Assignment 2

Due February 13^{th} 2009

Interprocess communication: shared memory, semaphores

The problem

Electronic voting is big news nowadays. While the issue is controversial, it may be a good idea to look at how to implement an e–voting system. A real–life e–voting system is fairly complex with cryptography being at its core. We will ignore this part (wrong course) and focus solely on the movement of information.

References

1. A technical presentation by LF Cranor
2. The opinion of the leading expert Rebecca Mercuri
3. wikipedia's 2 cents.
4. Commercial provider in Canada
5. A Canadian user of e–voting

The voting software

The e–voting software is made of 3 parts:

Validator: A voter presents to the Validator a key received from from the Election Board. This key is validated, recorded (to prevent duplicate use) and a random vid is generated for this voter.

Booth: The voter now invokes a program that accepts a vid and a vote. The vid and the voter's vote are forwarded to the Tallier.

Tallier: The vote is recorded (together with the vid) and a confirmation number is sent back to Booth (and given to the voter).

When voting is over, the Tallier posts all the votes (vote+vid) to allow voters to check whether the election was fair.

This scheme is not perfect. The Validator may be able to stuff ballots if it successfully guesses keys sent by the Election Board (this can be prevented by monitoring unsuccessful authentication attempts). Also, a conspiracy between the authors of the Validator and the Booth software could compromise the anonymity of the vote. Otherwise, it is not too bad: the outcome cannot be doctored, at least not easily.

Your task is to implement the component called Validator, a standalone program called as follows:

Validator Name key

Name: any sequence of characters.

key: an integer key that was sent to you by the Election Board.

Interpretation of the arguments

It costs millions of dollars to set up a real–life election. In these hard times we will avoid the expenses by trivialising the rules in the following way:

Name: the name is not checked if the key is valid and is used for the first time. Otherwise, the voter (possibly a criminal) is forced to show a real identification etc. This case is not done by the software but by the police.

key: Any integer value divisible by 9 is considered legal provided that it is not present in the global database of used keys.

Examples

Dobo 53217, MickeyMouse 2007 , Crook 2009
Note that the last key is invalid.

You will generate the keys at random. The names are needed anywhere in the assignment; if you really need them, make them Aaaaaaaa, Aaaaaaab, Aaaaaaac, etc.

Some rules

1. If Booth is running, file Booth pid pid will be present in the current directory and will contain the process id of Booth stored as a string. Note that the file has an unusual name.

2. If Booth is not present, your Validator must fork+exec it. Booth will initialise the two semaphores and send a SIGUSR1 to its parent (this signal must be caught or it is fatal).

3. Booth is expecting multiple instances of Validator running concurrently. Your implementation of Validator must work properly in the presence of other Validators (could be your own or somebody else's).

4. Communication between Validator and Booth is done using one shared memory segment identified by the following key:

   ```
   key = ftok( "/dev/null" , code ) ;
   ```

   where code is your personal code (a signup sheet will be available). (Important: you should test your code using a different file name first.)

5. Access to the shared memory area will be guarded by an array of two semaphores obtained using:

   ```
   semid = semget( key , 2 , 0600 ) ;
   ```

   where key is the same key returned by ftok. Note: IPC_CREAT — IPC_EXCL flags have to be thrown in somewhere, but not everywhere.

Beware: using /dev/null and using key twice is not necessarily portable. It might need to be replaced by more politically correct code on some systems (let us hope not).

The shared memory

One shared area is used:

```
struct VID {
    int code ;  // the current state of this entry
    // = 0 empty     (sem 0 == 1)
    // = 1 vid inside   (sem 1 == 0)
    // = 2 confirmation of vid     (sem 1 == 0)
    pid_t pid ; // Validator's pid
    int vid ;   // the vid to be recorded
    int confirmation ;  // passed back from Tallier
} ;
```

The interpretation is rather obvious; the one tricky part is the confirmation: Booth returns a number that only the sending Validator can decode without risk of error.

The role of semaphores

Two semaphores (S_0 and S_1) are used for synchronisation between Validator and Booth:

S_0: It guards the shared memory. When it is down, any process wishing to access the shared memory must wait (inside a semop).

S_1: It guards the contents of the shared memory. When it is down, no process is allowed to to overwrite the shared memory.

A process wishing to write to the shared memory must successfully lower both S_0 and S_1, do the write (without any other activities), and raise S_0.

Assignment requirements

1. A simplified copy of Booth is posted for your use. It will be used to test your program. Beware: it is very likely that the code of Booth will change a bit over time (errors are a fact of life).
2. It is your responsibility to understand the protocol used by Booth. Questions will be answered and suggestions will be welcome. Any student demonstrating a synchronisation error in the code of Booth will receive a bonus of 2 marks (per error).
3. To test your program modify it so that it generates internally a large number of vids and sends them to Booth. This variant will not require any arguments.
4. Your solution must be able to deliver successfully 100 voter identifiers running concurrently with other instances of Validator. It cannot crash or hang. Moreover, it cannot make any other process hang.
5. Your program should produce a terse but complete output showing what vids were sent and what confirmation numbers were received. At the present time it appears that all vids submitted will be accepted.

Submission rules

Submission rules are posted. They must be followed.

Grading

The assignment is worth 10 marks which are distributed as follows:

	action	marks
1	Booth is exec'ed	1
2	Validator delivers 1 vid correctly	1
3	Validator receives 1 confirmation correctly	1
4	100 vids and confirmations exchanged correctly	3
5	Two instances of Validator terminate correctly with 100 vids each	3
6	No shared objects left behind	1

In step 5 the two instances are started at almost the same time.