RECLAIMING SPACE IN FILES

Contents of today's lecture:

Reclaiming space in files
Reference: Folk, Zoellick and Riccardi, File Structures, 1998. Section 6.2

Motivation

Let us consider a file of records (fixed length or variable length).

We know how to create a file, add records to a file and modify the content of a record. These actions can be performed physically by using the various basic file operations we have seen (open a file for writing or appending, going to a position of a file using "seek" and writing the new information there).

What happens if records need to be deleted ?

There is no basic operation that allows us to "remove part of a file". Record deletion should be taken care by the program responsible for file organization.

Strategies for record deletion

How to delete records and reuse the unused space ?

1) Record Deletion and Storage Compaction

Deletion can be done by "marking" a record as deleted.

Ex.: - Place '*' at the beginning of the record; or
- Have a special field that indicates one of two states: "deleted" or "not deleted".

Note that the space for the record is not released, but the program that manipulates the file must include logic that checks if record is deleted or not.

After a lot of records have been deleted, a special program is used to squeeze the file - this is called **Storage Compaction**.

2) Deleting Fixed-Length Records and Reclaiming Space Dynamically

How to use the space of deleted records for storing records that are added later ?

Use an "AVAIL LIST", a linked list of available records.

- a header record (at the beginning of the file) stores the beginning of the AVAIL LIST (-1 can represent the null pointer).
- when a record is deleted, it is marked as deleted and inserted into the AVAIL LIST. The record space is in the same position as before, but it is logically placed into AVAIL LIST.

Ex.: After deletions the file may look like :

List head $\rightarrow$ 4

Edwards	Williams	*-1	Smith	*2	Sethi
0	1	2	3	4	5

If we add a record, it can go to the first available spot in the AVAIL LIST (RRN=4).

3) Deleting Variable Length Records

Use an AVAIL LIST as before, but take care of the variable-length difficulties.

The records in AVAIL LIST must store its size as a field.
RRN can not be used, but exact byte offset must be used.

List head → 33

Edwards|M|	Wu|F|	*-1|10|	Smith|M|	*15|30|
0	1	2	3	4
10 bytes	5 bytes	10 bytes	8 bytes	30 bytes

Addition of records must find a large enough record in AVAIL LIST.

Placement Strategies for New Records

There are several strategies for selecting a record from AVAIL LIST when adding a new record:

1. First-Fit Strategy
   - AVAIL LIST is not sorted by size.
   - First record large enough to hold new record is chosen.

   Example:
   AVAIL LIST: size=10, size=50, size=22, size=60
   record to be added: size=20
   Which record from AVAIL LIST is used for the new record ?

2. Best-Fit Strategy
   - AVAIL LIST is sorted by size.
   - Smallest record large enough to hold new record is chosen.

   Example:
   AVAIL LIST: size=10, size=22, size=50, size=60
   record to be added: size=20
   Which record from AVAIL LIST is used for the new record ?

3. Worst-Fit Strategy
   - AVAIL LIST is sorted by decreasing order of size.
   - Largest record is used for holding new record; unused space is placed again in AVAIL LIST.

   Example:
   AVAIL LIST: size=60, size=50, size=22, size=10
   record to be added: size=20
   Which record from AVAIL LIST is used for the new record ?

When choosing between strategies we must consider two types of fragmentation within a file:

Internal Fragmentation: wasted space within a record.
External Fragmentation: space is available at AVAIL LIST, but it is so small that cannot be reused.

For each of the following approaches, which type of fragmentation arises, and which placement strategy is more suitable?
When the added record is smaller than the item taken from AVAIL LIST:

- leave the space unused within the record
  type of fragmentation:
  suitable placement strategy:
- return the unused space as a new available record to AVAIL LIST
  type of fragmentation:
  suitable placement strategy:

Ways of combating external fragmentation:

- **Coalescing the holes** : if two records in AVAIL LIST are adjacent, combine them into a larger record.
- Minimize fragmentation by using one of the previously mentioned **placement strategies** (for example: worst-fit strategy is better than best-fit strategy in terms of external fragmentation when unused space is returned to AVAIL LIST).