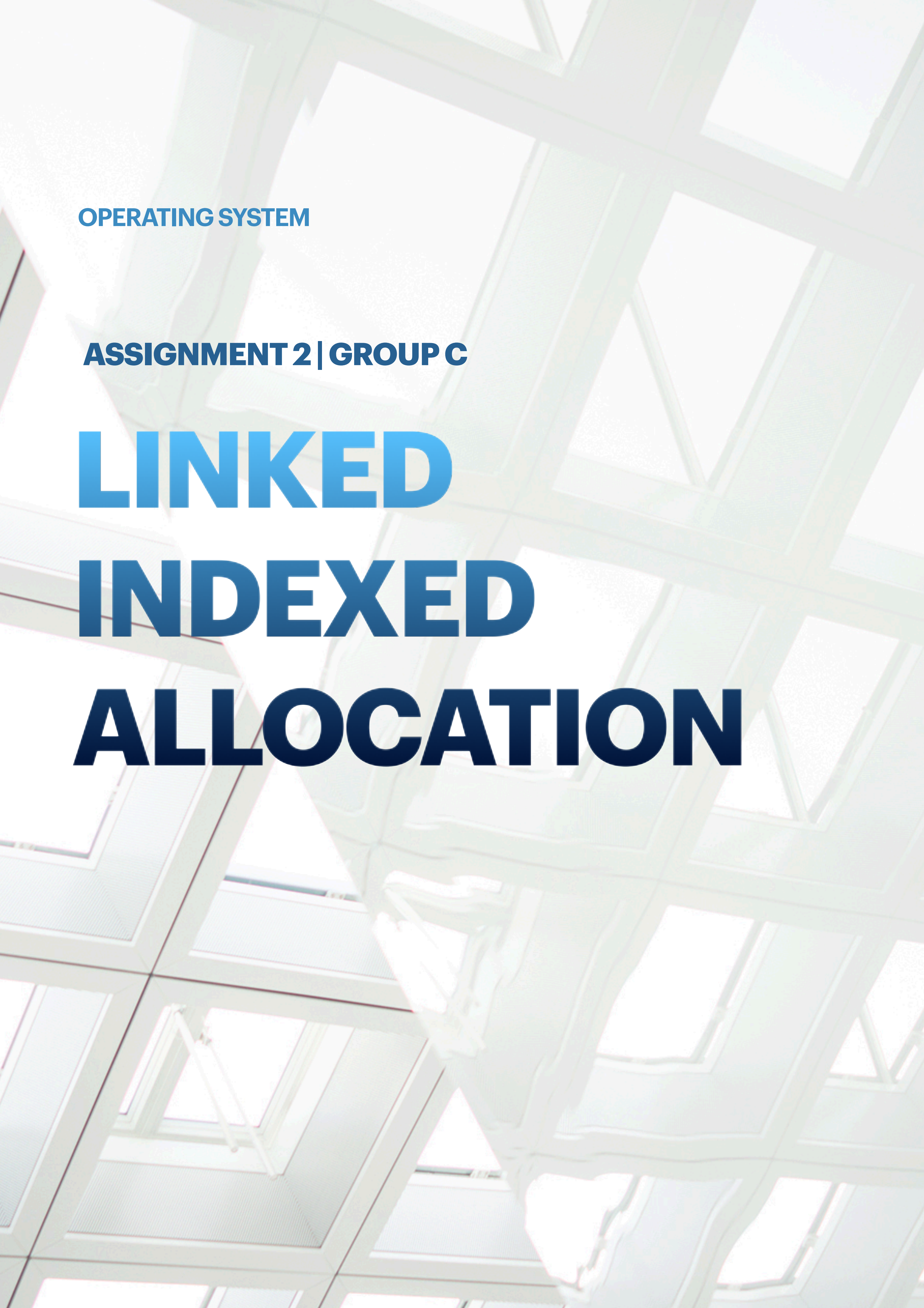


OPERATING SYSTEM

ASSIGNMENT 2 | GROUP C

LINKED INDEXED ALLOCATION

LINKED INDEXED ALLOCATION

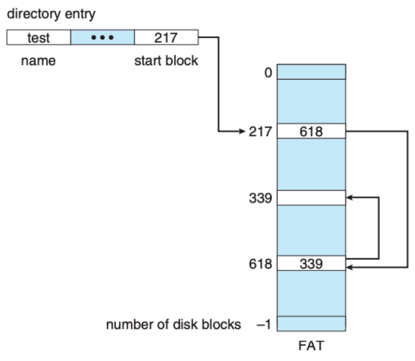
INTRODUCTION

The Linked indexed allocation is a file allocation method used in computer systems to organize and allocate space for files on a storage device, such as a hard disk. It is a combination of linked allocation and indexed allocation techniques.

WHAT DOES LINKED INDEX ALLOCATION SOLVE?

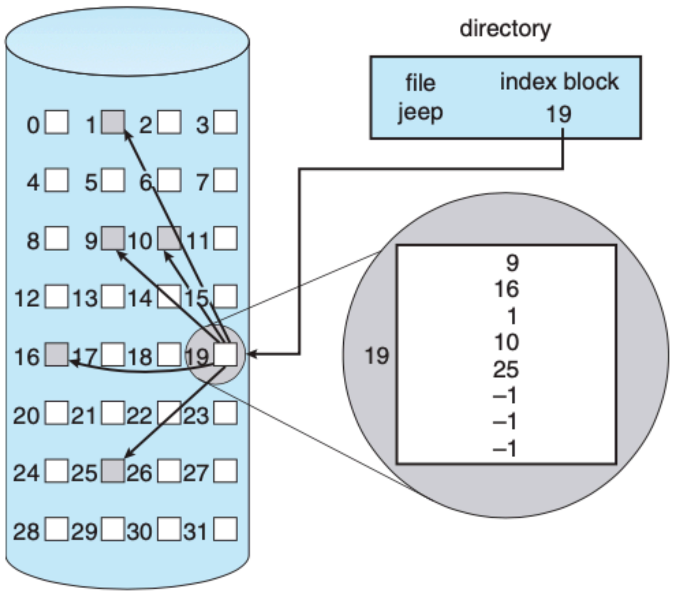
It solves external fragmentation and size declaration problems of contiguous allocation however, in the absence of FILE ALLOCATION TABLE, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order. **Indexed allocation** solves problem by bringing all the pointers together into one location the index block

FILE-ALLOCATION TABLE



Each file has its own index block, which is an array of disk-block addresses. The *i*th entry in the index block points to the *i*th block of the file. The directory contains the address of the index-block entry.This scheme is similar to the **paging scheme** When the file is created, all pointers in the index block are set to null. When the *i*th block is first written, a block is obtained from the free-space manager, and its address is put in the *i*th index-block entry. Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space. Indexed allocation does suffer from wasted space….

INDEXED ALLOCATION OF DISK SPACE



The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation. Consider a common case in which we have a file of only one or two blocks. With linked allocation, we lose the space of only one pointer per block. With indexed allocation, an entire index block must be allocated, even if only one or two pointers will be non-null. This point raises the question of how large the index block should be. Every file must have an index block, so we want the index block to be as small as possible. If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue. Mechanisms for this purpose include the following:

MECHANISMS TO SOLVE LARGE FILES ALLOCATION

- Linked scheme : An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we can link together several index blocks. For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses. The next address (the last word in the index block) is null (for a small file) or is a pointer to another index block (for a large file).

- Multilevel index : A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block. This approach could be continued to a third or fourth level, depending on the desired maximum file size. With 4,096-byte blocks, we could store 1,024 four-byte pointers in an index block. Two levels of indexes allow 1,048,576 data blocks and a file size of up to 4 GB.

- Combined scheme : Another alternative ,used inUNIX-based file systems, is to keep the first, say, 15 pointers of the index block in the file's inode. The first 12 of these pointers point to **direct blocks**; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small files (of no more than 12 blocks) do not need a separate index block. If the block size is 4 KB, then up to 48 KB of data can be accessed directly. The next three pointers point to **indirect blocks**. The first points to a **single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data. The second points to a **double indirect block**, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. The last pointer contains the address of a **triple indirect block**

PERFORMANCE

Indexed allocation is more complex. If the index block is already in memory, then the access can be made directly. However, keeping the index block in memory requires considerable space. If this memory space is not available, then we may have to read first the index block and then the desired data block. For a two-level index, two index-block reads might be necessary. For an extremely large file, accessing a block near the end of the file would require reading in all the index blocks before the needed data block finally could be read. Thus, the performance of indexed allocation depends on the index structure, on the size of the file, and on the position of the block desired.