CMSC 23000	Operating Systems	Homework 5
Autumn 2006		Due November 21

1. Suppose that your virtual memory hardware has the following page table entry format:

```
typedef struct {
    unsigned   hi20 : 20;  // high 20 bits of page address
    unsigned   r : 1;      // set if read access is allowed
    unsigned   w : 1;      // set if write access is allowed
    unsigned   valid : 1;  // set if page is resident in memory
    unsigned   unused : 9;
} pte_t;
```

   In this system, a memory access can generate a memory fault for several reasons

   (a) the accessed address is not resident (*i.e.*, the `valid` bit is 0).

   (b) A read from a page with a 0 `r` bit.

   (c) A write to a page with a 0 `w` bit.

   Describe how you might implement the NFU (*Not Frequently Used*) algorithm with aging using the memory-protection mechanism and the unused bits of the page-table entry. Your solution should how you deal with each of the above faults, how you pick victim pages, and any periodic tasks that are required.

   Note: you may assume that user programming model does not support read-only or write-only access to memory.

2. Consider the following simplified in-memory representation of an Unix-style inode:

```
#define BLOCK_SZB 512     // the size of a disk block in bytes
#define IDS_PER_BLOCK 128 // number of block ids in a block

typedef unsigned int Block_t;

typedef struct {
    dev_t    dev;
    uid_t    uid;
    gid_t    gid;
    offset_t length;
    Block_t  direct[10];    // direct access blocks
    Block_t  indirect1;     // one-level of indirection
    Block_t  indirect2;     // two-levels of indirection
    Block_t  indirect3;     // three-levels of indirection
} INode_t;
```

   (a) How large is the largest file that can be represented using a single inode? Give your answer in blocks.

(b) Assuming that block IDs are four bytes and that the following function returns a pointer to the in-memory cache of a disk block,

```
void *DiskBlock (dev_t dev, Block_t id);
```

implement the following procedure for mapping a file offset to the block containing it

```
Block_t OffsetToBlock (INode_t *inode, offset_t off);
```