

## Programming Assignment 3 - p2pbkup

Due Date - December 6, 2005

**Demo: between December 7-9, 2005, to be scheduled with the Teaching Assistant**

The final programming assignment will require you to integrate the code from first two programming assignments (MP1 and MP2) to design a peer-to-peer application. You will have to demonstrate this application to the teaching assistant.

So as to give you a choice of different possible implementations of `richftp` and `p2psearch`, we will allow you to form **new groups of up to 4 students**, and you can choose the most appropriate implementations for programming assignments 1 and 2, from among all the previous implementations by the students in your new group. **Start early on this project.**

There is no automatic testing for this assignment. Instead, you will have to demo a peer-to-peer application for us, and submit your code and a description of design.

**p2pbkup:** In this assignment, you will build a peer-to-peer backup system. Each user will specify a list of his or her friends, and any new or modified file is sent to some number of friends for backup. For the purposes of this assignment, each file should be backed up on two machines, neither of which is the origin machine.

**Filesystem structure:** You will maintain two directories: `~/important`, and `~/otherpeople`. Any file (not directory) placed in the first directory will be automatically backed up. The second folder is used to store other people's content.

As a simplifying assumption, you may assume that all peers are constantly connected to the network.

**Commands:** You will implement two programs. One is a daemon (`p2pbkupd`) that runs in the background to continuously monitor and backup the changed files. Another is a restoration software (`p2pbkup`) that restores all files to the most recent state.

**Deleted files:** In order to prevent the backups from growing indefinitely, files removed from the `~/important` directory are removed from the backup hosts as well. However, we want to allow a period of time before this purge, in case the deletion were accidental. In an actual system, we might set this timeout to 2 days, but for the purposes of this project, we ask that you set it to 2 minutes. The daemon should print a message to the console when a file is found to be deleted, and two minutes later it should print another message to indicate that the purge has been started.

**Extra Credit:** There are many interesting extensions to this problem. You may implement any of them, or a combination thereof, for extra credit, provided that you also have a methodology for demonstrating the additional features. The extra credit is limited to a maximum of 3% of the final grade.

1. Tit-for-tat. The basic structure for a backup network based on friendships is that you'll store arbitrary amounts of content for your friends. Unfortunately, different friends may have different amounts of available storage. Ensure that you store at most 10% more for friend A than friend A is storing for you. This may require reconsidering your file-to-friend mapping. Furthermore, you need to ensure that your peer is storing the information it claims to be. A fully-functional tit-for-tat system must implement some form of reliable verification.

2. Temporary departures/Churn. Currently we have assumed that no node ever leaves temporarily. If a node leaves the network, then the recovering host may not get all of its files. Also, a node that is gone will not receive the most recent copies of a file. You will need to be able to recover the most recent version of a file, even when all of the hosts associated with that file were down during the recovery process, and even when one of the backup peers was down during the last time the file changed.
3. Permanent departures. After a node has temporarily departed the network for a certain continuous amount of time, the daemon should consider these nodes permanently dead. The node should then make an effort to replicate files that were previously stored on the now removed machine.
4. Optimization Interactions. When (3) is implemented simultaneously with (1), offline hosts should suffer some probability of losing the backups that are stored in the network. By purging with a probability that is monotonically increasing in time, we slowly increase the amount of space available for tit-for-tat exchange with other hosts. When a host that was previously declared dead comes back online, some recovery process must be made to ensure the proper replication of that node's backups.

**Demo:** Your p2pbkup implementation should run across a cluster of hosts. Your final demo to us should be (i) interactive, and (ii) run with *at least three hosts* simultaneously in the p2p system.

**What you have to Turn in:** Hand in your code as usual. Also submit a writeup (4 pages maximum, at least 11 pt font) on the details of the particular designs that you chose.

**Where Do I Start?** After forming your group, the best place to start this project is to determine details of how the MP1 and MP2 interfaces will tie in, and what other functionality needs to be implemented (e.g., converting the MP\*() calls from MP2 into socket calls). As in the real world where different components and interfaces are designed are written by different programmers, this activity of merging MP1 and MP2 may get a little messy. You are allowed to modify any of the code that you have been previously given, or that students in your group have implemented.

**Hand in and Demo times** You will hand in the project just like for the other MPs. Hand in the C sources, the writeup, and the Makefile.

Demos are tentatively scheduled between December 7 to 9. The T.A. will solicit signup times closer to the dates. Exact times will be provided on the newsgroup, and each group will be asked to choose a time slot to demo the project they handed in.

**Managing a Large Group:** We recommend that one student in the group take charge as the "Project Manager" for this MP. The group/Project manager must decide how to plan and delegate the sub-tasks involved in this MP.

**START EARLY** on this project to avoid delays. Note that the due date is very close to the end of the semester.