**Inter-process Communication Services**

## Introduction

Recoverable virtual memory is a technique highly used in database design and on any data management that is required to make modification in many different instances atomically. Making use log file techniques, one can guarantee that an all or nothing operation will take place in case the program crashes, or a transaction gets aborted.

## Implementation

The key library used to implement practically all functionalities was C++ map. The map library can pair names, pointers, structs etc to values, and since you can template the values, the data type stop being a concern. Using the map data structure, three global structure were created, and served as backbone for the entire application: map_segname_to_seg, map_pt_to_trans and map_trans_to_regions. These structure were used in the following manner:

- **Map_segname_to_seg -** Maps the segment to its segname, provided upon mapping. It allows a segment to be searched for by its name.
- **Map_pt_to_trans -** Maps a segment pointer to its transaction. It allows a transaction to own a segment, and to access the segment by knowing the trans_id.
- **Map_trans_to_region -** Maps the regions about to be modified to its transaction. This structure takes a transaction id and pairs it with a list of regions, since a transaction can modify multiple data regions.

With the above structure, mapping and unmapping becomes as simple as indexing into them, providing, only, the key to the structure.

### The log file structure

The log file is structured by lines. The log file parser target two key items: trans_begin and trans_end. In between these values lie the values for the transaction. Once parsing, if the parser sees a pair of trans_begin and trans_end, it parses all modifications in between it, and writes it to a segment file. On the other hand, if the parser does not see the paired key values, it deletes all that is in between, and moves on to the next pair value.

## API

The API implements all functionalities necessary to make RVM possible.

- **rvm_init(const char *directory) -** initializes the the rvm structure, creating the directory and the rvm.log file necessary to run the library;
- **void *rvm_map(rvm_t rvm, const char *segname, int size_to_create) -** reads into the rvm directory and maps the files into virtual memory segments. If the segment does not yet exist, this functionality creates the file, and maps it to virtual memory;
- **Void rvm_unmap(rvm_t rvm, void *segbase) -** Unmaps a segment from memory;
- **void rvm_destroy(rvm_t rvm, const char *segname) -** Destroys a segment from a given rvm directory**;**
- **trans_t rvm_begin_trans(rvm_t rvm, int numsegs, void **segbases) -** initialises a transaction guaranteeing that no other transaction is modifying the same segment. One transaction is capable of modifying many segments at once**.**
- **void rvm_about_to_modify(trans_t tid, void *segbase, int offset, int size) -** specifies details about the modifications planned by the transaction. Every time a transaction call this function, a region structure is created and pushed into the region list owned by the transaction.
- **void rvm_commit_trans(trans_t tid) -** Writes all modifications to the log file, and releases all segments associated with the transaction, and well as perform deletion of mallocs used by the transaction**.**
- **void rvm_abort_trans(trans_t tid) -** Aborts a transaction restoring the segments to its previous state, before the transaction started. Also performs segment release and deletes memory allocated for the transaction**.**
- **void rvm_truncate_log(rvm_t rvm) -** Periodically shrinks log file, pushing data into other files in the system. The truncation guarantees that only transactions that successfully completed all its modifications are flushed into persistent files.

**API problems**

One problem that could have been better handled by the API is garbage collection. There are too many instances in which malloc was used, and not tracked well. There are key places in which a malloc can be freed, and because I could not find these spots, I simply decided not to free some memory, because I would break the code.