Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Prototype of the basic version of LinuxSSI
D2.2.7

Due date of deliverable: November 31^{th}, 2007
Actual submission date: January 10^{th}, 2007

Start date of project: June 1^{st} 2006

Type: Deliverable
WP number: WP2.2
Task number: T2.2.1

Responsible institution: INRIA
Editor & and editor's address: Matthieu Fertré
IRISA/INRIA
Campus de Beaulieu
35042 RENNES Cedex
France

Version 1.0 / Last edited by Matthieu Fertré / January 9^{th}, 2008

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	18/10/07	Matthieu Fertré	INRIA	Initial template
0.2	19/10/07	Matthieu Fertré	INRIA	Initial text for User guide chapter
0.3	19/10/07	Matthieu Fertré	INRIA	Initial text for installation chapter
0.4	19/10/07	Matthieu Fertré	INRIA	Initial text for prototype description
0.5	22/10/07	Matthieu Fertré	INRIA	Add NFSROOT documentation link in installation chapter, precise version of Kerrighed latest stable release in prototype description chapter
0.6	22/10/07	Matthieu Fertré	INRIA	User documentation about node(s) addition, removal.
0.7	23/10/07	Matthieu Fertré	INRIA	Introduction text
0.8	23/10/07	Matthieu Fertré	INRIA	Executive summary
0.9	26/10/07	John Mehnert-Spahn	UDUS	User documentation about checkpointing
0.10	26/10/07	Marko Novak	XLAB	User's documentation for customizable scheduler
0.11	29/10/07	Adrien Lebre	INRIA	kDFS documentation and minor typos
0.12	12/12/07	Christine Morin	INRIA	executive summary, introduction and conclusion
0.13	19/12/07	Matthieu Fertré	INRIA	comments from Jerome Robert
1.0	09/01/08	Matthieu Fertré	INRIA	comments from Ana Oprescu

Reviewers:
Jérôme Robert (EADS) and Ana Oprescu (VUA)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.2.1	Federation Management	INRIA*, XLAB, UDUS, NEC, SAP, ICT

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

LinuxSSI-XOS is the foundation layer of the XtreemOS cluster flavour. LinuxSSI-XOS is a standard Linux kernel modified in two ways: first, to incorporate the modifications related to VO support (the same kind of modifications as those used to build Linux-XOS[2]) and second, to integrate distributed resource management services to provide a single system image (i.e. modifications related to LinuxSSI).

This document relates to the prototype of LinuxSSI delivered at M18 (November 2007) and based on the specifications described in [2]. LinuxSSI leverages the existing Kerrighed SSI technology developed by INRIA in cooperation with EDF [9, 10, 11, 12, 14]. For XtreemOS-G services, a cluster executing LinuxSSI-XOS will be seen as a powerful PC executing Linux-XOS.

LinuxSSI leverages Kerrighed version 2.2.0 by offering the following functionalities: a distributed file system, kDFS, exploiting disks attached to cluster nodes [7], a customizable scheduler [3], a reconfiguration framework allowing a system administrator to add or stop nodes without rebooting the whole cluster [6], a mechanism to checkpoint/restart individual processes and parallel applications based on the shared memory communication paradigm [5].

This deliverable explains how to install and configure a LinuxSSI cluster. With the current prototype, NFSRoot, which allows to deploy a cluster on nodes in a diskless way, needs to be enabled. Then, installing LinuxSSI requires the configuration of a TFTP server, a DHCP server, and an NFS server. Moreover, the system image needs to be configured for diskless nodes. The whole installation process is described. This document also provides a user guide for the functionalities specific to LinuxSSI (not included in Kerrighed official release V2.2.0). In particular, we explain how to format and mount a kDFS partition, how to load and unload a probe or a scheduling policy based on the Pluggable Probes and Scheduling Policies (PlugProPol) framework. We also present the commands available for the system administrator to add or remove a cluster node without stopping the whole cluster. Finally, we present the checkpoint/restart command-line API.

Based on this first prototype of LinuxSSI, the next steps in our future work are three-folds. Our first priority for the coming months is to integrate LinuxSSI with Linux-XOS mechanisms to get the first LinuxSSI-XOS prototype. We also need to finalize some functionalities. Then we plan to implement advanced features as described in D2.2.2, D2.2.3, D2.2.4, D2.2.5, D2.2.6 deliverables [8, 5, 6, 7, 3].

The prototype described in this document results from the joint work carried out in the framework of the WP2.2 workpackage by INRIA, NEC, XLAB, UDUS, ICT and SAP and from discussion with other partners and with key developers from the Kerrighed open source community (external to XtreemOS consortium). INRIA, NEC, XLAB, UDUS, contributed to the implementation of LinuxSSI prototype.

Contents

Chapter 1

Introduction

LinuxSSI-XOS is the foundation layer of the XtreemOS cluster flavour. LinuxSSI-XOS is a standard Linux kernel modified in two ways: first to incorporate the modifications related to the VO support (the same kind of modifications as those used to build Linux-XOS and described in [2]) and secondly to integrate distributed resource management services to provide a single system image (that is to say modifications related to LinuxSSI).

This document relates to the prototype of LinuxSSI delivered at M18 (November 2007) and based on the specifications described in [2]. LinuxSSI leverages the existing Kerrighed SSI technology developed by INRIA in cooperation with EDF [9, 10, 11, 12, 14]. A Linux SSI operating system provides the illusion that a cluster is a virtual multiprocessor machine executing Linux. For XtreemOS-G services, a cluster executing LinuxSSI-XOS will be seen as a powerful PC executing Linux-XOS.

The prototype described in this document results from the joint work carried out in the framework of the WP2.2 workpackage by INRIA, NEC, XLAB, Düsseldorf University, ICT and SAP and from discussion with other partners and with key developers from the Kerrighed open source community (external to XtreemOS consortium). INRIA, NEC, XLAB, Düsseldorf University contributed to the implementation of LinuxSSI prototype.

This deliverable is organized as follows. First we present in Section 2 a brief description of the functionalities implemented in LinuxSSI current prototype. These functionalities are essentially a distributed file system exploiting disks attached to cluster nodes [7], a customizable scheduler [3], a reconfiguration framework allowing a system administrator to add or stop nodes without rebooting the whole cluster [6], mechanisms to checkpoint/restart processes and parallel applications based on the shared memory communication paradigm [5]. In Section 3, we present how to install and configure the current LinuxSSI prototype. In Section 4, a user guide is provided for the functionalities specific to LinuxSSI. Section 5 concludes summarizing the current state of LinuxSSI and stating our priorities in LinuxSSI implementation plan.

Chapter 2

Brief description of the prototype

LinuxSSI operating system provides the illusion that a cluster is a virtual multiprocessor machine executing Linux. For XtreemOS-G services, a cluster executing LinuxSSI-XOS will be seen as a powerful PC executing Linux-XOS. Thus, LinuxSSI-XOS is a standard Linux kernel modified in two ways: firstly to incorporate the modifications related to the VO support and secondly to integrate distributed resource management services to provide a single system image.

This prototype of LinuxSSI does not include VO support but focus on services needed for single system image. It is based on the latest development version of Kerrighed operating system with additional features developed in the framework of the WP2.2 by INRIA, NEC, XLAB, Düsseldorf University, ICT and SAP. Kerrighed technology has been originally developed by INRIA in collaboration with EDF and is now maintained by Kerlabs and the Kerrighed community.

In addition to the latest stable release of Kerrighed operating system (2.2.1), LinuxSSI 1.0 provides

- support of IPC semaphores and IPC message queues;
- the beta version of KDFS: a distributed file system;
- a global customizable scheduler;
- experimental checkpoint/restart of application with shared memory;
- experimental reconfiguration mechanisms allowing administrator to add or remove node(s) in the cluster.

Chapter 3

Installation and configuration notes

The whole installation and configuration procedure in this document is based on Kerrighed installation manual which can be found on:

http://www.kerrighed.org/wiki/index.php/UserDoc.

Every user who wants to install LinuxSSI is advised to first check Kerrighed documentation since installation procedure of LinuxSSI is very similar to the one of Kerrighed.

3.1 Enabling NFSRoot

Before we can start compiling and configuring LinuxSSI, we have to establish the infrastructure for booting computers via network (so-called "network boot" or "net boot"). Since version 2.1.0 of Kerrighed, the preferred way is to use cluster with diskless nodes for running LinuxSSI. This kind of cluster consists of a single server which has an image of LinuxSSI system stored on its hard disk and multiple clients. These clients don't have any operating system installed. Instead, they receive their system image from the server via network boot. This is accomplished by using so-called "NFSRoot" technique, which basically means that we are running an operating system with its root file system (i.e. "/" in Linux) mounted via NFS (Network File System).

The procedure which we are describing below is based on "Kerrighed on NFSROOT" instructions, which can be found on [1]. (They are the best place to look when trying to establish NFSRoot infrastructure). In order to enable NFSRoot, you have to setup three server daemons installed on your server:

- DHCP (Dynamic Host Configuration Protocol): this daemon assigns IP's to diskless clients and initiates network boot sequence. Under Debian, it is deployed by installing *dhcp3-server* package.

- TFTP (Trivial File Transfer Protocol): it allows clients to download configuration, kernels, ramdisks, etc. Under Debian, it is deployed by installing *tftpd-hpa* package.

- NFS: it is needed so diskless clients can mount their root file systems to the server. Under Debian, it is deployed by installing *nfs-kernel-server* package.

- BIND (Domain Name System (DNS) server): it is needed if you are planning to put LinuxSSI cluster on a sub-network which is separated from the rest of the company LAN.

3.1.1 Configuring TFTP server

To configure *tftpd-hpa* server, edit the **/etc/default/tftpd-hpa** file. If the file doesn't exist, create it:

```
# Default for tftpd-hpa
RUN_DAEMON="yes"
OPTIONS="-l -s /var/lib/tftpboot"
```

The "/var/lib/tftpboot" directory is the place on your server where you will store kernel, ramdisk and GRUB bootloader configuration files for your diskless clients.

Note: in order to enable network support already at boot time, you have recompile GRUB bootloader of your diskless clients to include drivers for their network cards. Precompiled versions of GRUB for various network cards can be found on Kerrighed web pages (http://kerrighed.gforge.inria.fr/download/grub/). There, each "pxegrub" file has a name of a network driver which is included into it, as its suffix. Patches which include support for some of the network cards into GRUB can also be found on GRUB mailing lists.

3.1.2 Configuring DHCP server

Below, an example of **/etc/dhcp3/dhcpd.conf** DHCP server configuration file is shown:

```
# DHCP server settings for LinuxSSI
#
### PART 1
# GRUB magic
option grub-menu code 150 = string;
# General options
option dhcp-max-message-size 2048;
use-host-decl-names on;
deny unknown-clients;
deny bootp;
### PART 2
option domain-name "xlabcluster.lan";
option ntp-servers ntp.network.net;
### PART 3
subnet 192.168.1.0 netmask 255.255.255.0 {
  option routers 192.168.1.1;
  option subnet-mask 255.255.255.0;
  option broadcast-address 192.168.1.255;
  option domain-name-servers 192.168.1.1;
  range 192.168.1.100 192.168.1.254;
  default-lease-time 21600;
  max-lease-time 43200;
}
### PART 4
group {
  filename "/pxegrub.via";
  option grub-menu = concat("(nd)/grub/", _
    host-decl-name);
  option root-path "/NFSROOT/LinuxSSI";
  host worker1 { fixed-address 192.168.1.101; _
    hardware ethernet 00:04:61:AA:6E:6E; _
    next-server 192.168.1.1; }
  host worker2 { fixed-address 192.168.1.102; _
    hardware ethernet 00:04:61:AA:70:95; _
    next-server 192.168.1.1; }
}
```

Part 1 of the configuration contains some basic settings for DHCP server and GRUB bootloader and you should leave them unchanged. In part 2, the domain name for the local subnet and Network Time Protocol (NTP) server are set. Here, it is sensible that you set only domain name and leave NTP server unchanged.

The part 3 contains configuration of a sub-network on which LinuxSSI cluster will be located. By setting it you prevent your DHCP server to interfere with other

parts of LAN (i.e. computers on the local network that don't belong to LinuxSSI cluster). If you have special DHCP server reserved just for Kerrighed, you generally don't want this server to assign IP addresses to the computers that don't belong to LinuxSSI cluster. Here, you specify the IP addresses of the gateway and DNS server for your local subnetwork.

In part 4, you should list all the computers that are a part of your LinuxSSI cluster. For each computer, you should specify a MAC address of its network card (i.e. *hardware ethernet*, for some motherboards you can find MAC addresses in BIOS by choosing the following sequence of options: *Integrated Peripherals* → *Onboard devices* → *VIA Lan MAC Address Input*), IP that should be assigned to it (i.e. *fixed-address*) and a server which will be in charge of providing system image for network booting (i.e.*next-server*, this is usually the DHCP server itself). This section also contains:

- a path to the PXE-enabled GRUB bootloader (i.e. *filename*). This path is relative to the TFTP daemon root (TFTP daemon root is set in /etc/default/tftp-hpa). To create a PXE-enabled GRUB bootloader, you have to compile it with "–enable-diskless" configuration switch enabled and with proper network card driver included. You can also get a precompiled version of PXE-enabled GRUB's from Kerrighed web page, as described above,
- a path to GRUB menu for each diskless node (i.e. *grub-menu*). This path is also relative to TFTP daemon,
- an absolute path to the root node of the system image (i.e. *root-path*). All the diskless nodes in LinuxSSI share the same system image.

3.1.3 Configuring NFS server

Here, we only determine which part of the system will be shared in which way. Most of it will be shared as read-only, except for "/tmp", "/var", "/dev" and "/root", which will be shared as read-write (we will be able to read and write to those directories). Edit file **/etc/exports** to add the following lines:

```
/NFSROOT/LinuxSSI *(ro,async,no_root_squash, \
    no_subtree_check)
/NFSROOT/LinuxSSI/tmp *(rw,sync,no_root_squash, \
    no_subtree_check)
/NFSROOT/LinuxSSI/var *(rw,sync,no_root_squash, \
    no_subtree_check)
/NFSROOT/LinuxSSI/dev *(rw,sync,no_root_squash, \
    no_subtree_check)
/NFSROOT/LinuxSSI/root *(rw,sync,no_root_squash, \
    no_subtree_check)
```

Once you have configured DHCP, TFTP and NFS servers, restart them to apply modifications:

```
# /etc/init.d/dhcp3-server restart
# /etc/init.d/tftpd-hpa restart
# /etc/init.d/nfs-kernel-server restart
```

3.1.4 Create and configure system image for diskless nodes

The next step is to install image of the system which will be used by diskless nodes. Below, you can see a list of commands that have to be executed in order to accomplish that:

```
# mkdir /NFSROOT
# debootstrap sid /NFSROOT/LinuxSSI \
    http://ftp.debian.org/debian
# chroot /NFSROOT/LinuxSSI
# mkdir /config
# passwd
# mount -t proc none /proc
# apt-get install dhcp3-common nfs-common nfsbooted
# apt-get install initramfs-tools
```

The most important command here is “debootstrap”. This is Debian-specific command, which takes care of downloading all Debian operating system from remote server and installs it into the specified directory. After successful installation, user has to chroot to the directory with the newly created image, set the root password and install all the necessary tools for performing network boot.

The next step is to edit the **/etc/fstab** file (note that this is actually the “/NFSROOT/LinuxSSI/etc/fstab” file, but since we changed root node to “/NFSROOT/LinuxSSI”, we must disregard that part of the path). Here you insert all the directories that should be mounted during boot time of every diskless node, including the ones that are mounted via NFS (here, we assume 192.168.1.1 is IP address of your NFS server).

```
# A swap partition
/dev/hda none swap sw 0 0
none /proc proc defaults 0 0
none /sys sysfs defaults 0 0
# the line below mounts ConfigFS file system which
# is needed by the PPSPF framework
none /config configfs defaults 0 0
#
# NFSROOT
# Following partitions are mounted in rw mode, for
# the moment, I have some troubles with lockd
# daemon that's why I added the 'nolock' params
192.168.1.1:/NFSROOT/LinuxSSI/dev /dev nfs \
  rw,hard,nolock 0 0
192.168.1.1:/NFSROOT/LinuxSSI/var /var nfs \
  rw,hard,nolock 0 0
192.168.1.1:/NFSROOT/LinuxSSI/tmp /tmp nfs \
  rw,hard,nolock 0 0
192.168.1.1:/NFSROOT/LinuxSSI/root /root nfs \
  rw,hard,nolock 0 0
#
# TMPFS
none /var/run tmpfs defaults     0 0
```

Create symbolic link to "/etc/network/if-up.d/mountnfs" script in order to automatically mount all the NFS-shared directories. By default, this mounting is performed when network interface is set up. But since the network interface is enabled from the beginning, when performing network boot, the scripts in "/etc/network/if-up.d" are never executed. To solve this issue we have to create a symbolic link in "/etc/rcS.d" directory.

```
# ln -sf ../network/if-up.d/mountnfs \
    /etc/rcS.d/S35mountnfs
```

Finally, include IP's and hostnames of all the computers that are members of LinuxSSI cluster (the server as well as all the diskless clients) in file **/etc/hosts**:

```
127.0.0.1 localhost.localdomain localhost
192.168.1.1 server.xlabcluster.lan server
192.168.1.101 worker1.xlabcluster.lan worker1
192.168.1.102 worker2.xlabcluster.lan worker2
```

3.2 LinuxSSI source download

Once we have establish the NFSRoot infrastructure we can start downloading LinuxSSI source code. We recommend source code to be stored somewhere inside system image for diskless nodes (i.e. the "/NFSROOT/LinuxSSI" directory tree), since this is the place where we will later perform LinuxSSI installation. So before you start downloading, do a chroot to "/NFSROOT/LinuxSSI" directory.

Before you start downloading, create a directory into which you will download sources (for example "/root/LinuxSSI-src") and move into it:

```
# mkdir /root/LinuxSSI-src
# cd /root/LinuxSSI-src
```

Download LinuxSSI source code from INRIA XtreemOS Gforge and copy it in the current directory. The archive is available at the following address:

https://gforge.inria.fr/frs/download.php/3668/linuxssi-0.9-alpha.tar.gz

Download Linux 2.6.20 kernel source code from *kernel.org* or a valid mirror:

```
# wget http://www.kernel.org/pub/linux/kernel/v2.6/\
    linux-2.6.20.tar.bz2
```

Decompress the Tarballs:

```
# tar xzf linuxssi-0.9-alpha.tar.gz
# tar xjf linux-2.6.20.tar.bz2
```

3.3 LinuxSSI installation

3.3.1 Prerequisites

In order to successfully compile LinuxSSI source code you must have the following Linux tools installed:

- automake (version 1.9 or newer),
- autoconf (version 2.59 or newer),
- libtool,
- rsync,
- pkg-config,

- gcc (version 3.3.x is recommended, although LinuxSSI compiles also with version 4.1)

The tools below are not mandatory, however it's good to install them as well:

- lsb-release: installs LinuxSSI startup scripts into /etc/init.d. These scripts are used for automatically load LinuxSSI module at boot time,
- xmlto: a tool to create man pages of LinuxSSI commands.

Note: since we will be installing LinuxSSI in the system image for diskless nodes, please make sure that you have chrooted to the "/NFSROOT/LinuxSSI" directory before you start installing all the tools mentioned above.

3.3.2 Installation

Below, you can find a list of commands that have to be executed in order to perform installation of LinuxSSI. First, you go to directory where you uncompressed LinuxSSI source files:

```
# cd /root/LinuxSSI-src
```

Next, configure the sources:

```
# cd linuxssi-0.9-alpha
# ./configure --with-kernel=/root/LinuxSSI-src/\
    linux-2.6.20
```

Next, patch the Linux kernel source with LinuxSSI:

```
# make patch
# make defconfig
```

Next, check the configuration of the Linux kernel (network card driver for instance):

```
# cd  /root/LinuxSSI-src/linux-2.6.20
# make menuconfig
# cd -
```

Next, build the sources and install them as root:

```
# make kernel
# make
# make kernel-install
# make install
```

This list of commands should successfully install LinuxSSI to your system image for diskless nodes. In case you have some problems, you should check Kerrighed mailing lists (http://www.kerrighed.org/wiki/index.php/Contact) and bug tracker (http://gforge.inria.fr/tracker/?group_id=69), where you could find a solution.

If the installation was successful, you should have 3 files in "/boot" directory of your system image for diskless nodes:

- vmlinuz-2.6.20-krg
- System.map-2-6-20-krg
- config-2.6.20

In the same directory, you should also create a RAM disk for the new kernel. This is an initial root file system that is mounted before the real root file system is available. It is loaded as a part of kernel boot procedure. Here is how to create it:

```
# cd /boot
# mkinitramfs -o initrd.img-2.6.20-krg 2.6.20-krg
```

When this is done, exit from the chrooted environment and copy the 4 files to the root directory of your TFTP server:

```
# exit
# cp /NFSROOT/LinuxSSI/boot/vmlinuz-2.6.20-krg \
    /var/lib/tftpboot
# cp /NFSROOT/LinuxSSI/boot/System.map-2.6.20-krg \
    /var/lib/tftpboot
# cp /NFSROOT/LinuxSSI/boot/config-2.6.20-krg \
    /var/lib/tftpboot
# cp /NFSROOT/LinuxSSI/boot/initrd.img-2.6.20-krg \
    /var/lib/tftpboot
```

3.3.3 Configuration

After we have successfully installed LinuxSSI, we have to make a "/var/lib/tftpboot/grub" directory. In it, we create a separate GRUB menu file for each diskless node (note:

filename of the grub file has to match the hostname of the corresponding node). GRUB file for "worker1" node (i.e. "/var/lib/tftpboot/grub/worker1" file) is shown as an example:

```
default 0
timeout 5

title LinuxSSI
root (nd)
kernel /vmlinuz-2.6.20-krg root=/dev/nfs ip=dhcp \
   nfsroot=192.168.1.1:/NFSROOT/LinuxSSI node_id=0 \
   session_id=0
initrd /initrd.img-2.6.20-krg
```

Other nodes' GRUB files are very similar to the one above. They differ only in "node_id" boot parameter, which represents LinuxSSI node ID. This has to be unique for every node in the cluster.

Once you have LinuxSSI properly installed, boot all the diskless nodes via network. Log to each of the nodes and check that LinuxSSI module was successfully loaded on it. This is done by using "lsmod" command. If LinuxSSI module is loaded on a particular node, its name will be present in the list of modules lsmod prints out. If this is not the case, you can still load it by invoking **modprobe kerrighed** command.

After you have made sure LinuxSSI module is loaded on all the diskless nodes, you can start LinuxSSI cluster by invoking **krgadm cluster start** command on one of the nodes.

Chapter 4

User's guide

4.1 Overview about using LinuxSSI

For basic usage of LinuxSSI, you can refer to Kerrighed documentation:

- Kerrighed User Manuel:
  http://www.kerrighed.org/wiki/index.php/V2.1.0_User_Manual
- Man pages of Kerrighed commands that can be consulted either with `man command` like for any Unix command or on the web page:
  http://www.kerrighed.org/wiki/index.php/UserDoc#Man_pages

4.2 Using advanced features of LinuxSSI

The following features are only available in LinuxSSI.

4.2.1 Using kDFS

The *kernel Distributed File System* aims at federating storage resources within a cluster in order to provide an integrated cluster file system for High Performance Computing. Design, implementation and current status of the first prototype is addressed in a distinct deliverable [7]. This section relates procedures to exploit a kDFS file system. The first paragraph addresses the `mkfs.kdfs` command, the second focuses on mounting a kDFS partition.
Note: Kerrighed has to be started to set/exploit a kDFS structure.

Formatting a kDFS partition

Associated to kDFS, a dedicated command, mkfs.kdfs has been implemented. This command formats a directory which can be used afterward in the kDFS physical structure. This command is released with the current kDFS prototype. Its syntax is the following one:

```
mkfs.kdfs DIRECTORY_PATHNAME ROOT_NODEID

DIRECTORY_PATHNAME: Absolute path to store kDFS meta-data an data.
ROOT_NODEID: kDFS root node id, localization of the kDFS root meta-data file.
```

`mkfs.kdfs` creates the kDFS "superblock" file for the node. This file is stored on the local native file system at `DIRECTORY_PATHNAME` if `KERRIGHED_NODEID` equals `ROOT_NODEID`, `mkfs.kdfs` creates the root meta-file which is the clusterwide entry point for the kDFS physical structure.

The table 4.1 describes creation of a kDFS structure distributed between two nodes:

On node A: (kerrighed nodeid = 1)	on Node B: (kerrighed nodeid = 2)
`mkfs.kdfs /KDFS 1`	`mkfs.kdfs /ANOTHER-KDFS 1`
Create kDFS local superblock Create kDFS root meta-file	Create kDFS local superblock

Table 4.1: Create a kDFS structure between two nodes

Mounting a kDFS partition

The `mount` command is the traditional one. We have still not extended it with specific kDFS parameters. Thus, for the moment, There are few limitations:

- One kDFS partition per node
- Only one mount point per node
- Diskless mechanisms not available

Users can mount a kDFS partition by the following command:

```
mount -t kdfs ALLOCATED_DIRECTORY MOUNT_POINT

ALLOCATED_DIRECTORY: native file system directory formatted with mkfs.kdfs
MOUNT_POINY: traditional mount point.
```

The table 4.2 describes kDFS mounting procedure from two nodes:

On node A: (kerrighed nodeid = 1)	on Node B: (kerrighed nodeid = 2)
`mount -t kdfs /KDFS /mnt/kfds`	`mount -t kdfs /ANOTHER-KDFS /mnt/kdfs`
`/mnt/kdfs` is now a global kDFS namespace for both nodes	

Table 4.2: Mount kDFS partitions

Finally, kDFS documentation is available on the Kerriged web site. Please visit http://www.kerrighed.org/wiki/index.php/KernelDevelKdFS

4.2.2 Taking advantage of the global scheduler

In the first implementation phase of XtreemOS project, we have implemented the Pluggable Probes and Scheduling Policies (PlugProPol) framework [4]. It is an infrastructure which enables user to write his own probes and scheduling policies and add them to the system in runtime (without the need to restart the whole cluster). If a user wants, for example, to monitor disk usage on his local machine, he only implements a proper probe and plugs it to PlugProPol in runtime. This makes the scheduling much more configurable since no reboot is needed.

All the probes and scheduling policies are implemented as Linux kernel modules, they run in kernel space and are able to access kernel data structures directly. No system calls are needed, which means such infrastructure induces less overhead than the one where probes and scheduling policies would be implemented in user space.

Since PlugProPol framework is based on ConfigFS file system, all its operations such as probes/scheduling policies loading and unloading, linking can be performed from user space using standard linux commands such as mkdir, rm, ln. This chapter contains a list of Linux commands which can be used with PlugProPol framework, along with their descriptions and examples of usage.

mkdir

Syntax: *mkdir <module_name>*

Arguments:

- *<module_name>*: name of the probe/scheduling policy module to load. This name must be equal to the name of the module file (the one with the ".ko" extension) which implements given probe/scheduling policy.

Description: by invoking this command a user requests PlugProPol framework to load a given probe or scheduling policy. In order to initiate probe loading, user has to invoke this command in */config/probes* directory. Similarly, he has to invoke the command in */config/schedulers* directory if he wants to load scheduling policy.

Probes and scheduling policies should be implemented as linux kernel modules [13] and should register themselves with PlugProPol framework in their initialization function. For an example of how to write a proper probe or scheduling policy see "mem_probe.c" and "echo_policy.c" in the Appendix. Once the probe/scheduling policy is loaded with PlugProPol the framework itself takes care of performing measurements (for the probes) and collecting them (for scheduling policies).

In order for a user to be able to load a particular probe or scheduling policy module with PlugProPol, he must first register it. This is done by inserting full

path to the module file (the one with the ".ko" extension) to the **/lib/modules/<kerrighed_kernel_version>/modules.dep** file, followed by a colon and full path to "kerrighed.ko" module file. Below you can see example entries for "mem_probe" and "echo_policy" (note that both path to the probe/scheduling module file and path to the "kerrighed.ko" file must be written in the same line for each entry):

```
/lib/modules/2.6.20-krgLinuxSSI/extra/mem_probe.ko: \
  /lib/modules/2.6.20-krgLinuxSSI/extra/kerrighed.ko

/lib/modules/2.6.20-krgLinuxSSI/extra/echo_policy.ko: \
  /lib/modules/2.6.20-krgLinuxSSI/extra/kerrighed.ko
```

Example:

```
# load mem_probe probe
mkdir /config/krg_scheduler/probes/mem_probe

# load echo_policy scheduling policy
mkdir /config/krg_scheduler/schedulers/test-scheduler
mkdir /config/krg_scheduler/schedulers/test-scheduler/\
    echo_policy
```

rmdir

Syntax:

- *rmdir* <*module_name*>
- *rm -rf* <*module_name*>

Arguments:

- <*module_name*>: name of the probe/scheduling policy to unload. This name must be equal to the name of the module file (the one with the ".ko" extension) which implements given probe/scheduling policy.

Description: this command is used for unloading probe or scheduling policy module that was loaded with mkdir command. Similarly to mkdir, the rmdir command has to be invoked invoked either in */config/probes* directory for probes or in */config/schedulers* directory for scheduling policies.

Example:

```
♯ unload mem_probe probe
rmdir /config/krg_scheduler/probes/mem_probe

♯ unload echo_policy scheduling policy
rmdir /config/krg_scheduler/schedulers/\
    test-scheduler/echo_policy
```

echo

Syntax: *echo* <*value*> > <*attribute_name*>

Arguments:

- <*value*>: the value which we want to assign to a given attribute. It can be given either in textual or in numerical form. If the value contains spaces it has to be surrounded by double quotes („).
- <*attribute_name*>: the name of the attribute to which we want to assign the value. The name must be given as an absolute path to the file which represents the attribute.

Description: this command assigns a value to the attribute specified by the attribute_name filename. Each attribute is represented by a separated file which is located in of the subdirectories in "/config/probes" or "/config/schedulers" directory. The command enables user to set probe and scheduling policy parameters in runtime and thus dynamically tune the scheduler.

Example:

```
♯ set probing period of mem_probe probe to 1 second.
echo 1000 > /config/krg_scheduler/probes/\
    mem_probe/probe_period
```

cat

Syntax: *cat* <*attribute_name*>

Arguments:

- <*attribute_name*>: the name of the attribute whose value we want to retrieve. The name must be given as an absolute path to the file which represents the attribute.

Description: this command retrieves a value of the attribute specified by the attribute_name filename. Each attribute is represented by a separated file which are located in of the subdirectories in "/config/probes" or "/config/schedulers" directory. The command enables user to get probe and scheduling policy parameters and is mostly used for reading properties of resources monitored by the probes.

Example:

```
♯ get total memory from mem_probe probe
cat /config/krg_scheduler/probes/mem_probe/\
    ram_total/value

♯ get total memory from echo_policy scheduling policy
cat /config/krg_scheduler/schedulers/test-scheduler/\
    echo_policy/port_mem_total/value
```

ln -s

source and sink must be directories (cannot link attributes).
Syntax: *ln -s* <*data_source*> <*data_sink*>

Arguments:

- <*data_source*>: source endpoint of the link which will provide us the data. This must be a directory (i.e. we cannot make symbolic links to attributes).
- <*data_sink*>: sink endpoint of the link which will consume the data. Like data_source, this must also be a directory (i.e. we cannot make symbolic links to attributes).

Description: by invoking the "ls -s" command, a user can connect one PlugProPol entity to another. By doing this he enables the "sink" entity to collect data from the "source" entity.

Only the connections that are defined below are possible:

- scheduling policy to probe,
- filter to probe,
- filter to another filter,
- scheduling policy to filter.

As soon as particular data source and sink are connected, PlugProPol framework takes care of data transfer between them.

Example:

```
# link port_mem_total endpoint of echo_policy
# scheduling policy to ram_total endpoint
# of mem_probe probe
ln -s /config/krg_scheduler/probes/mem_probe/ram_total \
    /config/krg_scheduler/schedulers/test_scheduler/\
    echo_policy/port_mem_total

# link port_mem_free endpoint of echo_policy scheduling
# policy to ram_free endpoint of mem_probe probe
ln -s /config/krg_scheduler/probes/mem_probe/ram_free \
    /config/krg_scheduler/schedulers/test_scheduler/\
    echo_policy/port_mem_free
```

4.2.3 Adding and removing node(s) in the cluster

Design, implementation and current status of the first prototype is addressed in a distinct deliverable [6].

The command used to manage node(s) addition and node(s) removal in the cluster is `krgadm`.

`krgadm` provides to the user a unique command to monitor the status of the cluster, to start a cluster, to make node(s) join or leave the cluster.

Nodes are identified either by their nodes' id, either by their MAC addresses.

Administrator can start a cluster with all the available nodes (Kerrighed module loaded) with the following command:

```
# krgadm cluster start
```

To add some nodes in a running cluster, for instance nodes 16 and 18, the following command must be used by the administrator:

```
# krgadm nodes add -n16:18
```

To remove some nodes in a running cluster, for instance nodes 21 and 34, the following command must be used by the administrator:

```
# krgadm nodes del -n21:34
```

More information can be found with `man krgadm` or `krgadm -h`. Some options described in `krgadm` manual or help are not yet implemented.

4.2.4 Checkpointing/Restarting application

Before the checkpoint and restart functionality can be used some prerequisites have to be met.

First, the directory */var/chkpt* must be created by the administrator with read and write permission enabled. After successfully taking a checkpoint, a directory will be created under the mentioned directory. The PID of the checkpointed process serves as directory name.

Second, a so called capability has to be set. Certain system features can be en/disabled with capabilities. Such a capability has to be switched on for checkpointing to work. The command for enabling the checkpointing functionality is:

```
# krgcapset -d +CHECKPOINTABLE
```

This command must be executed in the shell on which the process to be checkpointed will be started.

A checkpoint is created by issuing the following command:

```
# checkpoint PID
```

After successfully taking a checkpoint, a directory (checkpointed process' PID as directory name) should have been created under /var/chkpt. The following files will be created in this directory each including the serial number (SN) of the checkpoint in their name, e.g. '4711' for the file names mentioned below:

- *description_v4711.txt* (ascii file): short overview description of all files belonging to the checkpoint.
- *global_v4711.bin* (binary file): app_kddm_object KDDM object values as the applicationID, the serial number of the checkpoint and the kerrighed node mask.
- *node_5_v4711.bin* (binary file): description of local tasks involved in the checkpoint. The sample file belongs to node 5.
- *task_234_v4711.bin* (binary file): per task (e.g. PID='234') kernel structures such as registers, stack, signal mask, ...
- *task_mm_234_v4711.bin* (binary file): all pages of the process address space.

Repeated checkpointing of an application results in multiple files with the same base name but an SN increased by one at each time acheckpoint is taken. Checkpoints taken at different times can thus be distinguished.

An application is restarted by executing the following command:

```
# restart PID SN
```

Providing a SN allows to specify one out of many checkpoints taken during application execution.

For an application consisting of two or more processes, each of them having parent-child relationship, the PID of the common ancestor has to be provided to the restart command.

Chapter 5

Conclusion

LinuxSSI prototype described in this document is the result of the work in WP2.2 during the first 18 months of XtreemOS project. LinuxSSI has been developed based on Kerrighed technology in close collaboration with the key developers driving the Kerrighed open source community.

LinuxSSI leverages Kerrighed V2.2.0 extending it with the following features: kDFS, a distributed file system exploiting disks attached to cluster nodes, PlugProPol, a Pluggable Probes and Scheduling Policies framework allowing to dynamically load and unload scheduling policies without stopping the cluster, HotPlug, an infrastructure for the automatic reconfiguration of LinuxSSI distributed services when a system administrator requests a hot node addition or removal, a checkpoint/restart interface to store and restore the state of applications executed on top of LinuxSSI. These features, while being operational, should not be considered yet as production level software. XtreemOS consortium is actively pushing these features in the Kerrighed open source community to get them accepted in the Kerrighed mainstream development and integrated in Kerrighed releases. LinuxSSI can thus be considered as a research version of a Linux SSI operating system, Kerrighed being the production version. This document accompanying LinuxSSI software provides its installation and user manuals. LinuxSSI is packaged for RPM based Linux distributions in the framework of workpackage WP4.1.

Concerning LinuxSSI roadmap, one of our tasks is to improve the stability of the current LinuxSSI prototype. We will benefit from the feed-back of participants in WP4.2, who will experiment a number of applications on top of the current LinuxSSI prototype. In parallel, we plan in the very short term to integrate LinuxSSI and Linux-XOS mechanisms in order to build LinuxSSI-XOS, the foundation layer of the XtreemOS cluster flavour. We have started to adapt LinuxSSI kernel checkpointer to be used by the AEM service in XtreemOS-G. We have also started to design and implement LinuxSSI advanced features. Among these advanced features, we can cite the implementation of the DRMAA job submission interface on top of LinuxSSI, of file striping and redundancy in kDFS, of kDFS dynamic reconfiguration when the cluster configuration changes, of Infiniband high speed interconnect

support, of IPC object checkpoint/recovery, of reconfiguration mechanisms to allow LinuxSSI services to tolerate network link and node failures. We also plan to isolate KDDM mechanisms in order to push them in the mainstream Linux kernel development and to progressively remove limitations to LinuxSSI scalability.

As the collaboration with Kerrighed key developers has been very successful (avoiding duplication of work in particular) until now, so far we will continue to work in the same way in the following months for all the functionalities that we intend to push in Kerrighed mainstream development. This is essential for the exploitation of the LinuxSSI features in the future.

Bibliography

[1] Kerrighed on nfsroot. http://www.kerrighed.org/wiki/index.php/Kerrighed_on_NFSROOT.

[2] XtreemOS consortium. Specification of federation resource management mechanisms, November 2006.

[3] XtreemOS consortium. Design and implementation of a customizable scheduler. Deliverable D2.2.6, November 2007.

[4] XtreemOS consortium. Design and implementation of a customizable scheduler. Deliverable D2.2.6, November 2007.

[5] XtreemOS consortium. Design and implementation of basic checkpoint/restart mechanisms in linuxssi. Deliverable D2.2.3, November 2007.

[6] XtreemOS consortium. Design and implementation of basic reconfiguration mechanisms. Deliverable D2.2.4, November 2007.

[7] XtreemOS consortium. Design and implementation of high performance disk input/output operations in a federation. Deliverable D2.2.5, November 2007.

[8] XtreemOS consortium. Design and implementation of scalable mechanisms in linuxssi. Deliverable D2.2.2, November 2007.

[9] Pascal Gallard. *Conception d'un service de communication pour systèmes d'exploitation distribué pour grappes de calculateurs: mise en oeuvre dans le système à image unique Kerrighed*. Thèse de doctorat, IRISA, Université de Rennes 1, IRISA, Rennes, France, December 2004.

[10] Kerrighed website. http://www.kerrighed.org. http://www.kerrighed.org.

[11] Renaud Lottiaux. *Gestion globale de la mémoire physique d'une grappe pour un système à image unique : mise en œuvre dans le système Gobelins*. Thèse de doctorat, IRISA, Université de Rennes 1, December 2001.

[12] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, David Margery, Jean-Yves Berthou, and Isaac Scherson. Kerrighed and data parallelism: Cluster computing on single system image operating systems. In *Proc. of Cluster 2004*. IEEE, September 2004.

[13] O. Pomerantz P. Salzman, M. Burian. The linux kernel module programming guide. `http://www.tldp.org/LDP/lkmpg/2.6/html/book1.htm`, 2001.

[14] Geoffroy Vallée. *Conception d'un ordonnanceur de processus adaptable pour la gestion globale des ressources dans les grappes de calculateurs : mise en oeuvre dans le système d'exploitation Kerrighed*. Thèse de doctorat, IFSIC, Université de Rennes 1, France, March 2004.