```
class ContiguousAllocation:
    def __init__(self, total_blocks):
        self.total_blocks = total_blocks
        self.blocks = [None] * total_blocks

    def allocate(self, file_name, size):
        for i in range(self.total_blocks - size + 1):
            if all(self.blocks[i + j] is None for j in range(size)):
                for j in range(size):
                    self.blocks[i + j] = file_name
                print(f"File '{file_name}' allocated at blocks {i} to {i + size - 1}")
                return
        print(f"Not enough space to allocate file '{file_name}'")

    def deallocate(self, file_name):
        for i in range(self.total_blocks):
            if self.blocks[i] == file_name:
                self.blocks[i] = None
        print(f"File '{file_name}' deallocated.")

    def display(self):
        print("Disk blocks:", self.blocks)


contiguous = ContiguousAllocation(10)
contiguous.allocate("File1", 3)
contiguous.allocate("File2", 4)
contiguous.deallocate("File1")
contiguous.display()
```

```
import socket
import signal
import os

def handle_client(client_socket):
    data = client_socket.recv(1024)
    if data:
        client_socket.send(data)
    client_socket.close()

def server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8080))
    server_socket.listen(5)

    signal.signal(signal.SIGINT, lambda signum, frame: os._exit(0))

    while True:
        client_socket, addr = server_socket.accept()
        handle_client(client_socket)

if __name__ == '__main__':
    server()
```

```
class LinkedAllocation:
    def __init__(self, total_blocks):
        self.total_blocks = total_blocks
        self.blocks = [None] * total_blocks

    def allocate(self, file_name, size):
        prev_block = None
        for i in range(self.total_blocks):
            if self.blocks[i] is None:
                if prev_block is None:
                    self.blocks[i] = [file_name, None]
                else:
                    self.blocks[prev_block][1] = i
                    self.blocks[i] = [file_name, None]
                prev_block = i
                size -= 1
                if size == 0:
                    print(f"File '{file_name}' allocated.")
                    return
        print(f"Not enough space to allocate file '{file_name}'")

    def deallocate(self, file_name):
        for i in range(self.total_blocks):
            if self.blocks[i] and self.blocks[i][0] == file_name:
                next_block = self.blocks[i][1]
                while next_block is not None:
                    self.blocks[i] = None
                    i = next_block
                    next_block = self.blocks[i][1]
                self.blocks[i] = None
        print(f"File '{file_name}' deallocated.")

    def display(self):
        print("Disk blocks:", self.blocks)


linked = LinkedAllocation(10)
linked.allocate("File1", 3)
linked.allocate("File2", 2)
linked.deallocate("File1")
linked.display()
```

```
import socket

def client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 8080))

    client_socket.send(b'Hello, Server!')
    data = client_socket.recv(1024)
    print(f'Received from server: {data.decode()}')
    client_socket.close()

if __name__ == '__main__':
    client()
```

```
class IndexedAllocation:
    def __init__(self, total_blocks):
        self.total_blocks = total_blocks
        self.blocks = [None] * total_blocks
        self.index_blocks = {}

    def allocate(self, file_name, size):
        index_block = []
        for i in range(self.total_blocks):
            if self.blocks[i] is None:
                index_block.append(i)
                if len(index_block) == size:
                    self.index_blocks[file_name] = index_block
                    for block in index_block:
                        self.blocks[block] = file_name
                    print(f"File '{file_name}' allocated with index block {index_block}.")
                    return
        print(f"Not enough space to allocate file '{file_name}'")

    def deallocate(self, file_name):
        if file_name in self.index_blocks:
            index_block = self.index_blocks[file_name]
            for block in index_block:
                self.blocks[block] = None
            del self.index_blocks[file_name]
            print(f"File '{file_name}' deallocated.")
        else:
            print(f"File '{file_name}' not found.")

    def display(self):
        print("Disk blocks:", self.blocks)
        print("Index blocks:", self.index_blocks)


indexed = IndexedAllocation(10)
indexed.allocate("File1", 3)
indexed.allocate("File2", 4)
indexed.deallocate("File1")
indexed.display()
```