6.033 Computer System Engineering

Spring 2009

Hands-on 5: Write Ahead Log System

Intro to wal-sys

This hands-on assignment will give you some experience using a Write Ahead Log (WAL) system. This system corresponds to the WAL scheme described in Section 9.3 of the course notes. You should carefully read that section before attempting this assignment. You can do this hands-on on any computer that has a Perl language interpreter, but we will be able to answer your questions more easily if you run this on an Athena workstation. You can download the WAL system from here (if your browser displays the file in a window instead of saving it, use "File -> Save As" to save the file). The downloaded file is a Perl script named *wal-sys*. Before trying to run it, change its permissions to make it executable, for example by typing:

```
athena% chmod +x wal-sys
```

The *wal-sys* script can be run as follows:

```
athena% wal-sys [-reset]
```

Alternatively, you can run the script as:

```
athena% perl ./wal-sys [-reset]
```

Wal-sys is a simple WAL system that models a bank's central database, implementing redo logging for error-recovery. *Wal-sys* creates and uses two files, named LOG and DB, in the current working directory. The "LOG" file contains the log entries, and the "DB" file contains all of the installed changes to the database.

After you start *wal-sys*, you can enter commands to manage recoverable actions and accounts. There are also commands to simulate a system crash and to print the contents of the "LOG" and "DB" files. All the commands to *wal-sys* are case sensitive. Since *wal-sys* uses the standard input stream, you can use the system in batch mode. To do this, place your commands in a file ("cmd.in" for example) and redirect the file to *wal-sys*'s standard input:

```
athena% wal-sys -reset < cmd.in.
```

When using batch mode, make sure that each command is followed by a newline character (including the last one).

When you restart *wal-sys*, it will perform a log-based recovery of the "DB" file using the "LOG" file it finds in the current working directory. The **-reset**

option tells *wal-sys* to discard the contents of any previous "DB" and "LOG" files so that it can start with a clean initial state.

Commands interpreted by wal-sys

The following commands are used for managing recoverable actions and accounts:

- **`begin action_id`**
  Begin a recoverable action denoted by *action_id*. The *action_id* is a positive integer that uniquely identifies a given recoverable action.

- **`create_account action_id account_name starting_balance`**
  Create a new account with the given *account_name* and *starting_balance*. The first argument specifies that this operation is part of recoverable action *action_id*. The *account_name* can be any character string with no white spaces.

- **`credit_account action_id account_name credit_amount`**
  Add *credit_amount* to *account_name*'s balance. This command logs the credit and holds it in a buffer until an **`end`** command is executed for recoverable action *action_id*.

- **`debit_account action_id account_name debit_amount`**
  Reduce *account_name*'s balance by *debit_amount*. Like **`credit`**, this command logs the debit and holds it in a buffer until an **`end`** command is executed for recoverable action *action_id*.

- **`commit action_id`**
  Commit the recoverable action *action_id*. This command logs a commit record.

- **`checkpoint`**
  Log a checkpoint record.

- **`end action_id`**
  End recoverable action *action_id*. This command installs the results of recoverable action *action_id* to the "DB". It also logs an end record.

The following commands help us understand the dynamics of the WAL system:

- **`show_state`**
  Print out the current state of the database. This command displays the contents of the "DB" and "LOG" files.

- **`crash`**
  Crash the system. In this hands-on, we are only concerned about crash recovery, so this is the only command we will use to exit the program.

Using wal-sys

Start *wal-sys* with a reset:

```
athena% wal-sys -reset
```

and run the following commands (sequence 1):

```
begin 1
create_account 1 studentA 1000
commit 1
end 1
begin 2
create_account 2 studentB 2000
begin 3
create_account 3 studentC 3000
credit_account 3 studentC 100
debit_account 3 studentA 100
commit 3
show_state
crash
```

Wal-sys should print out the contents of the "DB" and "LOG" files, and then exit.

Use a text editor to examine the "DB" and "LOG" files and answer the following questions (do not run *wal-sys* again until you have answered these questions):

Question 1: *Wal-sys* displays the current state of the database contents after you type **`show_state`**. Why doesn't the database show *studentB*?

Question 2: When the database recovers, which accounts should be active, and what values should they contain?

Question 3: Can you explain why the "DB" file does not contain a record for *studentC* and contains the pre-debit balance for *studentA*?

Recovering the database

When you run *wal-sys* without the **`-reset`** option it recovers the database "DB" using the "LOG" file. To recover the database and then look at the results, type:

```
athena% wal-sys
> show_state
> crash
```

Question 4: What do you expect the state of "DB" to be after *wal-sys* recovers? Why?

Question 5: Run *wal-sys* again to recover the database. Examine the "DB" file. Does the state of the database match your expectations? Why or why not?

Question 6: During recovery, *wal-sys* reports the *action_id*s of those recoverable actions that are "Losers", "Winners", and "Done". What is the meaning of these categories?

Checkpoints

Start *wal-sys* with a reset:

```
athena% wal-sys -reset
```

and run the following commands (sequence 2):

```
begin 1
create_account 1 studentA 1000
commit 1
end 1
begin 2
create_account 2 studentB 2000
checkpoint
begin 3
create_account 3 studentC 3000
credit_account 3 studentC 100
debit_account 2 studentB 100
commit 3
show_state
crash
```

Note: the remainder of this assignment is only concerned with sequence 2. We will ask you to crash and recover the system a few times, but you should not run the sequence commands again. (Also note that in sequence 2, the command **debit_account 2 studentB 100** refers to action_id 2, not action_id 3! This is not a typo).

Question 7: Why are the results of recoverable action 2's `create_account 2 studentB 2000` command not installed in "DB" by the `checkpoint` command on the following line?

Examine the "LOG" output file. In particular, inspect the CHECKPOINT entry. Also, count the number of entries in the "LOG" file. Run *wal-sys* again to recover the database.

Question 8: How many lines were rolled back? What is the advantage of using checkpoints?

Note down the *action_id*s of "Winners", "Losers", and "Done". Use the `show_state` command to look at the recovered database and verify that the

database recovered correctly. Crash the system, and then run *wal-sys* again to recover the database a second time.

Question 9: Does the second run of the recovery procedure (for sequence 2) restore "DB" to the same state as the first run? What is this property called?

Question 10: Compare the *action_id*s of "Winners", "Losers", and "Done" from the second recovery with those from the first. The lists are different. How does the recovery procedure guarantee the property from Question 9 even though the recovery procedure can change? (Hint: Examine the "LOG" file).

Optional: Wal-sys has a hitherto unmentioned option: if you type **`wal-sys -undo`** it will perform undo logging and undo recovery. Try the above sequences again with undo logging to see what changes.