Universität des Saarlandes
Lehrstuhl für Softwaretechnik
Prof. Andreas Zeller, Stephan Neuhaus
Design of Secure Software Systems
Exercise 8

Total number of points obtainable: 20, plus 8 bonus points.

1 Handles

(12 points) In order to implement handles, you need two things: a mechanism to associate a handle with an object and a mechanism to turn a newly created object into a handle. The following exercise explores how this might be done. To do this, it implements buffer handles. A buffer handle is a handle to a buffer of fized size. The operations that are offered on this buffer handle make sure that no overflow can happen and that memory is properly zeroized before it is freed.

Write the implementation that goes with the following specification (also available at `http://www.st.cs.uni-sb.de/edu/secdesign/coding-samples.tar`). Lock buffers using *mlock*(2) and release them using *munlock*(2).

```c
#include <stdlib.h>

typedef int buffer_handle_t;

/* Errors that can occur when operating on buffer handles. The prefix
   'bh' exists to disambiguate this enum from other enums. */
typedef enum {
  bh_no_error,          /* No error has occurred */
  bh_invalid_handle,    /* This handle isn't known */
  bh_overflow,          /* Operation would overflow */
  bh_errno,             /* Library error, look in errno */
} bh_error_t;


/* Create new buffer.
 *
 * The buffer returned by this call will be locked in memory and will
 * not be swapped out. The buffer will contain only null bytes.
 *
 * Inputs
 *   size       Size of new buffer, in bytes
 *
 * Outputs
 *   handle     New buffer handle. This handle is valid only if
 *              returned error code is 'bh_no_error'.
 *
 * Errors
 *   bh_errno   A library call returned an error. See 'errno' for more
 *              detailed information.
 */
extern bh_error_t buffer_create(size_t size, buffer_handle_t *handle);

/* Release a buffer.
 *
 * This call releases a buffer that was previously allocated with
 * buffer_create(). It is an error to use the handle in functions
 * that take a handle as an input argument afterwards. The buffer is
 * zeroized befire returning its memory. If there are no more buffers
 * using the memory pages containing this buffer, the pages are
 * unlocked.
 *
 * Inputs
 *   handle     Handle of buffer to release
```

```
 *
 * Outputs
 *    none
 *
 * Errors
 *    bh_invalid_handle
 *              The HANDLE parameter is not currently associated with a
 *              valid handle.
 *    bh_errno  A library call returned an error. See 'errno' for more
 *              detailed information.
 */
extern bh_error_t buffer_release(buffer_handle_t handle);

/* Copy bytes into buffer.
 *
 * Inputs
 *    handle    Handle of buffer to copy into
 *    buffer    Contains bytes to copy into HANDLE
 *    size      Number of bytes to copy from BUFFER into HANDLE
 *    offset    Start copy at HANDLE byte OFFSET
 *
 * Outputs
 *    none
 *
 * Errors
 *    bh_invalid_handle
 *              The HANDLE parameter is not currently associated with a
 *              valid handle.
 *    bh_overflow
 *              The input parameters would cause bytes to be written
 *              beyond the end of the buffer. The operation is
 *              declined. If this error is returned, the contents of
 *              the buffer associated with HANDLE are unchanged.
 *    bh_errno  A library call returned an error. See 'errno' for more
 *              detailed information.
 */
extern bh_error_t buffer_copy_in(buffer_handle_t handle, void* buffer,
                                 size_t size, size_t offset);

/* Copy bytes out of buffer.
 *
 * Inputs
 *    handle    Handle of buffer to copy into
 *    buffer    Contains buffer into which bytes are copied
 *    size      Number of bytes to copy from HANDLE into BUFFER
 *    offset    Start copy at HANDLE byte OFFSET
 *
 * Outputs
 *    none
 *
 * Errors
 *    bh_invalid_handle
 *              The HANDLE parameter is not currently associated with a
 *              valid handle.
 *    bh_overflow
 *              The input parameters would cause bytes to be read
 *              beyond the end of the buffer associated with
 *              HANDLE. The operation is declined. If this error is
 *              returned, the contents of BUFFER are unchanged.
 *    bh_errno  A library call returned an error. See 'errno' for more
 *              detailed information.
 */
extern bh_error_t buffer_copy_out(buffer_handle_t handle, void* buffer,
                                  size_t size, size_t offset);
```

2 Fail Safe

(8 points) This code has some problems with error handling. Rewrite it so that it fails safely. This code is available for download at `http://www.st.cs.uni-sb.de/edu/secdesign/coding-samples.tar`. The functions *warning*(), *fatal*(), and *fatal_with_errno*() are not used in the code, but should be used by you.

```c
#include <stdarg.h>
#include <stdio.h>
#include <string.h>

/* The name of the program. */
static const char* program_name;

/* Prints a warning message. This function takes a printf(3) format
   string and argument list, i.e., you can call this function like
   printf. */
static void warning(const char* format, ...) {
  va_list args;

  fprintf(stderr, "%s: warning: ", program_name);

  va_start(args, format);
  vfprintf(stderr, format, args);
  va_end(args);
}

/* Prints an error message and exits with an error indication. This
   function takes a printf(3) format string and argument list, i.e.,
   you can call this function like printf. */
static void fatal(const char* format, ...) {
  va_list args;

  fprintf(stderr, "%s: fatal error: ", program_name);

  va_start(args, format);
  vfprintf(stderr, format, args);
  va_end(args);

  exit(1);
}

/* Like fatal() above, prints an error message and exits with an error
   indication. This function takes a printf(3) format string and
   argument list, i.e., you can call this function like printf.        In
   addition to printing the error message, it also looks at errno and
   prints a message describing the error. */
static void fatal_with_errno(const char* format, ...) {
  va_list args;

  fprintf(stderr, "%s: fatal error: ", program_name);

  va_start(args, format);
  vfprintf(stderr, format, args);
  va_end(args);

  perror(":");
  exit(1);
}


/* For an argument of the form "a/b/c", returns "c". If the argument
   contains no slash, the argument itself is returned. */
static const char* basename(const char* filename) {
  const char* slashpos = strrchr(filename, '/');
  return slashpos != 0 ? slashpos + 1 : filename;
}
```

```
int main(int argc, const char* argv[]) {
  FILE *input_file = fopen(argv[1], "r");
  FILE *output_file = fopen(argv[2], "w");
  int c;

  program_name = basename(argv[0]);

  while ((c = getc(input_file)) != EOF)
    putc(c, output_file);

  return 0;
}
```

3 Fail Secure

(8 bonus points) The following excerpt is pseudo code that you cannot compile and run. Nevertheless, it is apparent that it has some security problems. Try to rewrite the code so that it fails securely. This code is available for download at `http://www.st.cs.uni-sb.de/edu/secdesign/coding-samples.tar`

```
static const unsigned char fallback_key[] = { 0x01, 0x02, 0x03, 0x04 };

static const char *aes_name = "AES"; /* Name of AES encryption function */

static const cipher_t *aes = ...; /* Encryption with AES */
static const cipher_t *none = ...; /* No encryption */

/* Return 1 if the cipher with that name is available, 0 otherwise. */
extern int cipher_available(const char* cipher_name);

void encrypt(const unsigned char* plaintext, const char* password,
             char* ciphertext, size_t ciphertext_size,
             size_t* ciphertext_length) {
  unsigned char key[MAX_KEY_SIZE];
  const cipher_t *cipher;

  if (is_good_password(password))
    make_key_from_password(password, key);
  else if (sizeof(fallback_key) < MAX_KEY_SIZE) {
    memcpy(key, fallback_key, sizeof(fallback_key));
    memset(key + sizeof(fallback_key), '\0', MAX_KEY_SIZE - sizeof(fallback_key));
  } else
    memcpy(key, fallback_key, MAX_KEY_SIZE);

  if (cipher_available(aes_name)) /* Try Advanced Encryption Standard */
    cipher = aes;
  else
    cipher = none;

  encrypt_buffer(plaintext, cipher, key, ciphertext,
                 ciphertext_size, ciphertext_length);
}
```