2021 2-1 i233 Operating System Assignment#2 (Programing Project)

1. Create two programs, each with procedures (functions) that simulate the management of blocks in a storage system (SS), and a main procedure uses these procedures to allocate and free blocks repeatedly.
   (1) `bmc`: Bitmap based contiguous allocation. (Figure 4-10 on "Filesystems" pp.22)
   (2) `fbt`: Block table based allocation (Figure 4-12 on "Filesystems" pp.24)

2. Number of blocks to manage should be "`#define`"ed by a constant `SSBLK`. (So you can start with few blocks and blow it huge when it is working.)

3. Proposed API is as follows (`xxx` are substituted with `bmc` or `fbt`). You can come up with your original API design.
   - `int xxx_init()`
     Initialize everything including data structures for management.
     Returns `0` on success, `-1` on fail.
   - `int xxx_alloc(int nblks)`
     Performs allocation of `nblks`.
     Returns the block number of the first block of the allocated region on success allocation, `-1` if it couldn't allocate the requested blocks.
   - `int xxx_free(int blkno, int nblks)`
     Free `nblks` of blocks beginning with block `bn`.
     Returns `0` on success, `-1` on error.
   - `void xxx_dump()`
     Displays (prints) management data structure for debugging.
   - `void fbt_list(int blkno)`
     This is only for `fbt`.
     Displays (prints) a list of blocks beginning with `blkno`.
     If `blkno` is `-1`, displays a free block list.
   - `void xxx_verify()`
     A procedure that checks the integrity (validity) of the allocation.

4. Tests

There are numerous ways to test your programs, and you should come up with your own way of doing it. Following three methods are for reference.

(1) Simple test

Read a line with intended operation and parameter(s) and call `xxx_alloc()`, `xxx_free()` or other API procedures accordingly.

For example:

```
a 10    // Command requesting allocatinon of 10 blocks.
A 16 10 // The SS respond with output representing it has allocated
        // 10 blocks beginning with block 16.
```

```
d 16    // Command requesting printing a block list beginning with
        // block 16 (this is for fbt only).
…
d -1    // Command requesting printing a free block list.
…
d       // Command requesting full dumping of whole management structure.
…
f 16 10 // Command requesting freeing 10 blocks beginning with block 16.
F 16 10 // The SS respond with success.
a 4000  // Command requesting allocation of 4000 blocks.
A failed // SS respond with an error
…
```

(2) Random banging test

More realistic test by automatically generating numerous requests. Notice that the test routine has to remember parameters for allocated blocks (files), so that they can be used when freeing them, as shown in the following pseudo program.

(a) Prepare an array of structure, called files, to record block returned by `xxx_alloc()` and number of blocks requested.

(b) Randomly decide to allocate or to free blocks.

(c) If allocation, generate an appropriate blocks to request, and call `xxx_alloc()` to actually allocate them. Record the result in an unused entry in files.

(d) If free, decide which files entry to free, and call `xxx_free()` to actually free the blocks.

(e) Repeat (b) through (d) for appropriate number of trials, then upon termination of the test, call `xxx_dump()`, `fbt_list()` and other checking procedures to verify the integrity of the management structures.

※ A function `ssbanger()` is provided, which execute the above test automatically by passing pointers to API functions of `bmc` and `fbt`. The `ssbanger()` generates requests so that they are in biased-normal distribution (look more like an actual file size distribution.) Refer to the `ssbanger()` API for details.

(3) Random banging test — Multi-thread

In a real file system, file access requests arrive concurrently from multiple processes and threads running in parallel. We want to make sure that the implemented `bmc` and `fbt` can handle concurrent accesses correctly.

※ A multi-threaded extension of `ssbanger()`, `ssbanger_mt()`, will be provided later.

5. Additional requirements

`bmc` and `fbt` should execute appropriate error checking and exception handling. This will also serve a good purpose when debugging.

6. Notes on writing reports

The report should include “an appropriate” execution that demonstrate `bmc` and `fbt` are correctly implemented.

(1) The most important aspect of written reports for programming projects is how to let readers know that you have understood the every aspect of the target system. Literal expression and non-literal expression both play an important role.

(2) The next important aspect is the structure as a written material. Don’t just paste your program text.

(3) Completeness and fanciness of programs are also, but least important.

(4) It is suggested that you draw your figures by hand and paste it onto your report. Also requested that letters are not hand written. When pasting execution results, etc., avoid capturing the screen as an image and paste it as text. (The “script” command is useful for capturing results.)

Keep It Simple and Stupid, unless you find them necessary to express your professional understanding of the target system.

You can extend your programs to include the directory system (DS) by adding some more code to the above mentioned random banging test. (Hint: What if you extend the files structure to include a name for the corresponding allocation record?) Extension like this is another way to express your understanding.

7. Due date/time and Submission

Due: 8 Dec (Wed) 24:00 JST **(Strict!!)**

Submission: The method of submission will be announced on Slack later.

2021 2-1 i233 Operating System Assignment#2 (Programing Project)

1. ストレージシステム(SS)の領域管理をシミュレートする手続き群と、それらを用いて領域の要求と解放を繰り返すプログラムを、次の2種類作成せよ。
   (1) bmc: ビットマップで連続割り当て (Figure 4-10 on “Filesystems” pp.22)
   (2) fbt: ブロックテーブルでの管理 (Figure 4-12 on “Filesystems” pp.24)

2. 管理対象のブロックは、定数 SSBLK ブロックとする。(プログラム中の #define で決定する。)

3. API案を次に示す(xxx は bmc あるいは fbt に置き換えて読む)。各自の独自設計でも良い。
   - `int xxx_init()`
     管理構造そのものを含めた初期化を行う。
     成功なら 0、失敗した場合は -1 を返す。
   - `int xxx_alloc(int nblks)`
     nblks ブロックの割り当てを行う。
     割り当てた領域の先頭のブロック番号を返す。
     割り当てに失敗した場合は -1 を返す。
   - `int xxx_free(int blkno, int nblks)`
     ブロック bn から nblks 分の解放を行う。
     成功なら 0、何らかのエラーが発生した場合は -1 を返す。
   - `void xxx_dump()`
     管理構造の状態を表示する。
   - `void fbt_list(int blkno)`
     fbt でのみ実現する関数。
     blkno で始まるリストを表示する。
     blkno が -1 なら空き領域リストを印字する。
   - `void xxx_verify()`
     割当の一貫性を検査する関数。

4. テスト用手続き

さまざまなテスト用手続きが考えられるが、たとえば次の3種類を例示しておく。

(1) Simple test
標準入力から、動作の種類とパラメータを受け取り、xxx_alloc() と xxx_free() を呼び出す。例えば、

- `a 10`　　// 10ブロックの割り当てを要求するコマンド
  `A 16 10` // ブロック16からはじまる10ブロックの割り当てが行われたことを示す出力
- `d 16`　　// ブロック16から始まるブロックリストのダンプ印字を要求するコマンド(fbtのみ)
  …
- `d -1`　　// 空き領域のダンプ印字を要求するコマンド
  …
- `d`　　// 管理構造の全ダンプを要求するコマンド
  …
- `f 16 10` // ブロック16からはじまる10ブロックの解放を要求するコマンド
  `F 16 10` // ブロック16からはじまる10ブロックの解放が行われたことを示す出力
- `a 4000`　// 4000ブロックの割り当てを要求するコマンド
  `A failed` // 割り当てが失敗したことを示す出力
- …

(2) Random banging test (耐久試験)

自動的に多数の要求を実行する方式である。どのような要求を行って結果がどうなったのかを記憶しておく必要がある。

(a) `xxx_alloc()` が返した先頭ブロックと要求したブロック数を格納する構造体の配列 `files` と、ファイル数を表す整数 `nfiles` を用意する。`files` 配列は `-1` で初期化しておく。

(b) ブロックの割り当てか解放かを乱数で決定する。

(c) 割り当ての場合は妥当な割り当て要求を乱数を使って発生し、`xxx_alloc()` を呼び出して割り当てを行う。

(d) 解放の場合は、乱数を使って解放する `files` エントリを決定し、`xxx_free()` を呼び出して解法を行う。

(e) (b)-(d)を妥当な回数繰り返し、最後に `xxx_dump()` や `fbt_list()` を呼び出して管理構造の検査が行えるようにする。

※ `bmc` や `fbt` の API 関数へのポインタを渡すと、自動的に割当と解放を繰り返す関数 `ssbanger()` を別途用意してある。`ssbanger()` は、ファイルのサイズが正規分布となるように割当要求を生成する。詳細は `ssbanger()` の API を見よ。

(3) Random banging test — Multi-thread (マルチスレッド対応耐久試験)

現実のファイルシステムでは、並行動作する複数のプロセス・スレッドから同時並行でファイルアクセス要求が到着する。実装した `bmc` や `fbt` は、同時アクセスに対して問題なく処理が可能かを確かめておきたい。

※ `ssbanger()` のマルチスレッド拡張版 `ssbanger_mt()` を後日例示する。

5. 追加仕様

`bmc` と `fbt` は、適切な異常検出を行うものとする。これはデバッグにも役立つ。

6. レポート作成上の注意事項

レポートには `bmc` と `fbt` が正しく実現されていることを示す実行例が含まれていなければならない。

(1) レポートの最も重要な観点は「こいつわかってるな」と思わせる図版、説明表現である。
(2) レポートとしての「文章の構成や工夫」が二番目に重要である。作成したプログラムと実行結果を単に貼り付けただけでは不十分である。
(3) プログラムの完成度も重要であるが、こちらは三番手以降とする。
(4) 図版は手書きを貼り込んでもかまわないが、文章は手書きにしないように注意すること。プログラムの実行結果等の掲載時には、画像での画面キャプチャは避け、テキストとして貼り付けること。(結果取得には “script” コマンドが便利)

上にあげた耐久試験機能は、もう少し手を入れるとディレクトリシステムとして動作させることができるようになる。そのような拡張も歓迎する。(ヒント: `files` を、ファイル名を含む属性エントリのテーブルとするとどうなるか。)

7. 提出〆切・提出方法

- 〆切: 12/8 (水) 24:00 JST **(厳守)**
- 提出方法: 後日 slack にて周知