CMSC 23000 | **Operating Systems** | **Homework 4**
Autumn 2006 | | **Due November 14**

1. Often data is stored in encoded form on disk (*e.g.*, encrypted or compressed) and is decoded when it is read in. Your job is to implement a *lazy* decode mechanism for Linux that decodes the file in page-sized chunks on demand (*i.e.*, when the in-memory data is referenced). The interface to encoded file is as follows:

```
#define PAGE_SZB    4096

typedef int (*decoder_t)(int fd, void *dst, int nbytes);

typedef struct {
    offset_t    off;       // offset of page data in file
    size_t      szb;       // size of page data in bytes
} page_t;

typedef struct {
    int         fd;        // file descriptor for encoded file
    decoder_t   decodeFn;
    void        *base;     // base address of decoded data; will
                           // be page aligned
    int         nPages;    // number of pages in the file
    page_t      *map;      // map from page index to file offset
} encoded_file_t;
```

   where the `decoder_t` type is a function that reads and decodes the file's data. You should implement the following function:

```
void init (encoded_file_t *f);
```

   which will be passed an *initialized* `encoded_file_t` structure. Your implementation will require at least one additional function.

   Hint: On a Linux system, read the man pages for the `mprotect` and `sigaction` system calls.

2. Consider the following page-reference string:

   1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

   Fill in the following table with the number of page faults for the given physical memory sizes and replacement algorithms.

Number of Frames	**OPT**	**FIFO**	**LRU**	**LFU**
3				
4				
5				

   Assume that initially there are no resident pages.