$\boxed{\text{Burt Rosenberg}}$

Problem Set 5

OUT: 8 NOVEMBER, 1994
DUE: 22 NOVEMBER, 1994

Assignment

Write a C program to play the game of Rock Paper Scissors against another (program) opponent. Play many games against who is ever logged in, keep score and records. The most winning games wins. This assignment will exercise file sharing and communication protocols between programs. You will learn about file locking and public key cryptography. It serves as a model for some important new issues in programming which arise from the importance of parallelism and networks.

Players

Players are named A and B. The A player goes first, proposing a game. The B player then joins. Once A and B are so matched, play proceeds to the end of the single game.

Files

Communication between players is through a common file directory, the *spool directory.* Players must lock the spool directory before changing the contents of the directory, and must unlock the spool directory immediately afterwards. This is accomplished by creating, or failing to create, a common *lock file.* The communication continues through a series of files, the *protocol files*, also called the A, B, C, D and E files, according to the first letter of their names. The A player is must create and delete the A, C and E files, while the B player must create and delete the B and D files.

File Names

Each game has a unique number, the combination of a serial number and the UID's of players A and B. The A through E files derive their names from the letter A through E, the A player's UID and the serial number. The A player chooses a serial number, and it should be unique for player A. The file names are:

$$[\mathbf{A}|\mathbf{B}|\mathbf{C}|\mathbf{D}|\mathbf{E}]\,uid_A\textbf{.}sernum$$

As an example, `A1212.1102001` is the A file of player with UID 1212, and the game has sequence number 1102001.

Outline of Protocol

Any player can be an A player, proposing a game. It does so by creating an A file and waiting for another player to adopt a B position. The B player will respond with a B file. Player A will then create a C file, and delete its A file. In normal play, the protocol will continue in this manner, with players A and B alternating turns of waiting for the other player to post the next file in the protocol and deleting its previously posted file.

The contents of these files exchange information between players A and B. The intent of the A file is for the A player to announce its intent to play and to publish its play in encrypted form The intent of the B file is to announce player B's intent to accept play against A, to identify B to A, to signal that B has read the A file and recorded A's encrypted play, and to publish B's encrypted play. The C file announces to player B that player A has read the B file, has accepted B as opponent for this game, has recorded B's encrypted play, and reveals A's play in unencrypted form. Player B will respond to the C file with a D file. The D file announces to player A that player B has read the C file, accepts that the revealed play agrees with the encrypted play, and reveals player B's play. The final file of the sequence is file E. In response to file D, player A posts file E which signals that player A has read the D file and accepts that the revealed play agrees with the encrypted play. Seeing the E file, player B deletes the D file. Seeing the D file deleted, player A deletes the E file.

The protocol is now finished. The two parties have no further responsibilities to each other. It is important to observe the correct locking of the spool directory. If two players decide simultaneously to create B files, one will succeed in locking the directory first. The second player will lock the directory and observe the A file already has a responding B file and conclude that it was too late to join as A's opponent.

```
>>> start of protocol <<<
(Player A)         (Player B)
lock
create A
unlock
                    lock
                    create B
                    unlock
lock
create C
delete A
unlock
                    lock
                    create D
                    delete B
                    unlock
lock
create E
delete C
unlock
                    lock
                    delete D
                    unlock
lock
delete E
unlock
>>> end of protocol <<<
```

Directory Locking

Each player temporarily gets exclusive control over the spool directory by locking it. This is done by creating the lock file:

```
/u/1mth322s/e0l47v0m/spool/.lock
```

The player desiring to lock the directory first looks for the presence of the `.lock` file in the directory. It it does not exist, the directory is free to be locked. The player then creates a `.lock` in the spool file. It the create succeeds, the player has successfully locked the directory. If it fails, another user has simultaneously attempted to create a `.lock` file in the spool directory, and that user was judged to have "asked first".

After locking the directory, the player looks at the directory again and confirms the situation. The player than makes the required changes and unlocks the directory. The directory is unlocked simply by deleting the lock file. If the lock file is left behind, all play action will stop.

Glossary of Protocol Elements

To describe the contents of the protocol files in detail, this glossary of their contents is provided.

1. *sernum:* A seven digit integer chosen by player A. It must be unique as far as player A is concerned. When the sernum is combined with player A's UID, it is the unique identifier of the game.

2. uid_A: The UID of player A.

3. uid_B: The UID of player B.

4. cry_A: The encrypted version of player A's choice of Rock, Paper or Scissors.

5. cry_B: The encrypted version of player B's choice of Rock, Paper or Scissors.

6. key_A: The decryption key for cry_A.

7. key_B: The decryption key for cry_B.

8. *ack/nack:* The literal `OK` or `NO` to acknowledge or refuse a communication. Each protocol file ends with one of these.

Protocol

1. The **A**uid_A**.***sernum* file:
   Auid_A**.***sernum***:** cry_A **OK**

2. The **B**uid_A**.***sernum* file:
   Buid_A**.***sernum***:** uid_B cry_B **OK**

3. The **C**uid_A**.***sernum* file:
   Cuid_A**.***sernum***:** uid_B key_A *ack/nack*

4. The $\mathbf{D}uid_A\mathbf{.}sernum$ file:
   $\mathbf{D}uid_A\mathbf{.}sernum\mathbf{:}$ uid_B key_B *ack/nack*

5. The $\mathbf{E}uid_A\mathbf{.}sernum$ file:
   $\mathbf{E}uid_A\mathbf{.}sernum\mathbf{:}$ uid_B *ack/nack*

Error Paths

Rather than acknowledge, a player can negative-acknowledge a message by using `NO` instead of `OK` as the final two letters of a protocol file. When this occurs, the opposing player removes its file, and in response to this, the nack'ing player removes its file. The two players have no further responsibilities to each other.

To avoid cheating, two additional rules apply. A nack'ing C message need not contain a valid key_A. However a nack'ing D message must contain the valid key_B. Else player B can annul any game which it has lost and attempt to hide this loss from player A.

Encryption

Each player chooses a play, then a key to suit the play, then encrypts the key. It is claimed that from the encryption it is difficult to recover the key. Also, it is not possible that two different keys give the same encryption. Therefore, although disclosure of the encrypted key does not make the key public, it is impossible to change the key once the ecryption is revealed.

Choose key_l where $l = A$ or $l = B$ to be an integer in the range of 0 to 1,000,000,006 and so that,

$$key_l \pmod 3 = \begin{cases} 0 & \text{if player } l \text{ is Paper,} \\ 1 & \text{if player } l \text{ is Rock,} \\ 2 & \text{if player } l \text{ is Scissors.} \end{cases}$$

Calculate the encryption of key_l as follows:

$$5^{key_l} = cry_l \pmod{1,000,000,007},$$

where l can be either A or B. For your protection, key_l should be chosen randomly subject to the above constraints, and it should not be too small. The claim of secrecy rests on the difficulty of recovering key_l from cry_l. If key_l is small, an exhaustive search starting from 0 would find it quickly. If key_l it is random, however, the chances of its discovery are but one in a billion. Keys must be in the stated range. Keys outside this range are not legal play.

Keeping Score

There are several ways to cheat if a record of play is not kept. For instance, you can impersonate another player and play both sides to give yourself wins. The programs should keep a game log. A sample entry is,

$$\textit{time-and-date-stamp}\ \mathbf{A}uid_A\mathbf{.}seqnum\ uid_B\ cry_A\ cry_B\ key_A\ key_B\ \textit{ack/nack}$$

where the the time and date stamp is the creation date of the A file for this game, and the ack/nack determines whether the game was not aborted in error.