

# Distributed File Watcher

Joseph Cordina

May 23, 2011

This is the description for Assignment 2 of unit CSA 2205, System Programming, for the year 2010/2011. This assignment is worth 17 of the total mark (30) for this part of the unit. The deadline for this project is 29th May, 2011. You may do this assignment in a group of maximum 2 people, yet you may do this assignment individually also. A single mark will be allocated to the whole group, so choose your partner carefully and make sure that the work gets evenly distributed. Under no circumstances should code be shared outside your group. You will submit your assignment making use of the ASS system<sup>1</sup>. Those of you without login access, please contact me on joseph.cordina at um.edu.mt in due time (thus make sure you check you can submit at least two weeks before the deadline). Each team might be asked to present their implementation of your assignment during which your program will be executed and the design explained. Please be reminded that you cannot copy and plagiarise to ease your way to a final submission. While you may discuss ideas with others, do not steal. You can find more information at the University main website.

Read the plagiarism information well since after-wards no excuses will be accepted. Anyone resorting to such methods will be considered as trying to cheat and will even risk the removal from the degree program.

## 1 Submission Contents

You will submit your C files together with any accompanying header files (C language has to be used obviously) separately. Please do not use any tar or zip file formats. Additionally include any libraries you might have used. The compilation can assume the presence of 'gcc' on which your system will be compiled. Include with your submission *makefiles* that can compile your system<sup>2</sup>. You will need to provide two makefiles, one to generate the .a files (library file to be used by and linked to the clients) and another for the **servers**, if any. You can easily find details on how to generate a library file<sup>3</sup>. I will try your code by linking it with some sample code and running it on any architecture. Then I will run separate instances of clients on Free-BSD, and Linux machines in any order. With your submission, you will provide a file called report.txt which will be a very **brief** description of how your system works and provides the required functionality. Make sure you mention any special techniques or tools you made use of and also any considerations you have made to improve efficiency. More importantly, give details of the network protocol between the interacting processes. Please do not include code inside this file, since this is obviously not readable. Additionally, please note down any bugs your system might have. Listing bugs is common practise, and will benefit the marking process. This file should also contain references to resources you have made use of in your program. Finally, you will submit a file 'protocol.jpg' containing an image that describes in the most concise and clear manner possible your system's architecture and messages that are passed between the components.

---

<sup>1</sup>[www.cs.um.edu.mt/~jcord/ass](http://www.cs.um.edu.mt/~jcord/ass)

<sup>2</sup>You can find plenty of tutorials on the net that teach you how to set one up

<sup>3</sup>An example is <http://www.adp-gmbh.ch/cpp/gcc/create.lib.html>

Before submitting, please do clean up your code. Remove any dead code and useless comments. Every system calls output should be validated for errors and when appropriate, an error message reported to standard output. Also do place some comments to describe your structures in the code, at least to show that you understand how your own program works. Remember C is only unreadable if you make it unreadable. I cannot emphasise enough to be careful with your naming conventions and visual formatting.

## 2 The API to be implemented

First of all kudos to a certain Joseph Galea for the idea of this assignment :) As part of this assignment, you will be providing a library that provides a number of API calls that will allow a process to register itself as a *watcher* of a file, which can exist in a different server. Whenever a file changes, the *watcher* (or *watchers*) will be notified. This is done by a specific callback function being called. This function we will call the *file change handler*, and is registered by the client program. The file change handler function will be able to tell which file changed and how it has changed from the parameters passed in (ex which file has been written to, or deleted). The file change handler function will be called once per file change event. Whenever one registers a new watch, a new unique id is given which can be used to unregister the file watch. Note you do not need to implement functionality to watch directories, but only implement the ability to watch files. Note also that the *file change handler* might be called in a different process than the one that registered the handler. You can assume that the contents of the file change handler will be made re-entrant. The API to be implemented can be seen below:

```
#include <netinet/in.h>

#define MODIFICATION 1 // file written to
#define DELETION 2 // file monitored is deleted
#define ATTRIB_CHANGE 4 // attributes of file change

struct watch {
    struct in_addr ipaddress; // remote ip
    int port; //remote port
    char *pathname; //file to be watched
};
typedef struct watch watch_t;

// file change handler will be called
// when an event happens
// with parameters: watch details and the
// event that occurred on that watch
typedef void (*filechangehandler_t)(watch_t, int);

// takes a list of watches and a bitwise OR
// of the events to watch (the events can be MODIFICATION,
// DELETION, and/or ATTRIB_CHANGE). It also takes the callback
// function which will be called when the watched file changes
// returns a process scoped unique id for the watch or -1 on error
// On error no watch will be started and parameter watches will point to
// the watch that caused the error
int register_distributed_watch(watch_t** watches, int no_of_watches,
    int events_mask, filechangehandler_t filechangehandler);

// removes a distributed watch by watchid
```

```
// returns 0 if ok, -1 on error
int distributed_unwatch(int watchid);
```

You are expected to implement the API above and allow it to be compiled to a static library. All parameters are expected to be in network byte ordering. You will write a test program (which you do not have to submit with your deliverables) that makes use of this API. I will test your submission by creating sample client programs and link them to your library. To allow linking, a header file has been created, *file\_watch.h*, which contains the api above and can be found on the usual website. This header file cannot be changed and you do not have to include it in your submission. To ease your implementation, you may use inbuilt libraries to watch a particular file. Linux provides the *inotify* system call suite to watch local files and FreeBSD provides the *queue* system call suite. If you want to provide watch functionality over a system that does not provide a version of these libraries, you can always write your own (maybe through a system of file polling). Whatever your design, make sure you do not poll too often for file changes, otherwise this will tax the system and be considered as bad design.

You can assume that the remote system where the file is hosted is accessible over TCP/IP and that you have access to the remote machine to run your own server. If you need to execute a server process on a remote machine as part of your assignment, this has to be started with the following command

```
file_watch_server <portno>
```

You can assume that servers can be killed at any point and also that clients might be started prior to the start of servers. Thus make sure you implement your error handling properly.

### 3 An example client

What follows is a sample client that makes use of the library explained above. You can also find this file in the usual website.

```
#include "file_watch.h"

// server IP
#define IP_STR "193.188.34.81"
// server PORT
#define SERVER_PORT 6010
// filename that will be watched
#define FILENAME "/home/tom/a.txt"

// forward definition
void fw_handler(watch_t file_watch, int event);

main()
{
    struct in_addr server_addr;
    inet_aton(IP_STR,&server_addr);
    int server_port = htons(SERVER_PORT);

    // details of watch
    watch_t my_watch;
    my_watch.ipaddress = server_addr;
    my_watch.port = server_port;
```

```

my_watch.pathname = FILENAME;

watch_t *watch_pt = &my_watch;

// register watch
int first_watch = register_distributed_watch(
    &watch_pt, 1,
    DELETION | MODIFICATION,
    fw_handler);

// on error
if (first_watch == -1) {
    printf("Things went bad!\n");
    exit(-1);
}
// wait and then remove watch
sleep(60);
distributed_unwatch(first_watch);
}

// file watch handler code
void fw_handler(watch_t file_watch, int event)
{
    if (event == DELETION) {
        printf("%s has been deleted \n",FILENAME);
    } else if (event == MODIFICATION) {
        printf("%s has been modified \n",FILENAME);
    } else {
        printf("Ouch, dunno what happened !\n");
    }
}
}

```

The above will register a watch on file */home/tom/a.txt* and machine *193.188.34.81* for any modifications or deletions on that file. Note that it assumed that a server is waiting for connections on the remote machine on port 6010. When the file gets deleted or modified an appropriate message will be shown on screen.

## 4 What might matter

Marks will be allocated equally to a functional system that allows multiple clients to register file watches and also to a good and efficient design. I will run several concurrent clients watching for several events on several files on different machines. It goes without saying that if you find an application implementing something similar you cannot use it. Obviously having a system that is free of race conditions, deadlocks and still considerably efficient would be nice. For your own good, first design your system before starting the coding. It will result in a better system and a quicker turn around time. Instances of the watchers will have to communicate with servers through an Internet connection created from non-privileged user processes.

This system has to be implemented using BSD stream sockets running on the TCP/IP protocol. You can to make sure that inter-architecture connectivity is supported. You have to ensure that your code works at the very least on a combination of FreeBSD running on an itanium architecture and on Linux running on an i386 architecture. Both

the clients and the server processes must be fault tolerant. They can both be terminated with a CTRL-C or the kill command. Ensure that proper cleanup is performed in each case. Thus the server must terminate all its child processes (if any) and close all open sockets. The child must close all its connections properly before terminating. Ensure that you develop an appropriate protocol between the watchers and the services (if any).

You can also assume that no deadline extensions will be given (as per usual really).

## 5 And To Conclude

Feel free to make use of any optimisations you can think of. Optimisations you should consider are implementation optimisations (better data structures, less indirect references, etc) and also algorithmic optimisations. Keep the allocated time you spend on this project in perspective with the allocated marks. As a suggestion, first sit down and design your solution well before implementing it since this will aid you a lot in your programming.

Good luck and if you need anything, contact me on joseph dot cordina at um.edu.mt. Even better, make use of the discussion group to discuss ideas and ask questions. You can find a link to the discussion group on my website. Other students might help you out in the solution. Just make sure you do not post any code or given concrete solutions.

Note that you will be doing a lot of design work and scrapping lots of ideas, so do not expect to do this assignment in the last few days before the deadline. Do it slowly and the ideas will mature. Remember this is mainly a design assignment (like all good system tools) and you only do the coding after you are sure your design is correct.

I might post on the website a list of designs that impressed me (lets hope there are a couple :) ) !