On Process Auditing and Intrusion Detection

by Dr. Frederick B. Cohen ‡

In this paper, we examine methods for auditing processes and the use of these methods in automated intrusion detection.

Search terms: IT Audit, Intrusion Detection

‡ This research was funded by *Management Analytics*, PO Box 1480, Hudson, OH 44236, USA
Url http://all.net - Email fc@all.net - Tel 1+216-686-0090 - Fax 1+216-686-0092

1 Background and Introduction

Auditing of information technology (IT) is in a state of great flux. Attempts to form standards are only in their infant stages [?] and different organizations have different perspectives on the issues. One area of IT audit that has gotten increasing attention recently is the real-time analysis of of audit trails for intrusion detection. [?] [?] [?] [?] [?] [?] [?] [?] [?] [?] [?] Real-time techniques have been investigated in the literature from several perspectives. The major lines of consideration have been in the ability to detect known attack patterns, to detect anomalous behavior in the form of deviations from expected norms, to detect dispersed attacks where the patterns come from numerous systems in a network, and to respond to attacks automatically.

The major source of information used in all of the published audit-based real-time intrusion detection systems we have encountered is the system-produced audit trails that record *security-relevant events*. This is normally limited to the execution of programs, attempts to access files, error messages produced by programs, and attempts to initiate network or internal services. One way to consider these audit sources is that they all relate to files and error messages. Audit trails of this sort tend to record events of the form of a particular user attempting to open a particular file for a particular type of access at a particular time. (e.g., user *John* tried to open the *login* program for *execute* access on *Saturday* at 9PM). Another typical audit source is error messages produced by programs. (e.g., Oct 15 10:21:50 Sendmail AA22541: to=[ffc@all.net], addressee not found).

In this paper, we look at a slightly different real-time intrusion detection method wherein we try to account for processes rather than files and error messages. We concentrate our examples on Unix-based systems but the general concepts should apply to other operating environments as well.

First, we will discuss the basic principles used in developing an audit capability based on processes. Next, we will show an example of an automated process audit tool (developed as part of this research) to get a sense of what can and cannot be easily found using process-based intrusion detection. We then discuss near-real-time process-based intrusion detection and consider its limitations and advantages. Finally, we summarize results, draw conclusions, and propose further work.

2 The Basic Principle

The strategy adopted for our auditing process is to start with the full list of process information and systematically remove items that we can account for. Through the process of elimination, what we have left is a listing of things we cannot account for.

Underlying this approach is the idea that we should know what processes should and should not be in use by which users on our system. This is appropriate in many systems, including but not limited to firewall computers, production computers in many environments, process control computers, telecommunication switching systems, and so on.

In systems where this is not the case, we don't have to go quite so far. For example, we can have lists of processes that are definitely authorized and definitely not authorized. This gives us three levels of audit results; definitely acceptable, definitely not acceptable, and questionable.

We can also apply the anomalous behavior techniques used in other intrusion detection systems to detect out-of-pattern use based on process data.

Another way to understand the difference between this approach and other common auditing approaches is to consider the difference between theory and experiment. If audit trails were accurate enough, they could be analyzed to produce a theoretical state of the computer at any given moment. The theory is that by examining the sequence of auditable events, unexpected behavior can be found. The process audit approach experimentally looks at the actual state of the computer at a given moment to determine whether anything unexpected appears.

3 A Simple Example

A simple example of a process audit may help to demonstrate the sorts of things we can detect by looking at processes. The first step is to get a snapshot of process information from the machine being tested. An example of a process status (ps) listing is provided in the appendices. This listing will form the basis for the rest of this example.

3.1 Root Processes

Our analysis begins with processes owned by the *root* user, which is the user authorized to do anything on a *Unix*-based system.

The process numbers 0, 1, and 2, are normally used by the operating system for the *swapper*, the *init* process, and the *pagedaemon*. These form the basis for the proper operation of the *Unix* operating system and, subject to the particulars of the version of *Unix*, must always be present. If they are not present with the proper process identification numbers, owned by *root*, and with reasonable values for all of their parameters, something is wrong, and a severe error is produced. If there is more than one copy of any of these processes, a severe error is also produced. In this particular version of *Unix*, the following are proper values, and are thus eliminated from our list of suspects.

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
root         0  0.0  0.0    0    0 ?  D    Sep 16  1:41 swapper
root         1  0.0  0.2   56   48 ?  S    Sep 16  4:46 /sbin/init -
root         2  0.0  0.0    0    0 ?  D    Sep 16  0:01 pagedaemon
```

Several other processes that are commonly owned by *root* but that have non-fixed process numbers are also identified, checked for proper values, and then eliminated as well. These processes may vary widely from system to system, and are considered a part of the tuning process associated with such an audit tool. One each of the following processes is normal for most *Unix*-based systems:

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
root        60  0.0  0.0   80    0 ?  IW   Sep 16  0:00 portmap
root        75  0.0  0.0   72    0 ?  IW   Sep 16  1:53 syslogd
root       100  0.0  0.1   24   16 ?  S    Sep 16218:38 update
root       107  0.0  0.0   64    0 ?  IW   Sep 16  0:12 cron
root       123  0.0  0.0   56    0 ?  IW   Sep 16  2:03 inetd
```

The *portmap* and *inetd* processes are used to implement the internet protocol suite. The *syslog* daemon is used to log errors for the purposes of auditing, the *update* process is used to periodically update the disks to reflect cached updates, and the *cron* process is used to perform periodic or pre-arranged processing.

The next step in this analysis is to look for processes awaiting login. Normally, *Unix*-based systems create a *getty* process for each physical device that permits a login session. This process awaits a terminal connection, and then runs the *login* program to identify and authenticate users. Since policies differ from system to system and time to time, the default auditing tool we use simply lists the *getty* processes and asks the user to check them to make certain they are correct:

```
Verify that the following terminals should be actively awaiting login
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
root       127  0.0  0.0   56    0 co IW   Sep 16  0:00 - cons8 console (getty)
root       128  0.0  0.0   56    0 c  IW   Sep 16  0:00 - std.38400 ttyc (getty)
root       129  0.0  0.0   56    0 d  IW   Sep 16  0:00 - std.38400 ttyd (getty)
```

More sophisticated tools tuned for a particular system could identify each terminal not identified as awaiting login and determine whether or not they are attached to legitimate users. For example, if the console should only be used by one of a certain set of users, this can be verified automatically with such a system.

The process of elimination can be used to eliminate more and more of these processes. Whatever is left after all known-to-be legitimate processes have been eliminated, is suspect:

```
The following unidentified root processes are running.
Please verify that they are supposed to be running as root.
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
root        88  0.0  0.0  232    0 ?  IW   Sep 16  0:22 /usr/lib/sendmail -bd -q
root       154  0.0  1.9  248  592 b  S <  Sep 16 65:49 pppd [up, du0, ttyb, 38.
root        92  0.0  0.0   40    0 ?  IW   Sep 16  2:53 sh /etc/rc.local
root       112  0.0  0.0 1056    0 ?  IW   Sep 16  0:05 /u/w3/httpd
root     23791  0.0  0.0   24    0 ?  IW   10:47   0:00 sleep 120
root     20160  0.0  0.2   56   48 ?  S    Oct 13  2:17 in.rlogind
root     13097  0.0  0.0   56    0 ?  IW   Oct 11  0:56 in.rlogind
```

If desirable, additional checks can be used to identify processes that should never be run by *root*.

3.2 Errant Processes

There are a number of conditions that are considered undesirable in many *Unix* environments. For example, so-called *Zombie* processes can indefinitely remain in a state where they have been terminated but cannot be removed from the process table. As an example, this sometimes results from a process that has an unterminated child processes with opened files. We created a temporary zombie process to demonstrate:

```
PSL-02-01 - Found zombie process:
fc       29487  0.0  0.0    0    0 p1 Z    Sep 16  0:00 <defunct>
```

A similar problem faced by many networked computers is the existence of old network processes awaiting input or output on *TCP* channels that have no channel connected at the other end. This happens because many programs are not designed to prevent denial of services by having timeouts on all of the network-based I/O. If enough processes pile up, a situation can arise where no further network [processes can be initiated, resulting in denial of services. An undesired long-term network connection may also represent someone who is performing unauthorized activities.

```
Please kill any of these old processes that should not still be around.
The cause of these old processes may be important to track down.
fc 13098 0.0 0.0 40 0 p0 IW Oct 11  0:00 -sh (sh)
fc 14684 0.0 0.0 56 0 p0 IW Oct 11  0:00 -sh (sh)
fc 14685 0.8 0.8 104 240 p0 S Oct 11278:18 watcher (/bin/date;/u/fc
fc 20161 0.0 0.4 48 112 p1 S Oct 13  0:02 -sh (sh)
```

In this case, these processes represent a real-time intrusion detection display that has been operating for some time rather than an attack, but the principle is the same. As detection is tuned, such processes can be eliminated from the audit report, leaving only processes that are not accounted for. Anomalous processes should not be eliminated from all further testing, since there might be more than one anomaly associated with a given process.

3.3 Non-root System Processes

In most *Unix* systems, most accounts associated with system maintenance activities are unused most of the time. The next step in our prototype system is to look at these users to detect any unauthorized processes. In our test system, no such processes were operating.

```
PSL-04-03 - No additional processes found for privileged user root
PSL-04-03 - No additional processes found for privileged user daemon
PSL-04-03 - No additional processes found for privileged user sys
PSL-04-03 - No additional processes found for privileged user bin
PSL-04-03 - No additional processes found for privileged user uucp
PSL-04-03 - No additional processes found for privileged user news
PSL-04-03 - No additional processes found for privileged user audit
PSL-04-03 - No additional processes found for privileged user postmaster
PSL-04-03 - No additional processes found for privileged user sync
PSL-04-03 - No additional processes found for privileged user mud
```

3.4 User Processes

User processes are often harder to audit than system processes because, on many systems, there are few if any limits on what programs users may execute. In systems where this is not the case, it can be useful to look at these processes in more detail. Here is an example of something we can detect with this audit:

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
fc       23808  0.0  0.8   24  240 p0 S    10:47   0:00 cat /var/log/syslog
```

In this case, a normal user is accessing a file (/var/log/syslog) that should not be readable by normal users. This particular file is a system audit trail. How the user managed to do this is beyond the scope of automated audit of this sort but the detection of such an access is indicative of a detection that is common in this sort of audit. The rest of these processes don't give any obvious signs of abuse. The *watcher* program has a very long run-time, and if this were not in the profile for this machine, it might be worth investigating further.

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
fc       24001  7.7  1.8  216  536 p1 R    10:48   0:00 ps -aux
fc       20161  0.0  0.4   48  112 p1 S    Oct 13  0:02 -sh (sh)
fc       14685  0.0  0.0  104    0 p0 IW   Oct 11278:18 watcher (/bin/date;/u/fc
fc       13098  0.0  0.0   40    0 p0 IW   Oct 11  0:00 -sh (sh)
fc       14684  0.0  0.0   56    0 p0 IW   Oct 11  0:00 -sh (sh)
```

4 Near-real-time Intrusion Detection

Based on audit experience, it appears that most systems administrators don't examine every process on their system very often. It is common to find unterminated processes left for days or even weeks on computers that should have no such activities. It is not unusual to find processes running as *root* that should not be running with that privilege. In one recent firewall audit, we found several processes running that should not have ever been run on the particular host being examined. This led to a further test that indicated a substantial control breakdown.

Based on this anecdotal evidence, it would seem to be reasonable to run periodic checks of processes on computers, if only to assure that denial of service is not immanent. But this doesn't help very much in understanding the role of process-based audit for near-real-time intrusion detection.

There are at least two important reasons that process audit has not been widely used for intrusion detection. The first reason is that processed normally only produce audit records when files are used, and then, the only information is the file-based or error-based sort of information described earlier. The second reason is that process information tends to indicate misuse through the correlation of elapsed time, usage statistics, multiple processes started at different times, and the absence as well as presence of processes.

A good example of this is the non-termination of a process. If a process is awaiting input and none arrives, the process may remain active indefinitely without performing any obvious function. Similarly, if a process is only computing and not communicating, no audit trail will be generated through the normal methods.

If we tried to produce audit trails based on this sort of process information, we might encounter some other difficulties. For example, how often would we want to store information about a process that is still waiting for input? One of the challenges we face in this sort of auditing is what constitutes a security-relevant event. The answer seems to be related to the state of processes rather than that it was created at some point in the past.

It may also be theoretically possible to get most of the process information from examining audit trails. For example, if we audit all process creation and termination, we can tell what processes are active at any given time, but only by analyzing an unlimited amount of audit information. Considering that the same information may be readily available in the form of process status information stored by the operating system, this would seem to be a preferable source.

We are aware of two ways in which near-real-time intrusion detection has been done based on process audit:

- It has been used in an ad-hoc basis to track down anomalous behavior, such as in the tracking of the Internet virus in 1988. [**?**] [**?**] [**?**] In this case, the virus was detected based on denial of service behavior, and those who analyzed it began the process by examining the process table using the *ps* command under *Unix*. Once they found the processes creating the problem, they tracked the process to the file that created it and analyzed from there.
- It has been used in real-time to detect security flaws in an application. In 1994, we used a near-real-time process monitor to observe processes on a user-by-user basis in an Internet Service Provider's system. This was done as a security precaution to determine if any users were able to bypass a limited-function interface that provided menu-based service to anonymous users from over the Internet. On several occasions, the monitor detected flaws in the security system as they were beginning to be exploited. The near-real-time detection enabled the removal of those services before significant damage could be done.

The same tool used in the second example above has also been used to do real-time detection of attacks by specific users suspected of attacking systems and has been used to detect unauthorized programs being run by network-based users such as users coming in through World-Wide-Web browsers.

5 What Process Audit Currently Misses

This audit technique is clearly limited in terms of its detection capability. For example, in many operating environments, the process status information isn't guaranteed to be accurate. Under *Unix* an attacker can alter the command-line information associated with their own processes so as to cover up the actual name of the programs being run.

The process table changes continuously over time, so the process examining it gets only a brief snapshot of its appearance. Process status analysis is not trivial in the sense that it takes several seconds to get the information. Depending on the amount of processing spent and priority placed on the detection process, many short-lived processes may go undetected.

The information about a process provided in the process table indicates little of significance about the content of the process. Many programs can be abused without having a detectable impact on the process table. In effect, the signal to noise ratio is not as high as we might like it to be. This ultimately tracks to our limited knowledge about what is and is not legitimate in most systems.

6 Advantages of Processes Audit

There are four aspects of process audit that seem to be important in understanding its value as a near-real-time intrusion detection technology:

- It is independent of file-based and error-based audit so that it provides a redundant check.
- Process audit deals with how things are being used, not just that they are being used.
- Process audit deals with behavior over time.
- Process audit deals with resource consumption that can produce denial of services.

6.1 A Redundant Check

Process audit is independent of the protection mechanisms used for file-based and error-based audit. For example, if a privileged process is fooled into granting another process *superuser* privileges, neither file-based nor error-based audit is likely to provide any useful information because neither of these sorts of activities are involved in the attack.

Process audit has a chance of detecting such a change of status because it examines the operating system table that tracks the current status of a process rather than only observing the audit trails generated by the normal operating system operation. Since the process table is typically the basis for granting privileges to processes every time they are given runtime, it is likely that the process table will have information indicative of such an attack.

The idea of an independent audit is particularly important because the audit trail used by error-based and file-based audit is generated by the mechanisms that check for errors and accessibility of files. If these routines have been bypassed, they may not be able to produce audit trails indicative of the illicit activity. If normal audit mechanisms are working properly and have not been bypassed, then they should not permit any unauthorized activity.

6.2 How Things Are Used

Process audit deals with how the system is being used regardless of how that usage came to pass. This is substantially different from other audit techniques in that they tend to indicate only what was detected at a given point in time when a particular action was being started or ended. There is an implicit presumption in error-based and file-based audit that the system operates properly once a critical decision is made. Process audit makes no such assumption.

In the case of illicitly attained privileges, some failure in the protection system results in authority being granted when it should not be. If we can't trust the system to prevent unauthorized use, how can we be certain that it produces audit trails indicative of that use? We cannot.

In essence, process audit looks at the current state of a system, while error-based and file-based audit only looks at each decision as it is being made.

6.3 Behavior Over Time

Process audit holds the hope of observing the use of time by processes. Not only the total CPU usage can be tracked, but also the combination of different elements of usage. This type of behavior has been widely ignored in intrusion detection to date, in some part because of the fundamental limitations on detection of anomalous behavior by this technique.

For example, the use of a threshold on total time used by a process for detecting illicit behavior is very easily bypassed. All an attacker has to do is mimic the behavior of a legitimate process in order to bypass this technique. Nevertheless, for attacks that fail to embody this sort of defense, detection is quite straight forward.

In order for this method to work well, it is important to eliminate false positives. Sophisticated techniques can be used to model ratios of CPU usage to file access over the lifetime of a process. The basis for using statistical information for this purpose is beyond the scope of this paper and is essentially equivalent to the same problem as applied to statistical detection of anomalous behavior by other intrusion detection mechanisms.

A good example of what can be detected based on the use of time by processes is a computer virus such as the Internet virus. This virus used a great deal of computer time compared to almost any other program in the environment. Almost any reasonable threshold value would detect this. Similarly, process viruses [**?**] can easily be detected by such a technique.

6.4 Resource Consumption

Process audit provides state information about a system as opposed to change information. With error-based and file-based audit the audit trail lists changes which state can potentially be derived from. The advantage of having direct state information is that the state of the machine is often a better basis for a decision about protection than relatively static protection-state tables such as those used in access control. This is particularly important in the areas of denial of services and of detecting multiprocessing attacks.

- In the case of denial of service attacks, the ability of a particular decision to impact the system as a whole depends primarily on the available resources and the resources that can be consumed as a result of the decision. Rather than an a-priori guess about resource usage, (which may result in deadlock [?]) real-time decisions about which processes are allowed to continue operating allows the removal of processes that could cause denial of services as they become a threat.

- In the case of multiprocessing attacks, no single process consumes a great deal of resources, but in combination, they may result in denial. Combining information from multiple processes allows detection of threats as they occur without setting a-priori limits on the number of processes per user or the runtime of any particular process.

A good example of this is the detection of processes that have been running for more than one day as shown in the example earlier. In most World Wide Web servers, it would be a reasonable expectation that no process running on behalf of a browser would run for more than 15 minutes. This threshold when combined with the user ID of Web processes has been used to prevent denial of service threats and to terminate processes that are no longer connected to remote hosts.

7 Summary, Conclusions, and Further Work

Based on our initial experience with process-based audit techniques, it appears that there is great value in applying these techniques to near-real-time intrusion detection.

The value of these techniques goes beyond the application commonly used in error-based and file-based audit. Process audit appears to address integrity concerns to a limited extent and to apply to denial of service concerns which are poorly covered by other intrusion detection techniques in use today.

It seems clear that near-real-time process-based detection capability could be built into misuse detection systems along with error-based and file-based detection, and that the combined system would be more flexible and better suited to the overall intrusion detection challenge now faced in networked environments. But there are limitations.

Before near-real-time process-based detection becomes widely applicable, it will be important to develop substantial results and do further development. In the meanwhile, the limited results produced by relatively primitive tools are useful.

References

[1] Per Brinch-Hansen, Operating System Principles, Prentice Hall, Englewood Cliffs, N.J., 1973.

[2] F. Cohen, A Short Course on Systems Administration and Security Under Unix, ASP Press, 1990.

[3] J. Rochlis and M. Eichin. With Microscope and Tweezers: The Worm from MIT's Perspective. Communications of the Association for Computing Machinery 32, no. 6 (June 1989). [This paper describes how one team dissected the Internet Virus of 1988 and what that virus contained.]

[4] E. Spafford. Crisis and Aftermath. Communications of the Association for Computing Machinery, 32, no. 6 (June 1989). [This paper gives a perspective on the Internet Virus from one of the researchers who was granted access to the source code of the virus soon after it was decompiled.]

[5] F. Cohen. A Short Course on Computer Viruses, 2nd ed. New York: John Wiley, 1994. [This is widely considered one of the most comprehensive books on computer virus attacks and defenses, and it details most of the current state-of-the-art in this field.]

[6] Generally Accepted System Security Principles, GSSP Committee Chair OPA, Inc. 870 Market Street Suite 1001 San Francisco, CA 94102, This Exposure Draft represents the first edition; providing the background, framework, and pervasive principles of the GSSP.

[7] D. E. Denning. An Intrusion-Detection Model. IEEE CH2292. [This paper describes issues in automated intrusion detection as a method to detect Trojan horses, computer viruses, and other similar attacks against information systems and networks. It is one of the earliest works in this area.]

[8] T. F. Lunt. Automated Audit Trail Analysis and Intrusion Detection: A Survey. Eleventh National Security Conference. October 1988. [This paper surveys several intrusion detection technologies and reports on the state-of-the-art as of 1988.]

[9] T. F. Lunt and R. Jagannathan. A Prototype Real-Time Intrusion-Detection Expert System. IEEE Symposium on Security and Privacy, April 1988. This paper describes the IDES real-time intrusion detection system.]

[10] S.R. Snapp, J. Brentano, G.V. Dias, T.L. Goan, L.T. Heberlein, CL Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T.Grance, D.M. Teal, and D. Mansur. DIDS (Distributed Intrusion Detection System)—Motivation, Architecture, and an Early Prototype. Proceedings of the 14th National Computer Security Conference, October 1991:167–176. [Sets out several scenarios in which detection of attacks depends on information from multiple sources: doorknob attacks where multiple login trials are below the threshold on each system; chain and parallel attacks; and network browsing, where multiple files are examined on several different computers during a short period, and where the level on each machine is below an alarm threshold.]

[11] Teresa Cochran and Joseph M. Mellichamp, AUTORED: An Automated Error Recovery System for Network Management." IEEE Network Magazine, (March 1990). [This paper describes one mechanism for automating fault detection and recovery for network management.]

[12] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A Network Security Monitor. IEEE CH2884, 1990. [This paper describes the operation of a real-time network security monitor which detects certain sorts of attacks against local area networks in real-time.]

[13] Lundy Lewis. A Case-Based Reasoning Approach to the Management of Faults in Communications Networks. IEEE 11d.3.1.(1993):1422–1429. [This paper describes a system for automated detection and correction of select network faults in which the system learns from examples. It is limited in its value for malicious attacks, but is quite useful in day-to-day management of network operations.]

[14] Paul Proctor. White Paper to Implement Computer Misuse Detection on the Operational Support System (OSS). 1993. [This paper describes a misuse detection system and how it can be used to protect systems and networks against many current attacks.]

[15] Marcus J. Ranum. A Network Firewall. Washington Open Systems Resource Center. Greenbelt, MD (June 12, 1992). [In this paper, a method for detecting and limiting the scope of attacks from over the Internet is implemented.]

[16] T.F. Lunt. "A Survey of Intrusion Detection Techniques." Computers and Security V12:405–418. [This paper describes tools for automating audit data inspection which can operate off-line or in real-time to detect several types of attacks both from outside and inside the system.]

[17] P. Proctor. "A Computer Misuse Detection System." [This paper describes the combination of expert systems for known attack detection and statistical deviation detection for behavioral change detection as a method for automated misuse detection and demonstrates how many of the current network-based attacks are detected and countered in real-time by these mechanisms.]

A A Process Status Listing

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
fc       24001  7.7  1.8  216  536 p1 R    10:48   0:00 ps -aux
root        88  0.0  0.0  232    0 ?  IW   Sep 16  0:22 /usr/lib/sendmail -bd -q
root       123  0.0  0.0   56    0 ?  IW   Sep 16  2:03 inetd
root        60  0.0  0.0   80    0 ?  IW   Sep 16  0:00 portmap
root        75  0.0  0.0   72    0 ?  IW   Sep 16  1:53 syslogd
root        92  0.0  0.0   40    0 ?  IW   Sep 16  2:53 sh /etc/rc.local
root       100  0.0  0.1   24   16 ?  S    Sep 16218:38 update
root       154  0.0  1.9  248  592 b  S <  Sep 16 65:49 pppd [up, du0, ttyb, 38.
root       107  0.0  0.0   64    0 ?  IW   Sep 16  0:12 cron
root       127  0.0  0.0   56    0 co IW   Sep 16  0:00 - cons8 console (getty)
root       112  0.0  0.0 1056    0 ?  IW   Sep 16  0:05 /u/w3/httpd
root       128  0.0  0.0   56    0 c  IW   Sep 16  0:00 - std.38400 ttyc (getty)
fc       20161  0.0  0.4   48  112 p1 S    Oct 13  0:02 -sh (sh)
fc       14685  0.0  0.0  104    0 p0 IW   Oct 11278:18 watcher (/bin/date;/u/fc
root     23791  0.0  0.0   24    0 ?  IW   10:47   0:00 sleep 120
fc       13098  0.0  0.0   40    0 p0 IW   Oct 11  0:00 -sh (sh)
root     20160  0.0  0.2   56   48 ?  S    Oct 13  2:17 in.rlogind
root         2  0.0  0.0    0    0 ?  D    Sep 16  0:01 pagedaemon
root         1  0.0  0.2   56   48 ?  S    Sep 16  4:46 /sbin/init -
fc       23808  0.0  0.8   24  240 p0 S    10:47   0:00 cat /var/log/syslog
fc       14684  0.0  0.0   56    0 p0 IW   Oct 11  0:00 -sh (sh)
root     13097  0.0  0.0   56    0 ?  IW   Oct 11  0:56 in.rlogind
fc       23792  0.0  0.0   40    0 p0 IW   10:47   0:00 sh -c (/bin/date;/u/fc/b
root       129  0.0  0.0   56    0 d  IW   Sep 16  0:00 - std.38400 ttyd (getty)
root         0  0.0  0.0    0    0 ?  D    Sep 16  1:41 swapper
```