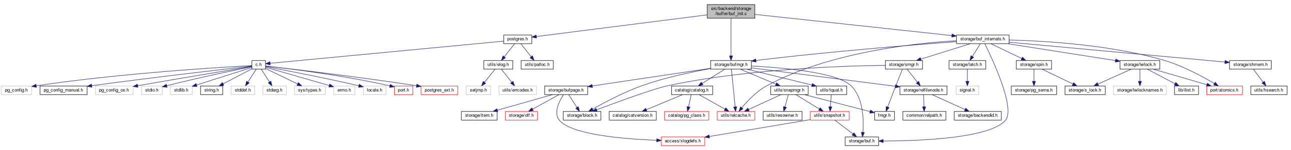
/src/backend/storage/buffer

buf_init.c

include dependency graph for buf_init.c



Functions:

- void **InitBufferPool**(void)

  -
    ```
    /*
    * Initialize shared buffer pool
    *
    * This is called once during shared-memory initialization (either in
    the
    * postmaster, or in a standalone backend).
    */
    ```

  - > - 设置4个bool变量：bool   foundBufs, foundDescs, foundIOLocks, foundBufCkpt;
    > - 将descriptor 对齐到cacheline boundary
    > - 将lwlocks对齐到 cacheline boundary
    > - 检查是否全部设置完毕，如果没有，初始化所有buffer的headers
    > - 然后将所有的buffer链接到一起，标记为unused

  - References LWLockTranche::array_base, LWLockTranche::array_stride, Assert, backend_flush_after, buf, BufferDesc::buf_id, BufferBlocks,BufferDescriptorGetContentLock, BufferDescriptorGetIOLock, BufferIOLWLockArray, CLEAR_BUFFERTAG, BufferDesc::freeNext,FREENEXT_END_OF_LIST, GetBufferDescriptor, i, LWLockInitialize(), LWLockRegisterTranche(), LWTRANCHE_BUFFER_CONTENT,LWTRANCHE_BUFFER_IO_IN_PROGRESS, LWLockTranche::name, NBuffers, offsetof, pg_atomic_init_u32(), ShmemInitStruct(), BufferDesc::state,StrategyInitialize(), BufferDesc::tag, BufferDesc::wait_backend_pid, and WritebackContextInit().

  - Referenced by CreateSharedMemoryAndSemaphores().

- **Size BufferShmemSize**(void)

  - compute the size of shared memory for the buffer pool including **data pages**, **buffer descriptors**, **hash tables**, etc.

  - References add_size(), mul_size(), NBuffers, PG_CACHE_LINE_SIZE, and StrategyShmemSize().

- Referenced by CreateSharedMemoryAndSemaphores().
- > ```
  > /*
  > * Multiply two Size values, checking for overflow
  > */
  > Size
  > mul_size(Size s1, Size s2);
  >
  > int         NBuffers = 1000;
  > ```

Variables

BufferDescPadded* BufferDescriptors //buffer描述符

> BufferDescPadded:
>
> #include buf_internals.sh // Internal definitions for *buffer manager* and the *buffer replacement*.
>
> ```
> //Concurrent access to buffer headers has proven to be more efficient if they're **cache line aligned**.So we force the start of the BufferDescriptors array to be on a cache line boundary and force the elements to be cache line sized.
>
> #define BUFFERDESC_PAD_TO_SIZE  (SIZEOF_VOID_P == 8 ? 64 : 1)
> typedef union BufferDescPadded
> {
>   //BufferDesc -- shared descriptor/state data for a single shared buffer.
>     BufferDesc  bufferdesc;
>     char        pad[BUFFERDESC_PAD_TO_SIZE];
> } BufferDescPadded;
> ```
>
> ```
> typedef struct BufferDesc
> {
>     BufferTag   tag;            /* ID of page contained in buffer */
>     int         buf_id;         /* buffer's index number (from 0) */
>
>     /* state of the tag, containing flags, refcount and usagecount */
>     pg_atomic_uint32 state;
>
>     int         wait_backend_pid;       /* backend PID of pin-count waiter */
>     int         freeNext;       /* link in freelist chain */
>
>     LWLock      content_lock;   /* to lock access to buffer contents */
> } BufferDesc;
> ```

char* BufferBlocks //Buffer实际的存储区域，类型为char*

LWLockMinimallyPadded* BufferIOLWLockArray = **NULL**

> ```
> /* LWLock, minimally padded */
> typedef union LWLockMinimallyPadded
> {
>     LWLock      lock;
>     char        pad[LWLOCK_MINIMAL_SIZE];
> } LWLockMinimallyPadded;
> ```

LWLockTranche BufferContentLWLockTranche

LWLockTranche BufferIOLWLockTranche

```
/*
 * Prior to PostgreSQL 9.4, every lightweight lock in the system was stored
 * in a single array.  For convenience and for compatibility with past
 * releases, we still have a main array, but it's now also permissible to
 * store LWLocks elsewhere in the main shared memory segment or in a dynamic
 * shared memory segment.

 **Each array of lwlocks forms a separate "tranche"**.

 *
 * It's occasionally necessary to identify a particular LWLock "by name"; e.g.
 * because we wish to report the lock to dtrace.  We could store a name or
 * other identifying information in the lock itself, but since it's common
 * to have many nearly-identical locks (e.g. one per buffer) this would end
 * up wasting significant amounts of memory.  Instead, each lwlock stores a
 * tranche ID which tells us which array it's part of.  Based on that, we can
 * figure out where the lwlock lies within the array using the data structure
 * shown below; the lock is then identified based on the tranche name and
 * computed array index.  We need the array stride because the array might not
 * be an array of lwlocks, but rather some larger data structure that includes
 * one or more lwlocks per element.
 */
typedef struct LWLockTranche
{
    const char *name;
    void       *array_base;
    Size        array_stride;
} LWLockTranche;
```

WriteBackContext BackendWritebackContext //保存写回时的上下文环境

```
/* struct forward declared in bufmgr.h */
typedef struct WritebackContext
{
    /* pointer to the max number of writeback requests to coalesce(合并) */
    int         *max_pending;

    /* current number of pending writeback requests */
    int          nr_pending;

    /* pending requests */
    PendingWriteback pending_writebacks[WRITEBACK_MAX_PENDING_FLUSHES];
} WritebackContext;
```

CkptSortItem* CkptBufferIds   //用于在checkpoint检查点时对每个file， sort它的buffer

```
/*
 * Structure to sort buffers per file on checkpoints.
 *
 * This structure is allocated **per buffer** in shared memory, so it should be
 * kept as small as possible.
 */
typedef struct CkptSortItem
{
    Oid          tsId;
    Oid          relNode;
    ForkNumber  forkNum;
    BlockNumber blockNum;
    int          buf_id;
} CkptSortItem;
```