```python
import socket
import sys

TCP_IP = "127.0.0.1"
FILE_PORT = 5005
DATA_PORT = 5006
buf = 1024
file_name = sys.argv[1]


try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((TCP_IP, FILE_PORT))
    sock.send(file_name)
    sock.close()

    print "Sending %s ..." % file_name

    f = open(file_name, "rb")
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((TCP_IP, DATA_PORT))
    data = f.read()
    sock.send(data)

finally:
    sock.close()
    f.close()
```

1. `import socket`: Isso importa o módulo `socket` para que você possa usar recursos de socket para comunicação de rede.
2. `import sys`: Este é um módulo que fornece acesso a algumas variáveis usadas ou mantidas pelo interpretador Python, como `sys.argv`, que permite acessar os argumentos da linha de comando.
3. `TCP_IP = "127.0.0.1"`: Define o endereço IP do servidor para `127.0.0.1`, que é o endereço de loopback. Isso significa que o servidor está sendo executado na mesma máquina em que este código está sendo executado.
4. `FILE_PORT = 5005` e `DATA_PORT = 5006`: Define as portas em que o servidor estará ouvindo para a transferência de informações. A porta `5005` é usada para enviar o nome do arquivo, enquanto a porta `5006` é usada para enviar os dados do arquivo.
5. `buf = 1024`: Define o tamanho do buffer para ler partes do arquivo em pedaços de 1024 bytes.
6. `file_name = sys.argv[1]`: Obtém o nome do arquivo a ser enviado como um argumento da linha de comando. Isso assume que o usuário deve fornecer o nome do arquivo como um argumento ao executar o programa.
7. O bloco `try` inicia aqui e é usado para tratar exceções que podem ocorrer durante a execução do código.

8. `sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`: Cria um objeto de soquete. Este é o soquete que será usado para se conectar ao servidor.
9. `sock.connect((TCP_IP, FILE_PORT))`: Tenta conectar o soquete ao endereço IP e porta especificados (no caso, `TCP_IP` e `FILE_PORT`). Isso estabelece a primeira conexão para enviar o nome do arquivo.
10. `sock.send(file_name)`: Envia o nome do arquivo para o servidor. Observe que isso pode causar um erro em Python 3.x, já que `send` espera bytes, e `file_name` é uma string. Para compatibilidade, você pode codificar a string em bytes, por exemplo: `sock.send(file_name.encode())`.
11. `sock.close()`: Fecha a primeira conexão.
12. `f = open(file_name, "rb")`: Abre o arquivo especificado no modo binário de leitura ("rb"). Isso prepara o arquivo para leitura.
13. `sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`: Cria outro objeto de soquete. Este será usado para enviar os dados do arquivo.
14. `sock.connect((TCP_IP, DATA_PORT))`: Tenta conectar o novo soquete ao endereço IP e porta (aqui, `DATA_PORT`) para enviar os dados do arquivo.
15. `data = f.read()`: Lê o conteúdo do arquivo para a variável `data`.
16. `sock.send(data)`: Envia os dados do arquivo para o servidor.
17. O bloco `finally` inicia aqui e é usado para garantir que os sockets e o arquivo sejam fechados, independentemente de qualquer exceção que possa ocorrer.
18. `sock.close()`: Fecha o segundo soquete usado para enviar dados.
19. `f.close()`: Fecha o arquivo que foi lido.

```python
import socket
import select

UDP_IP = "127.0.0.1"
IN_PORT = 5005
timeout = 3


sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, IN_PORT))

while True:
    data, addr = sock.recvfrom(1024)
    if data:
        print "File name:", data
        file_name = data.strip()

    f = open(file_name, 'wb')

    while True:
        ready = select.select([sock], [], [], timeout)
        if ready[0]:
            data, addr = sock.recvfrom(1024)
            f.write(data)
        else:
            print "%s Finish!" % file_name
            f.close()
            break
```

1. `import socket` e `import select`: Importa os módulos `socket` e `select` para lidar com comunicação de rede por UDP e operações de seleção (espera até que os dados estejam prontos para leitura).
2. `UDP_IP = "127.0.0.1"`: Define o endereço IP local, neste caso, o endereço de loopback (`localhost`).
3. `IN_PORT = 5005`: Define a porta em que o servidor UDP estará ouvindo.
4. `timeout = 3`: Define um tempo limite (em segundos) para esperar por dados de entrada antes de encerrar a transferência.
5. `sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`: Cria um socket UDP.
6. `sock.bind((UDP_IP, IN_PORT))`: Vincula o socket ao endereço IP e à porta especificados. O servidor está pronto para receber dados na porta e no endereço especificados.
7. Entra em um loop infinito (`while True`) para esperar por conexões UDP.
8. `data, addr = sock.recvfrom(1024)`: Recebe dados (nesse caso, o nome do arquivo) de um cliente UDP. `data` contém os dados recebidos e `addr` contém o endereço do cliente que enviou os dados.
9. `if data:`: Verifica se `data` contém algo (ou seja, se o nome do arquivo foi recebido com sucesso).
10. `print "File name:", data`: Exibe o nome do arquivo recebido no console.
11. `file_name = data.strip()`: Remove qualquer espaço em branco em excesso do nome do arquivo e armazena-o na variável `file_name`.

12. Abre o arquivo especificado no modo de escrita binária (`'wb'`). Este arquivo será usado para armazenar os dados recebidos.
13. Entra em um segundo loop infinito para receber os dados reais do arquivo.
14. `ready = select.select([sock], [], [], timeout)`: Usa a função `select.select()` para verificar se há dados prontos para serem lidos a partir do socket. Isso permite que o servidor aguarde por um determinado período (`timeout`) antes de verificar novamente os dados de entrada.
15. `if ready[0]:`: Verifica se há dados prontos para serem lidos do socket.
16. `data, addr = sock.recvfrom(1024)`: Recebe os dados (dados do arquivo) do cliente UDP.
17. `f.write(data)`: Escreve os dados no arquivo aberto. Isso acontece enquanto os dados do arquivo estiverem sendo recebidos.
18. Se nenhum dado estiver pronto para leitura após o tempo limite especificado, ele imprime uma mensagem indicando que a transferência foi concluída e fecha o arquivo.