IPS Installation Guide

Version 5.0 (October 1, 1992)

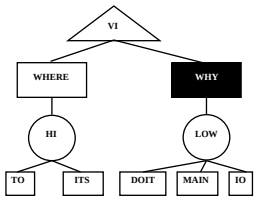
Jeff Hollingsworth

Bruce Irvin

Mike Mcmenemy

Barton P. Miller

University of Wisconsin-Madison



© Copyright 1990-1992 Jeff Hollingsworth, Barton Miller, Bruce Irvin. All Rights Reserved.

1. Introduction

This manual describes how to compile and install the IPS performance tool. Most of the installation of the software is automated and contained in the top level makefile. IPS consists of several major parts: user interface/master, slave analyst, instrumentation libraries, compiler drivers. Also, there are **optional** kernel modifications. These changes are not required, but can dramatically reduce the instrumentation overhead. This document describes the installation procedure for the supported architectures. A separate document called the *Guide to Porting IPS-2* describes how to port IPS to new machines. We also assume you are familiar with the ideas behind IPS-2. The paper in the April 1990 issue of *IEEE Transactions on Parallel and Distributed Systems* is a good place to start.

2. Minimum System Configuration

To compile and run IPS you will need at least one workstation (or X terminal) running the X Window package (version 11 release 3 or later). In addition, about 12 megabytes of disk space is required to build IPS. On machines that will run your instrumented applications, you will need enough disk space for the trace logs that IPS writes. The size of these logs varies greatly depending on the application and its runtime. Currently, IPS runs on VAX, DECstation, SPARC, Sequent Symmetry (Dynix and PTX), and Cray Y-MP (Unicos).

3. Loading IPS from Tape

IPS is distributed on a tape written in UNIX tar format. To load the tape onto your system type the following commands:

1.) `cd <location to put ips>`

2.) `tar xf <tape device name>`

The IPS source tree consists of several directories. Source code and object code are kept in separate directories. For example, the source for the user interface is in the `usri` directory, and the object code is in the directory `usri.<machine type>`. Where `<machine type>` is one of the supported machines (currently MIPS, SEQUENT, PTX, SPARC, CRAY, and VAX).

4. Compiling

The master, slave analyst, and the instrumentation libraries can all be compiled from a single makefile located in the root of the IPS source tree. Before compiling, two files should be edited: the configuration file `config.h`, and the `Makefile`. There are several parameters in `config.h`:

`SEQ_SLAVE` indicates if the slave analyst should be run as a sequential process on parallel machines. By default the slave runs as a parallel program whenever possible for CRAY, PTX, and DYNIX. This parameter should probably be left undefined.

`DEF_SLAVE` defines the default location for the slave analyst. This can be overridden via the ips*slave X resource.

`DEF_MOTD` defines the default location for the IPS message of the day. This can be overridden via the ips*motd X resource.

`DEF_SYSCTL` defines the default location for the IPS system control file. This can be overridden via the ips*sysctl X resource.

`IPS_DIR` defines which directory should contain the slave analyst and instrumentation libraries.

`APPLICATION_DEFAULT_DIR` defines the name of the directory to look for the applications defaults file for the IPS-2 user interface. On most systems this should be set to `/usr/misc/X11/lib/app-defaults`.

`X_VERSION` specifies which version of the X Window system is installed. Currently the only valid value is `1104`.

`ATSCC_VERSION` defines the version of Sequent's atscc compiler that is currently installed. (You can check in the directory /usr/xltr/c to see what version is installed).

The `Makefile` contains two parameters:

`OLD_COMPILER` specifies the location of the pcc based C compiler on Sequent machines. This is required to build the parallel programming library.

After the configuration file has been edited, type "`make all`" from the IPS root directory. This will compile the master, the slave analyst, and the instrumentation libraries. When completed, the files will be placed in the directory *bin.<machine type>*. The files in this directory need to be copied to the `IPS_DIR` directory. It is also possible to make *bin.<machine type>* a symbolic link to `IPS_DIR`, and then it is not necessary to copy items after compiling.

5. Application Default File

In addition to the compiled files, there is an X application default file that needs to be installed. The easiest place to put this file is in the application defaults directory (normally `/usr/lib/X11/app-defaults`). The file `Ips` located in the root of the IPS source tree should be installed in the application defaults directory with the name `ips`. If it is not possible to install the application defaults file in the default directory, it can be installed in any directory (as long as the file is called `Ips`). IPS users will then need to set the environment variable `XAPPLRESDIR` to that directory (or alternatively you can re-define the variable `APPLICATION_DEFAULT_DIR` in config.h to the directory where the application default file is installed). This application defaults file should work with color or black and white displays.

WARNING: If you do not install IPS in /usr/public/ips, you will need to edit the application default file, and change the definition of the `sysctl`, `slave`, and `motd` resources. See the users guide or the application default file for a description of these resources.

6. Other Items

Some window managers (twm and mwm for example) decorate windows with titlebars. IPS contains its own titlebars, and most users prefer disabling the window managers title bars for IPS windows. To do this for twm, add the following to your `.twmrc` file.

```
NoTitle {
    "ips"
}
```

For the mwm window manager, Add the line:

```
Mwm*ips*clientDecoration: -title
```

to your .Xdefaults file.

7. Summary

1.) Edit the file config.h.

2.) Edit the Makefile

3.) type `make all`.

4.) copy the files from *bin.<machine type>* to `IPS_DIR`.

5.) Edit the file `Ips`.

6.) copy `Ips` to `/usr/lib/X11/app-defaults`.

Appendix 1: Sequent Kernel Changes

The change to the Sequent kernel is to map the process's uarea into its address space. This change permits a process to read its elapsed process time at memory speed rather than needing a kernel call to get it.

Source Distribution

To install this change, the file `vm_pt.c` needs to be changed, and then a new kernel needs to be built. A diff of the file is located in:

`<IPS_DIR>/misc/kernel.SEQUENT/vm_pt.diff`.

The change consists of adding an `#include` of the file `<sys/seg.h>`, and then replacing the line:

```
*(int *) UAREAPTES(p) = *(int *) &Usrptmap[utab];
```

in the routine `vgetu` with the line:

```
*(int *) UAREAPTES(p) = *(int *) &Usrptmap[utab] | RW|PG_R|PG_M;
```

Binary Distribution

A compiled version of vm_pt.o is included in the IPS distribution. The following steps are required to build a new kernel using it.

```
#
# Iinstall file in the correct library.
#
 cd /usr/sys/conf
 mv OBJSlib.a OBJSlib.a.orig
 cp OBJSlib.a.orig OBJSlib.a
 cp <ips directory>/misc/kernel.SEQUENT/vm_pt.o /usr/sys/conf/.
 ar r OBJSlib.a vm_pt.o
 ranlib OBJSlib.a

# build the new kernel
 cd /usr/sys/<kernal name>
 make

# install the new kernel
 mv /dynix /dynix.bu
 cp <kernel name> /dynix
 sync; sync; reboot
```

Appendix 2: VAX (BSD and ULTRIX) Kernel Changes

To allow faster access of the system (wall) clock, you must change four of the kernel source files. A copy of the following code fragments is in the file: `<IPS_DIR>/misc/kernel.VAX/changes`.

In file `sys/vax/mem.c`:

After the `#include "uio.h"`, add the following two lines:

```
#include "ioctl.h"
#include "kernel.h"
```

After procedure `mmwrite()`, add the following lines:

```
mminit()
{
    char *timemem;
    register int *pte;
    extern struct pte Sysmap[6*NPTEPG];

    /*
     * Map memory so time structure is user readable
     */
    timemem = (char *)(((int) &time) & ~(0x80000000));
    pte = (int *)(Sysmap + btop( timemem ));

    *pte = (*pte & ~PG_PROT) | PG_URKW | PG_V;
    if(((int) timemem & PGOFSET)+sizeof(struct timeval) > NBPG) {
        /* have to map two pages */
        pte++;
        *pte = (*pte & ~PG_PROT) | PG_URKW | PG_V;
    }
}

/*ARGSUSED*/
mmioctl(dev, com, data, flag)
    register dev_t dev;
    caddr_t data;
{
    switch (com) {
    case MMGETTIMEADR:
        * (caddr_t *) data = (caddr_t) &time;
        break;
    default:
        return (-1);
    }
    return (0);
}
```

In file `sys/vax/machdep.c`:

After the call to procedure `configure()`, add the following line:

```
mminit();       /* do extra mm device initialization */
```

In file `sys/h/ioctl.h`:

At the very end of the file, add the following line:

```
#define MMGETTIMEADR    _IOR(m, 1, caddr_t)
```

In file `sys/machine/conf.c`:

In the initialization part of the declaration for `struct cdevsw`, the entry marked `/*3*/` must be changed.

FROM:

```
nulldev,        nulldev,        mmread,         mmwrite,        /*3*/
nodev,          nulldev,        nulldev,        0,
mmselect,       nodev,
```

TO:

```
nulldev,        nulldev,        mmread,         mmwrite,        /*3*/
mmioctl,        nulldev,        nulldev,        0,
mmselect,       nodev,
```

Note that the only change is on the 2nd line, first entry.

Finally, change the line that looks like:

```
int     mmread(),mmwrite();
```

to the following:

```
int     mmread(),mmwrite(),mmioctl();
```

Appendix 3: DECstation Kernel Changes

The directory `<IPS_DIR>/misc/kernel.MIPS` contains diffs for the files. The following files need to be modified for the DECstation 3100 (Ultrix version 3.1d, also called Ultrix Worksystem V2.1):

/sys/h/ioctl.h	include file for ioctl's
/sys/h/shm.h	shared memory include file
/sys/mips/machine/mem.c	routines for special /dev/null ioctl
/sys/mips/machine/conf.c	set-up for memory device
/sys/sys/init_main.c	call to configure memory device
/sys/sys/vm_pt.c	changes to page tables
/sys/sys/vm_smem.c	changes to shared memory

In the file *ioctl.h*:

Add these two new ioctl operation codes to the end of the file:

```
#define MMGETTIMEADR    _IOR(m, 1, caddr_t)             /* get timeval addr */
#define MMGETUADR       _IOR(m, 2, caddr_t)             /* get u addr */
```

In the file *shm.h*:

After this line:

```
#define SMNOSW  0100            /* Lock segment in memory */
```

Insert this line:

```
#define SMMAPU  0x400           /* segment used for u-area mapping */
```

In the file *mem.c*:

After this line:

```
#include "../mips/cpu.h"
```

Insert the following lines:

```
#include "../h/kernel.h"
#include "../h/ioctl.h"
#include "../h/ipc.h"
#include "../h/shm.h"
#include "../h/proc.h"

#define DWN(x) ((int)(x) & ~(CLBYTES-1))  /* make cluster page aligned */
int ips_wcsmid; /* shared memory id for ips wallclock */
int ips_usmid;
mminit()
{
        ips_wcsmid = vm_system_smget(DWN(&time), CLBYTES, 0444);
        if (ips_wcsmid == -1) {
          printf("error (ips_wcsmid)=%X0,
                 u.u_error);
                 }
}
```

© Copyright 1990 Mike Mcmenemy. All Rights Reserved.

```
mmioctl(dev, com, data, flag)
        dev_t dev;
        register caddr_t data;
{
    switch( com ) {
      case MMGETTIMEADR:
        {
          int time_offset; /* offset into page */
          char *smat();
          struct timeval *timeptr = 0;
          time_offset = ((int) &time - DWN(&time));

          if ((timeptr = (struct timeval *) smat(ips_wcsmid, 0, SHM_RDONLY)) ==
              (struct timeval *)-1)
            goto bad;

          *(caddr_t *) data = ((caddr_t) ((int) timeptr + time_offset));

          break;
        }
      case MMGETUADR:
        {
          int u_offset; /* offset into page for u area*/
          char *smat();
          struct user *uptr = 0;
          struct proc *p;
          struct smem *sp;
          struct pte *upte;

          extern struct smem *smconv();

          union pte_words {
            int hardware_word;
            struct pte whole_pte;
          };

#define Hard(pte)       (((union pte_words *)(pte))->hardware_word)

          p = u.u_procp;

          if (!p) {
            printf("no proc structure for process0);
            goto bad;
          }

          ips_usmid = vm_system_smget(DWN(&u), CLBYTES*2, 0444);
          if (ips_usmid == -1) {
            printf("error (ips_usmid)=%X0,
                   u.u_error);
          }

          u_offset = ((int) &u - DWN(&u));
```

```
            sp = smconv(ips_usmid, 0); /* get pointer to memory segment */
            sp->sm_flag |= SMMAPU;

            upte = sp->sm_ptaddr;
            Hard(upte) =  Hard(&u.u_procp->p_addr[0]) & PG_PFNUM
              | PG_UW | PG_V | PG_M;
            upte++;
            Hard(upte) =  Hard(&u.u_procp->p_addr[1]) & PG_PFNUM
              | PG_UW | PG_V | PG_M;

            if ((uptr = (struct user *) smat(ips_usmid, 0, SHM_RDONLY)) ==
                (struct user *)-1)
              goto bad;

            *(caddr_t *) data = ((caddr_t) ((int) uptr + u_offset));

            break;
          }
        default:
          return (-1);
        }

      return(0);

      bad:
      return(-1);

  }
```

In the file *conf.c*:

After this line:

```
int     mmread(), mmwrite();
```

Insert the line:

```
int mmioctl();
```

After these two lines:

```
/* Memory devices */
nulldev, nulldev, mmread,     mmwrite, /*2*/
```

Change this line:

```
nodev,       nulldev, nulldev, 0,
```

To this:

```
mmioctl, nulldev, nulldev, 0,
```

In the file *init_main.c*:

After this section of code:

```
/*
 * Configure the system.
 */
        configure();
```

Add the following line:

```
        mminit();  /* initialize memory device */
```

In the file *vm_pt.c*:

After the two lines:

```
        newu->u_ssize != p->p_ssize || newu->u_procp != p)
            panic("vgetu");
```

Add the following section of code:

```
        for (i = 0; i < p->p_smcount; i++) {
             union pte_words {
            int hardware_word;
            struct pte whole_pte;
             };

    #define   Hard(pte) (((union pte_words *)(pte))->hardware_word)

           if (p->p_sm[i].sm_p->sm_flag & SMMAPU) {
             {
            struct pte *upte;

            upte = p->p_sm[i].sm_p->sm_ptaddr;
            Hard(upte) =  Hard(&newu->u_procp->p_addr[0]) & PG_PFNUM
                 | PG_UW | PG_V | PG_M;
                upte++;
            Hard(upte) =  Hard(&newu->u_procp->p_addr[1]) & PG_PFNUM
                 | PG_UW | PG_V | PG_M;
             }
           }
        }
```

In the file *vm_smem.c*:

After this section of code:

```
        /* if first time, then fill */
        upte = sp->sm_ptaddr;
        if (upte->pg_v == 0) {
            num_ptes = btop(size);
```

Add the following code:

```
            if (sva == UADDR) {
              return (smid);
            } else
```