Universidade Federal de Alagoas
Instituto de Computação
Ciência da Computação
Disciplina de Redes de Computadores I
Professor Leandro Sales
Aluno Gean da Silva Santos
Turma 2014.2

Projeto de Redes de Computadores I
Automatic Backup System - Aubasys

1. Plataformas testadas

- Linux: Testado na distribuição Mint 17 Cinammon 64 bits.
- Windows: Testado nas versões: Windows 8.1 64 bits e Windows 7 Ultimate 64 bits.

2. Funcionalidades

O Aubasys é uma aplicação distribuída baseada na arquitetura cliente-servidor para transferência de arquivos do cliente para o servidor.

Na aplicação do cliente, é verificado periodicamente o diretório definido pelo usuário no arquivo *settingsclient.py* buscando atualizações em relação ao último backup. Quando é executado pela primeira vez, a aplicação lista os arquivos presentes (se houver) no diretório, os envia para o servidor e, depois, mantém o estado do diretório no último envio.

Na aplicação do servidor, cada nova conexão estabelecida por um cliente é executada em uma tarefa (thread). Durante a tarefa o servidor recebe e verifica os cabeçalhos enviados pela aplicação cliente e executa as operações correspondentes como: registrar novo usuário, preparar para recepção de novo backup e receber arquivos.

No servidor, cria-se uma pasta com o nome do cliente e dentro dessa cria-se mais duas pasta: “current” e “previus”. A cada atualização a aplicação cria, dentro da pasta previus, uma pasta com a data e hora corrente e coloca os arquivos recebidos nela. Então cria um link, na pasta current, com o mesmo nome do arquivo para cada arquivo recebido que aponta para esse. No caso de remoção de um arquivo, apenas é removido o link na pasta current.

É previsto que o programa (tanto servidor como cliente) parem se, ao iniciar, perceba que o arquivo de configuração não esteja preenchido.

Em algumas operações, mensagens são exibidas para acompanhamento da execução e outras são ainda escritas em um arquivo *debug.txt*. O debug em arquivo e as mensagens podem ser desativadas/ativadas no arquivo *tools.py*.

3. Dificuldades

- Manter coerência entre as transmissões de cabeçalhos e dados.
- Recepção de arquivos grandes no servidor: para isso foi utilizado um loop no servidor com o método *socket.recv(4096)*, ou seja, lê 4096 bytes a cada iteração, armazena os bytes lido em uma variável temporária acrescenta essa variável em outra e depois verifica se o recebido na variável temporária é menor que 4096, no caso verdade o loop cessa.
- Tratar as exceções para manter a aplicação executando: em diversos locais da aplicação (tanto no servidor como no cliente), exceções ou erros que ocorreram ou foram previstos foram tratadas para que escrevessem o ocorrido em arquivo debug e não parassem a execução da aplicação.
- Verificar espaço livre em disco: a aplicação servidor cotem um método que serviria para verificação do espaço para gravação de arquivos, mas não foi encontrado uma forma direta de obter esse resultado, mas a implementação feita usa uma forma incerta (vulgo gambiarra) que depende de apresentação do sistema; impedindo, assim, essa verificação.

4. Possíveis implementações

Realizar backup de subdiretórios, pois atualmente os diretórios dentro do diretório de backup são ignorados. Para sua realização, seria necessário adicionar mais uma linha a um cabeçalho indicando que será realizado upload de um diretório. Mais complexo é verificar o conteúdo desses diretórios, seria necessário mais um loop no cliente para cada diretório e subdiretório encontrado.

Criar leitura de entrada de teclado para interação com a aplicação.