



THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-96-07

AP/Linux - Initial Implementation

**Andrew Tridgell, Paul Mackerras,
David Sitsky and David Walsh**

June 1996

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical.Reports@cs.anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

<http://cs.anu.edu.au/techreports/>

Recent reports in this series:

- TR-CS-96-06 M. Hegland, M. H. Kahn, and M. R. Osborne. *A parallel algorithm for the reduction to tridiagonal form for eigendecomposition.* June 1996.
- TR-CS-96-05 Andrew Tridgell and Paul Mackerras. *The rsync algorithm.* June 1996.
- TR-CS-96-04 Peter Bailey and David Hawking. *A parallel architecture for query processing over a terabyte of text.* June 1996.
- TR-CS-96-03 Brendan D. McKay. *autoson - a distributed batch system for unix workstation networks (version 1.3).* March 1996.
- TR-CS-96-02 Richard P. Brent. *Factorization of the tenth and eleventh Fermat numbers.* February 1996.
- TR-CS-96-01 Weifa Liang and Richard P. Brent. *Constructing the spanners of graphs in parallel.* January 1996.

AP/Linux - Initial Implementation

Andrew Tridgell, Paul Mackerras, David Sitsky and David Walsh
Australian National University

May 6th, 1996

Abstract

The AP1000+ is a distributed-memory parallel computer based on SuperSPARC processors, which incorporates message-passing hardware which can be accessed safely from user mode. We are in the process of porting the Linux kernel to this machine and extending it to support execution of parallel programs. This report outlines the motivation and background of this effort, and describes the current status and future directions for the work. The reader may also refer to our WWW page at <http://cap.anu.edu.au/cap/projects/linux> for up to date information on the progress of the port.

1 Introduction

The standard operating system provided with the Fujitsu AP1000+ is CellOS. CellOS supports the execution of a single program at any given time. The CellOS kernel is short-lived in the sense that the machine is reset and the CellOS kernel is reloaded anew each time a program is run. CellOS provides multitasking in that several separate tasks may run on each cell; the tasks all execute in a single address space, and may only be created at the beginning of the execution of the user's program.

Although CellOS is suitable for many tasks, we found that the lack of many features which are available in a modern workstation environment was creating difficulties and hampering development. For example, FORTRAN programs compiled with the Sun f77 compiler could not use direct I/O, because of the lack of the mmap system call. And the fact that the kernel is reloaded afresh each time a program is run makes it more difficult to implement a robust and efficient file system.

Accordingly, we decided to develop a new operating system for the AP1000+ which would provide an environment much more like that found on a modern Unix workstation, with facilities to support the execution of parallel programs—specifically, fast, low-latency communications between processes running on different cells. Our aim is to demonstrate the feasibility and desirability of such an environment on the AP1000+ (and other parallel computers).

Specifically, our requirements for the new operating system included:

- Multi-user support.
- Support for parallel programs, including fast, low-latency communications facilities.
- A long-lived kernel, which cannot be crashed by user programs.
- A complete set of system calls (as in a workstation environment).
- Sophisticated memory management facilities.
- Well-developed user libraries.
- Robust and fast operation.
- Low resource overheads.

After some investigation we settled on the Linux operating system as the basis for this new operating system for the AP1000+. We have ported the Linux kernel to the AP1000+ and added features to support communications between user processes on different cells using the facilities provided by the AP1000+ message controller (MSC+).

2 The Fujitsu AP1000+

Fujitsu's AP1000+ multicomputer provides a powerful parallel processing platform, comprising up to 1024 *cells* (processor nodes), linked by three networks: a wormhole-routed 2-D torus network (the T-net) with a bandwidth of 25MB/s per link, a global broadcast network (the B-net) with a bandwidth of 50MB/s, and a synchronization network (the S-net). The cell processors are 50MHz SuperSPARC processors each with 16MB or 64MB of memory. The system is controlled by a *host* machine (a Sun workstation), which can communicate with the cells via the B-net and S-net.

Each cell includes a message controller (MSC+) which controls the flow of data between the T-net and memory. The MSC+ supports several kinds of data transfer, including (among others):

Put: Data is transferred from a specified address on this cell to a specified address on a remote cell.

Send: Data is transferred from a specified address on this cell to a ring buffer on a remote cell.

Get: Data is transferred from a specified address on a remote cell to a specified address on this cell.

User programs can initiate transfers by writing directly to MSC+ registers without requiring kernel intervention. The MSC+ provides memory protection based on the notion of "contexts" as used in the definition of the SPARC Reference MMU (memory management unit) (SRMMU). A context represents a mapping of virtual addresses to physical addresses. Generally each process running under Unix or a Unix-like operating system would have its own context. Contexts are referenced by a context-ID (a small integer), and are represented by a tree-structured page table stored in system memory.

Requests sent to the MSC+ specify all addresses as virtual addresses in the context of the current process. This context-ID is sent in the message header and used by the receiving MSC+ to map virtual addresses on the remote cell to physical addresses. Invalid or unmapped addresses cause the MSC+ to interrupt the CPU in order for it to either map in a page (as for a normal page fault), or for it to signal an error to the MSC+. In the first case, the transfer is then continued; in the second, it is aborted.

3 Linux

Linux is a modern operating system developed by a loosely-knit group of people on the internet, led by Linus Torvalds of the University of Helsinki, Finland. It provides a POSIX compliant kernel and user environment, along with all the features that are expected in a normal workstation environment, such as virtual memory, multi-user operation, support for network protocols such as TCP/IP, etc.

Linux is freely distributed under the Gnu Public License and is available widely on the Internet. The open availability of the source code has contributed enormously to the development of the system.

Linux is most widely used on Intel i386 style processors, although it is also available for a range of other processor architectures, including Alpha, M68000, MIPS, PowerPC and SPARC. The SPARC port of Linux has features which particularly interested us for the AP1000+, including:

- SunOS 4 binary compatability
- Solaris 2 binary compatibility (still being developed)
- very good performance
- SMP support
- support for a wide range of Sun workstations
- modular architecture support.

These features combine to form a very attractive system on which to base a new operating system for the AP1000+. The binary compatibility with SunOS is particularly important

because it allows us to use a wide range of unmodified commercial compilers such as Sun's f77 FORTRAN compiler to produce binaries which can run on the AP/Linux cells. This was done with great success for the PCCM2 climate model.

Interest in Linux as an operating system for high performance computing platforms is not new. Both Digital Equipment Corporation and Silicon Graphics have shown considerable interest in Linux, with Digital now offering a range of its high performance Alpha workstations pre-installed with Linux. SGI have also recently announced an effort to port Linux to their high end parallel machines.

Other high performance platforms using Linux include clusters of Intel based workstations and various medium-sized SMP machines. For details on some of these projects see <http://www.cs.cornell.edu/Info/People/mdw/hpc/hpc.html>.

4 Progress so far

Initially, the porting efforts concentrated on getting Linux running on one cell. We use the B-net to transport IP packets to and from the host, which acts as a gateway. As the cells have no physical console device, we have implemented a virtual console. Characters written to the virtual console are sent over the B-net to the host, where they can be displayed in a window and/or written to a file.

Subsequently, we have added facilities to support parallel programs. A parallel program consists of a process running on each cell; all the processes in a parallel program use the same context-ID, so that they can access the facilities of the MSC+ directly.

At the time of writing, the initial AP/Linux port is fast approaching the date when it will go into production use at ANU. The system is now able to run complex parallel applications using either MPI or APLib for communications.

Some major milestones that have been reached include:

1. Boot from host machine over B-net, mounting filesystems via NFS.
2. TCP/IP between the cells and to external machines via FDDI or B-net.
3. Source-level debugging of the kernel using remote GDB.
4. System able to recompile itself.
5. Wide range of SunOS applications working.
6. Able to run more than one parallel program at once.
7. MPI ported and working.
8. Hostless APLib written and working
9. Parallel applications ported and running: PCCM2 climate model, blackhole simulation, parallel sort.
10. Simple gang scheduling working.

There is still a lot more work to do, but the progress achieved in the two months spent so far is encouraging.

5 AP/Linux components

AP/Linux is based around a single Linux kernel on each cell. The kernels do communicate, but each has its own copy of most data structures like the process table, file table etc. Each cell is assigned its own IP address and hostname, and all filesystems are mounted from the host via NFS. The following sections detail the components which are specific to the AP1000+.

5.1 Bootstrap/gateway program

The design of the AP1000+ is such that the initial code to be run on the cells is supplied by a program running on the host (the cells have no ROM storage for a bootstrap program). Consequently, we have developed a small program, called "bootap", which first resets the

cells and sends a bootstrap program to the cells over the B-net, and then sends the kernel image. Subsequently it acts as a gateway for IP packets, handles console messages from the cells, and acts as a gateway for messages between the cells and remote kernel GDB processes.

This program plays a somewhat similar role to the “caren” program under Cellos, with one important difference: after the bootstrap code is sent to the cells, bootap goes into a simple service loop servicing requests from the cells as they arrive. Thus most operations are initiated by the cells rather than by the host. With this design, we can print debugging messages from almost any part of the AP/Linux kernel.

5.2 B-net driver

The primary functions provided by the B-net driver in AP/Linux are as follows:

- Sending console messages from the cells to the host.
- Sending and receiving IP packets over the B-net.
- Communicating with remote GDB.
- Passing apblock requests to the front end (see below).

Notably, we do not support any user-level communications over the B-net, because we cannot provide reliable transfers over the B-net when the cell kernels are being debugged with GDB. When a cell reaches a kernel breakpoint, it cannot process any incoming messages until it is continued, so it discards any incoming data other than commands from the remote GDB. For IP-based communications, this doesn’t matter, because IP communications are not guaranteed to be reliable; higher-level protocols retransmit if necessary.

5.3 MSC+ driver

The MSC+ driver provides the kernel support necessary for user programs to be able to use the MSC+. It handles the various error conditions that can occur in the MSC+, and manages the 3 sets of ringbuffer registers which are provided in the MSC+, allocating them to parallel processes on demand.

The MSC+ driver for AP/Linux is considerably more sophisticated than that in Cellos, because it attempts to recover from any error condition that a user program can generate. In a multi-user environment, it is not acceptable for the kernel to panic because of an error in a user’s program. Instead, an appropriate signal should be delivered to the process, or the process should be terminated if a fatal error condition occurs.

Particular tasks undertaken by the MSC+ driver include:

- Ring buffer management:
 - Allocating memory for a process’s ring buffer and mapping it into the process’s address space
 - Setting up the MSC+ ring buffer registers
 - Handling ring buffer overflows.
- Handling page faults that are detected by the MSC+ in translating virtual to physical addresses.
- Software support for the MSC+’s send queues:
 - Refilling the send queues from the overflow buffer
 - Removing and restoring partial MSC+ commands from the send queue on a context switch.

As a special case, the MSC+ driver interprets writes to particular range of addresses as an instruction to send a signal to the local parallel task. In this way, a parallel task can send a signal to other parallel tasks within the same parallel process on other cells.

There are still some problems associated with the handling of some error conditions in the MSC+ which we are investigating.

Although the MSC+’s ring buffer read pointer registers are designed to be mapped into the user process’s address space, we do not use this approach in AP/Linux. Instead the

process updates a “soft” copy of the read pointer as it consumes messages from the ring buffer. The kernel updates the real read pointer register (in the MSC+) when it receives a ring buffer overflow interrupt. This approach proved to be simpler to implement, given that there may be more than three parallel programs running at any time. This approach may also make it possible to allow programs to use ring buffers of other than the four sizes supported by the MSC+ hardware.

5.4 T-net driver

The T-net driver provides inter-kernel communication, providing a put and get operations using the MSC+’s system send queue. Currently this is only used by the gang scheduling and task synchronisation code, but it is expected that these functions will be used extensively as the kernels become more tightly integrated.

5.5 BIF driver

The BIF driver is a IP networking driver which sends its messages using the B-net interface. The driver is very simple because of the extensive support provided by the IP layers in the Linux kernel. The driver uses DMA whenever possible to minimise memory copying overheads. IP packets can be sent to any cell or to the host.

We have measured a throughput using TCP/IP of over 7MB/sec between cells and a little over 1MB/sec to the host. We will be looking at ways to improve these figures in the future.

5.6 FDDI driver

A FDDI driver has been written for AP/Linux to take advantage of the FDDI option board designed and built locally by Paul Mackerras. Currently, we have one FDDI interface, attached to cell 9, which acts as a gateway between the FDDI ring and the B-net (in addition to its other functions).

The FDDI board provides much faster access to the host, with measured bandwidths of around 7MB/sec.

5.7 APBlock driver

Since our AP1000+ does not currently have disks attached to the cells, we have written a simple block device driver which provides functionality like that of a disk drive. Data written to the apblock device is sent to the host and stored on the host’s filesystems. For read requests, a small message is sent to the host, which reads the data and returns it over the B-net.

This mechanism is relatively slow, but nevertheless important at present as it is currently the only device which we can use for swapping (i.e., backing store for paging). Thus it is essential for running programs, such as PCCM2, whose memory requirements exceed the physical memory available on the cells.

It is also expected that this driver will be used as a basis for the real DDV disk driver when disks are finally installed on the machine.

5.8 Task management

Under Linux, the unit of execution is a “task”. Normally there is only one task corresponding to each process, but there can be several.

Tasks are divided into two classes under AP/Linux. Normal serial tasks run just as they would on any other SparcLinux machine and use the low numbered process IDs and context IDs. Parallel tasks are assigned high numbered process IDs and context IDs. A “parallel process” consists of a parallel task on each of the cells. Parallel tasks are treated differently by several sections of the AP/Linux kernel:

- scheduling decisions for parallel tasks involve some communication between the cells;

- parallel tasks within a single parallel process are allocated the same process ID and context ID on each cell;
- parallel tasks have access to the MSC+ hardware for sending and receiving messages;
- parallel tasks can send signals to other parallel tasks in the same parallel process on other cells.

It is expected that the concepts of a parallel task and a parallel process will be strengthened as the operating system is developed. In particular, we plan to support execution of a parallel process on subsets of the cells (i.e., partitions), as well as on the whole machine.

5.9 Parallel scheduling

After initial experiments with the normal scheduling provided by Linux it was found that some sort of cooperative scheduling was required to achieve reasonable performance when several parallel processes were running simultaneously, particularly if they waited for messages to arrive by simply polling the ring buffer until a message was present.

We have written a simple gang scheduler which seems to alleviate the observed problems. It works by increasing the scheduling priority of a parallel task if the master cell is running the same parallel task. The master cell is currently cell 0. The master cell simply does a broadcast put of the task ID to all other cells when it switches to a new parallel task, and the other cells use this information in calculating the priorities of the parallel tasks.

There is much more work to be done on strategies for parallel scheduling, which we plan to do once we have some data on real usage patterns for the machine.

5.10 Asyncd daemon

The asyncd daemon is a kernel thread which provides a mechanism for handling page faults caused by non-local memory accesses. This overcomes a fundamental assumption in Unix-like operating systems that page faults will only be caused by the processor's instruction stream.

In a machine like the AP1000+, a page fault may be caused by put and get requests which arrive asynchronously from other cells. When this happens, the MSC+ interrupts the CPU to handle the page fault. However, the interrupt handler cannot call the kernel's page fault routines directly, as they are not designed to be called from interrupt level.

Instead, the interrupt handler adds the details of the page fault to a queue, and then wakes the asyncd thread. The asyncd then handles the page fault, calling a callback routine in the MSC+ driver when the operation has completed. The MSC+ driver then tells the MSC+ to either continue or abort the operation.

We expect that this sort of mechanism will be used extensively when remote paging and remote fork capabilities are added to the system. However, there is a potential deadlock situation because the incoming or outgoing channel to the T-net is blocked while the page fault is being resolved. Therefore it is not safe to rely on communication over the T-net to resolve page faults. Unfortunately this could easily occur, for example if a file on a parallel file system is mapped into a parallel task's address space, and the parallel file system uses the T-net in implementing read and write requests.

5.11 Paralleld daemon

The paralleld daemon provides user-level management of the process of creating parallel tasks. This daemon runs as a privileged parallel task on each cell and accepts requests to launch parallel programs. At present we use UDP to send the requests to the paralleld. The requests include parameters such as the user ID, values of environment variables, and command line arguments.

Paralleld launches the parallel process by calling an extended clone() interface, added to AP/Linux for this purpose. This interface handles the creation of a process with an appropriate process ID and context ID, and the initialisation of appropriate data structures.

Paralleld also handles the creation and management of sockets for stdin, stdout and stderr for the parallel process. Data received from stdout or stderr of any of the parallel tasks are

sent to the process which sent the original request to launch the parallel program, and data received from that process is sent to the stdin of all the parallel tasks. Paralleld monitors the parallel process and its connection to the initiating process, killing any disconnected tasks.

The user is able to launch parallel programs using the “prun” utility, with control over logging and stdin handling. Prun is able to launch any SparcLinux or SunOS binary, without the requirement for any special format for parallel binaries. This opens up the possibility for a wide range of parallel programs based on standard SunOS languages and utilities.

5.12 APlib library

A CellOS-compatible APlib library has been written to provide an interface for message passing which is familiar to current users of AP1000 systems.

The library accesses the MSC+ hardware directly from user space, giving high throughput and very low latency. The user can choose to wait for incoming messages either by polling (continually reading the ring buffer until a message arrives), or by a combination of polling and signals.

Currently only a hostless implementation of APlib has been implemented, although it is expected that the host-side calls will be added in the near future. A hostless model is quite useful, however, as all cells have access to filesystems and I/O facilities provided by the Linux operating system.

5.13 MPI library

The MPI system developed by David Sitsky for the AP1000+ has been successfully ported to the AP/Linux environment. The MPI implementation uses some APlib ringbuffer support functions in order to maintain compatibility for hybrid APlib/MPI programs, but accesses the MSC+ registers directly for speed-critical operations.

The MPI system has been used to port the PCCM2 climate model and parallel blackhole simulation quickly and easily to AP/Linux, demonstrating the ability of AP/Linux to run large applications.

6 Benchmark results

One of the features of SparcLinux is the high performance of the kernel. The implementation has stressed low latency in the implementation of system calls and associated functions to minimize overheads, despite the sophisticated functionality provided.

Benchmark results comparing SparcLinux with SunOS 4.1.3 and Solaris 2.5 on identical hardware demonstrate that Linux provides much higher performance in almost every area tested. Linux is faster than SunOS by up to a factor of 10 in some cases.

Figure 1 shows the results of the “lmbench” operating system benchmark run on a Sun SparcClassic workstation, using the same hardware for each of the operating systems shown. The results demonstrate that Linux is highly competitive with these mainstream operating systems.

Parallel benchmark tests comparing AP/Linux to CellOS have not yet been performed.

7 Future directions

In the near future we expect to open up the AP/Linux implementation for use by the current users of CellOS on the AP1000 and AP1000+ at ANU.

After this initial goal has been reached and the system has stabilized we expect to begin work on more extensive parallel extensions to the base operating system.

One of the goals of our research is to investigate the feasibility and desirability of implementing single system image capabilities on multicomputers of this type. This includes the ability to move running processes between processors, to access remote devices, memory and disk resources and to integrate the TCP/IP and process management subsystems across the machine.

L M B E N C H 1 . 0 S U M M A R Y

 Processor, Processes - times in microseconds

OS	Mhz	Null Syscall	Null Process	Simple Process	/bin/sh Process	Mmap lat	2-proc ctxsw	8-proc ctxsw
Linux 1.3.97	50	14	8.9K	38.3K	56K	354	86	101
SunOS 4.1.3_U	49	124	18.3K	63.9K	110K	470	152	262
SunOS 5.5	50	31	33.7K	148.2K	274K	596	174	205

Local Communication latencies in microseconds

OS	Pipe	UDP	RPC/ UDP	TCP	RPC/ TCP
Linux 1.3.97	300	1016	1752	1376	2598
SunOS 4.1.3_U	890	1375	2287	1573	2804
SunOS 5.5	530	1563	2080	1354	2398

Local Communication bandwidths in megabytes/second

OS	Pipe	TCP	File reread	Mmap reread	Bcopy (libc)	Bcopy (hand)	Mem read	Mem write
Linux 1.3.97	8	4.0	23.5	17.4	18	25	41	37
SunOS 4.1.3_U	4	2.0	19.5	8.2	18	24	41	36
SunOS 5.5	8	7.0	12.6	19.5	18	18	40	36

Memory latencies in nanoseconds
 (WARNING - may not be correct, check graphs)

OS	Mhz	L1 \$	L2 \$	Main mem	TLB	Guesses
Linux 1.3.97	50	20	170	180	659	No L2 cache?
SunOS 4.1.3_U	49	20	175	183	-1	No L2 cache?
SunOS 5.5	49	-	-	-	-	Bad mhz?

Figure 1: lmbench results on a SparcClassic

Another very important aspect of our work will be the development of the parallel I/O capabilities of the system, with emphasis on the parallel filesystem. We will start with the HiDIOS parallel filesystem developed under Cellos for the AP1000, but expect it will need many substantial changes in order to work satisfactorily in the much richer environment provided by Linux.

8 Conclusion

The work done so far on AP/Linux is encouraging as it demonstrates the viability of a long lived, robust, feature-rich operating system on the AP1000+.

It is expected that this work will provide the basis for future parallel operating system development on the AP+ at ANU, and will provide a good environment for users of the machine, overcoming the limitations of the Cellos environment.

We also hope that collaboration with other groups interested in multi-computer implementations of Linux will lead to the development of more advanced operating system ideas which go beyond the AP1000+ environment.

9 Acknowledgements

We would like to thank the many contributors to Linux for providing an excellent operating system on which to base our research. Special thanks go to David Miller (the Sparc dude) for his extensive help with details of the SparcLinux kernel. David's support has been invaluable.

We would also like to thank the Fujitsu High Performance Computing Division for their continuing support of the CAP Program, and the CRC for Advanced Computational Systems for their support of the Pious project.