Problem 1500: Design a File Sharing System

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

We will use a file-sharing system to share a very large file which consists of

m

small

chunks

with IDs from

1

to

m

.

When users join the system, the system should assign

a unique

ID to them. The unique ID should be used

once

for each user, but when a user leaves the system, the ID can be

reused

again.

Users can request a certain chunk of the file, the system should return a list of IDs of all the users who own this chunk. If the user receives a non-empty list of IDs, they receive the requested chunk successfully.

Implement the

FileSharing

class:

FileSharing(int m)

Initializes the object with a file of

m

chunks.

int join(int[] ownedChunks)

: A new user joined the system owning some chunks of the file, the system should assign an id to the user which is the

smallest positive integer

not taken by any other user. Return the assigned id.

void leave(int userID)

: The user with

userID

will leave the system, you cannot take file chunks from them anymore.

int[] request(int userID, int chunkID)

: The user

userID

requested the file chunk with

chunkID

. Return a list of the IDs of all users that own this chunk sorted in ascending order.

Example:

Input:

["FileSharing","join","join","join","request","request","leave","request","leave","join"]
[[4],[[1,2]],[[2,3]],[[4]],[1,3],[2,2],[1],[2,1],[2],[[]]]

Output:

[null,1,2,3,[2],[1,2],null,[],null,1]

Explanation:

FileSharing fileSharing = new FileSharing(4); // We use the system to share a file of 4 chunks.

fileSharing.join([1, 2]); // A user who has chunks [1,2] joined the system, assign id = 1 to them and return 1.

fileSharing.join([2, 3]); // A user who has chunks [2,3] joined the system, assign id = 2 to them and return 2.

fileSharing.join([4]); // A user who has chunk [4] joined the system, assign id = 3 to them and return 3.

fileSharing.request(1, 3); // The user with id = 1 requested the third file chunk, as only the user with id = 2 has the file, return [2] . Notice that user 1 now has chunks [1,2,3].

fileSharing.request(2, 2); // The user with id = 2 requested the second file chunk, users with ids [1,2] have this chunk, thus we return [1,2].

fileSharing.leave(1); // The user with id = 1 left the system, all the file chunks with them are no longer available for other users.

fileSharing.request(2, 1); // The user with id = 2 requested the first file chunk, no one in the system has this chunk, we return empty list [].

fileSharing.leave(2); // The user with id = 2 left the system.

fileSharing.join([]); // A user who doesn't have any chunks joined the system, assign id = 1 to them and return 1. Notice that ids 1 and 2 are free and we can reuse them.

Constraints:

1 <= m <= 10

5

0 <= ownedChunks.length <= min(100, m)

1 <= ownedChunks[i] <= m

Values of

ownedChunks

are unique.

1 <= chunkID <= m

userID

is guaranteed to be a user in the system if you

assign

the IDs

correctly

.

At most

10

4

calls will be made to

join

,

leave

and

request

.

Each call to

leave

will have a matching call for

join

.

Follow-up:

What happens if the system identifies the user by their IP address instead of their unique ID and users disconnect and connect from the system with the same IP?

If the users in the system join and leave the system frequently without requesting any chunks, will your solution still be efficient?

If all users join the system one time, request all files, and then leave, will your solution still be efficient?

If the system will be used to share

n

files where the

ith

file consists of

m[i]

, what are the changes you have to make?

Code Snippets

C++:

```
class FileSharing {
public:
FileSharing(int m) {

}

int join(vector<int> ownedChunks) {

}

void leave(int userID) {
```

```
}

vector<int> request(int userID, int chunkID) {

}
};

/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing* obj = new FileSharing(m);
* int param_1 = obj->join(ownedChunks);
* obj->leave(userID);
* vector<int> param_3 = obj->request(userID,chunkID);
*/
```

Java:

```
class FileSharing {

public FileSharing(int m) {

}

public int join(List<Integer> ownedChunks) {

}

public void leave(int userID) {

}

public List<Integer> request(int userID, int chunkID) {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing obj = new FileSharing(m);
* int param_1 = obj.join(ownedChunks);
* obj.leave(userID);
* List<Integer> param_3 = obj.request(userID,chunkID);
```

```
*/
```

Python3:

```
class FileSharing:

def __init__(self, m: int):


def join(self, ownedChunks: List[int]) -> int:


def leave(self, userID: int) -> None:


def request(self, userID: int, chunkID: int) -> List[int]:



# Your FileSharing object will be instantiated and called as such:
# obj = FileSharing(m)
# param_1 = obj.join(ownedChunks)
# obj.leave(userID)
# param_3 = obj.request(userID,chunkID)
```

Python:

```
class FileSharing(object):

def __init__(self, m):
"""
:type m: int
"""



def join(self, ownedChunks):
"""
:type ownedChunks: List[int]
:rtype: int
"""
```

```
def leave(self, userID):
"""
:type userID: int
:rtype: None
"""


def request(self, userID, chunkID):
"""
:type userID: int
:type chunkID: int
:rtype: List[int]
"""



# Your FileSharing object will be instantiated and called as such:
# obj = FileSharing(m)
# param_1 = obj.join(ownedChunks)
# obj.leave(userID)
# param_3 = obj.request(userID,chunkID)
```

JavaScript:

```
/**
* @param {number} m
*/
var FileSharing = function(m) {

};

/**
* @param {number[]} ownedChunks
* @return {number}
*/
FileSharing.prototype.join = function(ownedChunks) {

};

/**
* @param {number} userID
* @return {void}
```

```
*/
FileSharing.prototype.leave = function(userID) {

};

/**
* @param {number} userID
* @param {number} chunkID
* @return {number[]}
*/
FileSharing.prototype.request = function(userID, chunkID) {

};

/**
* Your FileSharing object will be instantiated and called as such:
* var obj = new FileSharing(m)
* var param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* var param_3 = obj.request(userID,chunkID)
*/
```

TypeScript:

```
class FileSharing {
constructor(m: number) {

}

join(ownedChunks: number[]): number {

}

leave(userID: number): void {

}

request(userID: number, chunkID: number): number[] {

}
}
```

```
/**
* Your FileSharing object will be instantiated and called as such:
* var obj = new FileSharing(m)
* var param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* var param_3 = obj.request(userID,chunkID)
*/
```

C#:

```
public class FileSharing {

public FileSharing(int m) {

}

public int Join(IList<int> ownedChunks) {

}

public void Leave(int userID) {

}

public IList<int> Request(int userID, int chunkID) {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing obj = new FileSharing(m);
* int param_1 = obj.Join(ownedChunks);
* obj.Leave(userID);
* IList<int> param_3 = obj.Request(userID,chunkID);
*/
```

C:

```
```

```
typedef struct {

} FileSharing;


FileSharing* fileSharingCreate(int m) {

}

int fileSharingJoin(FileSharing* obj, int* ownedChunks, int ownedChunksSize)
{

}

void fileSharingLeave(FileSharing* obj, int userID) {

}

int* fileSharingRequest(FileSharing* obj, int userID, int chunkID, int*
retSize) {

}

void fileSharingFree(FileSharing* obj) {

}

/**
* Your FileSharing struct will be instantiated and called as such:
* FileSharing* obj = fileSharingCreate(m);
* int param_1 = fileSharingJoin(obj, ownedChunks, ownedChunksSize);

* fileSharingLeave(obj, userID);

* int* param_3 = fileSharingRequest(obj, userID, chunkID, retSize);

* fileSharingFree(obj);
*/
```

Go:

```
type FileSharing struct {

}


func Constructor(m int) FileSharing {

}


func (this *FileSharing) Join(ownedChunks []int) int {

}


func (this *FileSharing) Leave(userID int) {

}


func (this *FileSharing) Request(userID int, chunkID int) []int {

}


/**
* Your FileSharing object will be instantiated and called as such:
* obj := Constructor(m);
* param_1 := obj.Join(ownedChunks);
* obj.Leave(userID);
* param_3 := obj.Request(userID,chunkID);
*/
```

Kotlin:

```
class FileSharing(m: Int) {

fun join(ownedChunks: List<Int>): Int {

}

fun leave(userID: Int) {
```

```
}

fun request(userID: Int, chunkID: Int): List<Int> {

}

}

/**
* Your FileSharing object will be instantiated and called as such:
* var obj = FileSharing(m)
* var param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* var param_3 = obj.request(userID,chunkID)
*/
```

Swift:

```
class FileSharing {

init(_ m: Int) {

}

func join(_ ownedChunks: [Int]) -> Int {

}

func leave(_ userID: Int) {

}

func request(_ userID: Int, _ chunkID: Int) -> [Int] {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* let obj = FileSharing(m)
* let ret_1: Int = obj.join(ownedChunks)
```

```
* obj.leave(userID)
* let ret_3: [Int] = obj.request(userID, chunkID)
*/
```

Rust:

```
struct FileSharing {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl FileSharing {

fn new(m: i32) -> Self {

}

fn join(&self, owned_chunks: Vec<i32>) -> i32 {

}

fn leave(&self, user_id: i32) {

}

fn request(&self, user_id: i32, chunk_id: i32) -> Vec<i32> {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* let obj = FileSharing::new(m);
* let ret_1: i32 = obj.join(ownedChunks);
* obj.leave(userID);
* let ret_3: Vec<i32> = obj.request(userID, chunkID);
*/
```

Ruby:

```
class FileSharing

=begin
:type m: Integer
=end
def initialize(m)

end


=begin
:type owned_chunks: Integer[]
:rtype: Integer
=end
def join(owned_chunks)

end


=begin
:type user_id: Integer
:rtype: Void
=end
def leave(user_id)

end


=begin
:type user_id: Integer
:type chunk_id: Integer
:rtype: Integer[]
=end
def request(user_id, chunk_id)

end


end
```

```
# Your FileSharing object will be instantiated and called as such:
# obj = FileSharing.new(m)
# param_1 = obj.join(owned_chunks)
# obj.leave(user_id)
# param_3 = obj.request(user_id, chunk_id)
```

PHP:

```
class FileSharing {
/**
* @param Integer $m
*/
function __construct($m) {

}

/**
* @param Integer[] $ownedChunks
* @return Integer
*/
function join($ownedChunks) {

}

/**
* @param Integer $userID
* @return NULL
*/
function leave($userID) {

}

/**
* @param Integer $userID
* @param Integer $chunkID
* @return Integer[]
*/
function request($userID, $chunkID) {

}
}
```

```
/**
* Your FileSharing object will be instantiated and called as such:
* $obj = FileSharing($m);
* $ret_1 = $obj->join($ownedChunks);
* $obj->leave($userID);
* $ret_3 = $obj->request($userID, $chunkID);
*/
```

Dart:

```
class FileSharing {

FileSharing(int m) {

}

int join(List<int> ownedChunks) {

}

void leave(int userID) {

}

List<int> request(int userID, int chunkID) {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing obj = FileSharing(m);
* int param1 = obj.join(ownedChunks);
* obj.leave(userID);
* List<int> param3 = obj.request(userID,chunkID);
*/
```

Scala:

```
class FileSharing(_m: Int) {

def join(ownedChunks: List[Int]): Int = {
```

```
}

def leave(userID: Int): Unit = {

}

def request(userID: Int, chunkID: Int): List[Int] = {

}

}

/**
* Your FileSharing object will be instantiated and called as such:
* val obj = new FileSharing(m)
* val param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* val param_3 = obj.request(userID,chunkID)
*/
```

Elixir:

```
defmodule FileSharing do
@spec init_(m :: integer) :: any
def init_(m) do

end

@spec join(owned_chunks :: [integer]) :: integer
def join(owned_chunks) do

end

@spec leave(user_id :: integer) :: any
def leave(user_id) do

end

@spec request(user_id :: integer, chunk_id :: integer) :: [integer]
def request(user_id, chunk_id) do
```

```
end
end

# Your functions will be called as such:
# FileSharing.init_(m)
# param_1 = FileSharing.join(owned_chunks)
# FileSharing.leave(user_id)
# param_3 = FileSharing.request(user_id, chunk_id)

# FileSharing.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

```
-spec file_sharing_init_(M :: integer()) -> any().
file_sharing_init_(M) ->
.

-spec file_sharing_join(OwnedChunks :: [integer()]) -> integer().
file_sharing_join(OwnedChunks) ->
.

-spec file_sharing_leave(UserID :: integer()) -> any().
file_sharing_leave(UserID) ->
.

-spec file_sharing_request(UserID :: integer(), ChunkID :: integer()) ->
[integer()].
file_sharing_request(UserID, ChunkID) ->
.


%% Your functions will be called as such:
%% file_sharing_init_(M),
%% Param_1 = file_sharing_join(OwnedChunks),
%% file_sharing_leave(UserID),
%% Param_3 = file_sharing_request(UserID, ChunkID),

%% file_sharing_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket:

```
(define file-sharing%
(class object%
(super-new)

; m : exact-integer?
(init-field
m)

; join : (listof exact-integer?) -> exact-integer?
(define/public (join owned-chunks)
)
; leave : exact-integer? -> void?
(define/public (leave user-id)
)
; request : exact-integer? exact-integer? -> (listof exact-integer?)
(define/public (request user-id chunk-id)
)))

;; Your file-sharing% object will be instantiated and called as such:
;; (define obj (new file-sharing% [m m]))
;; (define param_1 (send obj join owned-chunks))
;; (send obj leave user-id)
;; (define param_3 (send obj request user-id chunk-id))
```

Solutions

C++ Solution:

```
/*
* Problem: Design a File Sharing System
* Difficulty: Medium
* Tags: hash, sort, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class FileSharing {
```

```
public:
FileSharing(int m) {

}

int join(vector<int> ownedChunks) {

}

void leave(int userID) {

}

vector<int> request(int userID, int chunkID) {

}
};

/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing* obj = new FileSharing(m);
* int param_1 = obj->join(ownedChunks);
* obj->leave(userID);
* vector<int> param_3 = obj->request(userID,chunkID);
*/
```

Java Solution:

```
/**
* Problem: Design a File Sharing System
* Difficulty: Medium
* Tags: hash, sort, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class FileSharing {

public FileSharing(int m) {
```

```
}

public int join(List<Integer> ownedChunks) {

}

public void leave(int userID) {

}

public List<Integer> request(int userID, int chunkID) {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing obj = new FileSharing(m);
* int param_1 = obj.join(ownedChunks);
* obj.leave(userID);
* List<Integer> param_3 = obj.request(userID,chunkID);
*/
```

Python3 Solution:

```
"""
Problem: Design a File Sharing System
Difficulty: Medium
Tags: hash, sort, queue, heap

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

class FileSharing:

def __init__(self, m: int):
```

```
def join(self, ownedChunks: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class FileSharing(object):

def __init__(self, m):
"""
:type m: int
"""



def join(self, ownedChunks):
"""
:type ownedChunks: List[int]
:rtype: int
"""



def leave(self, userID):
"""
:type userID: int
:rtype: None
"""



def request(self, userID, chunkID):
"""
:type userID: int
:type chunkID: int
:rtype: List[int]
"""




# Your FileSharing object will be instantiated and called as such:
# obj = FileSharing(m)
# param_1 = obj.join(ownedChunks)
# obj.leave(userID)
```

```
# param_3 = obj.request(userID,chunkID)
```

JavaScript Solution:

```
/**
* Problem: Design a File Sharing System
* Difficulty: Medium
* Tags: hash, sort, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} m
*/
var FileSharing = function(m) {

};

/**
* @param {number[]} ownedChunks
* @return {number}
*/
FileSharing.prototype.join = function(ownedChunks) {

};

/**
* @param {number} userID
* @return {void}
*/
FileSharing.prototype.leave = function(userID) {

};

/**
* @param {number} userID
* @param {number} chunkID
* @return {number[]}
```

```
*/
FileSharing.prototype.request = function(userID, chunkID) {

};

/**
* Your FileSharing object will be instantiated and called as such:
* var obj = new FileSharing(m)
* var param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* var param_3 = obj.request(userID,chunkID)
*/
```

TypeScript Solution:

```
/**
* Problem: Design a File Sharing System
* Difficulty: Medium
* Tags: hash, sort, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class FileSharing {
constructor(m: number) {

}

join(ownedChunks: number[]): number {

}

leave(userID: number): void {

}

request(userID: number, chunkID: number): number[] {

}
```

```
}

/**
* Your FileSharing object will be instantiated and called as such:
* var obj = new FileSharing(m)
* var param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* var param_3 = obj.request(userID,chunkID)
*/
```

C# Solution:

```
/*
* Problem: Design a File Sharing System
* Difficulty: Medium
* Tags: hash, sort, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class FileSharing {

public FileSharing(int m) {

}

public int Join(IList<int> ownedChunks) {

}

public void Leave(int userID) {

}

public IList<int> Request(int userID, int chunkID) {

}
}
```

```
/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing obj = new FileSharing(m);
* int param_1 = obj.Join(ownedChunks);
* obj.Leave(userID);
* IList<int> param_3 = obj.Request(userID,chunkID);
*/
```

C Solution:

```
/*
* Problem: Design a File Sharing System
* Difficulty: Medium
* Tags: hash, sort, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/




typedef struct {

} FileSharing;


FileSharing* fileSharingCreate(int m) {

}

int fileSharingJoin(FileSharing* obj, int* ownedChunks, int ownedChunksSize)
{

}

void fileSharingLeave(FileSharing* obj, int userID) {

}
```

```
int* fileSharingRequest(FileSharing* obj, int userID, int chunkID, int*
retSize) {

}

void fileSharingFree(FileSharing* obj) {

}

/**
* Your FileSharing struct will be instantiated and called as such:
* FileSharing* obj = fileSharingCreate(m);
* int param_1 = fileSharingJoin(obj, ownedChunks, ownedChunksSize);

* fileSharingLeave(obj, userID);

* int* param_3 = fileSharingRequest(obj, userID, chunkID, retSize);

* fileSharingFree(obj);
*/
```

Go Solution:

```
// Problem: Design a File Sharing System
// Difficulty: Medium
// Tags: hash, sort, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

type FileSharing struct {

}


func Constructor(m int) FileSharing {

}
```

```
func (this *FileSharing) Join(ownedChunks []int) int {

}


func (this *FileSharing) Leave(userID int) {

}


func (this *FileSharing) Request(userID int, chunkID int) []int {

}


/**
* Your FileSharing object will be instantiated and called as such:
* obj := Constructor(m);
* param_1 := obj.Join(ownedChunks);
* obj.Leave(userID);
* param_3 := obj.Request(userID,chunkID);
*/
```

Kotlin Solution:

```
class FileSharing(m: Int) {

fun join(ownedChunks: List<Int>): Int {

}

fun leave(userID: Int) {

}

fun request(userID: Int, chunkID: Int): List<Int> {

}

}
```

```
/**
* Your FileSharing object will be instantiated and called as such:
* var obj = FileSharing(m)
* var param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* var param_3 = obj.request(userID,chunkID)
*/
```

Swift Solution:

```
class FileSharing {

init(_ m: Int) {

}

func join(_ ownedChunks: [Int]) -> Int {

}

func leave(_ userID: Int) {

}

func request(_ userID: Int, _ chunkID: Int) -> [Int] {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* let obj = FileSharing(m)
* let ret_1: Int = obj.join(ownedChunks)
* obj.leave(userID)
* let ret_3: [Int] = obj.request(userID, chunkID)
*/
```

Rust Solution:

```
// Problem: Design a File Sharing System
// Difficulty: Medium
// Tags: hash, sort, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

struct FileSharing {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl FileSharing {

fn new(m: i32) -> Self {

}

fn join(&self, owned_chunks: Vec<i32>) -> i32 {

}

fn leave(&self, user_id: i32) {

}

fn request(&self, user_id: i32, chunk_id: i32) -> Vec<i32> {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* let obj = FileSharing::new(m);
* let ret_1: i32 = obj.join(ownedChunks);
* obj.leave(userID);
* let ret_3: Vec<i32> = obj.request(userID, chunkID);
*/
```

Ruby Solution:

```
class FileSharing

=begin
:type m: Integer
=end
def initialize(m)

end


=begin
:type owned_chunks: Integer[]
:rtype: Integer
=end
def join(owned_chunks)

end


=begin
:type user_id: Integer
:rtype: Void
=end
def leave(user_id)

end


=begin
:type user_id: Integer
:type chunk_id: Integer
:rtype: Integer[]
=end
def request(user_id, chunk_id)

end


end
```

```
# Your FileSharing object will be instantiated and called as such:
# obj = FileSharing.new(m)
# param_1 = obj.join(owned_chunks)
# obj.leave(user_id)
# param_3 = obj.request(user_id, chunk_id)
```

PHP Solution:

```
class FileSharing {
/**
* @param Integer $m
*/
function __construct($m) {

}

/**
* @param Integer[] $ownedChunks
* @return Integer
*/
function join($ownedChunks) {

}

/**
* @param Integer $userID
* @return NULL
*/
function leave($userID) {

}

/**
* @param Integer $userID
* @param Integer $chunkID
* @return Integer[]
*/
function request($userID, $chunkID) {

}
}
```

```
/**
* Your FileSharing object will be instantiated and called as such:
* $obj = FileSharing($m);
* $ret_1 = $obj->join($ownedChunks);
* $obj->leave($userID);
* $ret_3 = $obj->request($userID, $chunkID);
*/
```

Dart Solution:

```
class FileSharing {

FileSharing(int m) {

}

int join(List<int> ownedChunks) {

}

void leave(int userID) {

}

List<int> request(int userID, int chunkID) {

}
}

/**
* Your FileSharing object will be instantiated and called as such:
* FileSharing obj = FileSharing(m);
* int param1 = obj.join(ownedChunks);
* obj.leave(userID);
* List<int> param3 = obj.request(userID,chunkID);
*/
```

Scala Solution:

```
class FileSharing(_m: Int) {

def join(ownedChunks: List[Int]): Int = {

}

def leave(userID: Int): Unit = {

}

def request(userID: Int, chunkID: Int): List[Int] = {

}

}

/**
* Your FileSharing object will be instantiated and called as such:
* val obj = new FileSharing(m)
* val param_1 = obj.join(ownedChunks)
* obj.leave(userID)
* val param_3 = obj.request(userID,chunkID)
*/
```

Elixir Solution:

```
defmodule FileSharing do
@spec init_(m :: integer) :: any
def init_(m) do

end

@spec join(owned_chunks :: [integer]) :: integer
def join(owned_chunks) do

end

@spec leave(user_id :: integer) :: any
def leave(user_id) do

end
```

```
@spec request(user_id :: integer, chunk_id :: integer) :: [integer]
def request(user_id, chunk_id) do

end
end

# Your functions will be called as such:
# FileSharing.init_(m)
# param_1 = FileSharing.join(owned_chunks)
# FileSharing.leave(user_id)
# param_3 = FileSharing.request(user_id, chunk_id)

# FileSharing.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
-spec file_sharing_init_(M :: integer()) -> any().
file_sharing_init_(M) ->
.

-spec file_sharing_join(OwnedChunks :: [integer()]) -> integer().
file_sharing_join(OwnedChunks) ->
.

-spec file_sharing_leave(UserID :: integer()) -> any().
file_sharing_leave(UserID) ->
.

-spec file_sharing_request(UserID :: integer(), ChunkID :: integer()) ->
[integer()].
file_sharing_request(UserID, ChunkID) ->
.


%% Your functions will be called as such:
%% file_sharing_init_(M),
%% Param_1 = file_sharing_join(OwnedChunks),
%% file_sharing_leave(UserID),
%% Param_3 = file_sharing_request(UserID, ChunkID),
```

```
%% file_sharing_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket Solution:

```
(define file-sharing%
(class object%
(super-new)

; m : exact-integer?
(init-field
m)

; join : (listof exact-integer?) -> exact-integer?
(define/public (join owned-chunks)
)
; leave : exact-integer? -> void?
(define/public (leave user-id)
)
; request : exact-integer? exact-integer? -> (listof exact-integer?)
(define/public (request user-id chunk-id)
)))

;; Your file-sharing% object will be instantiated and called as such:
;; (define obj (new file-sharing% [m m]))
;; (define param_1 (send obj join owned-chunks))
;; (send obj leave user-id)
;; (define param_3 (send obj request user-id chunk-id))
```