
```

مطالب تکمیلی

فایل مربوط به کد الگوریتم fifo پیوست شده است.