

Tunable Kernel Parameters

Philex Lin, 13 May 2002.

NCTU CIS Operating System lab.

2002 Linux kernel trace seminar

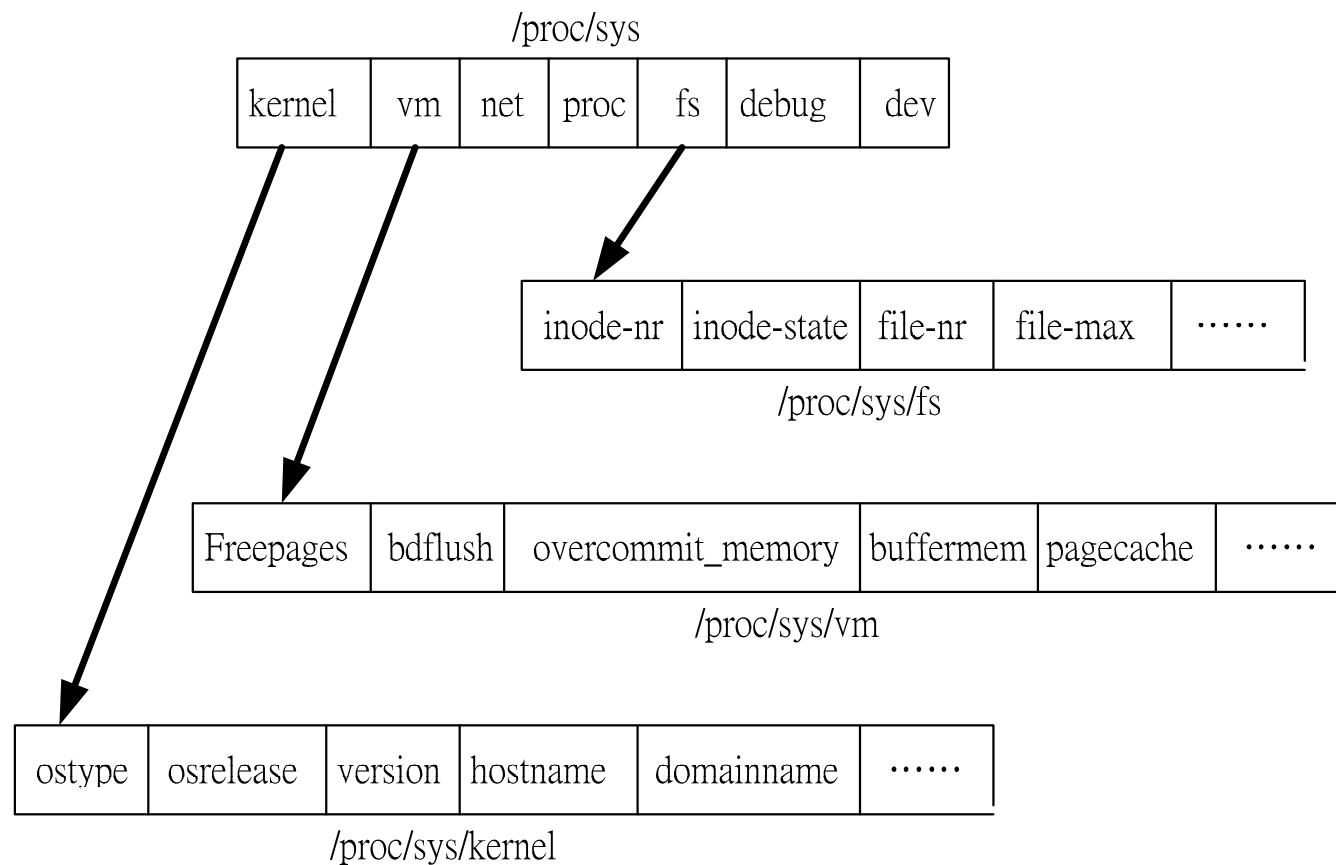
Outline

- | /proc file system and sysctl system call
- | Usage of /proc file system

/proc file system and sysctl system call

- I Two interfaces for tunable kernel parameters
 - /proc file system
 - sysctl system call
- I Difference for Read/Write kernel parameters between /proc file system and sysctl system call
 - /proc file system : accessible through shell commands , even shell scripts
 - Reading and writing through sysctl requires a C program

A partial tree of struct ctl_table



struct ctl_table

/* A sysctl table is an array of struct ctl_table: */

struct ctl_table

{

int ctl_name;

const char *procname;

void *data;

int maxlen;

mode_t mode;

ctl_table *child;

proc_handler *proc_handler;

ctl_handler *strategy;

struct proc_dir_entry *de;

void *extra1;

void *extra2;

};

```
static ctl_table root_table[] = {
    {CTL_KERN, "kernel", NULL, 0, 0555, kern_table},
    {CTL_VM, "vm", NULL, 0, 0555, vm_table},
#ifdef CONFIG_NET
    {CTL_NET, "net", NULL, 0, 0555, net_table},
#endif
    {CTL_PROC, "proc", NULL, 0, 0555, proc_table},
    {CTL_FS, "fs", NULL, 0, 0555, fs_table},
    {CTL_DEBUG, "debug", NULL, 0, 0555, debug_table},
    {CTL_DEV, "dev", NULL, 0, 0555, dev_table},
    {0}
};
```

/* Callback for text formatting */

/* Callback function for all r/w */

/* /proc control block */

Register_proc_table(1)

```
static void register_proc_table(ctl_table * table,
    struct proc_dir_entry *root)
{
    struct proc_dir_entry *de;
    int len;
    mode_t mode;

    for (; table->ctl_name; table++) {
        /* Can't do anything without a proc name. */
        if (!table->procname)
            continue;
        /* Maybe we can't do anything with it... */
        if (!table->proc_handler && !table->child) {
            printk(KERN_WARNING "SYSCTL: Can't register %s\n",
                table->procname);
            continue;
        }
    }
}
```

Register_proc_table(2)

.....

```
len = strlen(table->procname);  
mode = table->mode;
```

```
de = NULL;
```

```
if (table->proc_handler)
```

```
    mode |= S_IFREG;
```

```
else {
```

```
    mode |= S_IFDIR;
```

```
    for (de = root->subdir; de; de = de->next) {  
        if (proc_match(len, table->procname, de))  
            break;
```

```
    }
```

```
    /* If the subdir exists already, de is non-NULL */
```

```
}
```

.....

```
int proc_match(int len, const char  
*name, struct proc_dir_entry * de)  
{  
    if (!de || !de->low_ino)  
        return 0;  
    if (de->namelen != len)  
        return 0;  
    return !memcmp(name, de->name, len);  
}
```

Register_proc_table(3)

```
.....
if (!de) {
    de = create_proc_entry(table->procname, mode, table);
    if (!de)
        continue;
    de->data = (void *) table;
    if (table->proc_handler) {
        de->proc_fops = &proc_sys_file_operations;
        de->proc_iops = &proc_sys_inode_operations;
    }
}
table->de = de;
if (de->mode & S_IFDIR)
    register_proc_table(table->child, de);
}
```

```
struct proc_dir_entry *create_proc_entry(const char
*name, mode_t mode, struct proc_dir_entry *parent)
{ struct proc_dir_entry *ent;
  nlink_t nlink;
  if (S_ISDIR(mode)) {
      if ((mode & S_IALLUGO) == 0)
          mode |= S_IRUGO | S_IXUGO;
      nlink = 2;
  } else {
      if ((mode & S_IFMT) == 0)
          mode |= S_IFREG;
      if ((mode & S_IALLUGO) == 0)
          mode |= S_IRUGO;
      nlink = 1; }
  ent = proc_create(&parent,name,mode,nlink);
  if (ent) {
      if (S_ISDIR(mode)) {
          ent->proc_fops = &proc_dir_operations;
          ent->proc_iops = &proc_dir_inode_operations; }
      proc_register(parent, ent); }
  return ent; }
```

unregister_proc_table

```
static void unregister_proc_table(ctl_table * table, struct proc_dir_entry *root)
{
    struct proc_dir_entry *de;
    for (; table->ctl_name; table++) {
        if (!(de = table->de))
            continue;
        if (de->mode & S_IFDIR) {
            if (!table->child) {
                printk (KERN_ALERT "Help - malformed sysctl tree on free\n");
                continue;
            }
            unregister_proc_table(table->child, de);
            /* Don't unregister directories which still have entries.. */
            if (de->subdir)
                continue;
        }
        /* Don't unregister proc entries that are still being used.. */
        if (atomic_read(&de->count))
            continue;
        table->de = NULL;
        remove_proc_entry(table->procname, root);
    }
}
```

Do_rw_proc

```
static ssize_t do_rw_proc(int write, struct file * file, char * buf, size_t count, loff_t *ppos)
{
    int op;
    struct proc_dir_entry *de;
    struct ctl_table *table;
    size_t res;
    ssize_t error;

    de = (struct proc_dir_entry*) file->f_dentry->d_inode->u.generic_ip;
    if (!de || !de->data)
        return -ENOTDIR;
    table = (struct ctl_table *) de->data;
    if (!table || !table->proc_handler)
        return -ENOTDIR;
    op = (write ? 002 : 004);
    if (ctl_perm(table, op))
        return -EPERM;
    res = count;
    /*
     * FIXME: we need to pass on ppos to the handler.
     */
    error = (*table->proc_handler) (table, write, file, buf, &res);
    if (error)
        return error;
    return res;
}
```

```
/*
 * ctl_perm does NOT grant the superuser
 * all rights automatically, because
 * some sysctl variables are readonly even
 * to root.
 */
static int test_perm(int mode, int op)
{
    if (!current->euid)
        mode >>= 6;
    else if (in_egroup_p(0))
        mode >>= 3;
    if ((mode & op & 0007) == op)
        return 0;
    return -EACCES;}

```

Proc_dostring(1)

```
int proc_dostring(ctl_table *table, int write, struct file *filp,
                 void *buffer, size_t *lenp)
{
    size_t len;
    char *p, c;

    if (!table->data || !table->maxlen || !*lenp ||
        (filp->f_pos && !write)) {
        *lenp = 0;
        return 0;
    }

    if (write) {
        len = 0;  p = buffer;
        while (len < *lenp) {
            if(get_user(c, p++))
                return -EFAULT;
            if (c == 0 || c == '\n')
                break;
            len++;
        }
    }
}
```

.....

Proc_dostring(2)

```
.....
    if (len >= table->maxlen)
        len = table->maxlen-1;
    if(copy_from_user(table->data, buffer, len))
        return -EFAULT;
    ((char *) table->data)[len] = 0;
    filp->f_pos += *lenp;
} else {
    len = strlen(table->data);
    if (len > table->maxlen)
        len = table->maxlen;
    if (len > *lenp)
        len = *lenp;
    if (len)
        if(copy_to_user(buffer, table->data, len))
            return -EFAULT;
    if (len < *lenp) {
        if(put_user('\n', ((char *) buffer) + len))
            return -EFAULT;
        len++;    }
    *lenp = len;
    filp->f_pos += len;    }
return 0;}
```

Do_proc_dointvec(1)

```
static int do_proc_dointvec(ctl_table *table, int write, struct file *filp,
                           void *buffer, size_t *lenp, int conv, int op)
{
    int *i, vleft, first=1, neg, val;
    size_t left, len;

    #define TMPBUFLLEN 20
    char buf[TMPBUFLLEN], *p;

    if (!table->data || !table->maxlen || !*lenp ||
        (filp->f_pos && !write)) {
        *lenp = 0;
        return 0;
    }
    i = (int *) table->data;
    vleft = table->maxlen / sizeof(int);
    left = *lenp;
    for (; left && vleft--; i++, first=0) {
        if (write) {
            while (left) {
                char c;
                if(get_user(c,(char *) buffer))
                    return -EFAULT;
                if (!isspace(c))
                    break;
                left--;
                ((char *) buffer)++;
            }
        }
    }
}
```

Do_proc_dointvec(2)

```
.....
if (!left)
    break;
neg = 0;
len = left;
if (len > TMPBUFLen-1)
    len = TMPBUFLen-1;
if(copy_from_user(buf, buffer, len))
    return -EFAULT;
buf[len] = 0;
p = buf;
if (*p == '-' && left > 1) {
    neg = 1;
    left--, p++;
}
if (*p < '0' || *p > '9')
    break;
val = simple_strtoul(p, &p, 0) * conv;
len = p-buf;
if ((len < left) && *p && !isspace(*p))
    break;

if (neg)
    val = -val;
buffer += len;
left -= len;
switch(op) {
case OP_SET: *i = val; break;
case OP_AND: *i &= val; break;
```

Do_proc_dointvec(3)

```
.....
    case OP_OR: *i |= val; break;
    case OP_MAX: if(*i < val)
        *i = val;
        break;

    case OP_MIN: if(*i > val)
        *i = val;
        break;

    } else {
        p = buf;
        if (!first)
            *p++ = '\t';
        sprintf(p, "%d", (*i) / conv);
        len = strlen(buf);
        if (len > left)
            len = left;
        if(copy_to_user(buffer, buf, len))
            return -EFAULT;
        left -= len;
        buffer += len;
    }
}

if (!write && !first && left) {
    if(put_user('\n', (char *) buffer))
        return -EFAULT;
    left--, buffer++;
}
}
```

Do_proc_dointvec(4)

```
.....
if (write) {
    p = (char *) buffer;
    while (left) {
        char c;
        if(get_user(c, p++))
            return -EFAULT;
        if (!isspace(c))
            break;
        left--;
    }
}
if (write && first)
    return -EINVAL;
*lenp -= left;
filp->f_pos += *lenp;
return 0;
}
```

Proc_dointvec_minmax(1)

```
int proc_dointvec_minmax(ctl_table *table, int write, struct file *filp,
                        void *buffer, size_t *lenp)
{
    int *i, *min, *max, vleft, first=1, neg, val;
    size_t len, left;
    #define TMPBUFLLEN 20
    char buf[TMPBUFLLEN], *p;

    if (!table->data || !table->maxlen || !*lenp ||
        (filp->f_pos && !write)) {
        *lenp = 0; return 0;
    }
    i = (int *) table->data;
    min = (int *) table->extra1;
    max = (int *) table->extra2;
    vleft = table->maxlen / sizeof(int);
    left = *lenp;

    for (; left && vleft--; i++, first=0) {
        if (write) {
            while (left) {
                char c;
                if(get_user(c, (char *) buffer))
                    return -EFAULT;
                if (!isspace(c))
                    break;
                left--; ((char *) buffer)++;
            }
        }
    }
}
```

Proc_dointvec_minmax(2)

```
.....
    if (!left)
        break;
    neg = 0;    len = left;
    if (len > TMPBUFLEN-1)
        len = TMPBUFLEN-1;
    if(copy_from_user(buf, buffer, len))
        return -EFAULT;
    buf[len] = 0;    p = buf;
    if (*p == '-' && left > 1) {
        neg = 1;    left--, p++;
    }
    if (*p < '0' || *p > '9')
        break;
    val = simple_strtoul(p, &p, 0);
    len = p-buf;
    if ((len < left) && *p && !isspace(*p))
        break;
    if (neg)
        val = -val;
    buffer += len;    left -= len;

    if (min && val < *min++)
        continue;
    if (max && val > *max++)
        continue;
    *i = val;
```

Proc_dointvec_minmax(3)

```
.....
    } else {
        p = buf;
        if (!first)
            *p++ = '\t';
        sprintf(p, "%d", *i);
        len = strlen(buf);
        if (len > left)
            len = left;
        if(copy_to_user(buffer, buf, len))
            return -EFAULT;
        left -= len;    buffer += len;
    }
}
if (!write && !first && left) {
    if(put_user('\n', (char *) buffer))
        return -EFAULT;
    left--, buffer++;
}
if (write) {
    p = (char *) buffer;
    while (left) {
        char c;
        if(get_user(c, p++))
            return -EFAULT;
        if (!isspace(c))
            break;
        left--;
    }
}
if (write && first)
    return -EINVAL;
*lenp -= left;
filp->f_pos += *lenp;
return 0;
}
```

Do_sysctl

```
int do_sysctl(int *name, int nlen, void *oldval, size_t *oldlenp, void *newval, size_t newlen)
{
    struct list_head *tmp;
    if (nlen <= 0 || nlen >= CTL_MAXNAME)
        return -ENOTDIR;
    if (oldval) {
        int old_len;
        if (!oldlenp || get_user(old_len, oldlenp))
            return -EFAULT;
    }
    tmp = &root_table_headerctl_entry;
    do {
        struct ctl_table_header *head = list_entry(tmp, struct ctl_table_header, ctl_entry);
        void *context = NULL;
        int error = parse_table(name, nlen, oldval, oldlenp, newval, newlen, head->ctl_table, &context);
        if (context)                kfree(context);
        if (error != -ENOTDIR)      return error;
        tmp = tmp->next;
    } while (tmp != &root_table_headerctl_entry);
    return -ENOTDIR;
}
```

do_sysctl() → parse_table()

Parse_table

```
static int parse_table(int *name, int nlen, void *oldval, size_t *oldlenp,
                      void *newval, size_t newlen, ctl_table *table, void **context)
{
    int n;
repeat:
    if (!nlen)                return -ENOTDIR;
    if (get_user(n, name))    return -EFAULT;
    for ( ; table->ctl_name; table++) {
        if (n == table->ctl_name || table->ctl_name == CTL_ANY) {
            int error;
            if (table->child) {
                if (ctl_perm(table, 001))
                    return -EPERM;
                if (table->strategy) {
                    error = table->strategy(table, name, nlen, oldval, oldlenp, newval, newlen, context);
                    if (error)                return error;                }
                name++;    nlen--;    table = table->child; goto repeat;    }

            error = do_sysctl_strategy(table, name, nlen, oldval, oldlenp, newval, newlen, context);
            return error;    }
    }
    return -ENOTDIR;    }
```

do_sysctl() → parse_table() → do_sysctl_strategy()

Do_sysctl_strategy

/ Perform the actual read/write of a sysctl table entry. */*

```
int do_sysctl_strategy (ctl_table *table, int *name, int nlen, void *oldval, size_t *oldlenp, void *newval, size_t newlen, void **context)
```

```
{
    int op = 0, rc;
    size_t len;

    if (oldval)    op |= 004;
    if (newval)    op |= 002;
    if (ctl_perm(table, op)) return -EPERM;
    if (table->strategy) {
        rc = table->strategy(table, name, nlen, oldval, oldlenp, newval, newlen, context);
        if (rc < 0)    return rc;
        if (rc > 0)    return 0;    }

/* If there is no strategy routine, or if the strategy returns zero, proceed with automatic r/w */
    if (table->data && table->maxlen) {
        if (oldval && oldlenp) {
            get_user(len, oldlenp);
            if (len) {
                if (len > table->maxlen)    len = table->maxlen;
                if(copy_to_user(oldval, table->data, len)) return -EFAULT;
                if(put_user(len, oldlenp)) return -EFAULT;    }
            }
        if (newval && newlen) {
            len = newlen;
            if (len > table->maxlen) len = table->maxlen;
            if(copy_from_user(table->data, newval, len)) return -EFAULT; }
        }
    }
    return 0;
}
```

Register_sysctl_table

```
struct ctl_table_header *register_sysctl_table(ctl_table * table,
                                              int insert_at_head)
{
    struct ctl_table_header *tmp;
    tmp = kmalloc(sizeof(struct ctl_table_header), GFP_KERNEL);
    if (!tmp)
        return NULL;
    tmp->ctl_table = table;
    INIT_LIST_HEAD(&tmp->ctl_entry);
    if (insert_at_head)
        list_add(&tmp->ctl_entry, &root_table_header.ctl_entry);
    else
        list_add_tail(&tmp->ctl_entry, &root_table_header.ctl_entry);
#ifdef CONFIG_PROC_FS
    register_proc_table(table, proc_sys_root);
#endif
    return tmp;
}
```

Unregister_sysctl_table

```
void unregister_sysctl_table(struct ctl_table_header * header)
{
    list_del(&header->ctl_entry);
#ifdef CONFIG_PROC_FS
    unregister_proc_table(header->ctl_table, proc_sys_root);
#endif
    kfree(header);
}
```

Sysctl_string

```
int sysctl_string(ctl_table *table, int *name, int nlen, void *oldval, size_t *oldlenp,
                 void *newval, size_t newlen, void **context)
{
    size_t l, len;

    if (!table->data || !table->maxlen)                return -ENOTDIR;

    if (oldval && oldlenp) {
        if (get_user(len, oldlenp)) return -EFAULT;
        if (len) {
            l = strlen(table->data);
            if (len > l) len = l;
            if (len >= table->maxlen)    len = table->maxlen;
            if (copy_to_user(oldval, table->data, len)) return -EFAULT;
            if (put_user(0, ((char *) oldval) + len))    return -EFAULT;
            if (put_user(len, oldlenp))    return -EFAULT;    }
        }
    if (newval && newlen) {
        len = newlen;
        if (len > table->maxlen)                len = table->maxlen;
        if (copy_from_user(table->data, newval, len)) return -EFAULT;
        if (len == table->maxlen)    len--;
        ((char *) table->data)[len] = 0; }
    return 0;
}
```

Sysctl_intvec

```
int sysctl_intvec(ctl_table *table, int *name, int nlen, void *oldval, size_t
    *oldlenp, void *newval, size_t newlen, void **context)
{
    int i, *vec, *min, *max;
    size_t length;

    if (newval && newlen) {
        if (newlen % sizeof(int) != 0)        return -EINVAL;
        if (!table->extra1 && !table->extra2)  return 0;
        if (newlen > table->maxlen)           newlen = table->maxlen;
        length = newlen / sizeof(int);
        vec = (int *) newval;
        min = (int *) table->extra1;
        max = (int *) table->extra2;
        for (i = 0; i < length; i++) {
            int value;
            get_user(value, vec + i);
            if (min && value < min[i])        return -EINVAL;
            if (max && value > max[i])        return -EINVAL;
        }
    }
    return 0;
}
```

Usage of /proc file system(1)

```
create_proc_read_entry("scullmem",
                       0 /* default mode */,
                       NULL /* parent dir */,
                       scull_read_procmem,
                       NULL /* client data */);

int scull_read_procmem(char *buf, char **start, off_t offset,
                      int count, int *eof, void *data)
{
    int i, j, len = 0;
    int limit = count - 80; /* Don't print more than this */

    for (i = 0; i < scull_nr_devs && len <= limit; i++) {
        Scull_Dev *d = &scull_devices[i];
        if (down_interruptible(&d->sem))
            return -ERESTARTSYS;
        len += sprintf(buf+len, "\nDevice %i: qset %i, q %i, sz %li\n",
                       i, d->qset, d->quantum, d->size);
        for (; d && len <= limit; d = d->next) { /* scan the list */
```

Usage of /proc file system(2)

```
len += sprintf(buf+len, "  item at %p, qset at %p\n", d,
                  d->data);
if (d->data && !d->next) /* dump only the last item
                        - save space */
    for (j = 0; j < d->qset; j++) {
        if (d->data[j])
            len += sprintf(buf+len, "    % 4i: %8p\n",
                              j, d->data[j]);
    }
    }
    up(&scull_devices[i].sem);
}
*eof = 1;
return len;
}
```

Reference

- | **Linux Core Kernel Commentary**
second edition
- | **Understanding the LINUX KERNEL**
O'reilly
- | **Cross-Referencing Linux**
 - <http://lxr.linux.no/>
- | **Linux Device Drivers, 2nd Edition**
 - <http://www.xml.com/ldd/chapter/book/index.html>, chapter 4