MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
6.826 *Principles of Computer Systems*

February 8, 2000 **Due: Tuesday, February 15, 2000**

Problem Set 2

Instructions: There are **four** *problems in this problem set; please turn in each problem on a separate sheet of paper. Also give the amount of time you spend on each problem.*

Problem 1 (Memory Representation)

Consider a *sparse* representation of read/write memory[1]. We store a sequence of `(D, SEQ A)` pairs mapping data values to the addresses at which they are stored.

(a) Write a module `SparseMemory` that implements `Memory`[2] using this idea.

(b) Write an abstraction function AF that maps `SparseMemory` to `Memory`.

(c) Using AF, provide an informal argument that `SparseMemory` implements `Memory`.

Problem 2 (Cache Replacement Policies)

The procedure `BufferedDisk.MakeCacheSpace` in Handout 7 is undefined. Implement this procedure, using the following cache-replacement policies:

(a) LRU (Least-recently used)

(b) FIFO (First-in, first-out)

You may change other portions of the `BufferedDisk` module in order to implement these policies. Try to make the fewest changes possible.

For each of the two cache-replacement strategies you implemented, give a short English argument that the change in implementation preserves the abstraction function from `BufferedDisk` to `BDisk` given in Handout 7.

Problem 3 (Read-ahead Caching)

A *read-ahead* disk caching algorithm works as follows. When `read` is called and a cache miss occurs, we read from the disk n more blocks than we have to, and evict the cache as we would normally. The assumption is that given we have read block x, we will likely next read block $x + 1$, which would necessarily generate a cache hit.

Consider the `BufferedDisk` implementation of `BDisk` given in Handout 7.

(a) Based on `BufferedDisk`, write a new implementation of `BDisk` called `RADisk` that uses the read-ahead algorithm (with $n = 8$) described above. Do not include the procedures of `BufferedDisk` that are unchanged.

[1] This "inverted" representation would be good for a small-sized data set.
[2] See Handout 5.

(b) In order to write an abstraction function from `RADisk` to `BufferedDisk`, do you need to change the abstraction function from `BufferedDisk` to `BDisk` given in Handout 7? If so, make the appropriate changes. In either case, give a short English argument that there exists an abstraction function from `RADisk` to `BDisk`.

Problem 4

Ben Bitdiddle has been happily using the `BufferedDisk` implementation for a year now, but finds that his database application has exhausted the available space on any single physical disk.

(a) Using the `BufferedDisk` implementation, build an implementation of `Disk` called `DiskArray` that solves Ben's space problem by combining N physical disks into one large virtual disk N times bigger than one physical disk.

(b) After twenty years, advances in storage technology allow just one disk to fulfill Ben's storage needs, but now he needs to maintain different *versions* of the disk information. He comes to you with the following specification for a versioning disk:

```
CLASS VersioningDisk EXPORT Byte, Data, DA, E, DBSize, read, write,
  check, snap, revert, Crash =

% all type declarations, routines in Disk included here.
% note that the routines in Disk use the variable disk; we modify that
% variable to achieve the effect of versioning.  We only allow M
% versions to exist at once.

CONST M: Int := 10

TYPE Version = Int
  Disks = Version -> Dsk

VAR disks := Disks{0 -> disk}
  version := 0

APROC snapshot() -> Version = <<
  version := version + 1;
  disks := restrict(disks{version -> disk}, (version - M)..version);
  RET version >>

APROC revert(version) RAISES (VersionException) = <<
  disks!version =>
    disk := disks(version)
  [*] RAISE VersionException >>

END VersioningDisk
```

Write a simple implementation of Ben's specification for versioning disks that uses the `DiskArray` and an auxiliary data structure to map version numbers to the physical disks in the disk array.

(c) Ben realizes that every time he calls the `snapshot` method of your implementation in Part (b), a lot of time is spent copying the disk state. Modify your implementation to address this issue, and eliminate any unnecessary copying.