This is a summary of my notes on operating systems file allocation methods. In the context of operating systems file allocation refers to managing files on disk such that disk space is effectively utilized and files are accessed quickly.

Contiguous Allocation

1.
    1. All blocks of the file are consecutive on disk
    2. Deleting files leaves gaps
    3. Compacting disk can be very slow
    4. Simple and efficient indexing
    5. Random access well supported
    6. Difficult to grow: need to preallocate which wastes space
    7. Easy to map from logical to physical address

Let us assume we are going to read some position (pos) in a file. Assuming a block size of (1024) here is how to map from logical to physical address:

```
1 Block number on disk = file start block number + (pos/1024)
2 Offset in block = pos%1024
```

Linked Allocation

1.
    1. Linked list of file blocks
    2. Blocks can be scattered on disk
    3. No limit on file size
    4. File can grow easily
    5. Random access well supported
    6. Random access is not easy

Using the same assumptions as before here is how to map from logical to physical address:

```
1 block = start
2 for (j = 0; j < pos/1024; j++)
3 {
4    block = block->next
5 }
6 Offset in block = pos%1024
```

Linked Allocation Table in RAM

Same as the previous file allocation method but the table resides in memory. It is faster but need to copy the table to disk at some point and keep both copies consistent.

Index Block Allocation

1.
    1. File block addresses are stored in an array which is stored in a disk block
    2. Directory has a pointer to index block
    3. Both random and sequential access are easy
    4. File size limitation depends on array size for example if we have (256) pointers in the index block then the maximum file size will be 256KB assuming 1KB data blocks
    5. Space utilization is good

6. Files can grow or shrink easily

Using the same assumptions as before here is how to map from logical to physical address:

```
1 block = index[pos/1024]
2 offset = pos%1024
```

Linked Index Block Allocation

1.
    1. Similar to linked file blocks but using index blocks instead of data blocks
    2. Good for large files
    3. Both random and sequential access are easy
    4. Random access is slow for large files

Using the same assumptions as before and assuming an index block size of 256 here is how to map from logical to physical address:

```
1 index = start
2 block num = pos/1024
3 for (j = 0; j < block num/255; j++)
4 {
5    index = index->next
6 }
7 block = index[block num%255]
8 offset = pos%1024
```

Two Level Index Block Allocation

1.
    1. Allow larger files by creating an index of index blocks
    2. Overhead is now at least two blocks per file

Using the same assumptions as before here is how to map from logical to physical address:

```
1 block num = pos/1024
2 index = start[block num/256]
3 block = index[block num%256]
4 offset = pos%1024
```

Block Allocation with Extents

1. Reduce space consumed by index pointers because often consecutive blocks in file are sequential on disk
2. Store block and count instead of block and index
3. Faster read and write but complex look up

Summary

Here is a tabular summary of what we have mentioned:

Algorithm	Properties
Contiguous Allocation	All blocks of the file are consecutive on disk Deleting files leaves gaps Compacting disk can be very slow Simple and efficient indexing Random access well supported

Algorithm	Properties
	Difficult to grow Easy to map from logical to physical address
Linked Allocation	Linked list of file blocks Blocks can be scattered on disk No limit on file size File can grow easily Random access well supported Random access is not easy
Linked Allocation Table in RAM	Fast Table resides in memory Need to copy the table to disk and keep both copies consistent
Index Block Allocation	File block addresses are stored in an array which is stored in a disk block Directory has a pointer to index block Both random and sequential access are easy File size limitation depends on array size Space utilization is good Files can grow or shrink easily
Linked Index Block Allocation	Similar to linked file blocks but using index blocks instead of data blocks Good for large files Both random and sequential access are easy Random access is slow for large files
Two Level Index Block Allocation	Allow larger files by creating an index of index blocks Overhead is now at least two blocks per file
Block Allocation with Extents	Reduce space consumed by index pointers Store block and count instead of block and index Faster read and write but complex look up

Was it a useful article ? if so, please use the comments section below for questions, corrections or feedback. Thanks for reading.