Using Standard Unix Accounting Data for Chargeback Accounting

Don Koch
Gail Ridgley
Carl Meredith
SAS Institute Inc.

Although many sources, both native and third-party, of performance and measurement data for Unix systems are available today, due to a lack of consistency, accuracy and quality few are suitable for use in chargeback accounting systems. All Unix varieties include "standard" accounting utilities that use a single source of data for process accounting. This paper takes a closer look at native Unix accounting data and it's suitability for use as part of a chargeback accounting architecture.

Introduction

Unix servers continue to see significant growth in capacity, throughput, scalability and reliability. Today, high-end Unix servers are mainframe-class systems that include partitioning, high-availability, large-scale storage and high-speed I/O channels. They are being placed in well-managed raised-floor environments and cared for with best-of-breed practices. For many businesses, these practices include chargeback accounting.

Nearly all Unix systems include facilities for generating, managing and reporting on accounting data. A somewhat standard set of system calls, binary executables, shell scripts and log files comprise a suite of tools needed to accomplish these functions. This suite is referred to as **UAU** (Unix Accounting Utilities)**.**

The objectives of this paper are to assess the suitability of using UAU for chargeback accounting and to provide the necessary background for those who wish to pursue the use of this data for capacity planning, chargeback or performance management. This is not a tutorial on using UAU; rather we focus on the accounting data itself, with the goal of being able to use it in any performance data warehouse.

Chargeback Accounting Data

Chargeback accounting basically consists of two component: costing and billing [GEWU88] [HOFF85] [BROW95]. Accounting data must serve these two purposes. There are many methods for implementing costing and billing, but there are three requirements that all accounting data must satisfy in order to be useful for chargeback. These three requirements will be used to evaluate the suitability of Unix accounting data.

Accounting Data Requirements

1. Work and users are identifiable
2. Relative usage can be measured
3. Measurements are consistent

The first requirement ensures the compute workload can be associated with a place in an organization chart. This is our work-to-user mapping. In other words for each piece of work a computer does, can we identify the entity (person, department or division) that executed the work?

The second requirement addresses mapping of work to cost. It is not necessary, and most of the time impossible, for accounting data generated by computers to totally reflect the cost of providing a service. Service costs include much more than hardware and software. The second requirement says that accounting data needs to somehow reflect relative resource usage. Although CPU utilization may not directly correlate with the overall cost of a service, it may be a good measure of relative utilization of a service. Therefore, CPU utilization may be a good way to equitably distribute expenses at billing time. For a web server, the number of bytes transferred may be a more appropriate way to distribute expenses. A service that relies on an RDBMS may require some combination of disk utilization, CPU and I/O to accurately measure relative usage. We need to

know if the data contains enough information to assign relative utilization of a service to users.

The third requirement ensures that if I run the same job on the same system at a different time I would be billed the same. Consistency is actually very difficult to achieve for most metrics used for chargeback. For example if CPU utilization is used to determine relative usage, then a job's CPU time will depend on other system loads (unless you have the system to yourself). Will Unix accounting data contain consistent metrics?

Characterizing Data

Management data from computer and network systems all have some characteristics in common. The variables (or columns) fall into two broad classes: identification (ID) variables and analysis variables. ID variables include elements like machine name, interface number, userid, disk name, etc. The DATETIME variable (or date and time variables) is a special ID variable which is required and is used to derive other time-based ID variables such as HOUR, SHIFT, DOM (day of month), etc. Analysis variables include elements like CPU utilization, swap rate, disk queues, etc.

ID variables are used for observation (row) grouping and classification and to satisfy requirement 1, identification of work and users. Analysis variables are used to determine how much of each resource is being used, satisfying requirement 2, measurement of relative usage.

Chargeback Accounting in Context

Although UAU are meant to be used as a complete stand-alone solution it is more likely that you will want to integrate UAU data with your entire management architecture. To help place chargeback accounting in context of an overall performance and capacity management architecture, the block diagram in Figure 1 shows one possible view of chargeback accounting as a component of a comprehensive architecture.

On the left, the data sources include accounting, performance and network data. Network data is collected from a variety of devices such as routers, probes and switches. Business information is usually supplied from database extraction or by manual entry. It can include accounting codes, divisional affinities for devices, filesystems or userids, or any business-related information that will help identify the workload being measured.

The performance data warehouse (PDW) consolidates, organizes and manages many different sources of data. To the right is the information that is consumed by analysts, executives and users. That block represents the queries, reports and invoices that really interest people.

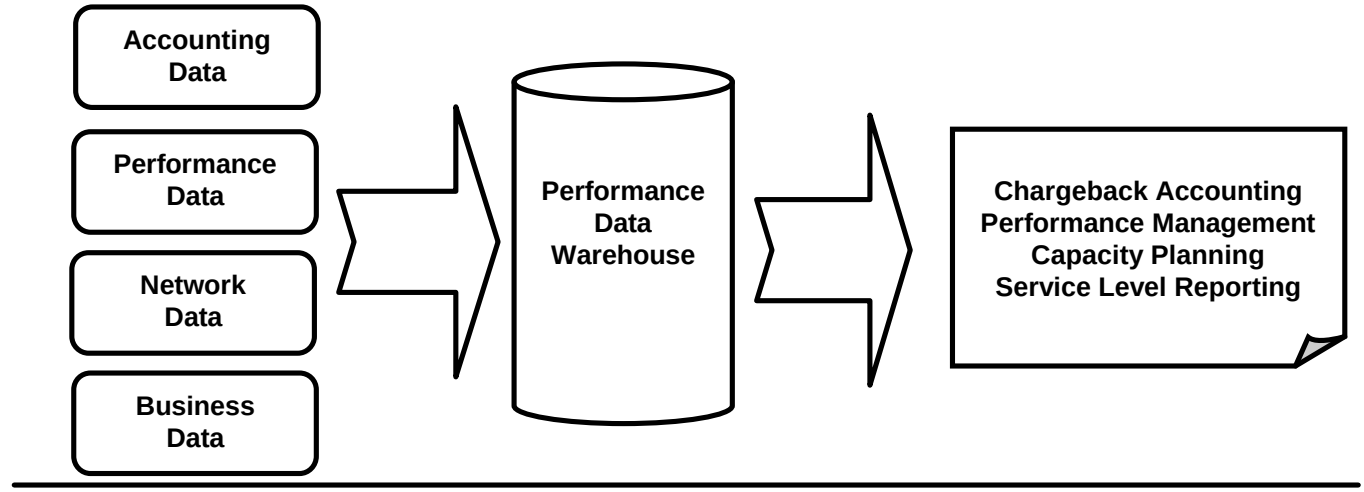

Figure 1. Chargeback in Context

Unix Accounting Overview

UAU evolved in a way similar to Unix itself. There are two branches in the evolution of Unix accounting: BSD and System V. Further, vendors may have implemented their own modifications and enhancements. Although there are differences among the Unix varieties within a particular branch (BSD or System V), their accounting systems are fundamentally the same. System V derivatives are the most common and are the ones we will be discussing in this paper. It would not be difficult to apply much of the information presented here to BSD type accounting systems.

Beyond accounting, UAU are often used for monitoring system usage, security management, troubleshooting and performance analysis. UAU encompass the following functions:

- Process Accounting
  - user and group ids of process owner
  - process begin and elapsed time
  - command name
  - memory used
  - bytes transferred
  - miscellaneous information
- Disk Accounting
  - user and group ids of files
  - number of blocks used by the user's files
- Connect Accounting
  - how long a user is logged in
  - which tty was used
  - system reboots
  - accounting software stop and starts
- Fee Calculations
  - include fees for special services
- Reports
  - daily and monthly summarization reports

Accounting records are written to the raw accounting files for process termination, user login, user logout, and system reboot. On a daily basis, the `runacct` utility is run to create cumulative or total accounting files and to create daily reports from the raw files. On a monthly basis, `monacct` is run to produce monthly summarization reports for billing purposes. Disk accounting is performed by the `dodisk` command. This command gathers disk utilization statistics at the time `it` runs. Fees for special charges are stored in `/var/adm/fee` file and incorporated into summary reports.

There are several shortcomings to using the UAU "out of the box". For instance, ad-hoc analysis it is not possible and supplied reports are not customizable. It is not possible to combine the UAU raw data with other data sources such as those from mainframes, PCs, networks or business processes for further analysis. In addition, bill preparation is not part of UAU. The intention is that each site will generate their billing from monthly reports. You would need to write bill preparation programs or buy software that will do that for you.

If you wish to use the Unix accounting facilities as provided by the Unix vendor, then the vendor's documentation [HEWL92] [SUNS93] provides all of the information you need to use UAU. We are assuming that you are interested in UAU data because you want to integrate that data into a comprehensive and more robust set of tools for doing chargeback accounting.

Unix Accounting Data

Figure 2 lists some of the important files that are included in UAU. Of these, `/var/adm/pacct` is the most useful for accounting. The `/var/adm/pacct` only exists if process accounting has been enabled on your system. The `/etc/passwd` and `/etc/group` files are needed to map UID (user id) and GID (group id), both integers, to user and group names. Only UID and GID are stored in the `/var/adm/pacct` file. Mapping of ID's to names is done at the time `/var/adm/pacct` is processed using SAS formats which are created from `/etc/passwd` and `/etc/group`.

File	Description
/var/adm/pacct	Process accounting data. One record per process, written upon process termination.
/var/adm/wtmp	Connect time accounting records. One record per tty login.
/etc/passwd	Maps UID (an integer) to user name. Contains other user-specific information.
/etc/group	Maps GID (an integer) to a group name. Also contains group affinities.

Figure 2. Accounting Data Files

Connect-time information is kept in `/var/adm/wtmp`. This file keeps login/logout times and port information. Today this information is not very useful for chargeback. It may have been in the days of green-screen terminal applications: users accessing Unix systems with X-Windows are logged multiple times and users logged in with `rsh` are not recorded at all in `/var/adm/wtmp`.

Process Accounting Data

Process accounting data is stored in binary file `/var/adm/pacct`. Sometimes an *n* is appended the filename indicating previous versions of the file. The `ckpacct` utility is responsible for creating versioned files. It executes regularly from `cron` to make sure the `pacct` file doesn't get too big. With the disk storage available today it probably isn't necessary enable versioning if the pacct file is processed nightly.

It is important to understand that a process accounting record is written to `/var/adm/pacct` by the kernel ***only*** after the process has completed [GLEN94]. This is only a problem when accounting is started and stopped, for long-running processes or ones that never terminate. Under normal circumstances process accounting is enabled at boot time and is not stopped until `/var/adm/pacct` is processed into the PDB or the system reboots. System reboots are infrequent, especially on high-end systems that demand 24x7 availability. There are few if any workloads that start prior to process accounting that you will want to include in chargeback accounting. For long-running processes that span processing of `/var/adm/pacct` (usually every 24 hours), they will eventually show up with the correct resource utilization in a subsequent day's log.

The table in Figure 3 lists the data elements in each record of `/var/adm/pacct` for HP-UX 9.0. Other Unix varieties will have a slightly different record layout so check the acct(4) man page for details. The diagram Figure 4 shows the binary record layout. You can also find the record layout details in `/usr/include/sys/acct.h` but without the descriptive details the man page contains. Many of the data types in Figure 3 are derived from c basic types. To determine what the base is for a derived type, look in `/usr/include/sys/types.h`.

In Figure 3, a "Unix Name" of "N/A" indicates an element in the SAS dataset that is not found in `/var/adm/pacct` but is derived from an element in `/var/adm/pacct` or is a supplied classification variable. One such classification variable that is not found in `/var/adm/pacct` is the name of the system (MACHINE) running the accounting software. We instantiated MACHINE using a SAS macro variable.

Unix Name	SAS Name	Typedef	Base type	Bytes	Description
ac_flag	ACCFLAG	char	char	1	Accounting flag: 01=fork, 02=super user, etc.
ac_stat	ACCSTAT	char	char	1	Process exit status
ac_uid	ACCUID	uid_t	unsigned short	2	User ID
ac_gid	ACCGID	gid_t	unsigned short	2	Group ID
ac_tty	ACCTTY	dev_t	unsigned int	4	Controlling tty
ac_btime	ACCBTM	time_t	unsigned int	4	Command begin time in seconds since 1/1/1970.
ac_utime	ACCUTM	comp_t[1]	unsigned short	2	User CPU time in hundredths of seconds
ac_stime	ACCSTM	comp_t	unsigned short	2	System CPU time in hundredths of seconds
ac_etime	ACCETM	comp_t	unsigned short	2	Elapsed time in hundredths of seconds
ac_mem	ACCMEM	comp_t	unsigned short	2	Memory usage in pages (1 KB)
ac_io	ACCIO	copm_t	unsigned short	2	Chars transferred by read or write
ac_rw	ACCRW	comp_t	unsigned short	2	Blocks read or written
ac_comm	ACCCOMM	char	char[]	8	Command name, array of 8 characters
N/A	ACCUSER	character	SAS character	32	User name (lookup ac_uid in /etc/passwd)
N/A	ACCGRP	character	SAS character	32	Group name (lookup ac_gid in /etc/group)
N/A	KCORMIN	numeric	SAS numeric	8	Memory usage in K core minutes
N/A	AVEMEM	numeric	SAS numeric	8	Average memory used (Kbytes)
N/A	MACHINE	character	SAS character	32	Name of system
N/A	DATETIME	numeric	SAS numeric	8	Derived from ACCBTM.

[1] Pseudo-floating point number; base-8 exponent in the high order 3-bits and fraction in the low order 13 bits; 0 to 1.3743895E+11.

Figure 3. HP-UX 9.0 pacct fields

	Octal Address, Low Byte							
High Byte	**_0**	**_1**	**_2**	**_3**	**_4**	**_5**	**_6**	**_7**
0_	ac_flag	ac_stat	ac_uid		ac_gid		padding	
1_	ac_tty				ac_btime			
2_	ac_utime		ac_stime		ac_etime		ac_mem	
3_	ac_io		ac_rw		ac_comm…			
4_	…ac_comm				N/A			

Figure 4. HP-UX 9.0 pacct binary record layout

A Closer Look at Memory Usage

Memory usage, ac_mem, represents the cumulative amount of memory a process uses while running. Memory usage is estimated by accumulating the number of 1 Kbyte memory pages a process is using at each system clock interrupt. Only memory resident pages of a process are counted; pages in swap space are not counted. A portion of shared code and data is added to a process's count by dividing the number of shared pages by the total number of processes using them.

A derived metric called "K core minutes" provides a combined measurement of the amount of memory used in KB and the length of time it was used in minutes. A long running program may use less memory than a short running program but still show a larger K core minutes. The units for K core minutes are in Kbyte-Minutes. It is calculated as

$$\text{KCORMIN} = \frac{(\text{ACCSTM} + \text{ACCUTM}) * \text{ACCMEM}}{60}$$

Another useful derived metric is an approximation of the average memory that the program required. It is calculated as

$$\text{AVEMEM} = \frac{\text{ACCMEM}}{(\text{ACCSTM} + \text{ACCUTM})}$$

See Figure 3. for the meaning of the abbreviations used in these formulae.

Disk Accounting Data

The UAU also include three programs to gather and report on disk utilization. Disk accounting data includes the name and ID of the user and the number of 512 byte blocks the user's files occupy. Disk information is gathered by the `dodisk` shell script that invokes `acctdusg` and `diskusg`. The information is stored in `/var/adm/acct/nite/disktacct` file.

Evaluation of the Data Source

An Example

Figure 6 is a 24-hour summary of system usage for each user that used the system. Figure 4 is the same data for the highest user, srcmgr, detailing usage by command. The metrics we chose to report on are ones that show some resource utilization. This data is from one of our software build machines. The nightly builds are executed by the srcmgr userid so it is not surprising that it is the highest user. The values in both tables are sums for an entire 24-hour period. Thus in Figure 6 the 'Chars xferd' value for 'cp' of 123,817,006 characters means that 123MB of data were copied (read and written) by srcmgr during the 24-hour period. If we were interested in performance management we could have chosen means instead of sums.

```
,                    ,Elapsed time,Chars xferd , Avg memory ,System time , User time  ,
,                    ,    SUM     ,    SUM     ,    SUM     ,    SUM     ,    SUM     ,
,User                ,            ,            ,            ,            ,            ,
,tstuser             ,     2178.66, 25493329.00,   215728.00,       20.22,      130.10,
,srcmgr              ,   773673.34, 10333205618, 39631358.00,     4926.02,    41262.80,
,daemon              ,       12.08,    89799.00,      607.00,        0.16,        0.11,
,root                ,     1295.14,  7601382.00,    16742.00,       43.70,       14.54,
```

Figure 5. System usage by user for one 24-hour period.

```
„ffffffffffffffffffff…ffffffffffff…ffffffffffff…ffffffffffff…ffffffffffff…ffffffffffff†
,                     ,Elapsed time,Chars xferd , Avg memory ,System time , User time  ,
,                     ‡ffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,                     ,    SUM     ,    SUM     ,    SUM     ,    SUM     ,    SUM     ,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,Cmd name             ,            ,            ,            ,            ,            ,
‡ffffffffffffffffffff‰            ,            ,            ,            ,            ,
,ar                   ,        2.36,  1512000.00,      324.00,        0.31,        0.16,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,bind                 ,     6972.00, 24167840.00,  1118508.00,      371.52,     1660.81,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,bldldarg             ,       15.05,   874076.00,    79871.00,        0.36,       13.12,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,bsh                  ,       25.98,  2147328.00,     4220.00,        0.63,        2.27,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,cc                   ,    33743.38, 17707682.00,   653278.00,       39.80,      100.89,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,cp                   ,      427.02,123817006.00,   395464.00,       89.95,       35.51,

                    . . . Some table rows removed to save space . . .

‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,rm                   ,      495.77,        0.00,  2172110.00,       25.69,      372.96,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,sh                   ,   284370.00, 10022130.00,  2396305.00,      763.94,      285.60,
‡ffffffffffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffffˆffffffffffff‰
,wc                   ,      287.56,746894683.00,   715445.00,       27.41,      209.03,
Šffffffffffffffffffff‹ffffffffffff‹ffffffffffff‹ffffffffffff‹ffffffffffff‹ffffffffffffŒ
```

Figure 6. System usage for userid=srcmgr for one 24-hour period.

The Three Data Requirements

How well does Unix accounting data satisfy the three requirements that accounting data must satisfy; namely, work and users are identifiable, relative usage can be measured and measurements are consistent? The first requirement is satisfied since we can clearly identify users (or groups if you like) and the processes that they execute. It's straightforward to map users and groups to a new ID variable that specifies organizational entity using SAS formats at process time. This new variable is then used to drive the billing process.

The accounting data can be used to indicate relative usage thus satisfying requirement 2. Although the variety of usage data that `/var/adm/pacct` contains may not be as robust as other accounting data sources (some may consider it anemic) we believe it provides enough information to fairly charge for system resources. It is possible to charge at different rates for memory (ACCMEM), CPU (ACCUTM + ACCSTM) and disk (ACCIO) or use some combination of these metrics.

We were unable to demonstrate significant inconsistency in our data when the same command was executed multiple times. This may be because our systems weren't heavily loaded or it may have to do with the types and numbers of workloads that were running during the test. This doesn't mean you shouldn't worry about consistency but it probably means that under normal circumstances the data will be consistent.

Data Shortcomings

When using UAU data, keep in mind that command names are limited to 8 characters so that commands with names greater than 8 characters will be truncated in `/var/adm/pacct`. It is not possible to distinguish between two commands with the same first eight characters.

Summary

It is possible to use UAU data in chargeback architectures since that data satisfies the three basic requirements of chargeback data. It is identifiable, equitable and consistent. For many shops, this may be your only source of data suitable for chargeback without an investment in third-party software or homegrown tools. The process accounting data can be included into any performance data warehouse once the binary records are parsed. The only homegrown tool that is needed is a program to read the process accounting data log.

References

[GEWU88] G. Gewurtz, “Chargeback Concepts,” CMG Conference Proceedings, 1988, pp. 617-619.

[HOFF85] F.W. Hoffman and W.L. Fuerst. “Designing and Using Chargeback Systems: A Tutorial,” CMG Conference Proceedings, 1985, pp. 586-588.

[BROW95] T. Browning. “Capacity Planning for Computer Systems,” AP Professional, 1995, pp. 58-63.

[HEWL92] “HP 9000 Series 800 Computers - System Administration Tasks,” Hewlett Packard, 1992, Chapter 12.

[SUNS93] “SunOS 5.2 Administering Security, Performance, and Accounting,” SunSoft, 1993, Chapter 10.

[GLEN94] Glenski, J.W. “Accounting for Long Running Unix Processes”, CMG Conference Proceedings, 1994, pp. 837-845.

Appendix: Sample Java program to process a pacct file.

```
/**
 * pacct2csv.java
 *
 * @param hostname, name of host which created the pacct file.
 * @param filename, name of pacct file.
 *
 * This is a very simple program to convert a pacct file to
 * an ASCII comma-separated variables file.  This will only
 * work on a pacct file created on HP-UX 9.0. The program will
 * run on any Java 1.1 platform.
 *
 * The hostname is a required parameter since a pacct file doesn't
 * contain the host name and hostname is a variable that is needed
 * as a column in the CSV file.
 *
 * Usage: 1. compile with "javac pacct2csv.java".
 *        2. run with "java pacct2csv host /var/adm/pacct"
 *           where host is the name of the host that created pacct.
 *
 * Disclaimer: This program is an example only. It is not guaranteed
 * to work correctly.
 */

import java.io.*;
import java.util.*;

public class pacct2csv {

  /**
   * Method to convert Unix accounting float to java 64-bit long int.
   * @param ac_Float is a 16-bit number:
   *        The high-order 3 bits compose a base-8 exponent.
   *        The low-order 13 bits compose the fraction part.
   * @return fraction * 8**exponenet
   */
  public static long toLong(short ac_float) {
    double exp, fra;
    long num;

    exp = ac_float>>>13;
    fra = ac_float & 0x1FFF;
    num = (long) (fra * Math.pow(8,exp));

    return(num);
  }

  public static void main(String args[]) {

    byte  ac_flag, ac_stat;
    short ac_uid, ac_gid, padding, ac_utime, ac_stime;
    short ac_etime, ac_mem, ac_io, ac_rw;
```

```
int   ac_tty, ac_btime;
byte  ac_comm[] = new byte[8];
long  uid, gid, utime, stime, etime, mem, io, rw;
GregorianCalendar cal = new GregorianCalendar();
String datetime = null;
StringBuffer command = null;
DataInputStream in = null;

/* Check arguments. */
if(args.length < 2) {
  System.err.println("Usage: pacct2csv hostname filename");
  System.exit(1);
}

String hostname = new String(args[0]);

/* Open input file. */
try {
  in = new DataInputStream(new FileInputStream(args[1]));
} catch(IOException e) {
  System.err.println("Could not open input file: " + args[1]);
  System.err.println(e);
  System.exit(1);
}

/* Print header record. */
System.out.println(
  "hostname,datetime,command,ac_flag,ac_stat,ac_uid,"+
  "ac_gid,ac_tty,utime,stime,etime,mem,io,rw");

try {
  while (true) {
    /* Read next record from pacct file */
    ac_flag   = in.readByte();
    ac_stat   = in.readByte();
    ac_uid    = in.readShort();
    ac_gid    = in.readShort();
    padding   = in.readShort();
    ac_tty    = in.readInt();
    ac_btime  = in.readInt();
    ac_utime  = in.readShort();
    ac_stime  = in.readShort();
    ac_etime  = in.readShort();
    ac_mem    = in.readShort();
    ac_io     = in.readShort();
    ac_rw     = in.readShort();
    in.read(ac_comm,0,8);

    /* Derive metrics */
    cal.setTime(new Date((long) ac_btime*1000));
    datetime = new String(cal.get(cal.MONTH)  + "/" +
                          cal.get(cal.DATE)   + "/" +
                          cal.get(cal.YEAR)   + ":" +
                          cal.get(cal.HOUR)   + ":" +
                          cal.get(cal.MINUTE) + ":" +
                          cal.get(cal.SECOND)
                          );
    utime = toLong(ac_utime);
    stime = toLong(ac_stime);
    etime = toLong(ac_etime);
    mem   = toLong(ac_mem);
    io    = toLong(ac_io);
    rw    = toLong(ac_rw);

    /* Read command string up to null termination. */
    command = new StringBuffer();
    for (int i=0; i < 8 && ac_comm[i] != '\000'; i++) {
      command.append((char)ac_comm[i]);
    }

    /* Print comma-separated metrics. */
```

```
        System.out.println(hostname + "," +
                           datetime + "," +
                           command  + "," +
                           ac_flag  + "," +
                           ac_stat  + "," +
                           ac_uid   + "," +
                           ac_gid   + "," +
                           ac_tty   + "," +
                           utime    + "," +
                           stime    + "," +
                           etime    + "," +
                           mem      + "," +
                           io       + "," +
                           rw
                           );
      }

    } catch (EOFException e) {
      /* Reached last record. */
    }
      catch(IOException e) {
      System.err.println("Error reading input: " + args[0]);
      System.err.println(e);
      System.exit(1);
    } finally {
      try {
       in.close();
      } catch(IOException e) {
       System.err.println(e);
       System.exit(1);
      }
    }
  }
}
```