Snapper 2.0 - Roadmap

Rumen Mitov

June 9, 2025

1 TODO [#A] Setup Snapper Project

DEADLINE: *<2025-06-16 Mon>*

`Effort: 4`

- ☒ Create run file and Makefile
- ☐ ~~Use Goa as a build tool~~
- ☐ Define data structures and `Snapper` object.

2 TODO [#C] Unit Tests

DEADLINE: *<2025-06-16 Mon>*

`Effort: 4`

The following unit tests simulate the virtual address space via a linked-list.

2.1 TODO Snapshot creation

1. Create a linked list of 1000 elements containing random integers.
2. Create a snapshot of the linked list.
3. If test is successful, there should be 1000 files in the snapshot directory of the generation.

2.2 TODO Snapshot successful recovery

2.2.1 Test #1

Note, this test requires a compressed archive of a generation storing integers in the range from 1 to 1000 in increasing order. The archive should be uncompressed before the test is ran, and the uncompressed files should be deleted after the test finishes.

1. Create an empty linked-list.
2. Recover each file from the generation into an element in the linked-list.
3. If test is successful, the linked-list should store all numbers from 1 to 1000 in ascending order.

2.2.2 Test #2

Note, this test requires a compressed archive containing two generations: one invalid, the second invalid. Both generations are snapshots of a linked-list containing all integers from 1 to 1000 in increasing order. However, the older, invalid one contains some files that are needed by the valid generation.

1. Create an empty linked-list.
2. Recover each file from the valid generation into an element in the linked-list.
3. If test is successful, the linked-list should store all numbers from 1 to 1000 in ascending order.

2.2.3 Test #3

Note, this test requires a compressed archive containing a generation and an incomplete snapshot named "current". The generation should store integers in the range from 1 to 1000 in increasing order.

1. Create an empty linked-list.
2. Recover each file from the generation into an element in the linked-list.
3. If test is successful, the linked-list should store all numbers from 1 to 1000 in ascending order and the "current" directory should have been removed.

2.3 TODO Snapshot unsuccessful recovery

Note, this test requires a compressed archive of a generation whose archive file has an invalid CRC.

1. Try to recover the generation.
2. If test is successful, the recovery should not be possible and an error should be written stating that the archive file is invalid.

2.4 TODO Snapshot purge

Note, this test requires a compressed archive of two valid generations. Both generations are snapshots of a linked-list containing all integers from 1 to 1000 in increasing order. However, the older one contains some files that are needed by the valid generation.

1. Purge the older generation.
2. Create an empty linked-list.
3. Trying to recover from the older generation should be unsuccessful.
4. Recover each file from the non-purged generation into an element in the linked-list.
5. If test is successful, the linked-list should store all numbers from 1 to 1000 in ascending order.

3 TODO [#B] Snapshot Creation

DEADLINE: *<2025-06-16 Mon>*

`Effort: 10`

1. Delete any unfinished snapshot generation called "current".
2. Initialize a new generation directory called "current".
3. Within the generation directory create the archive file and the snapshot directory.

4. Check if there is a valid prior generation (based on the timestamps). If there is, load the archive file's data into the `Snapper::Archiver` array.

5. Let h_i := `Snapper::Archiver[i]`. If `Snapper::Archiver[i]` contains redundant files, use the *first file* before the comma.

6. For each virtual page where the CRC of the file h_i does not match the CRC of page (or h_i does not exist):

   (a) Create new file, h_j, and save the binary contents of the page into this new file.

   (b) Initialize the snapshot file with the new CRC of the data, a reference count of 1, and the binary data of the page.

   (c) Update `Snapper::Archiver[i]` $\leftarrow$ *path(* h_j *)*, there *path()* is the path relative to <snapper-root>.

7. For each virtual page where CRC of the file h_i matches the CRC of the page:

   (a) If the file h_i has a reference count greater than or equal to **SNAPPER_REDUND**:

      i. Create a new file h_j as outlined in Step 6.

      ii. Increment the reference count for all files in `Snapper::Archiver[i]`.

      iii. Update `Snapper::Archiver[i]` $\leftarrow$ (*path(* h_j *)* || ',' || `Snapper::Archiver[i]`) - i.e. prepend the new file path, separated by a comma, to the string containing the redundant file copies.

   (b) If the file h_i has a reference count lower than **SNAPPER_REDUND**, increment the reference count of it and all other redundant files in `Snapper::Archiver[i]`.

8. Save `Snapper::Archiver` into the archive file and calculate its CRC.

9. Rename "current" to the current UNIX timestamp to signify that the generation is complete.

4 TODO #B Snapshot Recovery

DEADLINE: *<2025-06-23 Mon>*

`Effort: 10`

1. Choose a generation to boot from (by default the latest one).
2. Check if the generation is valid (i.e. has an archive file with a valid CRC). If not, recovery is not possible.
3. Load the archive file of the latest valid generation into `Snapper::Archiver`.
4. For each h ∈ `Snapper::Archiver` and for each redundant file, h_i ∈ h:
   (a) Check the CRC with the stored data.
   (b) If h_i does not exist or there is a mismatch with the CRC, try the next redundant file.
   (c) If there are no more redundant files to check, respond according to the configured policy.
   (d) If the CRC matches h_i, load the data of h_i into the corresponding page.

5 TODO #C Snapshot Purge

DEADLINE: *<2025-06-23 Mon>*

`Effort: 10`

Note, that when a file's reference count is decremented to 0, the file is removed. If a directory becomes empty as a result, it is removed.

1. Make sure the generation is valid (i.e. it has an archive file with a valid CRC).
2. If the archive file has an invalid CRC:
   (a) If **SNAPPER_INTEGR** is set to true, crash the system and ask the system administrator to replace the generation's corrupted archive file with a backup copy.
   (b) Otherwise, log an error message and boot the system into a clean state.
3. If the archive file has a valid CRC:
   (a) Load the archive file into `Snapper::Archiver`.
   (b) For each entry h ∈ `Snapper::Archiver` and for each file h_i ∈ h: decrement the file h_i's reference count.
   (c) Delete the archive file.

6 TODO [#C] XML Configuration Support

DEADLINE: *<2025-06-30 Mon>*

`Effort: 5`

- ☐ SNAPPER_ROOT
- ☐ SNAPPER_THRESH
- ☐ SNAPPER_INEGR
- ☐ SNAPPER_REDUND
- ☐ Retention::MAX_SNAPS
- ☐ Retention::EXPIRATION

7 TODO [#C] Integration Into PhantomOS

DEADLINE: *<2025-06-30 Mon>*

`Effort: 10`

8 TODO [#C] PhantomOS Snapshot Tests

DEADLINE: *<2025-07-07 Mon>*

`Effort: 5`

- ☐ Snapshot creation
- ☐ Snapshot recovery
- ☐ Snapshot purge