#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <sys/mman.h>

int main(int argc, char \*\*argv)

{

struct stat statbuf;

struct stat keybuf;

char buffer [20];

int key;

int data;

int output;

int count;

char ans;

int \* buf;

FILE \* keyfile;

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <sys/mman.h>

int main(int argc, char \*\*argv)

{

struct stat statbuf;

struct stat keybuf;

char buffer [20];

int key;

int data;

int output;

int count;

char ans;

int \* buf;

FILE \* keyfile;

FILE \* sourcefile;

FILE \* destfile;

if(geteuid() !=0)

{

printf("Root access is required to run this program\n\n");

exit(0);

}

if(argc<4)

{

printf("\n");

printf(" OTP 1.0 \n\n");

printf(" This program encrypts a file using a random key\n");

printf(" and generates an output file with the resulting\n");

printf(" cipher. Decryption is achieved by running the\n");

printf(" output file as source file with the same key.\n\n");

printf(" WARNING: The security of the encryption provided\n");

printf(" by this program is entirely dependent on the key\n");

printf(" file. The keyfile should meet the requirements\n");

printf(" below:\n");

printf(" - Be of the same size or larger than the\n");

printf(" source file.\n");

printf(" - Be completely random, preferably generated by \n");

printf(" a Hardware Random Number Generator.\n");

printf(" - NEVER be reused!\n\n");

printf(" The author takes no responsibility for use of\n");

printf(" this program. Available under GNU General Public\n");

printf(" Licence v.2\n\n");

printf(" USAGE: OTP <source file> <output file> <keyfile>\n\n");

return (0);

}

/\* Check number of arguments. \*/

if(argc>4)

{

printf("Too many arguments.\n");

printf("USAGE: OTP <source file> <output file> <keyfile>\n");

exit(1);

}

/\* Allocate memory required by processes \*/

buf = (int\*) malloc (sizeof(int));

if (buf == NULL)

{

perror("Error");

exit(1);

}

/\* Lock down pages mapped to processes \*/

printf("Locking down processes...\n\n");

if(mlockall (MCL\_CURRENT | MCL\_FUTURE) < 0)

{

perror("mlockall");

exit (1);

}

/\* Check if sourcefile can be opened. \*/

if((sourcefile = fopen(argv[1], "rb"))== NULL)

{

printf("Can't open source file\n");

perror("Error");

printf("USAGE: OTP <source file> <output file> <keyfile>\n");

exit (1);

}

/\* Get size of sourcefile \*/

fstat(fileno(sourcefile), &statbuf);

/\* Check if keyfile can be opened. \*/

if((keyfile = fopen(argv[3], "rb"))== NULL)

{

printf("Can't open keyfile.\n");

perror("Error");

printf("USAGE: OTP <source file> <output file> <keyfile>\n");

exit(1);

}

/\* Get size of keyfile \*/

fstat(fileno(keyfile), &keybuf);

/\* Check if keyfile is the same size as, or bigger than the sourcefile \*/

if((keybuf.st\_size) < (statbuf.st\_size))

{

printf("Source file is larger than keyfile.\n");

printf("This significantly reduces cryptographic strength.\n");

printf("Do you wish to continue? (Y/N)\n");

fgets(buffer, 20, stdin);

sscanf(buffer, "%c", &ans);

if(ans == 'n' || ans == 'N')

{

exit (1);

}

if(ans == 'y' || ans == 'Y')

{

 printf("Proceeding with Encryption/Decryption.\n");

 }

else

{

printf("No option selected. Exiting...\n");

exit (1);

}

}

/\* Check if destfile can be opened. \*/

if((destfile = fopen(argv[2], "wb"))== NULL)

{

printf("Can't open output file.\n");

perror("Error");

exit(1);

}

/\* Encrypt/Decrypt and write to output file. \*/

while(count < (statbuf.st\_size))

{

key=fgetc(keyfile);

data=fgetc(sourcefile);

output=(key^data);

fputc(output,destfile);

count++;

}

/\* Close files. \*/

fclose(keyfile);

fclose(sourcefile);

fclose(destfile);

printf("Encryption/Decryption Complete.\n\n");

/\* delete Source file option. \*/

printf("Do you wish to delete the source file? (Y/N)\n");

fgets(buffer, 20, stdin);

sscanf(buffer, "%c", &ans);

if(ans == 'y' || ans == 'Y')

{

 if ( remove(argv[1]) == 0)

 {

 printf("File deleted successfully.\n");

 }

 else

 {

 printf("Unable to delete the file.\n");

 perror("Error");

 exit(1);

 }

}

/\* delete keyfile option. \*/

printf("Do you wish to delete the keyfile? (Y/N)\n");

fgets(buffer, 20, stdin);

sscanf(buffer, "%c", &ans);

if(ans == 'y' || ans == 'Y')

{

 if ( remove(argv[3]) == 0)

 {

 printf("File deleted successfully.\n");

 }

 else

 {

 printf("Unable to delete the file.\n");

 perror("Error");

 exit(1);

 }

}

/\* cleanup \*/

printf("Releasing memory.\n");

free (buf);

return(0);

}