EECS 482 Final (Fall 1998)

You will have 110 minutes to work on this exam, which is closed book. There are 4 problems on 14 pages.

Read the entire exam through before you begin working. Work on those problems you find easiest first. Read each question carefully, and note all that is required of you. Keep your answers clear and concise, and carefully state all of your assumptions.

You are to abide by the University of Michigan/Engineering honor code. Please sign below to indicate that you have abided by the honor code on this exam.

Honor code pledge: I have neither given nor received aid on this exam.

Signature:___________________________

Name:___________________________________

Uniqname: ________________________________

Problem 1	______ out of 30
Problem 2	______ out of 20
Problem 3	______ out of 40
Problem 4	______ out of 10
Total:	**_______ out of 100**

1. Security (parts A-C) (approx. 30 minutes)

You are the owner of a bank and would like to increase the security of customer-teller interactions using private-key encryption. Your goal is to assure the teller that the customer at the teller window is who he/she claims to be, and to assure the customer that the teller is the authentic bank teller (there’s only one teller). Assume that each customer is given a unique private key, known only to them and the teller. Everyone carries a calculator to compute encryption and decryption.

Consider the following protocol for authenticating both customer and teller to each other.

Step 1. Customer sends the following message to the teller:

```
<name>, encrypt(<key>, “I am <name>, <random string>”)
```

notes:

`<name>` is the name of the customer

`<key>` is the private key for this customer

`<random string>` is a random string generated by the customer

`encrypt(<key>, X)` means encrypt the text `X` using the key value `<key>`

Step 2. Teller receives message and decrypt’s the encrypted part using <name>’s key. Teller checks that “I am <name>” appears in the decrypted message. If it does not appear, the teller stops the protocol.

Step 3. Teller sends the following message to the customer (without encryption):

```
<random string>
```

Step 4. Customer receives message and checks that the random string received from the teller is the same random string sent in Step 1.

A. Answer the following questions about the parts of the message in Step 1:

Why must `<name>` be sent unencrypted?

Why must “`I am <name>`” be present?

Why must “`I am <name>, <random string>`” be encrypted?

B. The above protocol does *not* meet its goal of assuring both customer and teller of the other's identity. Describe the attack(s) a villain could use to break the protocol. After the protocol completes, can the teller be sure the customer is who he/she claims to be? Can the customer be sure the teller is the real teller? Justify your answers (be specific).

C. Fix the protocol so that it does assure both customer and teller of the other’s identity. The fixed protocol must continue to use only private-key encryption and must use no more than two messages. Both the teller and the customer may keep additional state, but they should keep as little as possible.

2. Remote Procedure Call (approx. 20 minutes)

You have a program that consists of two functions: `main` and `compute`. `main` calls `compute` with two parameters. The first argument is a single character; the second argument is a pointer to a single character. The return value of compute is a single character. Here is the function prototype for compute:

```
char compute(char a, char *b);
```

You would like to modify the program so `main` and `compute` run on separate computers. You will use RPC to hide the fact that the program is running on two computers.

Your job for this question is to write (in C) the client-side and server-side RPC stub functions for `compute`. Assume the two computers are already connected via TCP sockets (the socket number on both client and server is `sock`). Assume the two computers are both Sun workstations. Your functions should be as simple as possible (don't worry about performance).

Use `send` and `recv` to communicate (you may assume `send` and `recv` will not return an error). Here are the function prototypes for `send` and `recv` (`flags` should be 0).

```
int send(int s, const char *msg, int len, int flags);
int recv(int s, char *buf, int len, int flags);
```

Client-side stub function:

Server-side stub function:

3. File Systems (parts A-E) (approx. 40 minutes)

Your job is to write a file system for a magnetic tape drive. The block size of the tape is 256 bytes. The tape drive supports the following operations:

- Seek: Seek moves the tape to a new position and must be used before every non-sequential access. All seeks take 10 seconds. Seeking beyond the current end of the tape will return an end-of-tape error.
- Read: Once the tape is at the right position, it can be read at a transfer rate of 1 MB/s. Reading beyond the current end of the tape will return an end-of-tape error.
- Append: The only way to write new data to the tape is by adding data to the end of the tape. Data can be written at 1 MB/s.

The file system you wish to store on this device has the following characteristics:

- The maximum file size is 12 KB. The file system has a fixed number of files (2^{16}), all of which are initially zero-length. Files are named by a number between 0 and 2^{16}-1.
- The file system supports the following operations: (1) read an existing block of a file, (2) modify an existing block of a file, (3) append a block to a file. Keep only the metadata you need to support these operations. Assume the computer has enough memory to cache all metadata.
- Reads and modifies show no temporal or spatial locality.
- Performance is the most important metric; you can assume that the workload will never fill the tape.
- The file system may not lose data on a crash, but the speed of post-crash recovery is of no concern. Assume that individual block writes are atomic, and that physical memory is lost on a crash.

Answer the following questions about your file system. Justify all your answers. Read the entire question before you answer any portion. You may find it helpful to use your answers for earlier parts of the question as components to answers of later parts.

A. Consider the initial state where all files are zero-length. Suppose the machine crashes before appending to any files. Describe the contents of the tape and the recovery process.

B. What allocation scheme does your file system use, and why? Hint: remember the allocation schemes we covered in class (contiguous, indexed, multi-level indexed). Describe the structure of a single file header. How many blocks are required to store the file header for a maximum-size file?

C. Suppose your file system is in the initial state (where all files are zero-length). Now, a user requests a one-block append to one of the files. What, exactly, must be written to the tape, and in what order? Suppose the system crashes any time during or after this operation, but before the next operation begins. Describe the recovery process for each possible case.

D. Consider the case where N operations—a mix of reads, appends, and modifications—have occured without an intervening crash. Suppose the next operation is a read. How is it satisfied?

E. Consider the case where N operations—a mix of reads, appends, and modifies—have occurred without an intervening crash. Suppose the next operation is a modification. How is it satisfied? Suppose the machine crashes at any point during this operation, but before the next one begins. Describe the recovery process for each possible case.

4. Systems Programming (approx. 10 minutes)

You want to add round-robin scheduling (with a quanta of 2 ms) to your Project 1 thread library. To do this, you start by calling `setitimer`. Write the code you would add to your thread library to cause a SIGALRM every 2 ms. We will grade both the correctness and the quality of your code. Comments will not be graded but may help us understand your code. The manual pages for the `setitimer` and `gettimeofday` library calls are included for your information.

```
NAME
     getitimer, setitimer - get or set value of interval timer

SYNOPSIS
     #include <sys/time.h>

     int getitimer(int which, struct itimerval *value);

     int setitimer(int which, const struct itimerval *value,
          struct itimerval *ovalue);

DESCRIPTION
     The system provides each process with two interval  timers,
     defined  in sys/time.h.  The getitimer() function stores the
     current value of the  timer  specified  by  which  into  the
     structure  pointed  to  by value.  The setitimer() call sets
     the value of the timer  specified  by  which  to  the  value
     specified  in  the  structure  pointed  to  by value, and if
     ovalue is not NULL, stores the previous value of  the  timer
     in the structure pointed to by ovalue.

     A timer value is defined by  the  itimerval  structure  (see
     gettimeofday(3C)  for  the  definition  of  timeval),  which
     includes the following members:

             struct timeval    it_interval;    /* timer interval */
             struct timeval    it_value;       /* current value */

     The it_value member indicates the time  to  the  next  timer
     expiration.   The it_interval member specifies a value to be
     used in reloading it_value when the timer expires.   Setting
     it_value  to  0 disables a timer, regardless of the value of
     it_interval.  Setting it_interval  to  0  disables  a  timer
     after its next expiration (assuming it_value is non-zero).

The two timers are:
     ITIMER_REAL          Decrements in  real  time.  A  SIGALRM
                          signal  is  delivered  when this timer
                          expires.


     ITIMER_VIRTUAL       Decrements in  process  virtual  time.
                          It  runs only when the process is exe-
                          cuting.   A   SIGVTALRM   signal    is
                          delivered   when   it  expires

RETURN VALUES
     If the calls succeed, 0 is returned.  If an error occurs, -1
     is returned, and an error code is placed in the global vari-
     able errno.

ERRORS
     The getitimer() and setitimer() functions will fail if:

     EINVAL         The specified number of  seconds  is  greater
                    than  100,000,000, the number of microseconds
                    is greater than or equal to 1,000,000, or the
                    which argument is unrecognized.

     EACCES         Either an unbound Solaris thread or  a  POSIX
                    thread  in local scheduling scope with a flag
                    other than ITIMER_REAL called setitimer().
```

```
NAME
     gettimeofday, settimeofday - get or set the date and time

SYNOPSIS
     #include <sys/time.h>

     int gettimeofday(struct timeval *tp, void * );

     int settimeofday(struct timeval *tp, void * );

DESCRIPTION
     The gettimeofday()  function  gets  and  the  settimeofday()
     function  sets the system’s notion of the current time.  The
     current  time  is   expressed   in   elapsed   seconds   and
     microseconds since 00:00 Universal Coordinated Time, January
     1, 1970.  The resolution of the  system  clock  is  hardware
     dependent;  the time may be updated continuously or in clock
     ticks.

     The  tp  argument  points  to  a  timeval  structure,  which
     includes the following members:

          long    tv_sec;    /* seconds since Jan. 1, 1970 */
          long    tv_usec;   /* and microseconds */

     If tp is a null pointer, the current time information is not
     returned or set.

     The second argument  to  gettimeofday()  and  settimeofday()
     should be a NULL pointer.

     Only the super-user may set the time of day.

RETURN VALUES
     A -1 return value indicates that an error occurred and errno
     has been set.

ERRORS
     The following error codes may be set in errno:

     EINVAL              tp specifies an invalid time.

     EPERM               A user other than  the  privileged  user
                         attempted to set the time or time zone.
```