Assignment 3

Due March 30th 2009

Threads, sockets and file system

The problem

Disks fail; they do so rarely but a great cost to their owners. Hence the need for backing up one's file system. You are given a simple but efficient backup system that stores copies of files with in the same file system. Your task is to convert it to a system that stores copies of your files over a network to some remote site (hoping to find them there when you need them).

The internal-backup.c program

A program called internal-backup.c is available. It is invoked as:

```
internal-backup
internal-backup .
internal-backup /dobo
```

The first invocation starts with a directory ./dummy which is convenient for debugging (don't even think of trying this program on any directory with valuable contents!!!).

This program performs an incremental backup of all recently modified files from all the subdirectories of a directory given as argument.

The program keeps a depth of backup. This implies that older backups will be saved and not overwritten by a new backup (up to a certain depth).

It traverses recursively the subdirectories of the directory given as argument and makes copies of all the regular files in a subdirectory called .bkp (note the dot).

The program saves the time of the backup so that only files modified since the previous backup are saved.

In its current form, the internal-backup.c program works in a correct setting. It does not check for all errors, so it probably bombs if given some malicious arguments. It does not bother with permissions, so it will attempt to write to a read–only directory (just one of its many shortcomings).

Some net work

Two more programs are posted:

backup-server.c will save the files being backed up using network communication (**TCP**). It is set up to listen on port 4950 for traffic sent to *127.0.0.1* and expects somebody to send the files to be backed up through this port (that somebody is called backup-client.c).

backup-client.c will send to the backup-server the files that need to be backed up. The program accepts a connection only if the connecting party successfully produces a user name and a password. These can be any unique words (do not use your regular passwords, please).

These programs use TCP; the network connection is fake (loopback instead of the real thing): the sender and receiver will be on the same computer (but communicating through a socket).

The two programs contain a deliberate error which you will need to fix: they assume that all the subdirectories being backed up were always handled together. As a result this sequence of executions:

```
backup-server &
backup-client ~/Courses/3110/Assignments
backup-server &
backup-client ~/Courses/3110
```

will not do the right thing (it will do no damage, if I got it right).

Your task

Your task consists of 3 separate steps:

1. Read the code of internal-backup.c in order to answer a set of questions (see appendix 1). This will also help you understand the logic of the program.
2. Rewrite the code so that it launches a separate pthread for each subdirectory it encounters.
3. Upgrade the two programs backup-server.c and backup-client.c so that they work correctly.

Part 1 is a prerequisite for the other two which are completely independent of each other.

Note: it takes an enormous amount of time to develop a fool–proof program. I did not bother and I do not expect you to bother (nobody is paying). Your programs should work in a correct setting (no disappearing files or directories, no duplicate copies of backup-client running, etc.). At the same time, all must remember that an industrial–strength program doing the same would take at least twice the amount of effort.

Assignment requirements

1. The code will work on a platform that is compatible with a Linux file system.
2. the programs will successfully create the first backup (when there was no /.bkp directory) as well as perform correctly subsequent incremental backups.
3. An incremental backup will not copy files that were not modified (but it may turn old backup versions into even older backup versions).
4. No junk files (e.g. **/.tmp29312**) will be left behind.
5. Post 4950 will be released by backup–server even when suffering from an untimely death.
6. You will not attempt to make the backup–server accept more than one client. you will not run two copies of backup–server at the same time.

Grading

The assignment is worth 20 marks which are distributed as follows:

5 marks All the questions in the appendix are answered very correctly.

1 mark backup–server writes a correct header file for the directory "officially" backed–up. Evidenced by running another backup immediately afterwards.

3 marks backup–server writes a correct header file for each subdirectory backed–up. Evidenced by running a backup for the subdirectory immediately afterwards.

5 marks backup–client sends to backup–server exactly the files it should back up from all sub–directories. It is not doing that now.

2 marks backup–client launches a new thread for each subdirectory.

1 mark All the threads terminate properly.

1 mark There are no .tmp files left anywhere after the programs terminate.

2 marks There are no .tmp files left anywhere after backup–server is forcibly terminated using a keyboard interrupt.

See next page for desserts.

Bonus 2 marks You attach a working program using cron to run your programs at regular times chosen by users.

Bonus 3 marks You attach a working program that retrieves a backed file from the backup database. The program will produce all the backed versions (one–by–one) until the user interactively picks one.

Appendix 1

Please answer the following questions:

1. How does the backup–server manipulate the password file? Is it easy to destroy someone else's backup in the current setup?
2. How does the program figure out what files belong to a given directory?
3. What does the program do when it encounters a link?
4. Why does it call **SaveOldBKP** before going through the files?
5. The program is recursive. Why?
6. The program will do poorly if two copies of it are run at the same time. Why?
7. The backup–server sends short messages of the form ACK. What for?
8. ACK is only 3 characters long. why is the backup–server insist on sending 4?
9. What is wrong with the third argument (100) of read in the first statement of handle_connection? Is there a correct value?
10. In backup_directory the server loops while recv returns a value greater than 0. Why 0?