

Análise Detalhada do `arquivo_raiz.exe` (originado de `arquivo_raiz.py`)

Objetivo Principal

O `arquivo_raiz.exe` tem como objetivo principal **centralizar e organizar arquivos** dentro de uma pasta raiz selecionada pelo usuário. Ele percorre recursivamente todas as subpastas (exceto as especificadas) da pasta raiz, move os arquivos encontrados para a pasta raiz e, durante o processo, aplica regras de sanitização, truncamento e resolução de conflitos de nomes aos arquivos (tanto os movidos quanto os que já estavam na raiz). Adicionalmente, remove as subpastas que ficaram vazias após a movimentação dos arquivos.

Funcionalidades Detalhadas

1. Seleção de Pasta Raiz (Interface Gráfica):

- Utiliza a biblioteca `tkinter` (`filedialog.askdirectory`) para apresentar uma janela ao usuário, permitindo que ele navegue e selecione a pasta principal onde a operação será realizada.
- Exibe mensagens informativas iniciais e de confirmação usando `tkinter.messagebox`.
- Se nenhuma pasta for selecionada, o script encerra a execução.

2. Processamento Recursivo de Arquivos (`os.walk`):

- Usa a função `os.walk(root_folder, topdown=True)` para percorrer a árvore de diretórios a partir da pasta raiz selecionada.
- `topdown=True` permite modificar a lista `dirs` em tempo real para evitar que `os.walk` entre em pastas excluídas.

3. Exclusão de Pastas Específicas:

- Possui uma lista `excluded_folders` (inicialmente contendo "erros" e "anos anteriores") que define quais nomes de pastas devem ser ignorados durante a varredura. A comparação é feita em minúsculas (`lower()`).
- A pasta de log (`self.log_folder`, por padrão "ERROS" dentro da pasta raiz) também é explicitamente ignorada para evitar que os próprios arquivos de log sejam processados.

4. Sanitização de Nomes de Arquivos (`_sanitize_filename`):

- Remove o prefixo "msg " (ignorando maiúsculas/minúsculas) do início dos nomes dos arquivos usando `re.sub(r'^msg\s+', '', filename, flags=re.IGNORECASE)`.
- Substitui caracteres inválidos em nomes de arquivos do Windows (`<>:"/\|?*`) por `_` (underscore) usando `re.sub(r'[<>:"/\|?*]', '_', sanitized)`.
- Remove caracteres de controle ASCII (0-31) usando `re.sub(r'[\x00-\x1f]', '', sanitized)`.
- Remove espaços em branco no início e no fim do nome (`strip()`).
- Se o nome do arquivo ficar vazio após a sanitização, gera um nome padrão com timestamp (`arquivo_renomeado_timestamp`).

- Normaliza números no início do nome para remover zeros à esquerda (ex: `001_arq.txt` -> `1_arq.txt`). Trata números muito grandes que não cabem em `int` mantendo-os como string e removendo zeros à esquerda se aplicável.
- Se o nome do arquivo ficar vazio após a sanitização, utiliza o nome de fallback `FALLBACK_SANITIZED_FILENAME` (definido como "arquivo_renomeado") e registra um log de `WARNING`.
- Registra um log de nível `INFO` se a sanitização efetivamente alterou o nome original (este log é gerado no `process_files_in_root` ao comparar `original_filename` com `sanitized_filename`).

5. Truncamento de Nomes de Arquivos (`_truncate_filename`):

- Verifica se o caminho completo do arquivo *no destino* (pasta raiz) excederia o limite `EFFECTIVE_MAX_PATH` (259) com uma margem de segurança `SAFE_PATH_MARGIN` (10). O limite efetivo para o nome do arquivo no destino é `EFFECTIVE_MAX_PATH - SAFE_PATH_MARGIN`.
- Se o caminho for muito longo, calcula o espaço disponível para o nome base do arquivo (descontando o caminho da pasta raiz, separador e extensão).
- Trunca o nome base do arquivo para caber no espaço disponível.
- Registra um log de nível `WARNING` informando o nome original e o nome truncado.
- Se o próprio caminho da pasta raiz for tão longo que não há espaço nem para um caractere no nome base, registra um log de `ERROR` e retorna o nome original (o erro provavelmente ocorrerá na etapa de mover/renomear).

6. Movimentação de Arquivos (`shutil.move`):

- Se um arquivo é encontrado em uma subpasta (que não seja excluída), ele é movido para a pasta raiz (`self.root_folder`) usando `shutil.move(str(source_path), str(destination_path))`.

7. Renomeação de Arquivos na Raiz (`Path.rename`):

- Se um arquivo já está na pasta raiz, mas seu nome precisa ser alterado devido à sanitização ou truncamento, ele é renomeado dentro da própria pasta raiz usando `source_path.rename(destination_path)`.

8. Tratamento de Duplicidade/Conflitos de Nomes:

- Antes de mover ou renomear, verifica se já existe um arquivo com o nome final (`current_final_filename`) na pasta raiz (`destination_path.exists()`).
- Se um conflito é detectado, entra em um loop `while`:
 - Gera um novo nome adicionando um timestamp (`_YYYYMMDDHHMMSSffffff`) ao nome base original (antes do truncamento/sanitização que causou o conflito).
 - Aplica o truncamento novamente a este novo nome com timestamp.
 - Verifica se o novo nome ainda causa conflito (caso extremamente raro). Se sim, loga um `ERROR` e desiste de processar o arquivo.
 - Se o novo nome é único, registra um log de `WARNING` informando a renomeação devido à duplicidade e usa esse novo nome.
 - O loop continua até `MAX_DUPLICATE_RESOLUTION_ATTEMPTS` (definido como 10) tentativas.

9. Logging Detalhado (logging):

- Configura um logger que salva as informações em um arquivo dentro da pasta `log_folder` (definida como "ERROS" dentro da pasta raiz). O nome do arquivo inclui um timestamp (`process_root_log_YYYYMMDDHHMMSS.log`).
- Níveis de Log Utilizados:
 - **INFO**: Registra sanitizações que alteraram nomes de arquivos.
 - **WARNING**: Registra truncamentos de nomes e renomeações devido a duplicidade.
 - **ERROR**: Registra falhas críticas como pasta raiz não encontrada, erros ao mover/renomear arquivos, falhas na resolução de nomes duplicados, erros ao verificar/remover pastas vazias.
- O formato do log inclui timestamp, nível do log e a mensagem (`%(asctime)s - %(levelname)s - %(message)s`).

10. Remoção de Pastas Vazias (remove_empty_folders):

- Após o processamento de todos os arquivos, se algum arquivo foi efetivamente movido, esta função é chamada.
- Utiliza `os.walk(self.root_folder, topdown=False)` para percorrer a árvore de diretórios de baixo para cima.
- Para cada pasta (exceto a raiz, a pasta de log e as pastas excluídas), verifica se está vazia (`not os.listdir(root)`).
- Se estiver vazia, tenta removê-la usando `os.rmdir(root)`.
- Registra um log de **ERROR** se houver problemas ao verificar ou remover uma pasta.

11. Feedback ao Usuário (Interface Gráfica):

- Utiliza `tkinter.messagebox` para mensagens iniciais de seleção de pasta e confirmação.
- Ao final do processo, exibe uma janela de resumo (`show_auto_close_message`) que detalha:
 - Número de arquivos renomeados na pasta raiz.
 - Número de arquivos movidos das subpastas para a raiz.
 - Número de erros ocorridos.
 - Status da remoção de pastas vazias (quantas foram removidas ou se nenhuma precisou ser).
- A janela de resumo fecha automaticamente após um tempo (10 segundos) ou pode ser fechada manualmente.
- Informa se ocorreram erros e instrui o usuário a verificar o arquivo de log.

Modo de Usar (para `arquivo_raiz.exe`)

1. **Execute o Programa:** Dê um duplo clique no arquivo `arquivo_raiz.exe`.
2. **Seleção da Pasta:** Uma mensagem inicial solicitará que você selecione a pasta raiz. Clique em "OK". Em seguida, uma janela de diálogo do sistema operacional será aberta. Navegue e selecione a pasta principal onde os arquivos serão centralizados e clique em "Selecionar pasta".
3. **Confirmação:** Uma mensagem confirmará a pasta selecionada e informará onde os logs serão salvos. Clique em "OK" para iniciar.
4. **Aguarde o Processamento:** O programa varrerá a pasta selecionada e suas subpastas, movendo, renomeando e sanitizando os arquivos. Pastas vazias remanescentes serão removidas.

5. **Verifique o Resumo Final:** Ao final, uma janela de resumo detalhará as ações realizadas e quaisquer erros. Esta janela fechará automaticamente após 10 segundos.
6. **Consulte os Logs:** Se erros foram reportados, verifique a subpasta "ERROS" (ou o nome configurado para logs) dentro da pasta raiz processada.

Fluxo de Execução

1. Exibe a janela para seleção da pasta raiz.
2. Configura o logger para salvar em `[Pasta Raiz]/ERROS/process_root_log_[timestamp].log`.
3. Exibe mensagens de confirmação da pasta selecionada e início do processo.
4. Inicia a varredura recursiva com `os.walk` a partir da pasta raiz.
5. Para cada arquivo encontrado:
 - Ignora se estiver na pasta de log.
 - Sanitiza o nome do arquivo.
 - Calcula o nome final truncado (considerando a pasta raiz como destino).
 - Verifica se o nome final já existe na pasta raiz. Se sim, tenta gerar um nome único com timestamp (e trunca novamente se necessário).
 - Se ocorrer erro irresolúvel na geração de nome único, loga `ERROR` e pula o arquivo.
 - Se o arquivo está em uma subpasta, move-o para a raiz com o nome final (`shutil.move`).
 - Se o arquivo já está na raiz, mas o nome final é diferente do original, renomeia-o na raiz (`os.rename`).
 - Loga erros (`ERROR`) se `move` ou `rename` falharem.
6. Após percorrer todos os arquivos, exibe um resumo das operações no console.
7. Se arquivos foram movidos, chama `remove_empty_folders` para limpar subpastas vazias (percorrendo de baixo para cima).
8. Prepara uma mensagem de resumo detalhada.
9. Exibe a mensagem de resumo em uma janela gráfica (`show_auto_close_message`) que se fecha automaticamente.

Constantes Importantes

- `EFFECTIVE_MAX_PATH = 259`: Limite prático para caminhos no Windows (`MAX_PATH` (260) - 1 para nulo).
- `SAFE_PATH_MARGIN = 10`: Uma margem subtraída do `EFFECTIVE_MAX_PATH` para garantir que o truncamento não resulte em um caminho exatamente no limite.
- `LOG_FOLDER_NAME = "ERROS"`: Nome padrão da pasta de logs.
- `DEFAULT_EXCLUDED_FOLDERS = ["erros", "anos anteriores"]`: Pastas ignoradas no processamento.
- `FALLBACK_SANITIZED_FILENAME = "arquivo_renomeado"`: Nome usado se a sanitização resultar em um nome vazio.
- `MAX_DUPLICATE_RESOLUTION_ATTEMPTS = 10`: Número máximo de tentativas para resolver nomes duplicados.

Requisitos Técnicos (para `arquivo_raiz.exe`)

- **Sistema Operacional:** O executável (`.exe`) é projetado para rodar em sistemas Windows.

- **Dependências:** Nenhuma instalação adicional é necessária para executar o arquivo `.exe`, pois todas as dependências (como Python e bibliotecas necessárias) estão empacotadas nele.

Este programa é uma ferramenta poderosa para organizar pastas com muitos arquivos dispersos em subdiretórios, garantindo nomes de arquivos válidos e gerenciando conflitos de forma robusta, ao mesmo tempo que fornece um registro detalhado das operações e erros através de logs e uma interface gráfica para feedback.