```
//
// Class Activity: Race Condition Vulnerability
//

Consider the following program:

#include <unistd.h>
#include <stdio.h>

int main()
{
    FILE *fp;
    if (!access("/tmp/XYZ", W_OK)) {
        fp = fopen("/tmp/XYZ", "a+");
        fprintf(fp,"Hello World!\n");
        fclose(fp);
    } else {
        fprintf(stderr,"permission denied\n");
    }
    return 0;
}


Q1. The access() function checks user's permissions to a file using which user ID?

        Real user ID (UID)
        Effective user (EUID)

Q2. The fopen() function checks user's permissions using which user ID?

        Real user ID (UID)
        Effective user (EUID)

Q3. Which countermeasure below eliminates the window between TOC and TOC operations?

    a) Make TOC and TOU operations atomic
    b) Add more race conditions to the code
    c) Enable sticky symlink protection
    d) Apply principle of least privilege when coding

Q4. Most TOCTTOU race condition vulnerabilities involve symbolic links inside the
    "/tmp" directory. Which countermeasure provides protection even if attackers
    can win the race condition in code that involves world-writable sticky directories?

    a) Make TOC and TOU operations atomic
    b) Add more race conditions to the code
    c) Enable sticky symlink protection
    d) Apply principle of least privilege when coding

Q5. Which countermeasure below makes it difficult for attackers to win the "race"?

    a) Make TOC and TOU operations atomic
    b) Add more race conditions to the code
    c) Enable sticky symlink protection
    d) Apply principle of least privilege when coding

Q6. Which countermeasure prevents attackers from doing anything inside the window
    between TOC and TOU?

    a) Make TOC and TOU operations atomic
    b) Add more race conditions to the code
    c) Enable sticky symlink protection
    d) Apply principle of least privilege when coding
```