ZD JOURNALS

Volume 6, Number 1 • January 2000
www.zdjournals.com/sun
US $9.00

Inside Solaris™

Tips & Techniques for users of Sun Solaris

A productive K desktop environment—KDE

by Rainer Dorsch

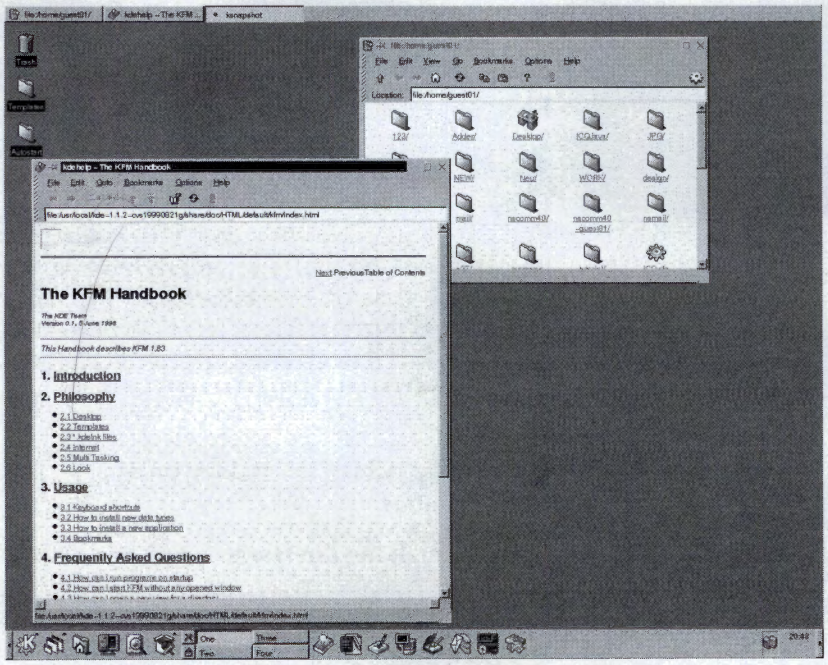
The K desktop environment (KDE) provides a desktop for UNIX similar to Microsoft Windows for PCs or MacOS for Apple computers. KDE can be configured completely by graphical interfaces, although the generated configuration files are human readable and can be edited with a simple editor. This feature makes Solaris much more attractive for users with no command line experience. In this article, well discuss KDE's many features, how to install KDE, some licensing issues, and much more.

KDE's many features

KDE provides many timely features, such as transparent network access, which is handy for both new and expert users. Further, we observed that many people use Solaris at work and Linux at home. Since KDE is the default desktop on many Linux distributions, the same user interface is available at work and at home. Although KDE is primarily developed under Linux, it supports many UNIX platforms, including Solaris.

KDE is now one of the largest open source projects, which was started in October 1996 by a small group of developers. GNU Public License (GPL) distributes KDE and it requires the QT library from Troll Tech, which can be downloaded (including source code) for free under the terms of the QT Public License (QPL).

The desktop shown in **Figure A**, can


kdehelp - The KFM Handbook
File Edit Goto Bookmarks Options Help
Location: file:/home/guest01/
Trash
Templates
Autostart
One
Two
Three
Four

file:/usr/local/kde-1.1.2-cvs19990821g/share/doc/HTML/default/kfm/index.html

Next PreviousTable of Contents

The KFM Handbook

The KDE Team
Version 0.1, 6 June 1998

This Handbook describes KFM 1.83

1. Introduction
2. Philosophy
   - 2.1 Desktop
   - 2.2 Templates
   - 2.3 *.kdelnk files
   - 2.4 Internet
   - 2.5 Multi Tasking
   - 2.6 Look
3. Usage
   - 3.1 Keyboard shortcuts
   - 3.2 How to install new data types
   - 3.3 How to install a new application
   - 3.4 Bookmarks
4. Frequently Asked Questions
   - 4.1 How can I run programs on startup
   - 4.2 How can I start KFM without any opened window

Figure A: *This is the main KDE desktop showing the KDE handbook.*

ZIFF-DAVIS a SOFTBANK company

spread out on several virtual screens and consists of three major elements:

- The *quick start bar* contains a K-button to start a program, which is comparable to Microsoft Windows' Start button. The appearance and features, such as the ability to automatically hide the bar, are configurable by a graphical user interface. Further, it allows you to switch between virtual screens.
- The *task bar* shows all running applications and allows you to switch between applications, even when running on different virtual screens.
- *Icons* also appear on the main desktop. These include icons such as trash and autostart.

KDE supports session management; therefore, when you shut down KDE, all KDE applications will be restarted and will be nearly in the same state as before shutting down. For example, in our case, it loaded the same files, but the cursor was at the beginning of the text. This feature must be supported by an application like the drag-and-drop protocol. So-called K-applications do this.

K-applications and KDE 2.0

Although KDE can run all X applications, K-applications support KDE goodies like the drag-and-drop protocol, session management, and a common user interface. The standard KDE distribution already contains many state-of-the-art tools, which are listed in Table A.

Besides the programs that come with the standard KDE distribution, there are many more K-programs available or under development. You'll find examples listed in Table B. At the time of writing, koffice was in alpha stage, and the first beta releases of kdeveloper were available.

Previous releases, starting from KDE 1.0 to 1.1.2, brought stability and small feature updates. The upcoming KDE 2.0 includes a transition to a CORBA-based communication infrastructure, allowing many new tightly interacting applications, like the components of koffice, the office tool of KDE, and konquerer, the new file manager and Web browser of KDE, embedding other applications to display Web data.

Table A: *The standard KDE distribution state-of-the-art tools*

Tool	Description
kfm	A file manager similar to Microsoft Explorer
kmail	A full-featured mail program
korganizer	A calendar tool compatible with Microsoft Outlook .vcs file format, as shown in Figure B
knews	A news reader
kedit	An editor
kview	A graphics display program
document previewers for postscript	Dvi and fax formats, shown in Figure C
kpaint	A simple graphics manipulation program
Network utilities	Shown in Figure D
A command line terminal	A window for entering interactive commands

Table B: *Other K-programs available or under development*

Tool	Description
koffice	A complete office program for KDE, containing a word processor, a spread sheet program, and a presentation program
kdevel	An integrated development environment, currently supporting the GNU development tools
StarOffice	An application that contains partial KDE support, like the drag-and-drop protocol

A sample session

Imagine you're looking for a tool to preview images and someone tells you that ghostview is the tool that you're looking for. You find that it's available at ftp://ftp.debian.de/debian/dists/stable/main/source/text/ghostview_1.5.orig.tar.gz. Traditionally, you would perform the following steps:

1. Ftp to ftp.debian.de, and log on as anonymous.
2. cd through the directories.
3. Download the archive file.
4. Check the table of contents of the downloaded file.

5. Extract the archive (with the obvious tar options).
6. Step into the archive.
7. Read the postscript manual by starting the imagetool program.
8. Finally, delete everything, because it wasn't what you expected.

With KDE, you're just a few clicks away from what you want:

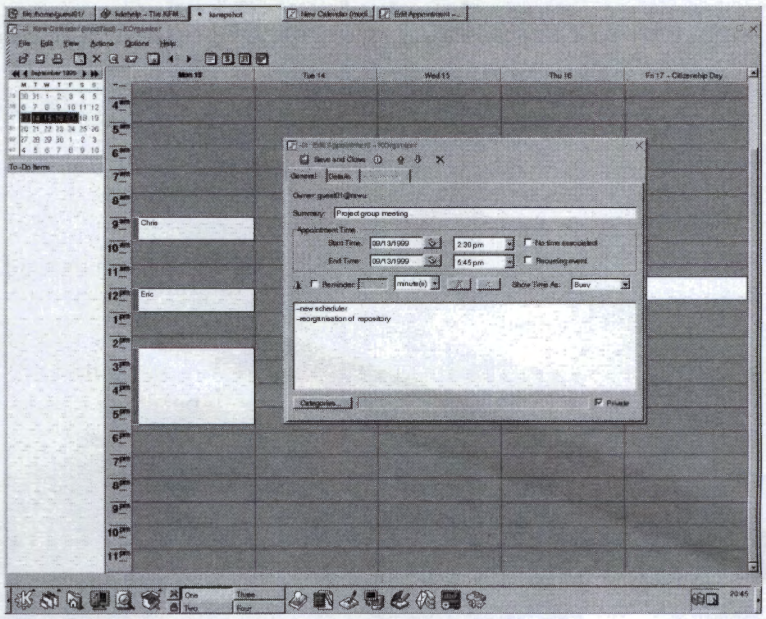
1. Open the URL in kfm (maybe you have it as bookmark).
2. Navigate though the ftp site, as it would be a local directory.
3. Step into the archive by clicking the archive.
4. Finally, click on the ps file, which opens a postscript previewer for you and displays the file.

If it isn't what you wanted, just close the postscript previewer. If you like it, drag it onto your desktop. **Figure E**, on the next page, shows the desktop with kfm and the postscript viewer. Note that this figure shows the fancy URL **ftp://ftp.debian.de/debian/dists/stable/main/source/text/ghostview_1.5.orig.tar.gz# tar:/ghostview-1.5.orig/ghostview.ps** in kfm, documenting that you loaded the file ghostview.ps, which is in the subdirectory hostview-1.5.orig in the ghostview subdirectory in the archive ghost-view_1.5.orig.tar.gz.

Installing KDE

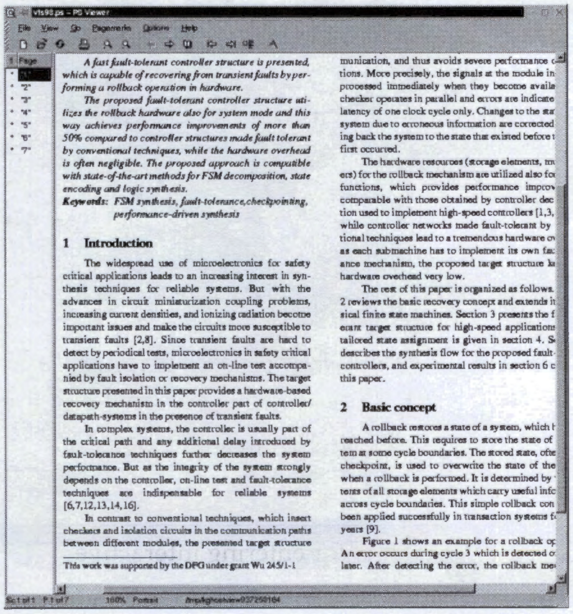
At the time of writing, KDE 1.1.1 is the latest official release. It's available as binary for Solaris (Intel and SPARC) and as source code. The release process of KDE 1.1.2 is nearly finished. KDE 2.0 is the current development version, which we didn't test. The only requirement to run KDE is a working version of the QT library. KDE 1.x requires QT 1.42 or 1.44. KDE 2.0 will require QT 2.0, which is licensed under the QPL open source license.

The simplest way to install KDE is to download the Solaris binary packages of QT and KDE from **ftp://ftp.kde.org/pub/kde/stable/latest/distribution/pkg/Solaris-2.6/Sparc/** (binaries for Solaris 7 are also available) or one of its mirrors from **www.kde.org/mirrors.html**, and install them using


Summary: Project group meeting
Start Time: 09/13/1999 2:30 pm
End Time: 09/13/1999 5:45 pm
Show Time As: Busy

-new scheduler
-reorganisation of repository

Figure B: *The KDE Organizer is editing an appointment.*


A fast fault-tolerant controller structure is presented, which is capable of recovering from transient faults by performing a rollback operation in hardware.

The proposed fault-tolerant controller structure utilizes the rollback hardware also for system mode and this way achieves performance improvements of more than 50% compared to controller structures made fault tolerant by conventional techniques, while the hardware overhead is often negligible. The proposed approach is compatible with state-of-the-art methods for FSM decomposition, state encoding and logic synthesis.

Keywords: *FSM synthesis, fault-tolerance, checkpointing, performance-driven synthesis*

1 Introduction

The widespread use of microelectronics for safety critical applications leads to an increasing interest in synthesis techniques for reliable systems. But with the advances in circuit miniaturization coupling problems, increasing current densities, and ionizing radiation become important issues and make the circuits more susceptible to transient faults [2,8]. Since transient faults are hard to detect by periodical tests, microelectronics in safety critical applications have to implement an on-line test accompanied by fault isolation or recovery mechanisms. The target structure presented in this paper provides a hardware-based recovery mechanism in the controller part of controller/datapath-systems in the presence of transient faults.

In complex systems, the controller is usually part of the critical path and any additional delay introduced by fault-tolerance techniques further decreases the system performance. But as the integrity of the system strongly depends on the controller, on-line test and fault-tolerance techniques are indispensable for reliable systems [6,7,12,13,14,16].

In contrast to conventional techniques, which insert checkers and isolation circuits in the communication paths between different modules, the presented target structure

This work was supported by the DFG under grant Wu 245/1-1

munication, and thus avoids severe performance c
tions. More precisely, the signals at the module in
processed immediately when they become availa
checker operates in parallel and errors are indicate
latency of one clock cycle only. Changes to the sta
system due to erroneous information are corrected
ing back the system to the state that existed before t
first occurred.

The hardware resources (storage elements, mu
ers) for the rollback mechanism are utilized also fo
functions, which provides performance improv
comparable with those obtained by controller dec
tion used to implement high-speed controllers [1,3,
while controller networks made fault-tolerant by
tional techniques lead to a tremendous hardware o
as each submachine has to implement its own fa
ance mechanism, the proposed target structure k
hardware overhead very low.

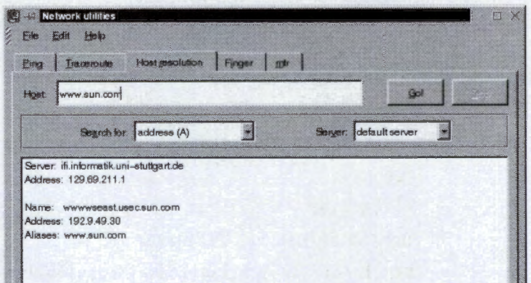
The rest of this paper is organized as follows.
2 reviews the basic recovery concept and extends i
sical finite state machines. Section 3 presents the f
erant target structure for high-speed applications
tailored state assignment is given in section 4. S
describes the synthesis flow for the proposed fault-
controllers, and experimental results in section 6 c
this paper.

2 Basic concept

A rollback restores a state of a system, which h
reached before. This requires to store the state of
tem at some cycle boundaries. The stored state, ofte
checkpoint, is used to overwrite the state of the
when a rollback is performed. It is determined by
tents of all storage elements which carry useful info
across cycle boundaries. This simple rollback con
been applied successfully in transaction systems f
years [9].

Figure 1 shows an example for a rollback op
An error occurs during cycle 3 which is detected o
later. After detecting the error, the rollback me

Figure C: *You can view postscript files with the document viewer.*


Host: www.sun.com
Search for: address (A)
Server: default server

```
Server:  ifi.informatik.uni-stuttgart.de
Address:  129.69.211.1

Name:   wwwwseast.usec.sun.com
Address:  192.9.49.30
Aliases:  www.sun.com
```

Figure D: *Here you can see a graphical representation of useful network tools.*

```
pkgadd -d qt-1.44-1.Solaris-2.6-Sparc.pkg
pkgadd -d kde-1.1.1-3.Solaris-2.6-Sparc.pkg
```

The script, which starts KDE, is placed in /usr/local/kde/bin/startkde. It's short and worth inspecting before starting KDE. The software is installed in /usr/local/qt and /usr/local/kde.

You can also compile your own version of KDE. You may want to do this if no Solaris binary is available, if you want a different configuration, or if you need a leading edge version because an important bug has been fixed. You can download the sources from www.troll.no/ (QT) and ftp://ftp.de.kde.org/pub/kde/stable/latest/distribution/tar/generic/source/ (KDE). KDE 1.1.1 consists of kdeadmin, kdebase, kdegames, kdegraphics, kdelibs, kdemultimedia, kdenetwork, kdesupport, kdetoys, kdeutils, and korganizer. kdesupport and kdelibs are essential to run KDE applications.

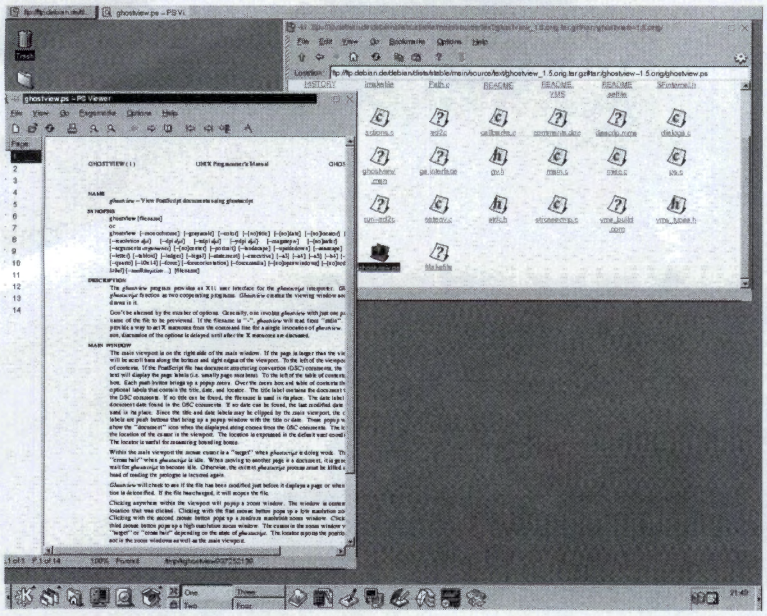

ghostview.ps - PS Viewer
File View Go Pagemedia Options Help
File Edit View Go Bookmarks Options Help
Location

Figure E: *Here we're checking the ghostview information page with the postscript previewer.*

Compiling a KDE 1.1.2 pre-release snapshot

Since KDE is improving continuously, please follow the installation instructions accompanying the version of KDE you download. We used the latest GNU C++ compiler (gcc, version 2.95.1).

Due to a limitation of Sun's assembler /usr/ccs/bin/as (http://egcs.cygnus.com/ml/gcc-bugs/1999-04/msg00618.html) of Solaris 2.6, it's essential to have a compiler using GNU binutils. It's also necessary to have GNU tar 1.12 or later installed on the system to have the step into tar files feature working. Further, either all users need it in their path variable before Sun's tar utility in /usr/bin, or after extracting the sources from the .tar.bz2 archive using

```
bunzip2 -c file.tar.bz2| tar xf
```

You have to replace the `tar` command in kdebase/kfm/kioslave/tar.cpp with its full path name

```
$ diff tar.cpp~ tar.cpp
77c77
<       QString Command( "tar-%sOxf - \"" );
---
>       QString Command( "/usr/local/bin/tar
  -%sOxf - \"" );
217c217
<     QString Command( "tar -%stvf -");
---
>     QString Command( "/usr/local/bin/tar
  -%stvf -");
376c376
<     QString cmdTempl( "tar -%srf %s \"%s\"" );
---
>     QString cmdTempl( "/usr/local/bin/tar
   -%srf %s \"%s\"" );
```

Next, we stepped into each subdirectory and executed

```
CFLAGS="-O -DNeedVarargsPrototypes"
 CXXFLAGS="-O -fpermissive" LDFLAGS="-s"
```

Listing A: *The Xresources.kde file*

```
!!#######################################################################
!!
!!   Xresources.kde
!!
!!   Configuration file for the KDE alternate desktop resources
!!
!!#######################################################################
!!
!!  This file used by the Solaris Desktop Login manager
Dtlogin*altDtsIncrement:          True
Dtlogin*altDtName:                KDE Desktop
Dtlogin*altDtKey:                 /usr/local/kde-1.1.2/bin/kwm
Dtlogin*altDtStart:               /usr/local/kde-1.1.2/dtlogin/Xsession.kde
Dtlogin*altDtLogo:                KDElogo
```

```
./configure  --prefix=/usr/local/kde-1.1.2
--enable-shared --with-quota
--with-qt-dir=/usr/local/qt-1.44
--with-extra-libs=/usr/openwin/lib
--with-pam
make
make install
```

kdesupport must be compiled and installed first, followed by kdelibs. resource.c in kdebase/kdm only compiles when explicitly compiled without the -DNeedVarargsPrototypes flag.

Next you can adjust global defaults. For example, if you want emacs to start up by default with a fixed font, edit /usr/local/kde-1.1.2/share/apps/kdisplay/app-defaults/Emacs.ad. See the "Further information" section later in this article if you want to install the current development or beta test version of KDE out of their version control system (CVS).

Integrating KDE into DTLogin

If you're running DTLogin as your login manager, three steps are necessary to integrate KDE into the window manager selection, containing by default CDE and OpenWindows.

First, edit Xresources.kde and install Xsession.kde and Xsession.kde2 to point to where your kde binaries are located. Next, execute the Install script. After rebooting, you should find a new selection available for KDE within DTLogin. Listings A through D show the files. These files and the pixmaps can be downloaded from www.01019freenet.de/rdorsch/dt_kde.tar.bz2.

License issues and GNOME

Binary distributions of KDE have a difficult license problem. Why? KDE is distributed under the terms of GPL. And it has to be, because of the reuse GPLed code within KDE. Now the problem comes with Troll Tech's license for QT-1.x. In contrast to a program such as Motif, the source code of QT is downloadable and runtime libraries for QT are available for free. It isn't, however, compatible with GPL, since the QT license is more restrictive in allowing developers to distribute modified copies of QT.

GPL says that for a binary distribution, everything needed to run the system has to be under a GPL license, except "the major components (compiler, kernel, and so on) of the operating system." As long as QT doesn't come

Listing B: *The Installscript for KDE*

```
#!/bin/sh
#
#  These files are used to install KDE as an startup option in the desktop
#  window manager.
#
#  Copy Xresources.kde to /usr/dt/config/C/Xresources.d/Xresources.kde
#
#  Copy KDElogo.bm and KDElogo.pm to /usr/dt/appconfig/icons/C/
#
#  The file startkde belongs in the KDE binary directory /css/apps/kde/bin
#
echo "Copying Xresources file to /usr/dt/config/C/Xresources.d"
cp Xresources.kde /usr/dt/config/C/Xresources.d/Xresources.kde
#
echo "Copying KDElogo bitmaps to /usr/dt/appconfig/icons"
cp KDElogo.*m /usr/dt/appconfig/icons/C
#
# Edit to point to where your kde binaries are.
#
echo "Copying Xsession.kde and Xsession.kde2 to /css/apps/kde/bin"
if [ ! -x /usr/local/kde-1.1.2/dtlogin ]; then
   mkdir /usr/local/kde-1.1.2/dtlogin
fi
cp Xsession.kde* /usr/local/kde-1.1.2/dtlogin
chmod a+x /usr/local/kde-1.1.2/dtlogin/Xsession.kde*
```

Listing C: *The Xsession.kde file*

```
#!/bin/ksh

# DTLogin variables
DTDSPMSG=/usr/dt/bin/dtdspmsg

if [ -z "$SESSIONTYPE" ]then
   export SESSIONTYPE="altDt"
fi

if [ -z "$DTSTARTIMS" ] then
   export DTSTARTIMS="False"
fi

if [ -z "$SDT_ALT_SESSION" ]
then
   export SDT_ALT_SESSION="/usr/local/kde-1.1.2/dtlogin/Xsession.kde2"
fi

if [ -z "$SDT_ALT_HELLO" ]
then
   if [ -x $DTDSPMSG ]; then
      export SDT_ALT_HELLO="/usr/dt/bin/dthello -string '
         `$DTDSPMSG -s 37 /usr/dt/lib/nls/msg/$LANG/dthello.cat 1
         'Starting the K Desktop Environment'`' &"
   else
      export SDT_ALT_HELLO="/usr/dt/bin/dthello
         -string 'Starting the K Desktop Environment' &"
   fi
fi

export SDT_NO_DSDM=""

/usr/dt/bin/Xsession
```

with the Solaris OS, it isn't formally allowed to distribute KDE in a binary form for Solaris (although it is done by the KDE project).

Listing D: *The Xsession.kde2 file*

```
#!/bin/ksh

# First a little namespace cleanup of vars associated with this
# (and Xsession.kde) scripts.

unset SDT_ALT_SESSION
unset SDT_ALT_HELLO
unset SDT_NO_DSDM

#KDE variables

KDEDIR=/usr/local/kde-1.1.2
export KDEDIR
# Paths of the necessary dynamiclibaries
LD_LIBRARY_PATH=/usr/local/lib:/usr/local/
    qt-1.44-2/lib:$KDEDIR/lib:/usr/local/
    gcc-2.95.1-binutils/lib:/usr/dt/
    lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
PATH=$PATH:$KDEDIR/bin
export PATH
MANPATH=$MANPATH:$KDEDIR/man
export MANPATH
XAPPLRESDIR=$XAPPLRESDIR:$KDEDIR/lib/app-defaults
export XAPPLRESDIR
TERM=dtterm
export TERM
MAIL=/var/mail/$USER
export MAIL

# initialize the configuration first.

kcontrol -init

# Start the common desktop tools in the background.
# The sleeps reduce disk usage during startup.
# kaudioserver will put itself in the background automagically

sleep 1 ; kaudioserver
(sleep 1 && exec kwmsound) &

# Add -ncols 96 after kfm if using a 8-bit display
(sleep 1 && exec kfm) &

(sleep 2 && exec krootwm) &
(sleep 1 && exec kpanel) &
(sleep 3 && exec kbgndwm) &

# finally, give the session control to the window manager

sleep 2 ; kwm
```

Distributing source code for Solaris is completely legal.

Due to many complaints, Troll Tech introduced a less restrictive QT license (QT public license, QPL) with QT 2.0. We expect the KDE team to also change its license somewhat before KDE 2.0 is released, which will finally eliminate all legal concerns about distributing binaries. An accurate discussion of the license issues is available at **www.debian.org/News/01998/19981008**. This site discusses the implications of the KDE license problems for Deb-ian GNU/Linux, which is also applicable for Solaris.

Due to the license issues of KDE, the GNOME project (**www.gnome.org**) was started shortly after KDE. GNOME builds on the GTK+ library (**www.gtk.org/**), which is distributed under the library GPL (LGPL), compatible with GPL. At the time of writing, both projects stimulate each other and it seems that KDE and GNOME want to cooperate on the protocol level.

Trouble under Solaris

Sun doesn't yet support KDE, but it runs with reasonable stability under Solaris, although we were able to crash it. We noticed that some features of KDE aren't supported under Solaris, since the majority of the developers run Linux. The less Solaris- (or UltraSparc hardware) specific a feature is, the more likely that it works. For example, showing memory statistics doesn't work on UltraSparc hardware under the K Control Center information panel.

During installation, KDE scans the system for non-KDE applications, which are integrated into the K-Panel. A real drawback is that Solaris system utilities, like the audiotool, aren't probed. If needed, applications like audiotool can be added by converting the CDE configuration manually.

Further, we noticed that the screen saver didn't work properly before KDE 1.1.1, and screen unlock didn't work in an NIS+-environment under Solaris 2.6, even in KDE 1.1.1 binary packages. These bugs were fixed in the 1.1.2 pre-release we compiled.

Further information

Since there is already a large KDE user and developer base, we were able to solve most problems by searching the KDE mailing list archive at **http://lists.kde.org**. Problems not previously discussed should be posted at the kde mailing list, and bugs should be reported with the bugtracking system at http://bugs.kde.org.

If you want to access the leading edge code base of the KDE project, you can access the CVS repository. Follow the instructions at www.kde.org/cvsup.html. Be forewarned, it can happen that the code doesn't even compile. To compile the tools from CVS, you need GNU autoconf and GNU automake. You can download them from ftp://ftp.uni-stuttgart.de/pub/gnu/.

Understanding the /proc file system

by Werner Klauser

The /proc file system (procfs) is an example of a virtual file system used by Sun's Solaris operating system. This article covers its implementation on Solaris 2.5 and its changes in Solaris 2.6 and Solaris 7.

/proc in Solaris 2.5

What first appears to be a normal file system beginning at /proc is in effect an illusion. These large files have the same names as the process IDs of the currently running processes, and don't really occupy disk space—they exist in the working memory and in the used swap space. These files contain information which the `ps(1)` command can show us. In this article, we'll discuss how to use these files and the /proc evolution between Solaris 2.5, Solaris 2.6, and 7.

Process tools

Process tools that are found in /usr/proc/bin/ are similar to some options of the `ps(1)` command, except that the output provided by these tools is much more detailed. In general, the process tools do the following:

- Provide control over processes, allowing users to stop or resume them.
- Display more details about processes, such as `fstat(2)` and `fcntl(2)` information, working directories, and trees of parent and child processes.

Let's discuss some of the more interesting tools.

Tools that control processes

Control of your processes can easily be accomplished with the commands in Table A. A practical use is to stop a process that is using too much CPU power and then restart it during noon time while everybody is eating lunch. Rather than starting another job that also uses a lot of CPU power, let `pwait(1)` inform you when the CPU-heavy job has finished with the following command:

```
$ /usr/proc/bin/pwait <pid_of_heavy_job1>
heavy_job2
$
```

`pwait(1)` waits until `heavy_job1` is finished before returning control back over to the shell. This means that `heavy_job2`, even though you have already typed its name, won't begin until `heavy_job1` is finished.

Tools that display process details

Even more detail is available to you with the process commands in Table B. Have you ever

Table A: *Commands to control your processes*

Command	Description
`pstop pid`	Stops the process
`pstart pid`	Restarts the process
`pwait [-v] pid`	Waits for the processes to end

wondered who's reading and/or writing a certain file? All you need is the file's inode—then use `pfiles(1)` to examine the running process to see which process(es) is using the file. Be certain to run the following command as root, because you need to examine all the system's processes. You must be root to do this:

```
# ls -i theFile
34277 theFile
# cd /proc
# /usr/proc/bin/pfiles * | view
search for "ino:34277"
#
```

The command `/usr/proc/bin/ptree(1)` will give you a useful display of the ancestors and children of the desired process. For example, we found our process ID using `ps`:

```
$ ps
   PID TTY        TIME CMD
   16149 pts/6    0:05 loggerTest
   15766 pts/6    0:00 ksh
```

And now we can see the ancestors of process 15766:

```
$ /usr/proc/bin/ptree 15766
178   /usr/sbin/inet -s
   15763 in.rlogind
   15765 login -d /dev/pts6 -r
   sunshine.klauser.ch
      15766 -ksh
         16149 bin/loggerTest
         16769 /usr/proc/bin/ptree 15766
$
```

Are you having problems unmounting a CD-ROM? Are you certain some process is sitting on it? `pwdx(1)` has no problem finding out who the culprit is:

Table B: *More process commands*

Command	Detail
`pfiles pid`	fstat(2) and fcntl(2) information for open files
`pldd pid`	Dynamic libraries linked into each process
`psig pid`	Signal actions
`ptree pid`	Process trees containing specific pids of fathers and children
`pwdx pid`	Current working directory

```
# cd /proc
# /usr/proc/bin/pwdx * | grep \/cdrom
2397:    /cdrom/sunsolve_3_0_7
# ps -ef | grep 2397
root  2397  2357  0   Mar 09 pts/5    0:00
➥ksh
root  2588  2397  0 10:24:29 pts/5    0:00
➥/bin/sh ./sunsolve
root  2652  2370  0 10:26:10 pts/3    0:00
➥grep 2397
#
```

It's amazing how the usage of these `proc(1)` tools can help you solve those little but messy problems!

Coding example

Beginning a program with `main(int argc, char *argv[])` and then looking at `argv[0]` will give you the name of how the program was called to execute it. But how does the `ps(1)` command get the process IDs and the names belonging to these process IDs?

The system command `getpid(2)` returns the process' ID. But now that you have its ID, how do you go on from here? This is where the /proc file system comes into use. Though these files only exist in a virtual world, they are readable. So all you need to do is execute a simple `open(2)` command:

```
#include  <stdio.h>
#include  <fcntl.h>
#include  <unistd.h>

main(int, char *argv[])
{
   int      ifd;
   char     sFilename[16];

   (void) sprintf(sFilename,
      "/proc/%d",
      getpid() );
   ifd = open(sFilename, O_RDONLY);
   (void) fprintf(stderr,
"file = %s, file descriptor = %d\n",
      sFilename, ifd);
   (void) close(ifd);

   return(0);
}
```

Notice that if you look at the files in /proc with `ls(1)`, you'll have the impression that their names are always five digits. That means if the process' ID is less than 10,000, it must be left padded with zeros (0). However, you don't need to take this into account. For example, opening /proc/123 has the same effect as opening /proc/00123! We only want information about the process, so its corresponding file is opened for reading only.

But just opening and then closing the virtual file doesn't help us too much. We need to do just a little bit more. Add the following lines to the example:

```
#include <sys/procfs.h>
...
   struct prpsinfo  tPrpsinfo;
...
   (void) ioctl(ifd,
      PIOCPSINFO,
      &tPrpsinfo);
   (void) fprintf(stderr,
      "processname = %s\n",
      tPrpsinfo.pr_fname);
```

Now we get the process name printed to `stderr`. This is just the beginning. But first an explanation of the above code lines.

The include file `<sys/procfs.h>` introduces our code example to the /proc file system. It includes, among other things, the variable type `struct prpsinfo`, which is a structure containing process information. Its element `pr_fname` is the process's name. It also contains the defined `PIOCPSINFO`, which gets us `ps(1)` information. Using `PIOCUSAGE` and a variable of type `struct prusage` can get us information concerning the process' usage of computer resources. This must be what `sar(1)` uses. Do take a look at this include file found in the /usr/include/sys/ directory.

Solaris 2.6 changes

The previous flat /proc file system has been restructured into a directory hierarchy that contains additional subdirectories for state information and control functions. Though the flat files no longer exist and are now directories, you can still open them as a flat file just as in Solaris 2.5. Another virtual file system surprise!

Solaris 7 changes

Some changes to Solaris 7 include advanced 64-bit support. Because 64-bit applications (processes) contain extended capabilities, a 32-bit program should be converted to a 64-bit program if you're using /proc files created by a 64-bit process.

Summary

Using the process file system /proc with its tools `proc(1)`, or easily accessing it through your own programs, quickly shows the benefits that its designers had in mind. You can open processes as files to actually read and gain useful information.

Developing Solaris knowledge bases

DOWNLOAD
ftp.zdjournals.com/sun

by Paul A. Watters

Solaris is a large and often bewildering operating system for technical support staff to know everything about. In addition, even the most experienced Solaris administrator often consults colleagues with expertise in a specific system area, so that they can draw on that expert's knowledge of the system. Of course, if that expert is sick or on vacation, it might be inconvenient to wait until he returns. Wouldn't it be much more efficient if the Solaris system could provide you directly with that expertise ?

One way to effectively pool the technical expertise and specialized knowledge of Solaris is to construct a knowledge base of all the facts and key relationships that are known about Solaris. This knowledge can be made readily available using an expert system shell, written in a logical programming language like Prolog. In this article, we'll develop a simple, but extensible, Prolog expert system that provides advice on how to install and configure slices and partitions in a hard drive.

What ever happened to artificial intelligence?

Systems based on artificial intelligence don't seem to have lived up to the grand expectations and predictions made a generation ago. No machine has passed the "turing test" of intelligence (that is, a machine whose input/output behavior is indistinguishable from that of a human), nor is it likely that Arthur C. Clarke's predictions in *2001: A Space Odyssey* about the intelligence of computers will be realized. However, it would be incorrect to assume that artificial intelligence hasn't made an impact on the way that software is developed, or that it doesn't have a practical application.

In fact, artificial intelligence as a program of research and development has seen the greatest success where researchers have asked the question "What do computers do better than humans?" One characteristic, called computational intelligence, is that computers can remember many more things than humans, and much more accurately. The other key characteristic of computational intelligence is the application of logic to solve problems. The formal logic employed by computer programs is neither soft nor fuzzy; it's based on the implementation and extension of simple logic functions like AND, OR, and NOT.

Prolog

Of course, we can only go so far with simple logic predicates. Fortunately, there are logical programming languages available for Solaris and other platforms that have been widely used for higher-level reasoning that humans often find difficult. One such language is Prolog, which stands for *pro*gramming in *log*ic.

Prolog is much more powerful than similar languages like Lisp, because it contains a built-in inference engine for automated reasoning. This means that applications requiring the use of formal reasoning can be rapidly implemented in Prolog, without needing to write a separate procedure for making inferences. There's also an ISO standard for Prolog, of which most versions of the language are now supersets.

Expert systems

Expert systems are automated reasoning systems which have become popular in recent years, from applications used to diagnose infectious diseases to the automated software configuration of mainframe computers. In each of these knowledge domains, experts are consulted in structured interviews to elicit their reactions to various situations.

For example, a doctor experienced with treating meningitis might be observed treating a patient, and the procedure that she follows is noted, while a system administrator might be asked to configure different systems with different operating requirements.

The kinds of knowledge that these experts use every day can thus be extracted, and often formalized, by identifying a hierarchical order of operations. This order of operations can then be implemented using Prolog, and made available to non-experts who may need access to expert knowledge.

For example, medical interns may be confronted with treating meningitis patients, but don't recognize a particular symptom. They could query an expert system with the suspected diagnosis and the unknown symptom to see whether the combination is known. If a new combination is identified, the knowledge base can be updated with that knowledge. Thus, knowledge bases continue to grow through time with experience. In the case of the system administrator, as new operating system releases are made, it's important to note any interactions between legacy software products and new systems, and for these to be entered into the knowledge base.

Limitations of expert systems

Of course, there are limitations to the application of pure logic in solving real-world problems. Expert systems have no mechanism, by themselves, to evaluate their internally generated worldview against the actual physical world. Thus, a change in external conditions that isn't entered into the knowledge base could reduce the effectiveness of the system in making diagnoses.

Alternatively, if the knowledge base contains no data or rules with which to deal with a problem, it can't provide advice (and therefore, may not be an expert at all). However, as long as we recognize that expert systems have limitations just as human experts do, it's possible to use them to solve real-world problems.

An artificial intelligence example

In this article, we present a simple expert system, developed in Prolog, for diagnosing why a disk partition or slice isn't visible to the system (a question that is often asked by part-time administrators or those from a plug-and-play background). The complete application is shown in **Listing B** on page 12.

The basic procedure involves installing the new disk; telling the system to reconfigure its hardware settings (for example, by booting with `boot -r` or `touch /reconfigure`); formatting the disk to create slices with the format command; creating a new filesystem on the slice with the `newfs` command; and mounting the disk slice after creating a mount point. While the procedure is straightforward for experienced administrators, it can seem daunting to novice users. As Solaris becomes more widely adopted on the Intel platform, it will become even more important for cost-effective, automated expertise to become readily available.

To run the program, you'll need to obtain a copy of Prolog for Solaris. One freely available and widely used implementation is the SWI-Prolog interpreter developed at the University of Amsterdam. To download it, point your browser to **www.swi.psy.uva.nl/projects/SWI-Prolog/**. After installation, you can start Prolog with the command `pl`. A banner like

```
Welcome to SWI-Prolog (Version 3.2.8)
Copyright University of Amsterdam

?-
```

will print. The prompt `?-` is the query prompt, and indicates that Prolog is ready to answer queries. Queries are always terminated with a period (.). To load the expert system saved in a file called sol_expert_system.pl, just type:

```
?- [sol_expert_system].
```

If the file is found, and its contents are legal Prolog code, then a message like

```
sol_expert_system compiled: 0.5 sec, 5,496
   bytes

yes
```

will print. The `yes` indicates that the query succeeded. If the file was unavailable or didn't contain Prolog code, then the query would probably return `no`.

To start the expert system, simply type the query

```
?- expert_system.
```

A sample session is shown in **Listing A**.

Listing A: *Sample expert system*

```
?- expert_system.
This program diagnoses why a disk is not visible to the system.
➥Answer all questions with Y for yes and N for no.

Was the device reported at bootup?
|: Y

Do you know if the disk has been formatted?
|: N
As the root user, use the 'format' command to prepare disk slices
➥and to modify the partition table.
```

The sequence of events in this session is quite straightforward, and the outcome is common enough in the world of system administration. The system asks the user whether the device is visible at boot time, to which the user replies yes. This is encouraging as it indicates that the device has been detected.

Of course, the first step after detection is formatting, which is why the program asks if the disk has been formatted. If it hasn't been formatted, then the user can format the disk, and then rerun the expert system to find out what to do next (that is, now that the user knows the disk has been formatted, he can learn how to create a UFS filesystem on individual slices).

The sequence of events, when `expert_system` is called, is fairly straightforward. Because the program is basically an implementation of logic, Prolog programs can be readily understood and maintained without an in-depth knowledge of the syntax of the language. After initialization, which clears all stored answers, the `analyze_problem` clause is called, which proceeds through a cycle of observing symptoms and diagnosing problems. If a symptom can't be diagnosed, then the query fails. However, in the situations we have included in our knowledge base, most should be covered by the rules given.

Symptoms are associated with logical conditions. For example, the observation that the device isn't visible

Listing B: *Our Prolog disk diagnostic application*

```
% Solaris Disk Expert System v1.0.0
% Uses the answers to three simple questions to determine
% why a disk is not visible to the system, e.g., by using df.
% To start the expert system, type 'expert_system.' at the
% prolog prompt.

% Clause which is used to start the system.
expert_system :-
   write('This program diagnoses why a disk is not
   ➥visible to the system.'),nl,
   write('Answer all questions with Y for yes and
   ➥N for no.'),nl,
   initialize,
   analyze_problem.

% List of symtpoms and the logical conditions associated
% with each symptom
symptom(reconfigure_dev_directory) :-
   observation(device_visible,no).

symptom(run_format) :-
   observation(device_visible,yes),
   observation(device_formatted,no).

symptom(newfs_created) :-
   observation(device_visible,yes),
   observation(device_formatted,yes),
   observation(newfs_installed,no).

symptom(ready_to_mount) :-
   observation(device_visible,yes),
   observation(device_formatted,yes),
   observation(newfs_installed,yes).

% List of questions to be asked.
ask_question(device_visible) :-
   write('Was the device reported at bootup?'),nl.

ask_question(device_formatted) :-
   write('Do you know if the disk has been formatted?'),nl.

ask_question(newfs_installed) :-
   write('Do you know if a file system has been prepared
   ➥on the disk?'),nl.

% Set of possible diagnoses and remedies.
diagnosis(reconfigure_dev_directory) :-
   write('You must tell the system to reconfigure its
   ➥/dev (devices)'), nl,
   write('directory by typing ''touch reconfigure''
   ➥before rebooting'), nl,
   write('or type ''boot -r'' at the boot prompt.').

diagnosis(run_format) :-
   write('As the root user, use the ''format'' command to
   ➥prepare disk slices'),nl,
   write('and to modify the partition table').

diagnosis(newfs_created) :-
   write('As the root user, you need to create a new file
   ➥system on your slice.'),nl,
   write('before it can be used. Please run ''mkfs''
   ➥before attempting to mount this disk.'), nl,
   write('If the file system will be UFS, you can use
   ➥the ''newfs'' command.').

diagnosis(ready_to_mount) :-
   write('Your disk is now ready to mount. Please determine
   ➥a mount point'),nl,
   write('(e.g., /u01) and mount with the command ''mount
   ➥raw-device /u01''.').


obtain_response(Response) :-
   get(Char),
   get0(_),
   match_ascii(Char,Response),
   !.

obtain_response(Response) :-
   nl,
   put(7),
   write('Type Y or N:'),
   obtain_response(Response).

match_ascii(89,yes).
match_ascii(121,yes).
match_ascii(78,no).
match_ascii(110,no).

analyze_problem :-
   symptom(D),
   nl,
   diagnosis(D),
   fail.

analyze_problem.

:- dynamic(stored_answer/2).

initialize :- retract(stored_answer(_,_)),fail.
initialize.

observation(Q,A) :- stored_answer(Q,A).

observation(Q,A) :- \+ stored_answer(Q,_),
   nl,nl,
   ask_question(Q),
   obtain_response(Response),
   asserta(stored_answer(Q,Response)),
   Response=A.
```

```
observation(device_visible,no).
```

leads to the symptom of a device reconfiguration being necessary

```
symptom(reconfigure_dev_directory)
```

This symptom is identified very early on in the execution of the program when the question

```
ask_question(device_visible) :-
   write('Was the device reported at bootup?')
```

is asked. Where there are multiple logical conditions which must be satisfied, they are specified with the logical operators AND (,), OR (;), and NOT (\==).

The best way to get a feel for how the expert system works is to try it out for yourself. The amount of detail included with each system can be substantially increased, and could include working examples like a man page.

More information

If you're interested in writing your own expert system, you should consult "Using expert systems to manage change and complexity in manufacturing," in Walter Reitman's book *Artificial Intelligence Applications for Business* and E. Shortliffe's book *Computer-Based Medical Consultation: Mycin*. Although dated, these are two classic implementations of expert systems.

Virtual memory and priority paging with Solaris 7

by Abdur Chowdhury and Andy Spitzer

Solaris provides many features to improve the performance of applications running on it. One of those performance enhancements is the virtual memory system. Virtual memory allows the creation and usage of memory objects larger than available physical memory. This is accomplished by temporarily storing data on a physical device, such as a disk, and swapping memory to and from the disk when needed to give the appearance of a much larger memory space.

In this article, we'll explain parts of the virtual memory system and why Sun added priority paging to improve the existing system. We'll also examine where priority paging can help and how to determine if it will help your system.

Virtual memory primer

There are two types of virtual memory algorithms used by Solaris today: swapping and demand paging. Swapping works on a per-process level. When free physical memory becomes really low, an algorithm selects a process to be swapped out. The operating system then swaps out that process' entire memory image to the swap device. Older versions of SunOS swapped out any process that hadn't run within the last 30 seconds, on the theory that a process that was sleeping that long wouldn't be needed soon.

Because of the coarseness of this algorithm, large amounts of memory are swapped in and out each time memory is needed. This can cause a huge performance hit, which is why Solaris doesn't rely on swapping alone.

The second type of virtual memory system is called demand paging. Physical memory is divided into pages of 4 KB or 8 KB in size, depending on the hardware architecture. Rather than swapping out the memory of an entire process, only a particular page of memory is selected by an NRU (Not Recently Used) algorithm or similar policy to be paged out. This gives the virtual memory manager a finer granularity with which to control physical memory usage. It also reduces the amount of physical memory needed to run a process, as only active pages end up in memory; the rest end up (or stay) on disk.

Solaris uses demand paging for the most part, but if the system gets really low on physical memory for more than 30 seconds, the Solaris virtual memory manager will switch to swapping and swap out an entire process to quickly free physical memory. Most system administrators monitor for swapping because it's a sign of not enough physical memory for the system.

An administrator may monitor the amount of swapping and paging on a machine via the perfmeter(1x) program, as well as by vmstat(1M) and others. *perfmeter* is a graphical program that displays the rates of paging and swapping on an Xserver. *vmstat* is a command line program. With the -S argument, vmstat reports on both paging and swapping.

Table A shows the output of vmstat. The pi column of vmstat shows page-ins (in kilobytes) since the last interval. The po column shows page-outs, and the si and so columns report swap-ins and swap-outs.

Sharing memory objects between processes

One of the enhancements to the virtual memory manager over the years is the added ability to share memory objects between processes. This feature drastically reduced the memory requirements of the system as a whole by being able to share large parts of code, such as executable images and shared libraries.

This brings us to one of the most important enhancements to the virtual memory manager: the ability to map file system objects directly into the virtual memory system. This ability speeds up many things. One performance improvement was the ability to map an executable file to the virtual memory and start running the process immediately without loading the entire executable image into memory.

As pages of code are needed, they are paged into physical memory by the virtual memory system. Unneeded pages remain on disk, and pages only used once (like startup code) are eventually paged out. Since the executable image has already been loaded into memory when a second copy of the same program is started, it may already be in memory, so another copy need not be paged in.

The two processes then share the same memory pages. A *copy on write* policy insures that if a process modifies a page another process is sharing, a copy of that page is made, and then updated. That page is no longer sharable as it is *dirty*. When a dirty page is paged out, it's written to the swap section of the disk. When a clean page is paged out, it need not be written to swap, as it can be reloaded from the original file.

Because the files may now be treated as memory pages, problems may arise from systems due to large amounts of file I/O. If a 1 GB file is read randomly, as each page of the file is read from disk for the process, it's mapped into physical memory. If no free physical memory is available, one of the existing pages of memory is paged out and the new page is read into physical memory.

This increases the probability that a given page of memory will be swapped out to disk. Memory used for file I/O shares the same pool of physical memory used for executable programs, and one may suck up all available physical memory at the expense of the other.

Priority paging enhancements

What does all this mean? It means that applications with large amounts of file system activity may adversely affect the system's performance by swapping out executable pages of code. To combat this consequence, a new feature called priority paging has been included into the virtual memory system. Priority paging allows a portion of the physical memory to be segmented for file system ob-

Table A: *Paging statistics can be displayed with vmstat*

```
vmstat -S 5
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  si  so pi po fr de sr s1 s1 s1 s1   in   sy   cs us sy id
 0 0 37  1360  5344   0   0 278 61 249 0 32 3  6  8  0  745  605  570 10  5 85
 0 0 59 1646584 9432  0   0 52 84 121 0  9  0  0  7  0  796 52495 572 28 10 62
 0 0 59 1647456 10024 0   0 35 20 20  0  0  0  0 10  0  760 52501 874 32 11 58
 0 0 59 1647120 11160 0   0 14 38 331 0 55  1  0  4  0  625 51718 495 28  8 65
 0 0 59 1647160 11088 0   0 19  1  1  0  0  3 21 11  1  989 50667 555 30  9 61
```

jects. This means that when reading a large file of non-executable code into memory, that executable code is less likely to be swapped out of main memory.

With this new policy, any system that makes heavy use of the file system can benefit by placing limits on the amount of physical memory that will be used for file system pages. Therefore, workstation applications will remain responsive even if large files are being processed. OLTP processes won't suffer from the need to reload executable code between transactions.

The virtual memory system uses a pageout scanner to find free pages of memory to swap out to temporary storage. The default pageout policies use a kernel variable called the lotsfree amount to determine when physical memory should be scanned to free pages. This can be thought of as a way to prevent swapping ahead of time, or a proactive approach to paging.

When the lotsfree limit has been reached, the pageout scanner must determine which pages to free; this is done by a two-handed clock algorithm. The first hand scans pages of memory and clears the reference bit. The second hand scans memory pages at a later time, and checks the reference bit. If it has been set, then the page has been used recently. If the bit isn't set, then it's a candidate for paging.

One last check is made before swapping the page. The last check looks at the po_share counter. The po_share counter is used to mark shared libraries and executable code. This is an attempt to protect shared libraries and executable code from swapping. It has worked well for systems that don't handle a large amount of file system activity. The po_share is just a counter and is decremented each time the scanner passes by. If no other pages have been found, then the shared library or executable code is swapped. Since file system activity may have been very recent, it's possible to bypass this protection and hurt system performance.

One last note about file system I/O: Solaris tried to eliminate simple file scans that would overwrite the virtual memory system by implementing a free-behind policy. When a file is sequentially read, the file pages are purged after the read in an attempt to save the virtual memory system.

About our contributors

Abdur Chowdhury is the senior computer scientist for IITRI in Rockville, Maryland. Abdur is also working on his Ph.D. in Computer Science at the Illinois Institute of Technology. He received his MS and BS in Computer Science at George Mason University in 1996 and 1994. Further, Abdur has authored many papers on process migration, fault tolerant routing protocols, and information retrieval topics. You can reach him at abdur@cs.iit.edu.

Rainer Dorsch received his master's degree in Physics in 1996 at the University of Ulm (Germany) and works now at the University of Stuttgart (Germany) on his Ph.D. in Computer Science. He started with UNIX in 1992, has worked with Linux at home since 1994, and has been heavily involved with Solaris administration since 1996. He welcomes your comments and criticisms. You may reach him via email at rainer.dorsch@informatik.uni-stuttgart.de.

Werner Klauser is an independent UNIX consultant working near Zurich, Switzerland. While not paragliding or roarin' around on his Harley chopper, he can be reached by email at klauser@klauser.ch or on his Web page at www.klauser.ch.

Andy Spitzer is a senior software engineer at Centigram Communications Corporation. He's the Chief Architect of Centigram's Enhanced Services Platform. You can reach him at woof@centigram.com.

Paul A. Watters is a research officer in the Department of Computing at Macquarie University, Australia. He can be reached at pwatters@mpce.mq.edu.au.

Inside Solaris

Tips & Techniques for users of Sun Solaris

Inside Solaris (ISSN 1081-3314) is published monthly by ZD Journals 500 Canal View Boulevard, Rochester, NY 14624.

Customer Relations

US toll free	(800) 223-8720
Outside of the US	(716) 240-7301
Customer Relations fax	(716) 214-2386

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to

ZD Journals Customer Relations
500 Canal View Boulevard
Rochester, NY 14623

Or contact Customer Relations via Internet email at journals@mail.training.com.

Editorial

Editor	Garrett Suhm
Assistant Editor	Jill Suhm
Managing Editor	Joe Froehlich
Copy Editors	Rachel Krayer
	Christy Flanders
	Ellyn McCasland
Contributing Editors	Abdur Chowdhury
	Rainer Dorsch
	Werner Klauser
	Andy Spitzer
	Paul A. Watters
Print Designer	Melissa Ribaudo

You may address tips, special requests, and other correspondence to

The Editor, *Inside Solaris*
500 Canal View Boulevard
Rochester, NY 14623

Editorial Department fax	(716) 214-2387

Or contact us via Internet email at inside_solaris@zdjournals.com.

Sorry, but due to the volume of mail we receive, we can't always promise a reply, although we do read every letter.

ZD Journals

General Manager	Kelly Baptiste
Manager of Customer Relations	Heather Loaicono
Manager of Operations	Cristal Haygood
Manager of Print Design	Charles V. Buechel
Manager of Product Marketing	Mike Mayfield
Product Marketing Manager	Brian Cardona

Postmaster

Periodicals postage paid in Rochester, NY and additional mailing offices.

Postmaster: Send address changes to

Inside Solaris
P.O. Box 92880
Rochester, NY 14692

Price

Domestic	$99/yr ($9.00 each)
Outside US	$119/yr ($11.00 each)

Our Canadian GST# is: R140496720. CPM# is: 1446703.

Back Issues

To order a back issue from the last six months, call Customer Relations at (800) 223-8720. Back issues cost $9.00 each, $11.00 outside the US. You can pay with MasterCard, VISA, Discover, or American Express. Collections of back issues are available on CD-ROM as well. Please call for more information.

Are you moving?

If you've moved recently or you're planning to move, you can guarantee uninterrupted service on your subscription by calling us at (800) 223-8720 and giving us your new address. Or you can fax us your label with the appropriate changes at (716) 214-2386. Our Customer Relations department is also available via email at journals@mail.training.com.

Coming up...

USPS ARMIN PS1 881 APPROVED POLY

- Using ASET for security
- Running Linux applications on Solaris X86
- The /etc/system file
- Acceptable use policies

PERIODICALS MAIL

2096

**************3-DIGIT 480
C: 7661905 00002096 04/00
[redacted]
34
78

Please include account number from label with any correspondence.

The new priority paging policy has added a new kernel variable called cachefree. The prior virtual memory system started scanning for free memory when the lotsfree threshold was reached. The new policy uses the cachefree threshold value to start scanning physical memory, and removes file system pages first to create free memory.

To enable the priority paging policy for Solaris 2.7, add the following line to your /etc/system file, and then reboot:

```
set priority_paging=1
```

Priority paging is available for Solaris 2.6 with a kernel patch 105181-09 and is enabled the same way.

Evaluations

The prior discussion may have lead to the conclusion that not all systems need this new policy and that changing your system at this point may not be beneficial. Solaris 2.7 has added a few kernel statistics to make determining whether to use the policy easier. To save you the time from writing code to read these values from the kernel, there's a utility called memstat, (see "Recommended readings" for the URL).

At this time, memstat is an unsupported utility. The same statistics may be added to vmstat in the future. The new statistics are executable page in, executable page out, executable page free, anonymous page in, anonymous page out, anonymous page free, file page in, file page out, and file page free. With these statistics, you can watch a running system and see if the executable pages out, or anonymous page out values change as file activity occurs.

As priority paging attempts to maintain executable pages in memory over data pages, the responsiveness of a workstation with priority paging enabled should increase. Responsiveness is subjective and hard to evaluate. It ultimately comes down to the workload and the amount of physical memory in the system, the speed of swap devices, etc.

One simple test is to open a browser and an xterm window. Cover the browser with the xterm, and then compile a large program. The compilation should generate a lot of file activity, perhaps causing the browser code to be paged out. Then, bring the browser window to the front, and watch how quickly it refreshes. With priority paging, the chances of file system activity pushing executable code out of primary memory are reduced, so the responsiveness of the browser may be better.

Conclusion

Certain applications and systems can benefit from priority paging. Systems running Oracle or some types of RDBMS will probably not see benefits from priority paging, simply because the database is bypassing the virtual memory system. Workstations with limited memory, however, may see great improvements. It simply comes down to first understanding what options you have at your disposal and then evaluating your particular situation to determine what's best for you.

Solaris hasn't made this policy the default, but may make it the default in future versions of the operating system if feedback from this policy is positive. Therefore, developers and system administrators must be aware of the policy and how to determine if it is beneficial for them. ZDJ

Recommended readings

- *The Solaris Memory System,* May 1998, Chapters 4-5
- R. McDougall, T. Vo, T. Pothier, "Priority Paging," **www.sun.com/sun-on-net/performance/priority_paging.html**
- The memstat utility, **www.sun.com/sun-on-net/performance/memstat.Z**