RENAN DOMINGOS MERLIN GRECA

WISYNC: SINCRONIZAÇÃO DE DIRETÓRIOS EM LAN

CURITIBA

2015

RENAN DOMINGOS MERLIN GRECA

WISYNC: SINCRONIZAÇÃO DE DIRETÓRIOS EM LAN

Monografia apresentada para obtenção do Grau de Bacharel em Ciência da Computação pela Universidade Federal do Paraná.
Orientador: Prof. Dr. Luiz Carlos P. Albini

CURITIBA

2015

SUMÁRIO

RESUMO

[resumo]

Palavras-chave:

ABSTRACT

[abstract]

Keywords:

CAPÍTULO 1

INTRODUÇÃO

Com o barateamento e a popularização de microcomputadores, é comum hoje um escritório, uma família ou até mesmo um indivíduo ter diversos computadores pessoais à sua disposição. Portanto, torna-se útil e às vezes necessário existir uma maneira de manter esses computadores "conversando" entre si, trocando informações e arquivos para que o usuário possa mudar de um para outro de forma coesa. Além disso, computadores portáteis (i.e., *laptops*) são mais vulneráveis do que computadores domésticos a roubos ou danos físicos, além de outros fatores que podem comprometer a integridade das informações neles contidas, o que torna necessária a realização de *backups* frequentes.

1.1 Proposta

Para trabalhar em diversos computadores ou manter cópias seguras de arquivos de forma conveniente, o presente projeto propõe um programa que, através de uma rede local (daqui em diante referida como LAN, de *Local Area Network*), compare diretórios em dois ou mais computadores distintos e realize a sincronização dos mesmos. Com sincronização, queremos dizer que o programa irá, para cada instância rodando:

- Copiar arquivos novos (adicionados desde a última execução) às outras instâncias;
- Apagar arquivos removidos nas outras instâncias; e
- Copiar alterações aos arquivos às outras instâncias, incluindo arquivos renomeados.

1.2 Trabalhos Similares

Atualmente existem alguns programas disponíveis na Internet que fazem operações desse tipo, mas em contextos um pouco diferentes.

Entre produtos comerciais, há diversos serviços de armazenamento em nuvem que suportam a sincronização de diretórios caso o usuário instale um aplicativo em cada computador. Exemplos desses produtos incluem Dropbox, Box, Google Drive, Apple iCloud Drive e Microsoft OneDrive. A principal diferença desses produtos ao atual projeto é a existência de um servidor mantendo os arquivos na "nuvem". A vantagem dessa decisão é que o usuário pode acessar seus arquivos facilmente de outros dispositivos como celulares e tablets - contudo, para a sincronização ocorrer é necessário que os computadores clientes estejam conectados à Internet e todos esses serviços impõem limites em bytes na quantidade de arquivos que podem ser sincronizados.

Outra categoria de programas que realizam operações semelhantes são os programas de *backup*. Um exemplo desses programas é o FreeFileSync [3], solução *open-source* disponível na web. Dados um par de diretórios A e B, programas de *backup* normalmente possuem três funcionalidades:

- Sincronização de dois sentidos, que torna A e B idênticos copiando as modificações de um para o outro e vice-versa;
- Sincronização de espelho, que faz o diretório B ser um clone do diretório A, incluindo arquivos removidos; e
- Sincronização de atualização, que atualiza B com os arquivos novos ou modificados de A, mas não remove arquivos excluídos em A.

Para o propósito desta monografia, o projeto irá focar no primeiro tipo de sincronização, a de dois sentidos. Os outros dois tipos também são interessantes e possivelmente serão adicionados ao escopo futuramente.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

2.1 Organização do Programa

Na versão atual, o WiSync é composto por três arquivos: `wisync.py`, `winet.py` e `wifiles.py`. Abaixo estão as funcionalidades de cada um:

- `wisync.py`: Arquivo principal do projeto, responsável por ler os parâmetros da linha de comando e controlar a execução do processo.
- `winet.py`: Contém a classe WiNet, que inclui os métodos e atributos necessários para fazer as partes em rede do programa, como hospedar e receber arquivos.
- `wifiles.py`: Contém a classe WiFiles, que inclui os métodos e atributos necessários fazer as partes que lidam com o sistema de arquivos do programa, como ler e comparar diretórios.

Para executar o WiSync, são necessárias algumas bibliotecas padrão do Python: `os`, `sys`, `argparse`, `time`, `json`, `datetime`. Também é usada uma versão modificada do programa `woof.py` [1] (distribuída sob a licença GNU General Public License), que é usada na hora de transmitir os arquivos entre os computadores.

2.1.1 Parâmetros de Execução

Para rodar o WiSync, utiliza-se o seguinte comando:

```
python wisync.py [-h] -d DIRECTORY [-n HOSTNAME] [-s]
```

Os argumentos são os seguintes:

- `-h, --help` Mostra o texto de ajuda do programa.

- `-d DIRECTORY, --directory DIRECTORY` Caminho para o diretório a ser sincronizado (de preferência, o caminho completo)

- `-n HOSTNAME, --hostname HOSTNAME` Nome de rede ou endereço IP do outro computador. Esse argumento é opcional e pode ser preenchido tanto pelo endereço IP do outro computador quanto pelo nome de rede. Por exemplo, `-n SgtPepper.local` e `-n 192.168.1.110` têm o mesmo efeito. Caso o argumento não esteja presente, o programa procura outras instâncias do WiSync na rede local.

- `-s, --server` Modo servidor. Se definido, esta instância será o "servidor", hospedando seus arquivos antes da instância remota.

2.2 Sobre a Transmissão dos Arquivos

A primeira etapa do desenvolvimento do programa foi definir o método e protocolo que seriam usados na hora de transmitir arquivos entre os computadores. Após alguma pesquisa, quatro alternativas foram consideradas:

- **Sockets via TCP**, que utilizaria a biblioteca padrão `socket` do Python para transmitir os dados dos arquivos a nível de camada de transporte. A implementação em Python é relativamente simples, mas requer muita atenção porque arquivos maiores têm que ser divididos em blocos para depois serem re-montados.

- **Secure Copy** (SCP), método que utiliza os mesmos padrões de segurança do protocolo SSH (*secure shell*) para garantir a segurança e confidencialidade dos dados. A maneira mais simples de fazer isso em Python seria invocando um comando da linha de comando, que não dá muito controle sobre a operação de dentro do programa.

- **File Transfer Protocol** (FTP) é um protocolo da camada de aplicação específico para o envio de arquivos. Contudo, ele se mostra mais eficaz em situações onde há servidor e clientes bem definidos: ou seja, um servidor que hospeda arquivos e clientes que podem acessar e alterar os arquivos do servidor.

- **Hypertext Transfer Protocol** (HTTP) é o protocolo padrão de envio de dados na Internet. É fácil de utilizar, mas os arquivos ficam temporariamente abertos a qualquer membro da rede.

Para o WiSync, o HTTP foi escolhido para realizar a troca de arquivos. Isso se deve principalmente à facilidade de implementar essa troca, e também ao fato que dois computadores podem rapidamente trocar seus papéis de cliente e servidor enquanto transmitem arquivos entre si. Como o objetivo do WiSync é ser usado em redes locais privadas, a abertura temporária dos arquivos na rede não é uma grande preocupação.

2.3 JSON

O *JavaScript Object Notation*, ou JSON, é um formato de armazenamento e troca de dados baseado na notação de objetos do JavaScript. Com o JSON, é possível armazenar objetos de um programa em um arquivo de texto, utilizando os tipos dictionary, array, string, number e boolean do JavaScript.

No WiSync, arquivos JSON são usados para armazenar e transmitir as listas de arquivos geradas pela leitura e comparação dos diretórios. O JSON foi escolhido por sua simples integração com Python e por ser fácilmente legível por um humano, facilitando a verificação dos dados durante os testes. A principal desvantagem do JSON também vem dessa facilidade de leitura, já que é possível alterar os dados armazenados por fora do WiSync e então gerar resultados inesperados.

CAPÍTULO 3

DESENVOLVIMENTO

Neste capítulo, será descrita o desenvolvimento e as soluções da implementação do WiSync para este trabalho.

A versão do WiSync entregue com este texto foi escrita em Python [2], versão 2.7, linguagem de programação interpretada originalmente lançada em 1991. Python foi escolhida por sua facilidade na implementação e extensa disponibilidade de bibliotecas de código aberto.

Originalmente, havia como objetivo tornar o programa compatível com os três sistemas operacionais mais comuns do mundo: Microsoft Windows, Apple OS X e Linux. Contudo, devido a diferenças inerentes na forma como o Windows funciona, optou-se por manter apenas compatibilidade com OS X e Linux. Para tal, o WiSync foi desenvolvido usando os seguintes computadores para testes:

Nome	"SgtPepper"	"Packard"
Sistema Operacional	OS X 10.11	Linux Mint 17
Processador	Intel Core i5-4308U	Intel Core i7-2600
RAM	8GB DDR3L	8GB DDR3
Armazenamento	SSD 512GB	HDD 320GB
Conectividade	Wi-Fi 802.11n 5GHz	Cabo Ethernet
IP local	192.168.1.110	192.168.1.132

3.1 Mudanças feitas ao woof

Como citado anteriormente, o programa `woof.py` [1] foi usado como biblioteca para auxiliar no envio de arquivos por HTTP. Para se adequar às necessidades do WiSync, algumas pequenas mas importantes mudanças tiveram que ser feitas:

- Após servir um arquivo ao outro computador, o endereço IP do computador remoto

é armazenado para poder ser utilizado novamente no futuro sem depender de outras buscas por endereços.

- Originalmente, o `woof.py` utilizava diversas *threads* para permitir o *download* simultâneo de arquivos. Como essa funcionalidade não é necessária para o WiSync e as múltiplas *threads* causavam consequências indesejadas, o código foi alterado para desabilitar isso.

3.2 Situações consideradas

Tratando-se das possíveis diferenças que podem existir entre um par de diretórios (vamos chamá-los de A e B), as seguintes possibilidades são consideradas:

- Arquivos existem em A mas não em B e vice-versa. Como demonstrado na figura 3.1, nessa situação basta copiar os arquivos do diretório que os contém para o que não os contém.

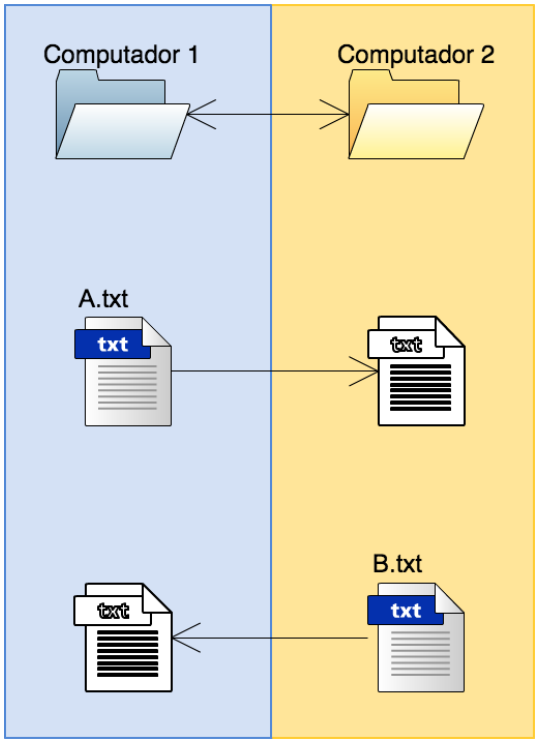

Figura 3.1: Caso onde os dois diretórios têm arquivos novos

- Arquivos com o mesmo nome existem em A e B mas têm conteúdo diferentes. Caso

da figura 3.2. Nesse caso o WiSync utiliza o histórico da última sincronização antes da atual e as informações de data de modificação dos arquivos. Se o arquivo já existia antes e a data de alteração de uma das versões atuais é igual à anterior (implicando que o arquivo não foi modificado desde então), a versão mais recente substitui a mais antiga. O mesmo processo se aplica a um arquivo que foi excluído desde a última sincronização.

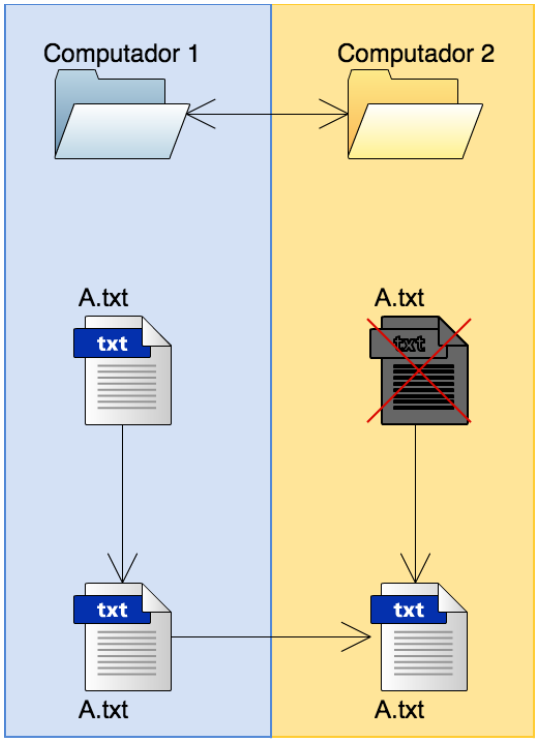

Figura 3.2: Caso onde um arquivo foi atualizado em apenas um dos diretórios

- Similar à situação acima, mas ambos arquivos têm datas de modificação diferentes às da última sincronização, ou não há informação sobre a última sincronização. Nesse caso, mostrado na figura 3.3, o WiSync copia ambas versões para o outro diretório, renomeando os arquivos para evitar conflitos e sobre-escritas.

- Caso um arquivo seja renomeado, o WiSync tratará essa situação como se fossem dois arquivos: um excluído e um novo. Caso o arquivo correspondente no diretório remoto não tenha sido alterado desde a última sincronização, este será excluído e uma nova cópia, com o novo nome, será criada. Isso é ilustrado na figura 3.4

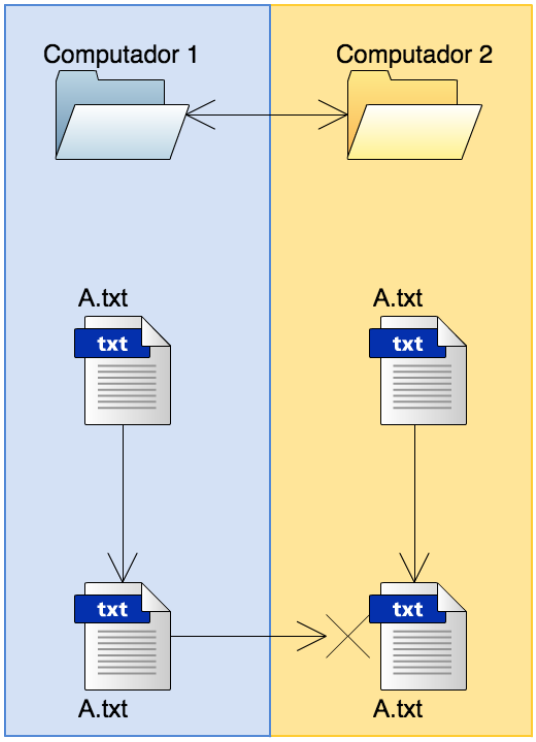

Figura 3.3: Caso onde há conflito de versões entre dois arquivos

3.3 Fluxo do programa

Durante a execução, o WiSync executa as seguintes operações:

1. Processa parâmetros e configura as classes WiFiles e WiNet, que preparam o que é necessário para lidar com arquivos e rede no programa.

2. As duas instâncias do programa trocam os arquivos JSON contendo informações sobre o diretório a ser sincronizado.

3. Uma das instâncias utiliza esses arquivos JSON e, se existir, o arquivo contendo informações do estado anterior do diretório para gerar um novo JSON contendo a lista de arquivos que devem ser transferidos ou excluídos em cada um dos diretórios.

4. A instância que realizou a comparação começa a enviar os arquivos de acordo com a lista contida no JSON e, ao mesmo tempo, a outra instância recebe esses arquivos.

5. As instâncias trocam e a segunda passa a enviar arquivos para a primeira.

6. Ambas instâncias excluem seus próprios arquivos de acordo com o JSON.

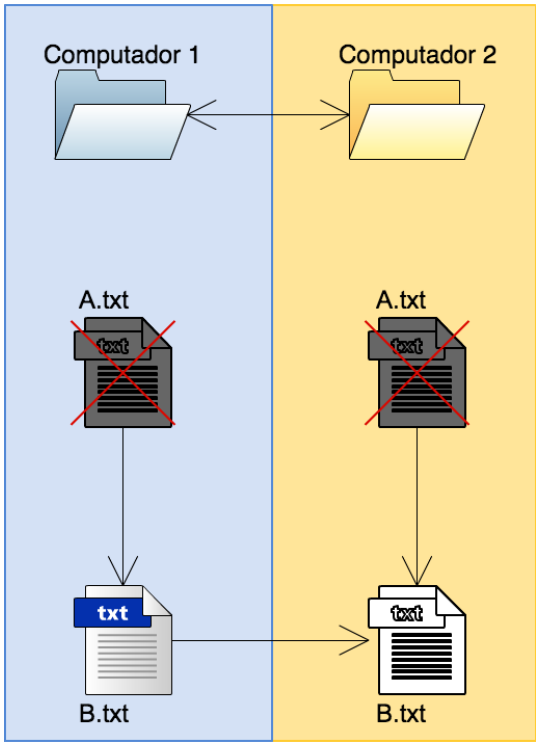

Figura 3.4: Caso onde um arquivo foi renomeado

7. Um novo JSON é gerado contendo as informações dos diretórios após a sincronização, que será usado para sincronizações futuras.

3.4 Dificuldades encontradas

Durante o desenvolvimento do WiSync, algumas dificuldades foram encontradas. A seguir elas são descritas e são apresentadas as soluções encontradas.

3.4.1 Waits e timeouts

Talvez a parte mais difícil do desenvolvimento do WiSync tenha sido levar em consideração o fato de que o programa roda em dois computadores diferentes, com velocidades diferentes e que se comunicam apenas ocasionalmente através da troca de arquivos. Para evitar erros, é necessário garantir que ambos computadores estejam no mesmo passo ao mesmo tempo.

Um erro que aconteceu com frequência durante o desenvolvimento, por exemplo, é que um computador procurava o arquivo do outro antes deste ter preparado a hospedagem do arquivo. Ou seja, o arquivo não estava disponível ainda e o primeiro computador

retornava um erro de endereço inválido. Para evitar isso, foi adicionado uma pausa de 1 segundo logo antes de tentar acessar um arquivo do outro computador. Este tempo permite que o outro computador termine de preparar a hospedagem. Essa solução não é infalível, pois pode ser que o computador demore mais ainda para hospedar e, neste caso, o erro acontece de qualquer forma. Uma solução para isso seria criar um laço onde se tenta diversas vezes acessar o arquivo remoto mas, nos testes, isso teve um desempenho pior do que a pausa.

3.4.2 Datas de criação/modificação

Como explicado anteriormente, o WiSync utiliza metadados dos arquivos - a data modificação em particular - para decidir quais devem ser copiados. Contudo, isso gerou um problema que não havia sido previsto: quando um arquivo é copiado para o outro, a data de modificação da cópia é a data de sincronização - não a data de modificação do arquivo original. Ou seja, numa sincronização subsequente, a cópia é entendida como uma versão nova do arquivo e copiada novamente para o computador original.

Para resolver isso, foi usada a função `os.utime` do Python, dessa forma configurando as datas de criação e modificação do arquivo criado para serem iguais às do arquivo de origem. Uma preocupação que surgiu em relação a isso foi a possibilidade de, devido a configurações diferentes entre os computadores, um arquivo ter data de criação e/ou modificação futura ao relógio do sistema. Contudo, testes foram realizados no Linux e no OS X e isso não se mostrou um problema.

3.4.3 Ordem dos arquivos

As listas de arquivos que o WiSync são armazenadas usando o tipo Dictionary, do Python, para que torne-se simples indexar os arquivos utilizando os nomes deles. Contudo, um dicionário, ao ser percorrido, não tem uma sequência definida a ser seguida. Portanto, na hora de enviar e receber arquivos, o dicionário é convertido para uma lista de tuplas, ordenada alfabeticamente com os nomes dos arquivos. Isso garante que ambos computadores enviem e recebam arquivos na mesma ordem.

CAPÍTULO 4

RESULTADOS

[resultados]

CAPÍTULO 5

CONCLUSÃO

[conclusão]

BIBLIOGRAFIA

[1] Simon Budig. Woof. Disponível em: http://www.home.unix-ag.org/simon/woof.html.

[2] Guido van Rossum. Python. Disponível em: https://www.python.org/.

[3] ZenJu. Freefilesync. Disponível em: http://sourceforge.net/projects/freefilesync/.