The Good, the Bad, and the Ugly Aspects of Installing New OS Releases

Barry Sharp
Boeing Shared Services Group - Technical Services
High Performance Computing - Engineering Operating Systems
The Boeing Company, P.O. Box 3707 MC 7J-04, Seattle, WA 98124-2207, USA
barry.sharp@boeing.com

Abstract

Many SGI/Cray sites do not have the luxury of having multiple like systems with one isolated for validating new OS releases prior to installing them as their production system. The difficulty of doing this on a single system without endangering the running production work is explored along with how Boeing has developed its processes over time to safeguard production while still being able to adequately QA new OS releases.

1.0 Introduction

When our T916 UNICOS system is taken out of production for QA test time it is necessary to preserve all aspects of the production environment. Preserving production consists of checkpointing all executing NQS batch work and saving it along with any NQS queued jobs. When a QA test system is loaded into the production machine a special NQS configuration is setup to avoid production work from being restarted. There are many other things such as cron, accounting, accessing or modifying files that were in use by the checkpointed production work that must also be considered to avoid QA testing from interfering with the successful recovery of production work at a later time.

The usefulness of the UNICOS guest system (UNICOS under UNICOS or UuU) is discussed and how it can be configured to operate safely alongside the host production work.

This paper outlines how an adequate level of protection is provided during QA testing of new OS releases on a single system used for production work, and how the test environment, planning, and configuration process ensures the new system installed is identical in all respects to the one tested.

2.0 Background

Prior to our UNICOS experiences we ran COS on an X-MP. The concerns we have today with system testing stem back to this system. In the COS startup module we had inserted a great deal of local code that serviced the requirements for protecting production work. We

called this local feature the “Freeze/Thaw” facility. Freezing the system consisted of preserving all executing work environments during the system shutdown process, with thawing being the restoration and resumption of the frozen system. Multiple levels of freeze/thaw operations were incorporated into the design, allowing for example, a production system to be frozen, a test system to be started and subsequently frozen and followed by thawing the production system. The frozen test system could then be thawed at a later date and reused, discarded or even frozen again. The COS system lended itself easily to this technique because all work that had rolled (swapped to disk) was restartable unless some file modification(s) restricted it. During shutdown all jobs that had never rolled, were rolled to make them restartable. Files (COS referred to them as datasets) being referenced by the work could be conveniently locked to avoid them from being modified during the frozen or test period. The list of conveniences offered by COS goes on and on. In the very worst case, the system freeze process took no longer than 15 minutes to complete. Most were accomplished in less than 5.

Achieving the same degree of protection for production work under UNICOS is far more difficult and thwart with pitfalls. During the early days with UNICOS important jobs that had been in execution for days were lost (all jobs are important from the customers’ viewpoint), queued jobs mysteriously disappeared, non rerunnable production queued jobs were inadvertently started during test sessions, and resource accounting data at times was erroneously processed.

All of these aspects are viewed with great disdain, causes a high cost penalty, and at times is embarrassing for the data center. None of this was good for our public relations! Our customers demand predictability and reliability.

Like it or not, the Good, the Bad, and the Ugly aspects of this UNICOS problem had to be addressed.

3.0 The Good Aspects

The good aspects can be arrived at by listing features that have been exclusively designed for preserving work and those that help facilitate the preservation of work across a system shutdown and restart.

1. Job session checkpointing and restarting.
2. Multi-level Security (MLS) - in particular Access Control Lists (ACLs).
3. Voluntary and mandatory file locking.
4. File systems provide a limited method for hiding data by not having them mounted.
5. NQS configurations are easily modified. The NQS checkpoint directory can be made different from production under a test system. NQS job queues can have different names from production.
6. The ability to send system shutdown and restart signals to processes. Over time we expect more of our codes to change to respond to these signals so that the checkpointing overheads can be reduced.

7. The ability to /etc/warn users about a pending system shutdown
8. The ***ping*** command for interrogating connectivity to a system
9. The ability to shutdown a portion of the workload such as NQS batch
10. The UDB facility for restricting system access via the /etc/udbrstrict command. I have always wondered why the ‘e’ was left out of udbrstrict.
11. The ability to restrict system services such as rsh, ftp, and DMF by simply not starting them. This really is complementary to the overall UNIX system design by not having all system services concentrated in a monolithic kernel.
12. The very powerful shell programming language.
13. The wealth of commands available.
14. The FTA facility is an admirable addition to the UNICOS toolkit. It provides reliable file transfer between hosts and will recover across a system interruption.
15. Last, and certainly not the least, is the UuU facility. Today, the UuU is being used from 4 to 8 hours per day for our UNICOS 10 and Year2000 testing.

The list can be extended further, but that is not the point. It is sufficient to say that there are many good aspects of UNICOS that, if understood, used correctly, and if human ingenuity and creativity is used, can offer much solace in securing production work during test time. Of course, there also needs to be a strong desire to do so in the first place.

4.0 The Bad Aspects

Again, the best way to illustrate the bad aspects is to list problems found to be annoying but not insurmountable. That is, problems that do not constitute a real threat to securing production work while testing a new system.

1. Checkpoints and restarts are very time consuming. The author has seen some NQS shutdowns take as much as one hour. Much of this is probably due to our work usage of large amount of SDS space.
2. Checkpoints have too many restrictions and error messages are issued in typical UNIX fashion (i.e., too cryptic).
3. Mandatory file locks evaporate while a job is in a checkpointed state. They are however, re-established during job restart, but this means during the time a job is checkpointed the files in question are unlocked! Whether this constitutes a real threat to data corruption is debatable. The best course of action is to warn the user community about this aspect.
4. It seems a shame that the SWAPDEV cannot be employed in some manner to hold the checkpointed information. For a large system such as ours with 512 Mw memory and 2048 Mw SSD/SDS the SWAPDEV is some 43 Gigabytes in size (primarily due to allowing memory and SDS to be over subscribed plus allowances made for fragmentation).

5. The vanilla UNICOS system reports very little information on successes, failures, and progress during system shutdown procedures.
6. Unlike COS, UNICOS does not provide a Simulator. Remember CSIM under COS?
7. Unlike the COS CSIM, the UuU requires or steals precious resources from the host system, such as memory, disk space, SDS, and console zip connections. In addition, there is always the risk that UuU will crash in some peculiar way that makes it affect the host system. A recent experience has shown UuU to have this affect if incorrectly configured. It is fair to add here that UuU is considered an invaluable asset for enabling new systems testing. Harping back to CSIM again, only one guest can be running at a given time whereas executing multiple CSIMs (all simulating different system builds) were possible.
8. A site needs to somehow advertise to all system components if system is a test system as opposed to production.
9. A fair amount of NQS internal knowledge is required to adequately construct automatic procedures to safeguard production NQS work. It has taken several years to feel comfortable that all aspects in this area have been dealt with.
10. The vanilla system does not preserve the NQS queue states across a shutdown/restart. For example, if the operator stops a queue named ‘ABC’ prior to an NQS shutdown then, when it is restarted hopefully the operator remembers not to start the ‘ABC’ queue. Our experience shows that this is not the case, especially if a shift change has occurred during the shutdown period.
11. The vanilla system does not preserve the tape drive states across a shutdown/restart. This is especially important when tape drives are part of a shared resource pool.
12. The /etc/shutdown and its brother scripts .pre, .mid, and .pst need extensive ‘bullet-proofing’ and/or additions. For example, we have added such things as the logging of critical steps to the log file /etc/debug by using the subscript LOGIT, improving the chances of all user processes being killed during shutdown by calling killall three times instead of only once, and prevent an invalid time on /etc/shutdown (this time must be a reasonable number).

5.0 The Ugly Aspects

The ugly aspects boil down to those things that are considered very difficult to impossible to resolve and those that are considered to involve great risk to production work, ranging from loss of work to potential data corruption.

5.1 File systems

Production file systems are considered to contain data that for safety reasons must at all times be beyond the reach of corruption. If at all possible these file systems are never mounted during the initial phases of testing a new system. A new system should be

installed on separate disks with separate data paths. This implies the need for a capital investment. Only after sufficient testing should production file systems be mounted, and even then, they should only be mounted if they are easily or completely recoverable to cover against the unforeseen problem(s).

The test system's file systems should be configured as a mirror image of production so that performance or configuration anomalies are highlighted well before the time comes for installing the new system. This file system replication will also aid with any benchmarking activities.

5.2 Severe checkpointing problems

Checkpoints can consume vast amounts of disk space. In some cases the needed disk space is unnecessary. For example, if prior to an NQS shutdown a large job had been 'held' via the ***qmgr*** directive and subsequently 'released' then a checkpoint file would already exist. This is also true for long running jobs that have been previously checkpointed and restarted across a 'test window'. When the NQS shutdown is performed a new checkpoint file is created for the job alongside the old one which is removed only if the current job's checkpoint succeeds. Thus, unnecessary disk space can be consumed during this activity.

This aspect of consuming unnecessary disk space can obviously cause other job checkpoints to fail, due to lack of disk space, which leads to lost work. The old job checkpoint images should be removed before new checkpoint images are begun so long as one of the following conditions is met:

1. The old job checkpoint image is no longer valid (maybe the chkpnt_util(1) could be used for this validation)
2. The old job checkpoint image is still valid, the job can be checkpointed (i.e. satisfies all the checkpointable requirements), and there's enough disk space for the new checkpoint image.

SPR 711575 was posted on this problem.

Interestingly enough, DMF provides some relief in this area. It is possible to migrate checkpoint files to place them in off-line (IFOFL) state (see UNICOS ***man sh*** and IRIX/DMF 2.6 ***dmattr*** command). Having checkpoint files in this state in no way interferes with the checkpointing facility. Jobs may be recovered successfully from migrated checkpoint files. Obviously, it just may take more time.

5.3 Reporting of failed and/or lost jobs across shutdown/restart

As mentioned above, during our early years with UNICOS many job failures were seen as a result of a system and/or NQS shutdown and restart. By 'seen' I mean that customers would notice their jobs had disappeared or had been rerun. The data center, for the most part, was ignorant about the work loss until notified by the customer.

The NQS job failures are reported in the NQS log file and /usr/adm/syslog/demonlog. It is very time consuming and tedious to scan for these job failures and perform some analysis as to the reasons after every shutdown/restart sequence.

A great deal of effort is required to create an automatic job shutdown and restart recovery reporting facility. Maintenance of such a facility may also be high if SGI/Cray modifies the job failure message texts in the log files. Even with these problems in mind, a reporting mechanism was considered essential enough to warrant the effort to develop one. This facility was embedded in our Health Monitoring System which is mentioned some in the paper “Advancing with UNICOS Toward IRIX”, Section 2.24, presented at the 1998 Stuttgart CUG. An example of the report is shown below. This report is electronically mailed to our Technical Help Desk and internal support analysts for review.

NQS SHUTDOWN/RESTART JOB RECOVERY STATUS INFORMATION
(Mon May 25 05:18:45 PDT 1998)

SUMMARY INFO:

==> **ERRORS** - Shutdown error(s) found
==> AOK - There were no Restart errors
==> **WARN** - There may have been Jobs lost

Details covering these errors are given below.

>>>>>>>>> NQS Shutdown Status Information <<<<<<<<

NQS Shutdown Started at 05/24 15:01:36
NQS Shutdown Completed at 05/24 15:07:00
(FYI - NQS Shutdown took approx 355 secs)

Number of jobs executing when shutdown invoked - 13
Number of jobs queued when shutdown invoked - 135

==> N.B. There were shutdown problems -

Request <45317.triton>: NOT checkpointed, chkpnt(2) failed, errno=<17>.
Explanation: Error #17 - File exists
Checkpoint/restart filename already existed.

Job Name/Owner=acf/mdm9274, Rerunnable=yes, Restartable=yes
Job will be rerun when NQS restarted
Job had consumed 31008 secs out of a possible 170000 secs
Job had a remaining cpu time of 138992 secs
Email was forwarded to user mdm9274 indicating action taken

CPU lost to rerunnable job checkpoint failures was 31008 secs
CPU lost to all of the job checkpointing failures was 31008 secs

>>>>>>>>> NQS Restart Status Information <<<<<<<<

NQS Restarted at approx 05/25 05:18:03

All NQS jobs restarted successfully.

```
>>>>>>>>>  NQS Queued Jobs Status Information  <<<<<<<<<

==> Request 47153.triton was not found using /usr/bin/qstat shortly after system restart
==> Queued job 47153.triton may have been lost across NQS Shutdown/Restart.
   Please investigate. Job details are:
   Jobname=acldaemon.job, Owner=tranair, Queue=Msm_Csm@triton

==> N.B. The above lost input job requires attention.
```

5.4 Stopping DMF

When the system is shutdown the Data Migration Facility (DMF) needs to be gracefully stopped and terminated. Ideally, all active DMF requests need to be preserved and subsequently restarted or re-queued during system restart. Unfortunately, all the active dmget-s are aborted during DMF shutdown. This results in lost work.

Obviously the solution is for these dmget-s to be recoverable or should be allowed to complete (without new ones being started). A fair amount of anxiety surrounded this issue during our early days with UNICOS. To resolve this, local modifications to DMF were made to provide a reasonable request flushing period. A -f flag to ***dmdstop*** was added which allows the daemon to complete requests in progress within a 10 minute window before terminating. The reader is referred to the paper “Advancing with UNICOS Toward IRIX”, Section 2.10, presented at the 1998 Stuttgart CUG, for further information on this DMF modification.

5.5 Network connections

Checkpoints will fail if any process within a job session has an open socket (i.e. network connection). This problem becomes a very severe one if customers employ long duration connections such as can exist for ***remsh/rsh***. This problem had to be solved by modifying ***remsh*** to be recoverable across a system and/or NQS shutdown/restart. The reader is referred to the paper “Advancing with UNICOS Toward IRIX”, Section 2.18, presented at the 1998 Stuttgart CUG, for further information on this ***remsh*** modification.

There are other type network connections, but our experience has shown the one mentioned above to be the most prevalent at our site.

5.6 MPI and PVM

Applications that employ the MPI or PVM programming styles cannot be checkpointed. This is due to this type application having open sockets. Although this is not currently a problem at our site on the UNICOS system, we do expect it to become a severe problem for the Origin2000 IRIX system. The MPI and PVM is popular among our users on the Origin2000 for performance and portability reasons.

The effort for resolving this problem is very high. These type programs need to be, from the outset, coded to provide themselves with restartable files on a periodic basis. They must also be programmed so that they can catch and process a define signal (SIGSHUTDN for UNICOS) that then allows them to prepare for a system shutdown and then either lay dormant awaiting for checkpoint and the system recovery signal (SIGRECOVERY for UNICOS) or terminating after first re-submitting themselves to NQS for restart. All of this adds much complexity to the program, especially if portability needs to be accommodated. Most users will probably not go to such lengths.

5.7 Third Party Applications

Some third party codes employ a network license manager for licensing requirements. Unless they are sensitive to shutdown and restart signals for releasing and re-acquiring the license, they cannot be checkpointed successfully. Even if they are, they must also deal with re-acquiring a new license during recovery when there may be none available. There have been many failures of this kind seen in the past.

The solution to this is for the third party vendor to acknowledge and resolve the issues.

6.0 Configuring and Planning for the Test Environment

Before much can be done in this area there needs to be a clear understanding on how you intend to

- separate production and test system
- review new vendor and/or local modifications (new features and bugfixes)
- integrate and control vendor and/or local code modifications and new releases
- build the test system
- QA and test the new system
- ensure jobs checkpointed on the old system are successfully resumed in the new system
- install the new system for production
- ensure installing a new system can be done quickly (in minutes) versus hours
- ensure the new production system is identical to the one QAed and tested
- provide a contingency plan in the event the new system fails miserably

6.1 Separation between production and test systems

Our site decided to invest capital to configure two distinct and identical sets of file systems. They are referred to as the ‘A’ and ‘B’ file systems. Each of these sets contain five file systems - /dev/dsk/{rootA, usrA, srcA, localA, uspool} and /dev/dsk/{rootB, usrB, srcB, localB, uspoolB}. The missing ‘A’ for uspool in the A set is not a typo error. If the current production system resides, say on the As, the new system will be built and tested

on the Bs. If the current production system resides on the Bs, the new system will be built and tested in the /dev/dsk/{rootA, usrA, srcA, localA, uspoolB} file system set. When the test system is ready for production the current test system is made production except for the /dev/dsk/uspoolB. The /dev/dsk/uspool is *always* the production uspool file system. This is done so that NQS jobs can be recovered in the new production system. File system /dev/dsk/uspoolB is always used for the test system. However, we do at times run a test system using the production /dev/dsk/uspool when 'testing with users' and have user production file systems mounted. More on this subject later.

When switching from the test system to the production system, besides always using the file system /dev/dsk/uspool, certain other files from the previous production system are copied into the new production system to provide a seamless transition. A partial list of these files is provided as an example.

/etc/.cipher
/etc/acid
/etc/csainfo
/etc/debug
/etc/dumpdates
/etc/exports
/etc/group
/etc/hosts
/etc/issue
/etc/udb
/etc/wtmp
/usr/adm (whole directory is dump'ed and restored to hold onto security logs)

In summary, the A and B file systems alternate between production and test.

Switching from test to production is done in a matter of minutes, and we are assured that what we install as production is identical to what we tested.

The sequencing technique for switching between the As and Bs is shown in Figure 1.

7.0 Contingency plan

When switching from test to production systems the old production file system set is left unmounted (except for the old production /rootX file system) and remains intact for one week. This provides for a rapid fall-back (can be done within 45 minutes) in case the new production system fails badly.

The old production /rootX file system is mounted on the new production system to assist with the recovery of jobs that were checkpointed under the old production system. Without this, they will fail recovery because they may have been using files located in /rootX, such a ***sh*** and ***csh*** (which is very likely). After one week it is expected that no jobs exist that are using the old production /rootX file system.

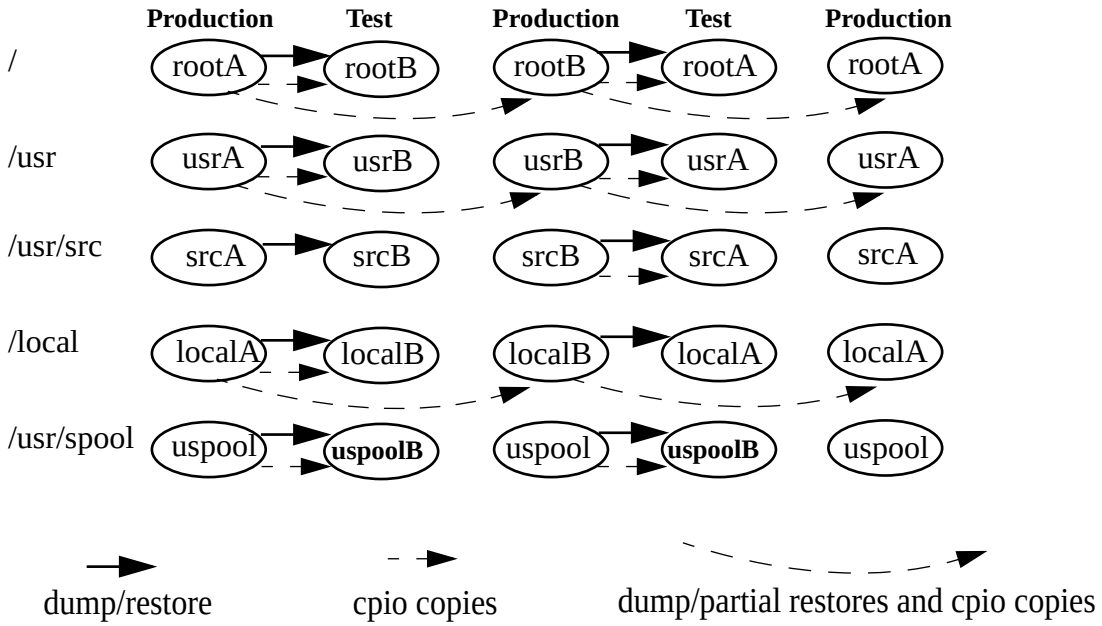

Figure 1

8.0 Safeguards Employed

To ensure the design described in Section 6.0 is followed to the ‘letter’ a substantial document was created that very carefully outlines and describes in great detail all that must be done. This document contains information on, for example:

- Building Local and Cray software
- Updating system PLs
- Generating system binaries and new kernel
- System testing
- Returning to a previous Menu System edition
- Changing existing mods and adding in new mods
- Purging a mod
- Re-adding mods
- How to build and install accounting commands
- Regenerating system binaries
- Installing new system for production
- NQE 3.x installation steps (new for us in UNICOS 10)

There are a huge number of things that have accumulated over the years that need doing before, during, and after a test session to provide safeguards for the production work.

Only the most noteworthy ones will be listed here.

Understandably, after a test session (and also after PM sessions) some things just may not be setup quite right. It could be that the person doing the testing left in a hurry, did not follow all the correct steps for returning the system to production (a process similar to a pilot's pre flight checklist), the test system simply trashed something pertaining to the production environment, or other external services, needed for production, are down because UNICOS testing completed earlier than was expected (it is not uncommon for multiple systems within the data center to be in test mode at the same time).

1. There are times, after sufficient stand-alone testing has been done, when it is necessary to have real live users also test things, This is referred to as 'testing with users'. When testing with users all user file systems are mounted along with the /dev/dsk/uspool and the production NQS directories/files are sequestered for their protection and to make production jobs (both executing and queued jobs) invisible to the test system. When returning to production these directories/files are made available in order to recover production work. An example of sequestering is using an ACL on the production NQS checkpoint directory to deny root access and defining another for the test system use.
2. When system is returned to production after a test period many sanity checks are performed before production is allowed to proceed to ensure all the test protections have been rescinded. For example, the production NQS checkpoint directory is accessible by root and is the same one that existed immediately prior to the production system shutdown.
3. Preserving the NQS queue and tape drive states across the test period.
4. Ensuring Silo tapes are configured 'up' before DMF and NQS are started. Jobs can fail if needed resources are not available and DMF will use external tape drives if Silo tapes are not 'up'.
5. Ensure that during testing the production root crontab is inactive and reversing the action when returning to production.
6. Ensure that production resource accounting is never run during test time.
7. During test time special NQS queue names are used which are different from the production NQS queue names. In the remote likelihood that test jobs are recovered/rerun when returning to production this prevents them from executing because of item 3. above.
8. Testing on different physical disk devices provides an almost foolproof method for safeguarding the production file system data. At times, in the past, it has been necessary to place the production disk devices in read-only mode to increase our assurance that the production data is beyond being corrupted by some errant test system procedure. This was done for example when testing the new NC1 filesystem structure released some time back.
9. Ensure all required user file systems are mounted and the high-level file system trees have same files (same as when production shutdown) and have correct ownership and permissions before production system is resumed. It does not take much for errors in this area to cause all executing and queued work to quickly evaporate.

10. Ensure that all external services such as STK Silo, data center File Servers are functional before starting production.
11. Ensure system dump areas have the proper identification headers. Without these a system dump will fail!

9.0 The UNICOS Guest system - UuU

The focus in this section is to arm the reader with information that can be used to avoid pitfalls that are easily fallen into when first starting to use the UuU or guest system feature.

Attempting to employ UuU for performing early testing on new systems may fail because resources needed by UuU cannot be made available by the host. This is caused by such things as memory in use by the host workload or simply because a judgement call is made that the UuU resource usage would degrade production work to poor service levels. It is important to note UuU usage is not free in this respect. The default is to make 10 attempts at getting the required memory before the guest startup aborts with insufficient memory. Any schedule for installing new systems that rely on UuU usage should take into account that UuU usage may not always be possible. When it is however, it is very beneficial in that it provides a method for allowing internal and real users the opportunity to test things during regular working hours without interrupting the production service. That is, both facilities can be provided simultaneously. This is viewed as a ‘real’ cost savings.

Currently the UuU facility is being used heavily to support early UNICOS 10 and the Year 2000 testing. For the Year 2000 it is strongly suggested that separate file systems be used with UuU to avoid future time-stamps from interfering with normal production activities.

In general, to operate UuU safely it is suggested that it use its own file systems as apposed to importing them from the host system using NFS. The i/o performance is also much better when UuU has its own file systems. This may require a capital investment for additional disk.

If the guest system is started while its root file system is mounted on the host system the guest system will panic during the boot phase. It would therefore, be advisable to always ensure the guest system’s file systems are all unmounted on the host before starting the guest boot operation.

When using NFS for importing production file systems into the guest it will be necessary to have the lockd(8) and statd(8) daemons active if guest customers require file locking across NFS. Bad press may result from guest customers who discover file locking is not doing what it does for them when running on the host production system! A past experience gave rise to potential data corruption.

The CPU and memory usage impact that UuU has on the host system is tracked by sar, so if there is any concern this data can be used to evaluate the issues being raised.

As we configure our guest system to use the host system's dump area (to conserve our disk expenses) it is always necessary to re-establish the dump headers after the guest system has been stopped. This is handled by our Health Monitoring System which checks the system dump areas (local mods provide us with two system dump areas) periodically. The check method use is shown below.

```
# ---------------------------------------------------------------
# Check for presence of dump header in the system disk dump areas.
# This should only be done if the guest OS is not executing.
# ---------------------------------------------------------------
#
# The command to test for the presence of a dump header is:
#
# /etc/cpdmp -n -i /dev/pdd/swapB_0
#
# Besides printing a message, this will exit with a return code of 0 or 2 if
# the dump header is present, 1 if it is not. To rewrite the dump header, the
# command is:
#
# /etc/bb /dev/dsk/swapB_0 | /etc/mkdmp -b /dev/dsk/swapB_0
#
# Note the /dev/pdd on the cpdmp command and the /dev/dsk on the other commands.
#
# If you want to check the other dump area, and you might as well, the commands
# would be:
#
# /etc/cpdmp -n -i /dev/pdd/dumpB
# /etc/bb /dev/dsk/dumpB | /etc/mkdmp -b /dev/dsk/dumpB
#
# ------------------------------------------------------------------------
# Programming note:
# Avoid /etc/cpdmp from writing to /etc/dump.log by using the -l option.
# ------------------------------------------------------------------------
```

Our experiences with UuU todate have been good and we are very satisfied. It is a shame that the feature took so long in coming. UuU is highly recommended to others for performing the kind of tasks mentioned above.

10.0 Summary

The difficulties of providing safeguards for a UNICOS production system against problems arising from testing new OS releases on a single machine have been explored and

shown to be full of pitfalls for a site with little UNICOS experience. These pitfalls have been discussed and some solutions for avoiding them have been offered.

With very careful planning and applying these solutions it is possible to provide a high degree of protection for the production system with minimal capital expenditure. Of course, there really is no substitute for a Test Machine - what luxury that would be.

The use of the UNICOS guest system for augmenting the testing of new OS releases has been discussed and indicated to be a very useful, cost effective, and a reliable feature.

The Good, the Bad, and the Ugly aspects surrounding this subject have been discussed so that others, including SGI/Cray, are suitably informed. Maybe SGI/Cray will address these aspects in IRIX.

11.0 Acknowledgments

Much of the background material given in this paper came from my associates at Boeing within the Technical Services organization. I wish to extend my thanks to them all in providing their time during the preparation of this paper. In particular I want to thank Mark Lutz, Bill Matson, Claude Asher, and Roger Harlow for putting up with all my questions that no doubt stretched their memories and patience at times.

The quality of our test environment is largely due to the dedication and hard work these people exhibit.