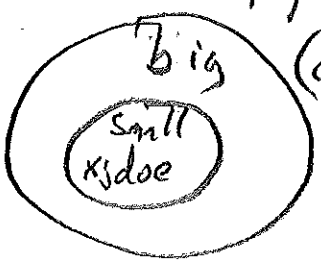
O.S. HW 6 Ch7 Exercises: 7.8, 7.11, 7.14
Pgm Proj. 7.1 & 7.2

7.8



(a) big - read allow
small - read deny

(b) jdoe is a member of small. Granting jdoe individual read will not allow him read because is small read write is deny. We would need to change small's read to allow, but all of small would then be granted read access.

(c) If jdoe was listed first with read allowed, he could read since the kernel uses the first ACL that applies to decide access right.

(d) It is more flexible. The case in (b) & (c) above illustrate this point.

(e) It is more restrictive since any deny would be deny. Files would be "safer."

(f) You could argue either way here, but I like the added flexibility of the kernel's approach.

1.5

7.11 File + directory owned by user 37 and group 53
both have permissions rw- r-x --x
(owner, group, other)

Group 53 members: 37, 42, and 71.

(a) Which user(s) may read the file?
- 53 group members only: 37, 42, 71
- Owner 37 cannot access file since it does not have execute permission of the directory
- other has no read access of the file.

(b) None, only the owner has write permission, but without execute permission of the directory cannot access it to write.

(c) All but the owner.

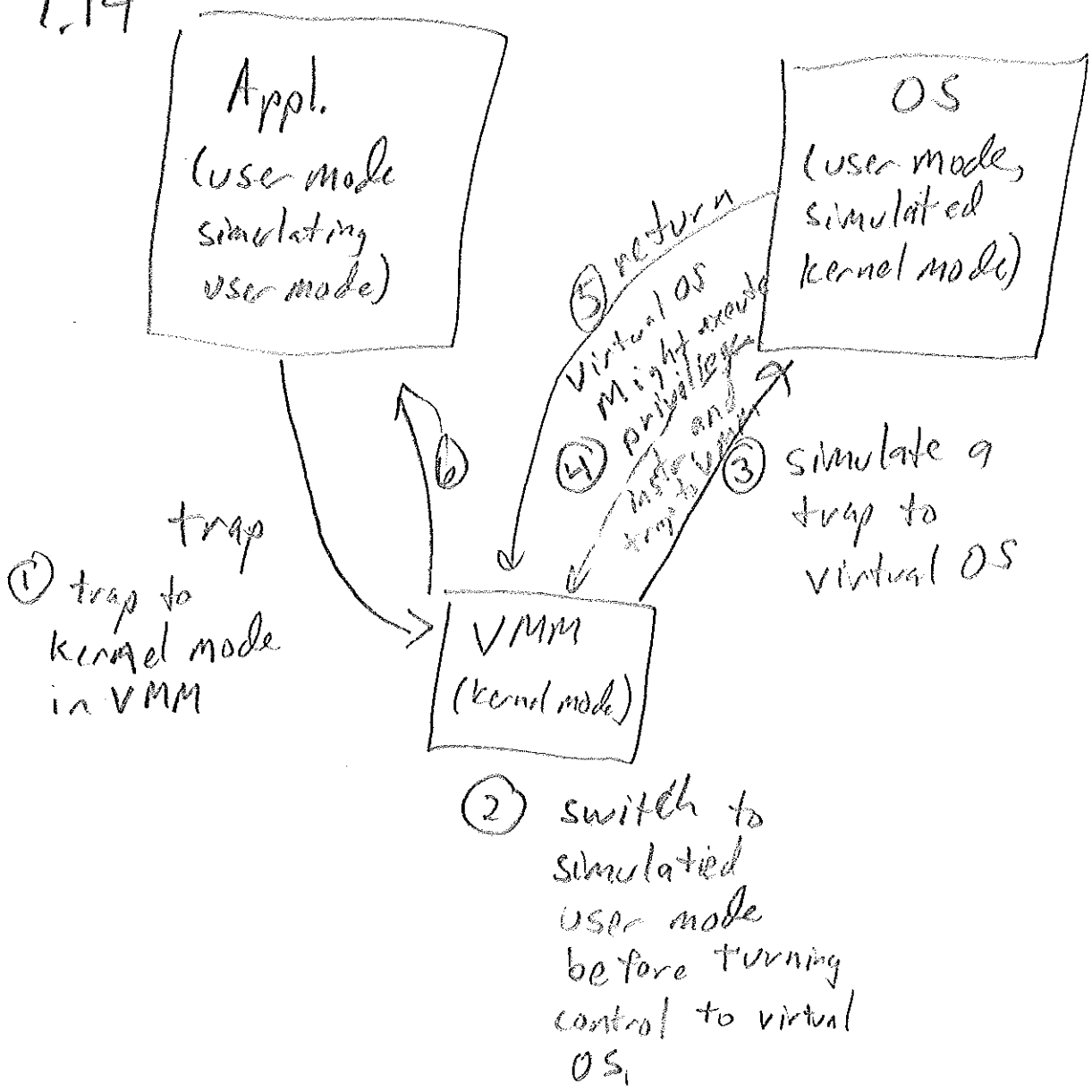
(d) Run as 85 or 37 if file has setuid bit set.

(e)

(f) T: (i) (iii) (iv) (vi)
F: (ii) (v)

1.5

7.14



⑤ Virtual OS return to appl. would invoke VMM, VMM change to user mode

⑥ simulating user mode before returning to appl.

```
// Solution Programming Project 7.1.
// Share the while loop code.  Use the if statement to
// sent the appropriate string to print
#include <unistd.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <cstdlib>

using namespace std;

int main(){
  string parentChildString;
  int loopCount = 5; // each process will get its own loopCount
  cout << "I am still only one process." << endl;

  pid_t returnedValue = fork();
  if(returnedValue < 0){
    // still only one process
    perror("error forking"); // report the error
    return -1;
  } else if (returnedValue == 0){
    // this must be the child process
    parentChildString =  "I am the child process.";
  } else {
    // this must be the parent process
    int pid = static_cast<int>(returnedValue);
    char str[12];
    sprintf(str, "%d", pid); // str now contains pid #
    string pidString( str );
    parentChildString = "I am the parent process; my child's ID is "
      +  pidString + ".";

  }
  while(loopCount > 0){
    cout << parentChildString << endl;
    loopCount--; // decrement parent's counter only
    sleep(1);
  }
  return 0;
}
```

```
//Solution to Programming Project 7.2
// Move shared "return 0;" statement to child and parent code

#include <unistd.h>
#include <stdio.h>
#include <iostream>
using namespace std;

int main(){
  int loopCount = 5; // each process will get its own loopCount
  cout << "I am still only one process." << endl;
  pid_t returnedValue = fork();
  if(returnedValue < 0){
    // still only one process
    perror("error forking"); // report the error
    return -1;
  } else if (returnedValue == 0){
    // this must be the child process
    while(loopCount > 0){
      cout << "I am the child process." << endl;
      loopCount--; // decrement child's counter only
      sleep(1); // wait a second before repeating
    }
    return 0;
  } else {
    // this must be the parent process
    while(loopCount > 0){
      cout << "I am the parent process; my child's ID is "
           << returnedValue << "." << endl;
      loopCount--; // decrement parent's counter only
      sleep(1);
    }
    return 0;
  }
}
```