

POHMELFS high performance network filesystem. Transactions, failover, performance.

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-05/msg06012.html>

- *From:* Evgeniy Polyakov <johnpol@xxxxxxxxxxxx>
 - *Date:* Tue, 13 May 2008 21:45:24 +0400
-

Hi.

I'm please to announce POHMEL high performance network filesystem.
POHMELFS stands for Parallel Optimized Host Message Exchange Layered File System.

Development status can be tracked in filesystem section [1].

This is a high performance network filesystem with local coherent cache of data and metadata. Its main goal is distributed parallel processing of data. Network filesystem is a client transport. POHMELFS protocol was proven to be superior to NFS in lots (if not all, then it is in a roadmap) operations.

This release brings following features:

- * Fast transactions. System will wrap all writings into transactions, which will be resent to different (or the same) server in case of failure.

Details in notes [1].

- * Failover. It is now possible to provide number of servers to be used in round-robin fasion when one of them dies. System will automatically reconnect to others and send transactions to them.

- * Performance. Super fast (close to wire limit) metadata operations over the network. By courtesy of writeback cache and transactions the whole kernel archive can be untarred by 2-3 seconds (including sync) over GigE link (wire limit! Not comparable to NFS).

Basic POHMELFS features:

- * Local coherent (notes [5]) cache for data and metadata.
- * Completely async processing of all events (hard and symlinks are the only exceptions) including object creation and data reading.
- * Flexible object architecture optimized for network processing. Ability to create long pathes to object and remove arbitrary huge directoris in single network command.
- * High performance is one of the main design goals.
- * Very fast and scalable multithreaded userspace server. Being in userspace it works with any underlying filesystem and still is much faster than async ni-kernel NFS one.

Roadmap includes:

- * Server extension to allow storing data on multiple devices (like creating mirroring),

POHMELFS high performance network filesystem. Transactions, failover, performance.

first by saving data in several local directories (think about server, which mounted remote dirs over POHMELFS or NFS, and local dirs).

- * Client/server extension to report lookup and readdir requests not only for local destination, but also to different addresses, so that reading/writing could be done from different nodes in parallel.
- * Strong authentication and possible data encryption in network channel.
- * Async writing of the data from receiving kernel thread into userspace pages via copy_to_user() (check development tracking blog for results).

One can grab sources from archive or git [2] or check homepage [3]. Benchmark section can be found in the blog [4].

The nearest roadmap (scheduled or the end of the month) includes:

- * Full transaction support for all operations (only writeback is guarded by transactions currently, default network state just reconnects to the same server).
- * Data and metadata coherency extensions (in addition to existing commented object creation/removal messages). (next week)
- * Server redundancy.

Thank you.

1. POHMELFS development status.

<http://tservice.net.ru/~s0mbre/blog/devel/fs/index.html>

2. Source archive.

<http://tservice.net.ru/~s0mbre/archive/pohmelfs/>

Git tree.

<http://tservice.net.ru/~s0mbre/archive/pohmelfs/pohmelfs.git/>

3. POHMELFS homepage.

<http://tservice.net.ru/~s0mbre/old/?section=projects&item=pohmelfs>

4. POHMELFS vs NFS benchmark.

http://tservice.net.ru/~s0mbre/blog/devel/fs/2008_04_18.html

http://tservice.net.ru/~s0mbre/blog/devel/fs/2008_04_14.html

http://tservice.net.ru/~s0mbre/blog/devel/fs/2008_05_12.html

5. Cache-coherency notes.

http://tservice.net.ru/~s0mbre/blog/devel/fs/2008_04_21.html

http://tservice.net.ru/~s0mbre/blog/devel/fs/2008_04_22.html

Signed-off-by: Evgeniy Polyakov <johnpol@xxxxxxxxxxx>

fs/Kconfig | 2 +

fs/Makefile | 1 +

fs/pohmelfs/Kconfig | 6 +

fs/pohmelfs/Makefile | 3 +

fs/pohmelfs/config.c | 148 +++++

fs/pohmelfs/dir.c | 1009 ++++++

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
fs/pohmelfs/inode.c | 1543 ++++++
fs/pohmelfs/net.c | 800 ++++++
fs/pohmelfs/netfs.h | 426 ++++++
fs/pohmelfs/path_entry.c | 278 ++++++
fs/pohmelfs/trans.c | 469 ++++++
11 files changed, 4685 insertions(+), 0 deletions(-)
```

```
diff --git a/fs/Kconfig b/fs/Kconfig
index c509123..59935cd 100644
--- a/fs/Kconfig
+++ b/fs/Kconfig
@@ -1566,6 +1566,8 @@ menuconfig NETWORK_FILESYSTEMS
```

```
if NETWORK_FILESYSTEMS
```

```
+source "fs/pohmelfs/Kconfig"
+
config NFS_FS
tristate "NFS file system support"
depends on INET
diff --git a/fs/Makefile b/fs/Makefile
index 1e7a11b..6ce6a35 100644
--- a/fs/Makefile
+++ b/fs/Makefile
@@ -119,3 +119,4 @@ obj-$(CONFIG_HPPFS) += hppfs/
obj-$(CONFIG_DEBUG_FS) += debugfs/
obj-$(CONFIG_OCFS2_FS) += ocfs2/
obj-$(CONFIG_GFS2_FS) += gfs2/
+obj-$(CONFIG_POHMELFS) += pohmelfs/
diff --git a/fs/pohmelfs/Kconfig b/fs/pohmelfs/Kconfig
new file mode 100644
index 0000000..ac19aac
--- /dev/null
+++ b/fs/pohmelfs/Kconfig
@@ -0,0 +1,6 @@
+config POHMELFS
+tristate "POHMELFS filesystem support"
+help
+POHMELFS stands for Parallel Optimized Host Message Exchange Layered File System.
+This is a network filesystem which supports coherent caching of data and metadata
+on clients.
diff --git a/fs/pohmelfs/Makefile b/fs/pohmelfs/Makefile
new file mode 100644
index 0000000..aa415a3
--- /dev/null
+++ b/fs/pohmelfs/Makefile
@@ -0,0 +1,3 @@
@@ -0,0 +1,3 @@
+obj-$(CONFIG_POHMELFS) += pohmelfs.o
+
+pohmelfs-y := inode.o config.o dir.o net.o path_entry.o trans.o
diff --git a/fs/pohmelfs/config.c b/fs/pohmelfs/config.c
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
new file mode 100644
index 0000000..0f3503b
--- /dev/null
+++ b/fs/pohmelfs/config.c
@@ -0,0 +1,148 @@
+/*
+ * 2007+ Copyright (c) Evgeniy Polyakov <johnpol@xxxxxxxxxxxx>
+ * All rights reserved.
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/kernel.h>
+#include <linux/connector.h>
+#include <linux/list.h>
+#include <linux/mutex.h>
+
+#include "netfs.h"
+
+/*
+ * Global configuration list.
+ * Each client can be asked to get one of them.
+ *
+ * Allows to provide remote server address (ipv4/v6/whatever), port
+ * and so on via kernel connector.
+ */
+
+static struct cb_id pohmelfs_cn_id = {.idx = POHMELFS_CN_IDX, .val = POHMELFS_CN_VAL};
+static LIST_HEAD(pohmelfs_config_list);
+static DEFINE_MUTEX(pohmelfs_config_lock);
+
+int pohmelfs_copy_config(struct pohmelfs_sb *psb)
+{
+    struct pohmelfs_config *c, *dst;
+    int err = -ENODEV;
+
+    mutex_lock(&pohmelfs_config_lock);
+    list_for_each_entry(c, &pohmelfs_config_list, config_entry) {
+        if (c->statectl.idx != psb->idx)
+            continue;
+
+        dst = kzalloc(sizeof(struct pohmelfs_config), GFP_KERNEL);
+        if (!dst) {
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ err = -ENOMEM;
+ goto err_out_unlock;
+ }
+
+ memcpy(&dst->state.ctl, &c->state.ctl, sizeof(struct pohmelfs_ctl));
+
+ mutex_lock(&psb->state_lock);
+ list_add_tail(&dst->config_entry, &psb->state_list);
+ mutex_unlock(&psb->state_lock);
+ err = 0;
+ }
+ mutex_unlock(&pohmelfs_config_lock);
+
+ return err;
+
+err_out_unlock:
+ mutex_unlock(&pohmelfs_config_lock);
+
+ mutex_lock(&psb->state_lock);
+ list_for_each_entry_safe(dst, c, &psb->state_list, config_entry) {
+ list_del(&dst->config_entry);
+ kfree(dst);
+ }
+ mutex_unlock(&psb->state_lock);
+
+ return err;
+}
+
+static void pohmelfs_cn_callback(void *data)
+{
+ struct cn_msg *msg = data;
+ struct pohmelfs_ctl *ctl;
+ struct pohmelfs_cn_ack *ack;
+ struct pohmelfs_config *c;
+ int err;
+
+ if (msg->len < sizeof(struct pohmelfs_ctl)) {
+ err = -EBADMSG;
+ goto out;
+ }
+
+ ctl = (struct pohmelfs_ctl *)msg->data;
+
+ err = 0;
+ mutex_lock(&pohmelfs_config_lock);
+ list_for_each_entry(c, &pohmelfs_config_list, config_entry) {
+ struct pohmelfs_ctl *sc = &c->state.ctl;
+
+ if (sc->idx == ctl->idx && sc->type == ctl->type &&
+ sc->proto == ctl->proto &&
+ sc->addrlen == ctl->addrlen &&
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ !memcmp(&sc->addr, &ctl->addr, ctl->addrlen)) {
+ err = -EEXIST;
+ break;
+ }
+ }
+ if (!err) {
+ c = kzalloc(sizeof(struct pohmelfs_config), GFP_KERNEL);
+ if (!c) {
+ err = -ENOMEM;
+ goto out;
+ }
+
+ memcpy(&c->state.ctl, ctl, sizeof(struct pohmelfs_ctl));
+ list_add_tail(&c->config_entry, &pohmelfs_config_list);
+ }
+ mutex_unlock(&pohmelfs_config_lock);
+
+out:
+ ack = kmalloc(sizeof(struct pohmelfs_cn_ack), GFP_KERNEL);
+ if (!ack)
+ return;
+
+ memcpy(&ack->msg, msg, sizeof(struct cn_msg));
+
+ ack->msg.ack = msg->ack + 1;
+ ack->msg.len = sizeof(struct pohmelfs_cn_ack) - sizeof(struct cn_msg);
+
+ ack->error = err;
+
+ cn_netlink_send(&ack->msg, 0, GFP_KERNEL);
+ kfree(ack);
+ }
+
+int __init pohmelfs_config_init(void)
+{
+ return cn_add_callback(&pohmelfs_cn_id, "pohmelfs", pohmelfs_cn_callback);
+ }
+
+void __exit pohmelfs_config_exit(void)
+{
+ struct pohmelfs_config *c, *tmp;
+
+ cn_del_callback(&pohmelfs_cn_id);
+
+ mutex_lock(&pohmelfs_config_lock);
+ list_for_each_entry_safe(c, tmp, &pohmelfs_config_list, config_entry) {
+ list_del(&c->config_entry);
+ kfree(c);
+ }
+ mutex_unlock(&pohmelfs_config_lock);
+ }
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
diff --git a/fs/pohmelfs/dir.c b/fs/pohmelfs/dir.c
new file mode 100644
index 0000000..41e1f8f
--- /dev/null
+++ b/fs/pohmelfs/dir.c
@@ -0,0 +1,1009 @@
+/*
+ * 2007+ Copyright (c) Evgeniy Polyakov <johnpol@xxxxxxxxxxxx>
+ * All rights reserved.
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/kernel.h>
+#include <linux/fs.h>
+#include <linux/jhash.h>
+
+#include "netfs.h"
+
+/*
+ * Each pohmelfs directory inode contains a tree of childrens indexed
+ * by offset (in the dir reading stream) and name hash and len. Entries
+ * of that hashes are called pohmelfs_name.
+ *
+ * This routings deal with it.
+ */
+static int pohmelfs_cmp_offset(struct pohmelfs_name *n, u64 offset)
+{
+ if (n->offset > offset)
+ return -1;
+ if (n->offset < offset)
+ return 1;
+ return 0;
+}
+
+static struct pohmelfs_name *pohmelfs_search_offset(struct pohmelfs_inode *pi, u64 offset)
+{
+ struct rb_node *n = pi->offset_root.rb_node;
+ struct pohmelfs_name *tmp;
+ int cmp;
+
+ while (n) {
+ tmp = rb_entry(n, struct pohmelfs_name, offset_node);
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```

+
+ cmp = pohmelfs_cmp_offset(tmp, offset);
+ if (cmp < 0)
+ n = n->rb_left;
+ else if (cmp > 0)
+ n = n->rb_right;
+ else
+ return tmp;
+ }
+
+ return NULL;
+}
+
+static struct pohmelfs_name *pohmelfs_insert_offset(struct pohmelfs_inode *pi,
+ struct pohmelfs_name *new)
+{
+ struct rb_node **n = &pi->offset_root.rb_node, *parent = NULL;
+ struct pohmelfs_name *ret = NULL, *tmp;
+ int cmp;
+
+ while (*n) {
+ parent = *n;
+
+ tmp = rb_entry(parent, struct pohmelfs_name, offset_node);
+
+ cmp = pohmelfs_cmp_offset(tmp, new->offset);
+ if (cmp < 0)
+ n = &parent->rb_left;
+ else if (cmp > 0)
+ n = &parent->rb_right;
+ else {
+ ret = tmp;
+ break;
+ }
+ }
+
+ if (ret)
+ return ret;
+
+ rb_link_node(&new->offset_node, parent, n);
+ rb_insert_color(&new->offset_node, &pi->offset_root);
+
+ pi->total_len += new->len;
+
+ return NULL;
+}
+
+static int pohmelfs_cmp_hash(struct pohmelfs_name *n, u32 hash, u32 len)
+{
+ if (n->hash > hash)
+ return -1;

```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ if (n->hash < hash)
+ return 1;
+
+ if (n->len > len)
+ return -1;
+ if (n->len < len)
+ return 1;
+
+ return 0;
+}
+
+static struct pohmelfs_name *pohmelfs_search_hash(struct pohmelfs_inode *pi, u32 hash, u32 len)
+{
+ struct rb_node *n = pi->hash_root.rb_node;
+ struct pohmelfs_name *tmp;
+ int cmp;
+
+ while (n) {
+ tmp = rb_entry(n, struct pohmelfs_name, hash_node);
+
+ cmp = pohmelfs_cmp_hash(tmp, hash, len);
+ if (cmp < 0)
+ n = n->rb_left;
+ else if (cmp > 0)
+ n = n->rb_right;
+ else
+ return tmp;
+ }
+
+ return NULL;
+}
+
+static struct pohmelfs_name *pohmelfs_insert_hash(struct pohmelfs_inode *pi,
+ struct pohmelfs_name *new)
+{
+ struct rb_node **n = &pi->hash_root.rb_node, *parent = NULL;
+ struct pohmelfs_name *ret = NULL, *tmp;
+ int cmp;
+
+ while (*n) {
+ parent = *n;
+
+ tmp = rb_entry(parent, struct pohmelfs_name, hash_node);
+
+ cmp = pohmelfs_cmp_hash(tmp, new->hash, new->len);
+ if (cmp < 0)
+ n = &parent->rb_left;
+ else if (cmp > 0)
+ n = &parent->rb_right;
+ else {
+ ret = tmp;
+ }
+ }
+
+ return ret;
+}
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ break;
+ }
+ }
+
+ if (ret) {
+ printk("%s: exist: ino: %llu, hash: %x, len: %u, data: '%s', new: ino: %llu, hash: %x, len: %u, data: '%s'.\n",
+ __func__, ret->ino, ret->hash, ret->len, ret->data,
+ ret->ino, new->hash, new->len, new->data);
+ return ret;
+ }
+
+ rb_link_node(&new->hash_node, parent, n);
+ rb_insert_color(&new->hash_node, &pi->hash_root);
+
+ return NULL;
+}
+
+static void __pohmelfs_name_del(struct pohmelfs_inode *parent, struct pohmelfs_name *node)
+{
+ rb_erase(&node->offset_node, &parent->offset_root);
+ rb_erase(&node->hash_node, &parent->hash_root);
+}
+
+/*
+ * Remove name cache entry from its caches and free it.
+ */
+static void pohmelfs_name_free(struct pohmelfs_inode *parent, struct pohmelfs_name *node)
+{
+ __pohmelfs_name_del(parent, node);
+ list_del(&node->sync_del_entry);
+ list_del(&node->sync_create_entry);
+ kfree(node);
+}
+
+/*
+ * Free name cache for given inode.
+ */
+void pohmelfs_free_names(struct pohmelfs_inode *parent)
+{
+ struct rb_node *rb_node;
+ struct pohmelfs_name *n;
+
+ for (rb_node = rb_first(&parent->offset_root); rb_node;) {
+ n = rb_entry(rb_node, struct pohmelfs_name, offset_node);
+ rb_node = rb_next(rb_node);
+
+ pohmelfs_name_free(parent, n);
+ }
+}
+
+/*
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ * When name cache entry is removed (for example when object is removed),
+ * offset for all subsequent childrens has to be fixed to match new reality.
+ */
+static int pohmelfs_fix_offset(struct pohmelfs_inode *parent, struct pohmelfs_name *node)
+{
+ struct rb_node *rb_node;
+ int decr = 0;
+
+ for (rb_node = rb_next(&node->offset_node); rb_node; rb_node = rb_next(rb_node)) {
+ struct pohmelfs_name *n = container_of(rb_node, struct pohmelfs_name, offset_node);
+
+ n->offset -= node->len;
+ decr++;
+ }
+
+ parent->total_len -= node->len;
+
+ return decr;
+}
+
+/*
+ * Fix offset and free name cache entry helper.
+ */
+void pohmelfs_name_del(struct pohmelfs_inode *parent, struct pohmelfs_name *node)
+{
+ int decr;
+
+ decr = pohmelfs_fix_offset(parent, node);
+
+ dprintk("%s: parent: %llu, ino: %llu, decr: %d.\n",
+ __func__, parent->ino, node->ino, decr);
+
+ pohmelfs_name_free(parent, node);
+}
+
+/*
+ * Insert new name cache entry into all caches (offset and name hash).
+ */
+static int pohmelfs_insert_name(struct pohmelfs_inode *parent, struct pohmelfs_name *n)
+{
+ struct pohmelfs_name *name;
+
+ name = pohmelfs_insert_offset(parent, n);
+ if (name)
+ return -EEXIST;
+
+ name = pohmelfs_insert_hash(parent, n);
+ if (name) {
+ parent->total_len -= n->len;
+ rb_erase(&n->offset_node, &parent->offset_root);
+ return -EEXIST;
+ }
```

```

+ }
+
+ list_add_tail(&n->sync_create_entry, &parent->sync_create_list);
+
+ return 0;
+}
+
+/*
+ * Allocate new name cache entry.
+ */
+static struct pohmelfs_name *pohmelfs_name_clone(unsigned int len)
+{
+ struct pohmelfs_name *n;
+
+ n = kzalloc(sizeof(struct pohmelfs_name) + len, GFP_KERNEL);
+ if (!n)
+ return NULL;
+
+ INIT_LIST_HEAD(&n->sync_create_entry);
+ INIT_LIST_HEAD(&n->sync_del_entry);
+
+ n->data = (char *)(n+1);
+
+ return n;
+}
+
+/*
+ * Add new name entry into directory's cache.
+ */
+static int pohmelfs_add_dir(struct pohmelfs_sb *psb, struct pohmelfs_inode *parent,
+ struct pohmelfs_inode *npi, struct qstr *str, unsigned int mode, int link)
+{
+ int err = -ENOMEM;
+ struct pohmelfs_name *n;
+ struct pohmelfs_path_entry *e = NULL;
+
+ n = pohmelfs_name_clone(str->len + 1);
+ if (!n)
+ goto err_out_exit;
+
+ n->ino = npi->ino;
+ n->offset = parent->total_len;
+ n->mode = mode;
+ n->len = str->len;
+ n->hash = str->hash;
+ sprintf(n->data, str->name);
+
+ if (!(str->len == 1 && str->name[0] == '.') &&
+ !(str->len == 2 && str->name[0] == '.' && str->name[1] == '.')) {
+ mutex_lock(&psb->path_lock);
+ e = pohmelfs_add_path_entry(psb, parent->ino, npi->ino, str, link);

```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ mutex_unlock(&psb->path_lock);
+ if (IS_ERR(e)) {
+ err = PTR_ERR(e);
+ goto err_out_free;
+ }
+ }
+
+ mutex_lock(&parent->offset_lock);
+ err = pohmelfs_insert_name(parent, n);
+ mutex_unlock(&parent->offset_lock);
+
+ if (err)
+ goto err_out_remove;
+
+ return 0;
+
+err_out_remove:
+ if (e) {
+ mutex_lock(&psb->path_lock);
+ pohmelfs_remove_path_entry(psb, e);
+ mutex_unlock(&psb->path_lock);
+ }
+err_out_free:
+ kfree(n);
+err_out_exit:
+ return err;
+ }
+
+/*
+ * Create new inode for given parameters (name, inode info, parent).
+ * This does not create object on the server, it will be synced there during writeback.
+ */
+struct pohmelfs_inode *pohmelfs_new_inode(struct pohmelfs_sb *psb,
+ struct pohmelfs_inode *parent, struct qstr *str,
+ struct netfs_inode_info *info, int link)
+{
+ struct inode *new = NULL;
+ struct pohmelfs_inode *npi;
+ int err = -EEXIST;
+
+ dprintk("%s: creating inode: parent: %llu, ino: %llu, str: %p.\n",
+ __func__, (parent)?parent->ino:0, info->ino, str);
+
+ err = -ENOMEM;
+ new = iget_locked(psb->sb, info->ino);
+ if (!new)
+ goto err_out_exit;
+
+ npi = POHMELFS_I(new);
+ npi->ino = info->ino;
+ err = 0;
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+
+ if (new->i_state & I_NEW) {
+ dprintk("%s: filling VFS inode: %lu/%llu.\n",
+ __func__, new->i_ino, info->ino);
+ pohmelfs_fill_inode(npi, info);
+
+ if (S_ISDIR(info->mode)) {
+ struct qstr s;
+
+ s.name = ".";
+ s.len = 1;
+ s.hash = jhash(s.name, s.len, 0);
+
+ err = pohmelfs_add_dir(psb, npi, npi, &s, info->mode, 0);
+ if (err)
+ goto err_out_put;
+
+ s.name = "..";
+ s.len = 2;
+ s.hash = jhash(s.name, s.len, 0);
+
+ err = pohmelfs_add_dir(psb, npi, (parent)?parent:npi, &s,
+ (parent)?parent->vfs_inode.i_mode:npi->vfs_inode.i_mode, 0);
+ if (err)
+ goto err_out_put;
+ }
+ }
+
+ if (str) {
+ if (parent) {
+ err = pohmelfs_add_dir(psb, parent, npi, str, info->mode, link);
+
+ dprintk("%s: %s inserted name: '%s', new_offset: %llu, ino: %llu, parent: %llu.\n",
+ __func__, (err)?"unsuccessfully":"successfully",
+ str->name, parent->total_len, info->ino, parent->ino);
+
+ if (err)
+ goto err_out_put;
+ } else {
+ mutex_lock(&psb->path_lock);
+ pohmelfs_add_path_entry(psb, npi->ino, npi->ino, str, link);
+ mutex_unlock(&psb->path_lock);
+ }
+ }
+
+ if (new->i_state & I_NEW) {
+ mutex_lock(&psb->path_lock);
+ list_add_tail(&npi->inode_entry, &psb->inode_list);
+ mutex_unlock(&psb->path_lock);
+
+ unlock_new_inode(new);
+ }
```

```

+ if (parent)
+ mark_inode_dirty(&parent->vfs_inode);
+ mark_inode_dirty(new);
+ }
+
+ return napi;
+
+err_out_put:
+ printk("%s: putting inode: %p, napi: %p, error: %d.\n", __func__, new, napi, err);
+ iput(new);
+err_out_exit:
+ return ERR_PTR(err);
+}
+
+/*
+ * Receive directory content from the server.
+ * This should be only done for objects, which were not created locally,
+ * and which were not synced previously.
+ */
+static int pohmelfs_sync_remote_dir(struct pohmelfs_inode *pi)
+{
+ struct pohmelfs_sb *psb = POHMELFS_SB(pi->vfs_inode.i_sb);
+ struct netfs_state *st = pohmelfs_active_state(psb);
+ struct netfs_cmd *cmd = &st->cmd;
+ long ret = msecs_to_jiffies(25000);
+ unsigned path_size;
+ int err;
+
+ dprintk("%s: dir: %llu, state: %lx: created: %d, remote_synced: %d.\n",
+ __func__, pi->ino, pi->state, test_bit(NETFS_INODE_CREATED, &pi->state),
+ test_bit(NETFS_INODE_REMOTE_SYNCED, &pi->state));
+
+ if (!test_bit(NETFS_INODE_CREATED, &pi->state))
+ return -1;
+
+ if (test_bit(NETFS_INODE_REMOTE_SYNCED, &pi->state))
+ return 0;
+
+ mutex_lock(&st->state_lock);
+
+ mutex_lock(&st->psb->path_lock);
+ err = pohmelfs_construct_path_string(pi, st->data, st->size);
+ mutex_unlock(&st->psb->path_lock);
+ if (err < 0)
+ goto err_out_unlock;
+
+ dprintk("%s: syncing dir: %llu, data: '%s'.\n", __func__, pi->ino, (char *)st->data);
+
+ cmd->cmd = NETFS_READDIR;
+ cmd->start = 0;
+ path_size = cmd->size = err + 1;

```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ cmd->ext = 0;
+ cmd->id = pi->ino;
+ netfs_convert_cmd(cmd);
+
+ err = netfs_data_send(st, cmd, sizeof(struct netfs_cmd), 1);
+ if (err)
+ goto err_out_unlock;
+
+ err = netfs_data_send(st, st->data, path_size, 0);
+ if (err)
+ goto err_out_unlock;
+
+ mutex_unlock(&st->state_lock);
+
+ ret = wait_event_interruptible_timeout(st->thread_wait,
+ test_bit(NETFS_INODE_REMOTE_SYNCED, &pi->state), ret);
+ dprintk("%s: awake dir: %llu, ret: %ld.\n", __func__, pi->ino, ret);
+ if (ret <= 0) {
+ err = -ETIMEDOUT;
+ goto err_out_exit;
+ }
+
+ return 0;
+
+err_out_unlock:
+ mutex_unlock(&st->state_lock);
+err_out_exit:
+ clear_bit(NETFS_INODE_REMOTE_SYNCED, &pi->state);
+
+ return err;
+}
+
+/*
+ * VFS readdir callback. Syncs directory content from server if needed,
+ * and provide info to userspace.
+ */
+static int pohmelfs_readdir(struct file *file, void *dirent, filldir_t filldir)
+{
+ struct inode *inode = file->f_path.dentry->d_inode;
+ struct pohmelfs_inode *pi = POHMELFS_I(inode);
+ struct pohmelfs_name *n;
+ int err = 0, mode;
+ u64 len;
+
+ dprintk("%s: parent: %llu.\n", __func__, pi->ino);
+
+ err = pohmelfs_sync_remote_dir(pi);
+ if (err)
+ return err;
+
+ while (1) {
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ mutex_lock(&pi->offset_lock);
+ n = pohmelfs_search_offset(pi, file->f_pos);
+ if (!n) {
+ mutex_unlock(&pi->offset_lock);
+ err = 0;
+ break;
+ }
+
+ mode = (n->mode >> 12) & 15;
+
+ dprintk("%s: offset: %llu, parent ino: %llu, name: '%s', len: %u, ino: %llu, mode: %o/%o.\n",
+ __func__, file->f_pos, pi->ino, n->data, n->len,
+ n->ino, n->mode, mode);
+
+ len = n->len;
+ err = filldir(dirent, n->data, n->len, file->f_pos, n->ino, mode);
+ mutex_unlock(&pi->offset_lock);
+
+ if (err < 0) {
+ dprintk("%s: err: %d.\n", __func__, err);
+ err = 0;
+ break;
+ }
+
+ file->f_pos += len;
+ }
+
+ return err;
+}
+
+const struct file_operations pohmelfs_dir_fops = {
+ .read = generic_read_dir,
+ .readdir = pohmelfs_readdir,
+};
+
+/*
+ * Lookup single object on server.
+ */
+static int pohmelfs_lookup_single(struct pohmelfs_inode *parent,
+ struct qstr *str, u64 ino)
+{
+ struct pohmelfs_sb *psb = POHMELFS_SB(parent->vfs_inode.i_sb);
+ struct netfs_state *st = pohmelfs_active_state(psb);
+ struct netfs_cmd *cmd = &st->cmd;
+ long ret = msecs_to_jiffies(25000);
+ unsigned path_size;
+ int err;
+
+ mutex_lock(&st->state_lock);
+ set_bit(NETFS_COMMAND_PENDING, &parent->state);
+
+ }
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ mutex_lock(&st->psb->path_lock);
+ err = pohmelfs_construct_path_string(parent, st->data, st->size - str->len - 2);
+ mutex_unlock(&st->psb->path_lock);
+ if (err < 0)
+ goto err_out_unlock;
+
+ path_size = err;
+ path_size += sprintf(st->data + path_size, "%s", str->name) + 1 /* 0-byte */;
+
+ cmd->cmd = NETFS_LOOKUP;
+ cmd->size = path_size;
+ cmd->ext = 0;
+ cmd->id = parent->ino;
+ cmd->start = ino;
+ netfs_convert_cmd(cmd);
+
+ err = netfs_data_send(st, cmd, sizeof(struct netfs_cmd), 1);
+ if (err)
+ goto err_out_unlock;
+
+ err = netfs_data_send(st, st->data, path_size, 0);
+ if (err)
+ goto err_out_unlock;
+
+ mutex_unlock(&st->state_lock);
+
+ ret = wait_event_interruptible_timeout(st->thread_wait,
+ !test_bit(NETFS_COMMAND_PENDING, &parent->state), ret);
+ if (ret <= 0) {
+ err = -ETIMEDOUT;
+ goto err_out_exit;
+ }
+
+ return 0;
+
+err_out_unlock:
+ mutex_unlock(&st->state_lock);
+err_out_exit:
+ clear_bit(NETFS_COMMAND_PENDING, &parent->state);
+
+ printk("%s: failed: parent: %llu, ino: %llu, name: '%s', err: %d.\n",
+ __func__, parent->ino, ino, str->name, err);
+
+ return err;
+}
+
+/*
+ * VFS lookup callback.
+ * We first try to get inode number from local name cache, if we have one,
+ * then inode can be found in inode cache. If there is no inode or no object in
+ * local cache, try to lookup it on server. This only should be done for directories,
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ * which were not created locally, otherwise remote server does not know about dir at all,
+ * so no need to try to know that.
+ */
+struct dentry *pohmelfs_lookup(struct inode *dir, struct dentry *dentry, struct nameidata *nd)
+{
+ struct pohmelfs_inode *parent = POHMELFS_I(dir);
+ struct pohmelfs_name *n;
+ struct inode *inode = NULL;
+ unsigned long ino = 0;
+ int err;
+ struct qstr str = dentry->d_name;
+
+ str.hash = jhash(dentry->d_name.name, dentry->d_name.len, 0);
+
+ dprintk("%s: dir: %p, dir_ino: %lu/%llu, dentry: %p, dinode: %p, "
+ "name: '%s', len: %u, dir_state: %lx.\n",
+ __func__, dir, dir->i_ino, parent->ino,
+ dentry, dentry->d_inode, str.name, str.len, parent->state);
+
+ mutex_lock(&parent->offset_lock);
+ n = pohmelfs_search_hash(parent, str.hash, str.len);
+ if (n)
+ ino = n->ino;
+ mutex_unlock(&parent->offset_lock);
+
+ #ifdef POHMELFS_FULL_DIR_RESYNC_ON_LOOKUP
+ mutex_lock(&parent->offset_lock);
+ n = pohmelfs_search_offset(parent, 3);
+ if (n) {
+ struct rb_node *rb_node;
+ for (rb_node = &n->offset_node; rb_node;) {
+ n = rb_entry(rb_node, struct pohmelfs_name, offset_node);
+ rb_node = rb_next(rb_node);
+
+ pohmelfs_name_free(parent, n);
+ }
+ }
+
+ parent->total_len = 3;
+ mutex_unlock(&parent->offset_lock);
+
+ clear_bit(NETFS_INODE_REMOTE_SYNCED, &parent->state);
+
+ err = pohmelfs_sync_remote_dir(parent);
+ if (err)
+ return NULL;
+
+ mutex_unlock(&parent->offset_lock);
+ n = pohmelfs_search_hash(parent, str.hash, str.len);
+ if (n)
+ ino = n->ino;
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ mutex_unlock(&parent->offset_lock);
+
+ inode = ilookup(dir->i_sb, ino);
+ if (!inode)
+ return NULL;
+
+ dentry = d_splice_alias(inode, dentry);
+ iput(inode);
+
+ return dentry;
+#else
+ if (ino) {
+ inode = ilookup(dir->i_sb, ino);
+ dprintk("%s: first lookup ino: %lu, inode: %p, name: '%s', hash: %x.\n",
+ __func__, ino, inode, str.name, str.hash);
+ if (inode)
+ return d_splice_alias(inode, dentry);
+ }
+
+ if (!test_bit(NETFS_INODE_CREATED, &parent->state))
+ return NULL;
+
+ err = pohmelfs_lookup_single(parent, &str, ino);
+ if (err)
+ return NULL;
+
+ if (!ino) {
+ mutex_lock(&parent->offset_lock);
+ n = pohmelfs_search_hash(parent, str.hash, str.len);
+ if (n)
+ ino = n->ino;
+ mutex_unlock(&parent->offset_lock);
+ }
+
+ if (ino) {
+ inode = ilookup(dir->i_sb, ino);
+ dprintk("%s: second lookup ino: %lu, inode: %p, name: '%s', hash: %x.\n",
+ __func__, ino, inode, str.name, str.hash);
+ if (!inode) {
+ printk("%s: No inode for ino: %lu, name: '%s', hash: %x.\n",
+ __func__, ino, str.name, str.hash);
+ //return NULL;
+ return ERR_PTR(-EACCES);
+ }
+ } else {
+ dprintk("%s: No inode number : name: '%s', hash: %x.\n",
+ __func__, str.name, str.hash);
+ }
+
+ return d_splice_alias(inode, dentry);
+#endif
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+}
+
+/*
+ * Create new object in local cache. Object will be synced to server
+ * during writeback for given inode.
+ */
+struct pohmelfs_inode *pohmelfs_create_entry_local(struct pohmelfs_sb *psb,
+ struct pohmelfs_inode *parent, struct qstr *str, u64 start, int mode)
+{
+ struct pohmelfs_inode *npi;
+ struct netfs_state *st = pohmelfs_active_state(psb);
+ int err = -ENOMEM;
+
+
+ dprintk("%s: name: '%s', mode: %o, start: %llu.\n",
+ __func__, str->name, mode, start);
+
+ mutex_lock(&st->state_lock);
+
+
+ st->info.mode = mode;
+ st->info.ino = start;
+
+
+ if (!start)
+ st->info.ino = psb->ino++;
+
+
+ st->info.nlink = S_ISDIR(mode)?2:1;
+ st->info.uid = current->uid;
+ st->info.gid = current->gid;
+ st->info.size = 0;
+ st->info.blocksize = 512;
+ st->info.blocks = 0;
+ st->info.rdev = 0;
+ st->info.version = 0;
+
+
+ npi = pohmelfs_new_inode(psb, parent, str, &st->info, !!start);
+ if (IS_ERR(npi)) {
+ err = PTR_ERR(npi);
+ goto err_out_unlock;
+ }
+
+
+ mutex_unlock(&st->state_lock);
+
+
+ return npi;
+
+err_out_unlock:
+ mutex_unlock(&st->state_lock);
+ dprintk("%s: err: %d.\n", __func__, err);
+ return ERR_PTR(err);
+}
+
+/*
+ * Create local object and bind it to dentry.
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ */
+static int pohmelfs_create_entry(struct inode *dir, struct dentry *dentry, u64 start, int mode)
+{
+ struct pohmelfs_sb *psb = POHMELFS_SB(dir->i_sb);
+ struct pohmelfs_inode *npi;
+ struct qstr str = dentry->d_name;
+
+ str.hash = jhash(dentry->d_name.name, dentry->d_name.len, 0);
+
+ npi = pohmelfs_create_entry_local(psb, POHMELFS_I(dir), &str, start, mode);
+ if (IS_ERR(npi))
+ return PTR_ERR(npi);
+
+ d_instantiate(dentry, &npi->vfs_inode);
+
+ dprintk("%s: parent: %llu, inode: %llu, name: '%s', parent_nlink: %d, nlink: %d.\n",
+ __func__, POHMELFS_I(dir)->ino, npi->ino, dentry->d_name.name,
+ (signed)dir->i_nlink, (signed)npi->vfs_inode.i_nlink);
+
+ return 0;
+}
+
+/*
+ * VFS create and mkdir callbacks.
+ */
+static int pohmelfs_create(struct inode *dir, struct dentry *dentry, int mode,
+ struct nameidata *nd)
+{
+ return pohmelfs_create_entry(dir, dentry, 0, mode);
+}
+
+static int pohmelfs_mkdir(struct inode *dir, struct dentry *dentry, int mode)
+{
+ int err;
+
+ inode_inc_link_count(dir);
+ err = pohmelfs_create_entry(dir, dentry, 0, mode | S_IFDIR);
+ if (err)
+ inode_dec_link_count(dir);
+
+ return err;
+}
+
+/*
+ * Remove entry from local cache.
+ * Object will not be removed from server, instead it will be queued into parent
+ * to-be-removed queue, which will be processed during parent writeback (parent
+ * also marked as dirty). Writeback will send remove request to server.
+ * Such approach allows to remove vey huge directories (like 2.6.24 kernel tree)
+ * with only single network command.
+ */
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+static int pohmelfs_remove_entry(struct inode *dir, struct dentry *dentry)
+{
+ struct pohmelfs_sb *psb = POHMELFS_SB(dir->i_sb);
+ struct inode *inode = dentry->d_inode;
+ struct pohmelfs_inode *parent = POHMELFS_I(dir), *pi = POHMELFS_I(inode);
+ struct pohmelfs_name *n;
+ int err = -ENOENT;
+ struct qstr str = dentry->d_name;
+
+ str.hash = jhash(dentry->d_name.name, dentry->d_name.len, 0);
+
+ dprintk("%s: dir_ino: %llu, inode: %llu, name: '%s', nlink: %d.\n",
+ __func__, parent->ino, pi->ino,
+ str.name, (signed)inode->i_nlink);
+
+ mutex_lock(&parent->offset_lock);
+ n = pohmelfs_search_hash(parent, str.hash, str.len);
+ if (n) {
+ pohmelfs_fix_offset(parent, n);
+ if (test_bit(NETFS_INODE_CREATED, &pi->state)) {
+ __pohmelfs_name_del(parent, n);
+ list_add_tail(&n->sync_del_entry, &parent->sync_del_list);
+ } else
+ pohmelfs_name_free(parent, n);
+ err = 0;
+ }
+ mutex_unlock(&parent->offset_lock);
+
+ if (!err) {
+ mutex_lock(&psb->path_lock);
+ pohmelfs_remove_path_entry_by_ino(psb, pi->ino);
+ mutex_unlock(&psb->path_lock);
+
+ pohmelfs_inode_del_inode(psb, pi);
+
+ mark_inode_dirty(dir);
+
+ inode->i_ctime = dir->i_ctime;
+ if (inode->i_nlink)
+ inode_dec_link_count(inode);
+ }
+ dprintk("%s: inode: %p, lock: %ld, unhashed: %d.\n",
+ __func__, pi, inode->i_state & I_LOCK, hlist_unhashed(&inode->i_hash));
+
+ return err;
+}
+
+/*
+ * Unlink and rmdir VFS callbacks.
+ */
+static int pohmelfs_unlink(struct inode *dir, struct dentry *dentry)
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+{
+ return pohmelfs_remove_entry(dir, dentry);
+}
+
+static int pohmelfs_rmdir(struct inode *dir, struct dentry *dentry)
+{
+ int err;
+ struct inode *inode = dentry->d_inode;
+
+ dprintf("%s: parent: %llu, inode: %llu, name: '%s', parent_nlink: %d, nlink: %d.\n",
+ __func__, POHMELFS_I(dir)->ino, POHMELFS_I(inode)->ino,
+ dentry->d_name.name, (signed)dir->i_nlink, (signed)inode->i_nlink);
+
+ err = pohmelfs_remove_entry(dir, dentry);
+ if (!err) {
+ inode_dec_link_count(dir);
+ inode_dec_link_count(inode);
+ }
+
+ return err;
+}
+
+/*
+ * Link creation is synchronous.
+ * I'm lazy.
+ * Earth is somewhat round.
+ */
+static int pohmelfs_create_link(struct pohmelfs_inode *parent, struct qstr *obj,
+ struct pohmelfs_inode *target, struct qstr *tstr)
+{
+ struct super_block *sb = parent->vfs_inode.i_sb;
+ struct pohmelfs_sb *psb = POHMELFS_SB(sb);
+ struct netfs_state *st = pohmelfs_active_state(psb);
+ struct netfs_cmd *cmd = &st->cmd;
+ unsigned path_size = 0;
+ struct inode *inode = &parent->vfs_inode;
+ int err;
+
+ err = sb->s_op->write_inode(inode, 0);
+ if (err)
+ return err;
+
+ mutex_lock(&st->state_lock);
+
+ mutex_lock(&st->psb->path_lock);
+ err = pohmelfs_construct_path_string(parent, st->data, st->size - obj->len - 1);
+ if (err > 0) {
+ path_size = err;
+
+ path_size += sprintf(st->data + path_size, "%s", obj->name);
+
+ }
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ cmd->ext = path_size - 1; /* No | symbol */
+
+ if (target) {
+ err = pohmelfs_construct_path_string(target, st->data + path_size, st->size - path_size - 1);
+ if (err > 0)
+ path_size += err + 1;
+ }
+ }
+ mutex_unlock(&st->psb->path_lock);
+
+ if (err < 0)
+ goto err_out_unlock;
+
+ cmd->start = 0;
+
+ if (!target) {
+ if (tstr->len > st->size - path_size - 1) {
+ err = -ENAMETOOLONG;
+ goto err_out_unlock;
+ }
+
+ path_size += sprintf(st->data + path_size, "%s", tstr->name) + 1 /* 0-byte */;
+ cmd->start = 1;
+ }
+
+ dprintk("%s: parent: %llu, obj: '%s', target_inode: %llu, target_str: '%s', full: '%s'.\n",
+ __func__, parent->ino, obj->name, (target)?target->ino:0, (tstr)?tstr->name:NULL,
+ (char *)st->data);
+
+ cmd->cmd = NETFS_LINK;
+ cmd->size = path_size;
+ cmd->id = parent->ino;
+ netfs_convert_cmd(cmd);
+
+ err = netfs_data_send(st, cmd, sizeof(struct netfs_cmd), 1);
+ if (err)
+ goto err_out_unlock;
+
+ err = netfs_data_send(st, st->data, path_size, 0);
+ if (err)
+ goto err_out_unlock;
+
+ mutex_unlock(&st->state_lock);
+
+ return 0;
+
+err_out_unlock:
+ mutex_unlock(&st->state_lock);
+
+ return err;
+}
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+
+/*
+ * VFS hard and soft link callbacks.
+ */
+static int pohmelfs_link(struct dentry *old_dentry, struct inode *dir,
+ struct dentry *dentry)
+{
+ struct inode *inode = old_dentry->d_inode;
+ struct pohmelfs_inode *pi = POHMELFS_I(inode);
+ int err;
+ struct qstr str = dentry->d_name;
+
+ str.hash = jhash(dentry->d_name.name, dentry->d_name.len, 0);
+
+ err = inode->i_sb->s_op->write_inode(inode, 0);
+ if (err)
+ return err;
+
+ return pohmelfs_create_link(POHMELFS_I(dir), &str, pi, NULL);
+}
+
+static int pohmelfs_symlink(struct inode *dir, struct dentry *dentry, const char *symname)
+{
+ struct qstr sym_str;
+ struct qstr str = dentry->d_name;
+
+ str.hash = jhash(dentry->d_name.name, dentry->d_name.len, 0);
+
+ sym_str.name = symname;
+ sym_str.len = strlen(symname);
+
+ return pohmelfs_create_link(POHMELFS_I(dir), &str, NULL, &sym_str);
+}
+/*
+ * POHMELFS inode operations.
+ */
+const struct inode_operations pohmelfs_dir_inode_ops = {
+ .link = pohmelfs_link,
+ .symlink = pohmelfs_symlink,
+ .unlink = pohmelfs_unlink,
+ .mkdir = pohmelfs_mkdir,
+ .rmdir = pohmelfs_rmdir,
+ .create = pohmelfs_create,
+ .lookup = pohmelfs_lookup,
+};
diff --git a/fs/pohmelfs/inode.c b/fs/pohmelfs/inode.c
new file mode 100644
index 0000000..3fb11ac
--- /dev/null
+++ b/fs/pohmelfs/inode.c
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
@@ -0,0 +1,1543 @@
+/*
+ * 2007+ Copyright (c) Evgeniy Polyakov <johnpol@xxxxxxxxxxx>
+ * All rights reserved.
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/module.h>
+#include <linux/backing-dev.h>
+#include <linux/fs.h>
+#include <linux/jhash.h>
+#include <linux/hash.h>
+#include <linux/ktime.h>
+#include <linux/mm.h>
+#include <linux/mount.h>
+#include <linux/pagemap.h>
+#include <linux/pagevec.h>
+#include <linux/parser.h>
+#include <linux/swap.h>
+#include <linux/slab.h>
+#include <linux/statfs.h>
+#include <linux/writeback.h>
+
+#include "netfs.h"
+
+static struct kmem_cache *pohmelfs_inode_cache;
+
+/*
+ * Removes inode from all trees, drops local name cache and removes all queued
+ * requests for object removal.
+ */
+void pohmelfs_inode_del_inode(struct pohmelfs_sb *psb, struct pohmelfs_inode *pi)
+{
+ struct pohmelfs_name *n, *tmp;
+
+ mutex_lock(&pi->offset_lock);
+ pohmelfs_free_names(pi);
+
+ list_for_each_entry_safe(n, tmp, &pi->sync_create_list, sync_create_entry) {
+ list_del_init(&n->sync_create_entry);
+ list_del_init(&n->sync_del_entry);
+ kfree(n);
+ }
```

```

+ }
+
+ list_for_each_entry_safe(n, tmp, &pi->sync_del_list, sync_del_entry) {
+ list_del_init(&n->sync_create_entry);
+ list_del_init(&n->sync_del_entry);
+ kfree(n);
+ }
+ mutex_unlock(&pi->offset_lock);
+
+ dprintk("%s: deleted stuff in ino: %llu.\n", __func__, pi->ino);
+}
+
+/*
+ * Sync inode to server. If @wait is set, it will wait for acknowledge.
+ * Returns zero in success and negative error value otherwise.
+ * It will gather path to root directory into structures containing
+ * creation mode, permissions and names, so that the whole path
+ * to given inode could be created using only single network command.
+ */
+static int pohmelfs_write_inode_create(struct inode *inode, struct netfs_trans *trans, int wait)
+{
+ struct pohmelfs_inode *pi = POHMELFS_I(inode);
+ struct pohmelfs_sb *psb = POHMELFS_SB(inode->i_sb);
+ int err = -ENOMEM, size;
+ struct netfs_cmd *cmd;
+ void *data;
+ unsigned int cur_len = trans->data_size - netfs_trans_cur_len(trans);
+
+ dprintk("%s: started ino: %llu, wait: %d.\n", __func__, pi->ino, wait);
+
+ cmd = netfs_trans_add(trans, cur_len);
+ if (IS_ERR(cmd)) {
+ err = PTR_ERR(cmd);
+ goto err_out_exit;
+ }
+
+ data = (void *) (cmd + 1);
+
+ mutex_lock(&psb->path_lock);
+ err = pohmelfs_construct_path(pi, data, cur_len - sizeof(struct netfs_cmd));
+ mutex_unlock(&psb->path_lock);
+ if (err < 0)
+ goto err_out_unroll;
+
+ size = err;
+
+ netfs_trans_fixup_last(trans, size + sizeof(struct netfs_cmd) - cur_len);
+
+ if (size) {
+ cmd->start = 0;
+ cmd->cmd = NETFS_CREATE;

```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ cmd->size = size;
+ cmd->id = pi->ino;
+ cmd->ext = !!wait;
+
+ netfs_convert_cmd(cmd);
+ }
+
+ dprintk("%s: completed ino: %llu, size: %d.\n", __func__, pi->ino, size);
+ return 0;
+
+err_out_unroll:
+ netfs_trans_fixup_last(trans, cur_len);
+err_out_exit:
+ clear_bit(NETFS_INODE_CREATED, &pi->state);
+ dprintk("%s: completed ino: %llu, err: %d.\n", __func__, pi->ino, err);
+ return err;
+}
+
+static void pohmelfs_write_trans_complete(struct netfs_trans *t, int err)
+{
+ unsigned i;
+
+ dprintk("%s: t: %p, trans_gen: %u, trans_size: %u, data_size: %u, trans_idx: %u, iovec_num: %u, err:
+d.\n",
+ __func__, t, t->trans_gen, t->trans_size, t->data_size, t->trans_idx, t->iovec_num, err);
+
+ for (i = 0; i < t->iovec_num-1; i++) {
+ struct page *page = t->data[i+1];
+
+ if (!page)
+ continue;
+
+ dprintk("%s: completed page: %p, size: %lu.\n",
+ __func__, page, page_private(page));
+
+ if (err) {
+ SetPageDirty(page);
+ TestClearPageWriteback(page);
+ }
+ end_page_writeback(page);
+ kunmap(page);
+ page_cache_release(page);
+ }
+ netfs_trans_exit(t);
+}
+
+static int pohmelfs_writepages(struct address_space *mapping, struct writeback_control *wbc)
+{
+ struct inode *inode = mapping->host;
+ struct pohmelfs_inode *pi = POHMELFS_I(inode);
+ struct pohmelfs_sb *psb = POHMELFS_SB(inode->i_sb);
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ struct netfs_state *st = pohmelfs_active_state(psb);
+ struct backing_dev_info *bdi = mapping->backing_dev_info;
+ int ret = 0;
+ int done = 0;
+ int nr_pages;
+ pgoff_t index;
+ pgoff_t end; /* Inclusive */
+ int scanned = 0;
+ int range_whole = 0;
+
+ if (wbc->nonblocking && bdi_write_congested(bdi)) {
+ wbc->encountered_congestion = 1;
+ return 0;
+ }
+
+ if (wbc->range_cyclic) {
+ index = mapping->writeback_index; /* Start from prev offset */
+ end = -1;
+ } else {
+ index = wbc->range_start >> PAGE_CACHE_SHIFT;
+ end = wbc->range_end >> PAGE_CACHE_SHIFT;
+ if (wbc->range_start == 0 && wbc->range_end == LLONG_MAX)
+ range_whole = 1;
+ scanned = 1;
+ }
+retry:
+ while (!done && (index <= end)) {
+ unsigned int i = min(end - index, (pgoff_t)1024);
+ unsigned int cur_len, added = 0;
+ void *data;
+ struct netfs_cmd *cmd, *cmds;
+ struct netfs_trans *trans;
+
+ trans = netfs_trans_alloc_for_pages(i);
+ if (!trans) {
+ ret = -ENOMEM;
+ goto err_out_break;
+ }
+
+ cmds = (struct netfs_cmd*)(trans->data + trans->iovec_num);
+
+ nr_pages = find_get_pages_tag(mapping, &index,
+ PAGECACHE_TAG_DIRTY, trans->iovec_num-1, (struct page **)&trans->data[1]);
+
+ dprintk("%s: t: %p, nr_pages: %u, end: %lu, index: %lu, max: %u.\n",
+ __func__, trans, nr_pages, end, index, trans->iovec_num);
+
+ if (!nr_pages)
+ goto err_out_break;
+
+ trans->complete = &pohmelfs_write_trans_complete;
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+
+ ret = netfs_trans_start_empty(trans, NETFS_TRANS_SYNC);
+ if (ret)
+ goto err_out_break;
+
+ cmd = netfs_trans_add(trans, sizeof(struct netfs_cmd));
+ if (IS_ERR(cmd)) {
+ ret = PTR_ERR(cmd);
+ goto err_out_reset;
+ }
+
+ trans->trans_size -= sizeof(struct netfs_cmd);
+
+ if (!test_bit(NETFS_INODE_CREATED, &pi->state)) {
+ ret = pohmelfs_write_inode_create(inode, trans, 0);
+ if (ret)
+ goto err_out_reset;
+ }
+
+ cur_len = trans->data_size - netfs_trans_cur_len(trans);
+
+ cmd = netfs_trans_add(trans, cur_len);
+ if (IS_ERR(cmd)) {
+ ret = PTR_ERR(cmd);
+ goto err_out_reset;
+ }
+
+ data = (void*)(cmd + 1);
+
+ mutex_lock(&psb->path_lock);
+ ret = pohmelfs_construct_path_string(pi, data, cur_len - sizeof(struct netfs_cmd));
+ mutex_unlock(&psb->path_lock);
+ if (ret < 0)
+ goto err_out_reset;
+
+ netfs_trans_fixup_last(trans, ret + 1 + sizeof(struct netfs_cmd) - cur_len);
+
+ cmd->id = pi->ino;
+ cmd->start = 0;
+ cmd->size = ret + 1;
+ cmd->cmd = NETFS_OPEN;
+ cmd->ext = O_RDWR;
+
+ netfs_convert_cmd(cmd);
+
+ ret = 0;
+
+ scanned = 1;
+ for (i = 0; i < nr_pages; i++) {
+ struct page *page = trans->data[i+1];
+ struct iovec *io;
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+
+ lock_page(page);
+
+ if (unlikely(page->mapping != mapping)) {
+ dprintk("%s: page: %p, page_mapping: %p, mapping: %p.\n",
+ __func__, page, page->mapping, mapping);
+ continue;
+ }
+
+ if (!wbc->range_cyclic && page->index > end) {
+ done = 1;
+ dprintk("%s: false.\n", __func__);
+ continue;
+ }
+
+ if (wbc->sync_mode != WB_SYNC_NONE)
+ wait_on_page_writeback(page);
+
+ if (PageWriteback(page) ||
+ !clear_page_dirty_for_io(page)) {
+ dprintk("%s: page: %p, writeback: %d.\n", __func__, page, PageWriteback(page));
+ continue;
+ }
+
+ set_page_writeback(page);
+ unlock_page(page);
+
+ cmd = &cmds[i];
+
+ cmd->id = pi->ino;
+ cmd->start = page->index << PAGE_CACHE_SHIFT;
+ cmd->size = page_private(page);
+ cmd->cmd = NETFS_WRITE_PAGE;
+ cmd->ext = 0;
+
+ trans->trans_size += cmd->size + sizeof(struct netfs_cmd);
+
+ netfs_convert_cmd(cmd);
+
+ io = &trans->iovec[++trans->trans_idx];
+ io->iov_len = sizeof(struct netfs_cmd);
+ io->iov_base = cmd;
+
+ io = &trans->iovec[++trans->trans_idx];
+ io->iov_len = page_private(page);
+ io->iov_base = kmap(page);
+
+ added++;
+
+ dprintk("%s: added trans: %p, idx: %u, page: %p, addr: %p [High: %d], size: %lu.\n",
+ __func__, trans, trans->trans_idx, page, io->iov_base,
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ !!PageHighMem(page), page_private(page));
+
+ if (ret || (--(wbc->nr_to_write) <= 0))
+ done = 1;
+ if (wbc->nonblocking && bdi_write_congested(bdi)) {
+ wbc->encountered_congestion = 1;
+ done = 1;
+ }
+ }
+
+ if (added) {
+ ret = netfs_trans_finish(trans, st);
+ if (ret)
+ netfs_trans_exit(trans);
+ } else {
+ netfs_trans_reset(trans);
+ netfs_trans_exit(trans);
+ }
+
+ if (ret)
+ break;
+
+ continue;
+
+err_out_reset:
+ netfs_trans_reset(trans);
+err_out_break:
+ netfs_trans_exit(trans);
+ break;
+ }
+
+ if (!scanned && !done) {
+ /*
+ * We hit the last page and there is more work to be done: wrap
+ * back to the start of the file
+ */
+ scanned = 1;
+ index = 0;
+ goto retry;
+ }
+
+ dprintk("%s: range_cyclic: %d, range_whole: %d, nr_to_write: %lu, index: %lu, ret: %d.\n",
+ __func__, wbc->range_cyclic, range_whole, wbc->nr_to_write, index, ret);
+
+ if (wbc->range_cyclic || (range_whole && wbc->nr_to_write > 0))
+ mapping->writeback_index = index;
+
+ if (!ret)
+ set_bit(NETFS_INODE_CREATED, &pi->state);
+ return ret;
+ }
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+
+/*
+ * Removes given child from given inode on server.
+ */
+static int pohmelfs_remove_child(struct pohmelfs_inode *parent, struct pohmelfs_name *n)
+{
+ struct pohmelfs_sb *psb = POHMELFS_SB(parent->vfs_inode.i_sb);
+ struct netfs_state *st = pohmelfs_active_state(psb);
+ int err, path_size;
+ struct netfs_cmd *cmd = &st->cmd;
+
+ mutex_lock(&st->state_lock);
+ mutex_lock(&psb->path_lock);
+ err = pohmelfs_construct_path_string(parent, st->data, st->size - n->len);
+ mutex_unlock(&psb->path_lock);
+ if (err < 0)
+ goto err_out_unlock;
+
+ path_size = err + sprintf(st->data + err, "%s", n->data) + 1 /* 0-byte */;
+
+ dprintk("%s: dir: %llu, ino: %llu, path: '%s', len: %d, mode: %o, dir: %d.\n",
+ __func__, parent->ino, n->ino, (char *)st->data, path_size,
+ n->mode, S_ISDIR(n->mode));
+
+ cmd->cmd = NETFS_REMOVE;
+ cmd->id = n->ino;
+ cmd->start = parent->ino;
+ cmd->size = path_size;
+ cmd->ext = S_ISDIR(n->mode);
+
+ netfs_convert_cmd(cmd);
+
+ err = netfs_data_send(st, cmd, sizeof(struct netfs_cmd), 1);
+ if (err)
+ goto err_out_unlock;
+
+ err = netfs_data_send(st, st->data, path_size, 0);
+ if (err)
+ goto err_out_unlock;
+
+ mutex_unlock(&st->state_lock);
+
+ return 0;
+
+err_out_unlock:
+ mutex_unlock(&st->state_lock);
+
+ return err;
+}
+
+/*
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ * Removes all childs, marked for deletion, on server.
+ */
+static int pohmelfs_write_inode_remove_children(struct inode *inode)
+{
+ struct pohmelfs_inode *pi = POHMELFS_I(inode);
+ int err, error = 0;
+ struct pohmelfs_name *n, *tmp;
+
+ dprintk("%s: parent: %llu, del_list_empty: %d.\n",
+ __func__, pi->ino, list_empty(&pi->sync_del_list));
+
+ if (!list_empty(&pi->sync_del_list)) {
+ mutex_lock(&pi->offset_lock);
+ list_for_each_entry_safe(n, tmp, &pi->sync_del_list, sync_del_entry) {
+ list_del_init(&n->sync_del_entry);
+ list_del_init(&n->sync_create_entry);
+
+ err = pohmelfs_remove_child(pi, n);
+ if (err)
+ error = err;
+
+ kfree(n);
+ }
+ mutex_unlock(&pi->offset_lock);
+ }
+ return error;
+}
+
+/*
+ * Writeback for given inode.
+ */
+static int pohmelfs_write_inode(struct inode *inode, int sync)
+{
+ int err = 0;
+
+ dprintk("%s: started ino: %llu.\n", __func__, POHMELFS_I(inode)->ino);
+
+ if (!test_bit(NETFS_INODE_CREATED, &POHMELFS_I(inode)->state)) {
+ struct netfs_state *st = pohmelfs_active_state(POHMELFS_SB(inode->i_sb));
+ struct netfs_trans *trans = st->trans;
+
+ err = netfs_trans_start(trans, NETFS_TRANS_SYNC);
+ if (err)
+ goto out;
+
+ err = pohmelfs_write_inode_create(inode, trans, sync);
+ if (err) {
+ netfs_trans_reset(trans);
+ goto out;
+ }
+ }
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+
+ err = netfs_trans_finish(trans, st);
+ if (err)
+ goto out;
+ set_bit(NETFS_INODE_CREATED, &POHMELFS_I(inode)->state);
+
+out:
+ netfs_trans_exit(trans);
+ }
+
+ pohmelfs_write_inode_remove_children(inode);
+
+ return err;
+}
+
+/*
+ * It is not exported, sorry...
+ */
+static inline wait_queue_head_t *page_waitqueue(struct page *page)
+{
+ const struct zone *zone = page_zone(page);
+
+ return &zone->wait_table[hash_ptr(page, zone->wait_table_bits)];
+}
+
+/*
+ * Read/write page request to remote server.
+ * If @wait is set and page is locked, it will wait until page is unlocked.
+ */
+static int netfs_process_page(struct page *page, __u32 cmd_op, __u32 size, int wait)
+{
+ struct inode *inode = page->mapping->host;
+ struct pohmelfs_sb *psb = POHMELFS_SB(inode->i_sb);
+ struct pohmelfs_inode *pi = POHMELFS_I(inode);
+ struct netfs_state *st = pohmelfs_active_state(psb);
+ struct netfs_cmd *cmd = &st->cmd;
+ int err, path_size;
+
+ if (unlikely(!size)) {
+ SetPageUptodate(page);
+ unlock_page(page);
+ return 0;
+ }
+
+ #if 0
+ {
+ SetPageUptodate(page);
+ unlock_page(page);
+ return 0;
+ }
+
+ #endif
+}
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+
+ mutex_lock(&st->state_lock);
+
+ mutex_lock(&psb->path_lock);
+ err = pohmelfs_construct_path_string(pi, st->data, st->size);
+ mutex_unlock(&psb->path_lock);
+ if (err < 0)
+ goto err_out_unlock;
+
+ path_size = err + 1;
+
+ cmd->id = pi->ino;
+ cmd->start = page->index << PAGE_CACHE_SHIFT;
+ cmd->size = size + path_size;
+ cmd->cmd = cmd_op;
+ cmd->ext = path_size;
+
+ dprintk("%s: path: '%s', page: %p, ino: %llu, start: %llu, idx: %lu, cmd: %u, size: %u.\n",
+ __func__, (char *)st->data, page, pi->ino, cmd->start, page->index, cmd_op, size);
+
+ netfs_convert_cmd(cmd);
+
+ err = netfs_data_send(st, cmd, sizeof(struct netfs_cmd), 1);
+ if (err)
+ goto err_out_unlock;
+
+ err = netfs_data_send(st, st->data, path_size, cmd_op == NETFS_WRITE_PAGE);
+ if (err)
+ goto err_out_unlock;
+
+ if (cmd_op == NETFS_WRITE_PAGE) {
+ err = kernel_sendpage(st->socket, page, 0, size, MSG_WAITALL | MSG_NOSIGNAL);
+ if (err < 0)
+ goto err_out_unlock;
+
+ SetPageUptodate(page);
+ unlock_page(page);
+
+ mutex_unlock(&st->state_lock);
+
+ return 0;
+ }
+
+ mutex_unlock(&st->state_lock);
+
+ err = 0;
+ if (wait && TestSetPageLocked(page)) {
+ long ret = msecs_to_jiffies(5000);
+ DEFINE_WAIT_BIT(wait, &page->flags, PG_locked);
+
+ for (;;) {
```

POHMELFS high performance network filesystem. Transactions, failover, performance.

POHMELFS high performance network filesystem. Transactions, failover, performance.

```
+ prepare_to_wait(page_waitqueue(page), &wait.wait, TASK_INTERRUPTIBLE);
+
+ dprintk("%s: page: %p, locked: %d, uptodate: %d, error: %d.\n",
+ __func__, page, PageLocked(page), PageUptodate(page),
+ PageError(page));
+
+ if (!PageLocked(page))
+ break;
+
+ if (!signal_pending(current)) {
+ ret = schedule_timeout(ret);
+ if (!ret)
+ break;
+ continue;
+ }
+ ret = -ERESTARTSYS;
+ break;
+ }
+ finish_wait(page_waitqueue(page), &wait.wait);
+
+ if (!ret)
+ err = -ETIMEDOUT;
+
+ dprintk("%s: page: %p, uptodate: %d, locked: %d, err: %d.\n",
+ __func__, page, PageUptodate(page), PageLocked(page), err);
+
+ if (!PageUptodate(page))
+ err = -EIO;
+
+ if (PageLocked(page))
+ unlock_page(page);
+ }
+
+ return err;
+
+err_out_unlock:
+ mutex_unlock(&st->state_lock);
+
+ SetPageError(page);
+ unlock_page(page);
+
+ dprintk("%s: page: %p, start: %llu/%llx, size: %u, err: %d.\n",
+ __func__, page, cmd->start, cmd->start, cmd->size, err);
+
+ return err;
+}
+
+static int pohmelfs_readpage(struct file *file, struct page *page)
+{
+ ClearPageChecked(page);
+ return netfs_process_page(page, NETFS_READ_PAGE, PAGE_CACHE_SIZE, 1);
+}
```

```
+}
+
+/*
+ * Write begin/end magic.
+ * Allocates a page and writes inode if it was not synced to server before.
+ */
+static int pohmelfs_write_begin(struct file *file, struct address_space *mapping,
+ loff_t pos, unsigned len, unsigned flags,
+ struct page **pagep, void **fsdata)
+{
+ struct inode *inode = mapping->host;
+ struct page *page;
+ pgoff_t index;
+ unsigned start, end;
+ int err;
+
+ *pagep = NULL;
+
+ index = pos >> PAGE_CACHE_SHIFT;
+ start = pos & (PAGE_CACHE_SIZE - 1);
+ end = start + len;
+
+ page = __grab_cache_page(mapping, index);
+
+ dprintk("%s: page: %p pos: %llu, len: %u, index: %lu, start: %u, end: %u.\n",
+ __func__, page, pos, len, index, start, end);
+ if (!page) {
+ err = -ENOMEM;
+ goto err_out_exit;
+ }
+
+ if (!PageUptodate(page)) {
+ if (start && test_bit(NETFS_INODE_CREATED, &POHMELFS_I(inode)->state)) {
+ err = pohmelfs_readpage(file, page)
```