```
/*
 * Semantics for shared and private memory areas are different past the end
 * of the file. A shared mapping past the last page of the file is an error
 * and results in a SIGBUS, while a private mapping just maps in a zero page.
 *
 * The goto's are kind of ugly, but this streamlines the normal case of having
 * it in the page cache, and handles the special cases reasonably without
 * having a lot of duplicated code.
 *
 * WSH 06/04/97: fixed a memory leak and moved the allocation of new_page
 * ahead of the wait if we're sure to need it.
 */
static unsigned long filemap_nopage(struct vm_area_struct * area, unsigned long address, int no_share)
{
	struct file * file = area->vm_file;
	struct dentry * dentry = file->f_dentry;
	struct inode * inode = dentry->d_inode;
	unsigned long offset, reada, i;
	struct page * page, **hash;
	unsigned long old_page, new_page;

	new_page = 0;
	offset = (address & PAGE_MASK) - area->vm_start + area->vm_offset;
	if (offset >= inode->i_size && (area->vm_flags & VM_SHARED) && area->vm_mm == current->mm)
		goto no_page;

	/*
	 * Do we have something in the page cache already?
	 */
	hash = page_hash(inode, offset);
	page = __find_page(inode, offset, *hash);
	if (!page)
		goto no_cached_page;

found_page:
	/*
	 * Ok, found a page in the page cache, now we need to check
	 * that it's up-to-date.   First check whether we'll need an
	 * extra page -- better to overlap the allocation with the I/O.
	 */
	if (no_share && !new_page) {
		new_page = page_cache_alloc();
		if (!new_page)
			goto failure;
	}

	if (PageLocked(page))
		goto page_locked_wait;
	if (!PageUptodate(page))
		goto page_read_error;

success:
	/*
	 * Found the page, need to check sharing and possibly
	 * copy it over to another page..
	 */
	old_page = page_address(page);
	if (!no_share) {
		/*
		 * Ok, we can share the cached page directly.. Get rid
		 * of any potential extra pages.
		 */
		if (new_page)
			page_cache_free(new_page);

		flush_page_to_ram(old_page);
		return old_page;
	}

	/*
	 * No sharing ... copy to the new page.
	 */
	copy_page(new_page, old_page);
	flush_page_to_ram(new_page);
	page_cache_release(page);
	return new_page;

no_cached_page:
	/*
	 * Try to read in an entire cluster at once.
	 */
	reada   = offset;
	reada >>= PAGE_CACHE_SHIFT + page_cluster;
	reada <<= PAGE_CACHE_SHIFT + page_cluster;
	for (i = 1 << page_cluster; i > 0; --i, reada += PAGE_CACHE_SIZE)
		new_page = try_to_read_ahead(file, reada, new_page);

	if (!new_page)
		new_page = page_cache_alloc();
	if (!new_page)
		goto no_page;

	/*
	 * During getting the above page we might have slept,
	 * so we need to re-check the situation with the page
	 * cache.. The page we just got may be useful if we
	 * can't share, so don't get rid of it here.
	 */
	page = find_page(inode, offset);
	if (page)
		goto found_page;

	/*
	 * Now, create a new page-cache page from the page we got
	 */
	page = page_cache_entry(new_page);
	new_page = 0;
	add_to_page_cache(page, inode, offset, hash);

	if (inode->i_op->readpage(file, page) != 0)
		goto failure;

	goto found_page;

page_locked_wait:
	__wait_on_page(page);
	if (PageUptodate(page))
		goto success;

page_read_error:
	/*
	 * Umm, take care of errors if the page isn't up-to-date.
	 * Try to re-read it _once_. We do this synchronously,
	 * because there really aren't any performance issues here
	 * and we need to check for errors.
	 */
	if (inode->i_op->readpage(file, page) != 0)
		goto failure;
	wait_on_page(page);
	if (PageError(page))
		goto failure;
	if (PageUptodate(page))
		goto success;

	/*
	 * Things didn't work out. Return zero to tell the
	 * mm layer so, possibly freeing the page cache page first.
	 */
failure:
	page_cache_release(page);
	if (new_page)
		page_cache_free(new_page);
no_page:
	return 0;
}
```