A Linux Task Manager

Andrew McRae

amcrae@employees.org

Abstract

The Netdevices product is an embedded cluster of CPUs that are treated externally as a single router. Internally, services and applications can run on and migrate between different CPUs.

To facilitate this internal cluster, a Task Manager was developed that provided a high level of process management, such as active process watchdogs, service migration to different CPUs, automatic process restart, process grouping, group start/stop etc.

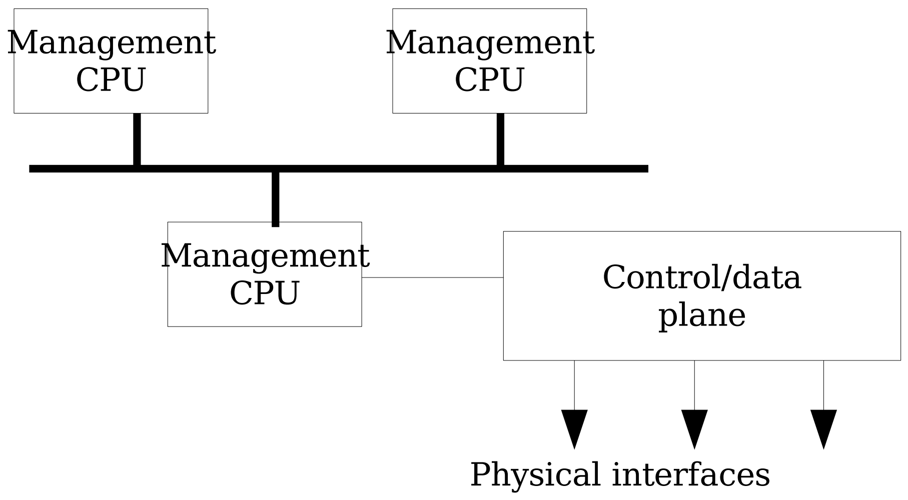
Another concept that was implemented was a services directory, where processes automatically discoveres which CPU provides selected services via internal IPC, and to rehome clients when services move to different CPUs due to CPU hotswap or failure.

This paper describes the technical details underpinning this Task Manager.

The Problem

The NetDevices SG-8 is a state-of-the-art services gateway that incorporates many different networking applications in a single integrated device, offering a high degree of manageability, performance and robustness.

Physically, the SG-8 consists of a number of separate hardware hot-swappable modules, interconnected with a high speed industry standard PCI-Express bus. A unique feature is the introduction of a cluster of management orientated CPUs, separating the management functions of the device from the control and data plane. These CPUs communicate via an internal high speed LAN:

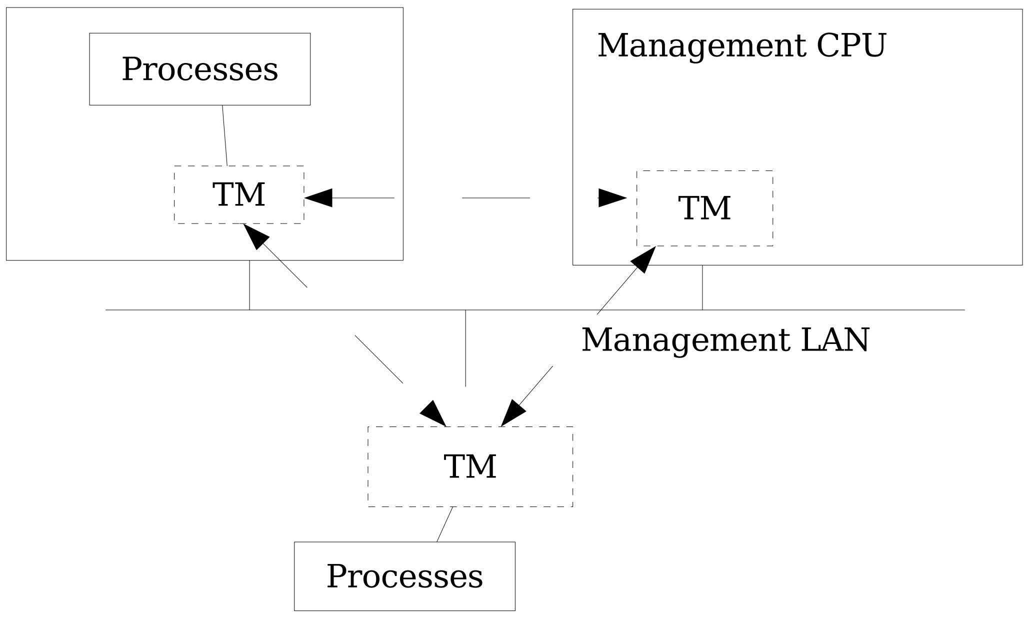


The management CPUs execute as a standardised cluster, each potentially running a range of different applications and features. A key feature of this cluster is the ability for any management CPU to be removed as part of a I/O card hot swap, and for the applications running on that CPU to be migrated to other CPUs. Note that *process migration* is not supported, but instead *service migration*, where the facilities provided by that application will be restarted and made available on another CPU.
Another requirement is the detection of and recovery from hardware failure of modules in the system, so that if a management CPU fails due to some hardware cause, other CPUs can then provide the services that the failed CPU provided.
However, studies have shown that a high percentage of unplanned equipment outages is caused not by hardware failure, but by software failure, so an important aspect of the monitoring and control is to quickly detect and recover from software failure.
To enable these features, a Task Manager is used to provide the following support:

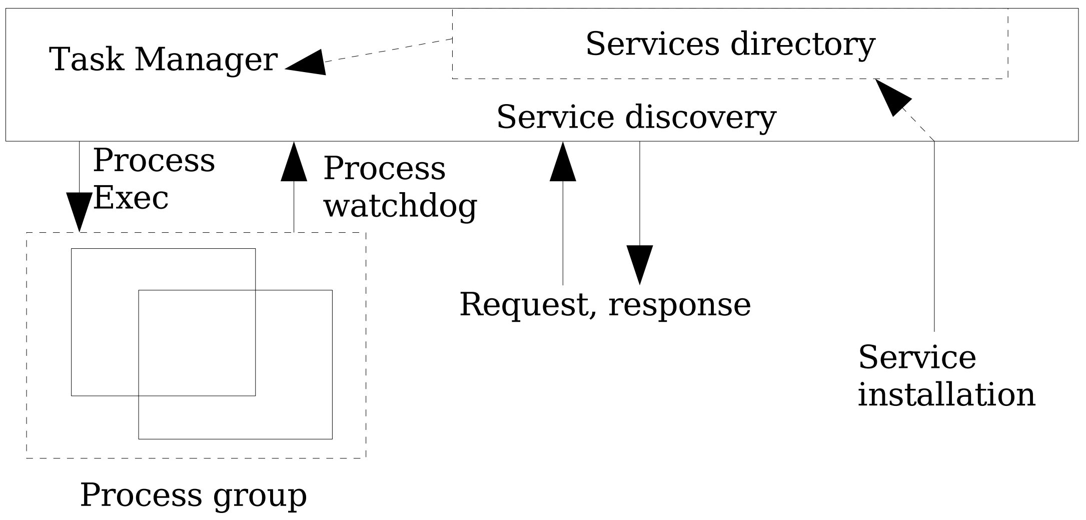
- Inter-module monitoring and detection of failed or removed CPUs.
- Internal processing monitoring, restart, watchdogs, and control.
- Provision of a *services database*, allowing processes to discover where selected services are running within the cluster of CPUs.
- Notification of CPU or service migration, so that processes connected to services can rehome to new instances of the services.

Overview

The Task Manager's (TM) role is to supervise and control the execution of Linux processes in the NetD system. It also provides an internal database of services that are mapped to processes so that client software can discover on which host and port a particular service is running.
Internally, the system appears as a loosely coupled cluster of CPUs, interconnected using a 100Mbit LAN. The IPC is IP based, with each CPU appearing on the internal network as a separate host.



Each CPU executes as part of it's startup sequence an instance of a Task Manager process. These TMs will automatically discover other TM's present in the system, and will from then on exchange service information, and monitor the health of the TM's.
On each CPU, the TM appears as:



The TM clients that interface to the TM connect to it via a Unix domain socket.
Each management plane CPU runs a single instance of the TM, and these instances communicate amongst themselves in a peer fashion to track and monitor the process groups and services that are running on all the management CPUs.

The TMs running on the management CPUs will co-ordinate amongst themselves to ensure that the necessary groups are available.
An interface is provided so that processes can be informed of changes, such as hosts that fail, or services that migrate, or services that are enabled or disabled.
Whilst this framework is implemented in an embedded system, the same concepts and framework works perfectly well in a more explictly distributed cluster, where separate hosts and servers interconnected on a LAN can use the services of the TM to provide a high level of process monitoring and control, and service migration can be used to provide a high level of redundancy.

Services Directory

The concept of *services* is that processes running on the management CPUs can be considered as offering a particular *service* to client software. This service can be identifed with a particular process or set of processes running on one or more of the CPUs. Often, only one instance of a service can be executing at any one time, but multiple CPUs may have the *capability* of running the process offering this service e.g there may be be two hosts in the system that are configured to run as a web server, but only one of these hosts should actually running this service. The process that can offer this service can be mapped to an Service Identifier in the Services Directory of the TM, so that the TM can track the availability of this service, and in the event of detecting the failure of the process, inform the other TMs of the removal of the service. The failure of the process may trigger the start of a process in another CPU that can offer this service as a backup. Services are automatically coupled to the process they are added from, so that if the process dies, the service is automatically removed (unless the process has set a specific flag indicating that the service is decoupled from the process).
The role of the Services Directory in the TM is:

- To allow external programs to install a service into a TM.
- To respond to requests from client software requesting the location of a service, and providing the appropriate information for the client to connect to the service (i.e a list of host addresses and optionally port numbers).
- To track the status of services running in other TMs, removing or adding services as necessary.
- To provide monitoring of selected services to processes that request it.

Services themselves do not need to be associated with a process that is started and monitored by the TM, but can refer to separately executed processes e.g an inetd initiated process. The services directory can simply be used as a registration database for responding to service requests from clients, directing the client to specific hosts. Service proxies can be installed that point to services on other hosts. Services also have some context data such as a TCP or UDP port number and host address indicating where the service is.
Earlier incarnations of the TM separated the services directory from the process management and control into a separate subsystem, but the coupling of services with processes naturally allows integration into the TM framework.

Process Management

The TM's role is to start and monitor processes, and in the event of detecting process failure, restart the process. Processes are managed as groups, where one or more process is executing within a process group (this is not to be confused with the Linux process group). Groups as a whole can be started, stopped, or deleted. Processes within a group can be flagged as being **critical** to that group, so that if a particular process fails to continue running (i.e it has been restarted but continues to fail within a certain time window), that group will be stopped (that is, all the processes in that group will be forcibly stopped). Processes can also be flagged as being **dependent** upon the start order within a group, so that if a process fails, the entire group is stopped and then restarted again.
Processes registered with the TM to be executed can be started with a supplied argument list.
If any process dies or is killed because of a watchdog timeout, it is automatically restarted.
Processes do not need to be associated with a particular group, but can belong to a default ***local*** group (a reserved internal group). When a process is defined without being associated with a group, it provides local management of this process as part of the local internal group.
A process group can be flagged as **negotiable**, which means that the TM will negotiate with other TMs about which TM will run this group – it is assumed for negotiated groups, that only *one* instance of the group is to be running in the system as a whole. If a TM discovers that another TM has stopped running a group that is flagged as negotiable, then this TM will start a negotiation with the other TM's to run this group.
A typical flow of events for the process management would be:

- A process is started as part of a group.
- At some stage the process crashes or exits.
- The TM will detect the process exit, and reap the exit status.
- The TM will log the failure, and in the event that other processes are part of this group and the failed process is flagged as being a critical process, the other processes are forcibly terminated.
- If the process is flagged to be restarted, then an attempt is made to restart the process.

- If the process is detected as unstable (several crashes in a row within a certain timeframe), and the process is marked as being critical to this group, the group is stopped and marked unrunnable.
- Another TM, once it sees the group has stopped running, can decide to run the group (if it is flagged as negotiable).

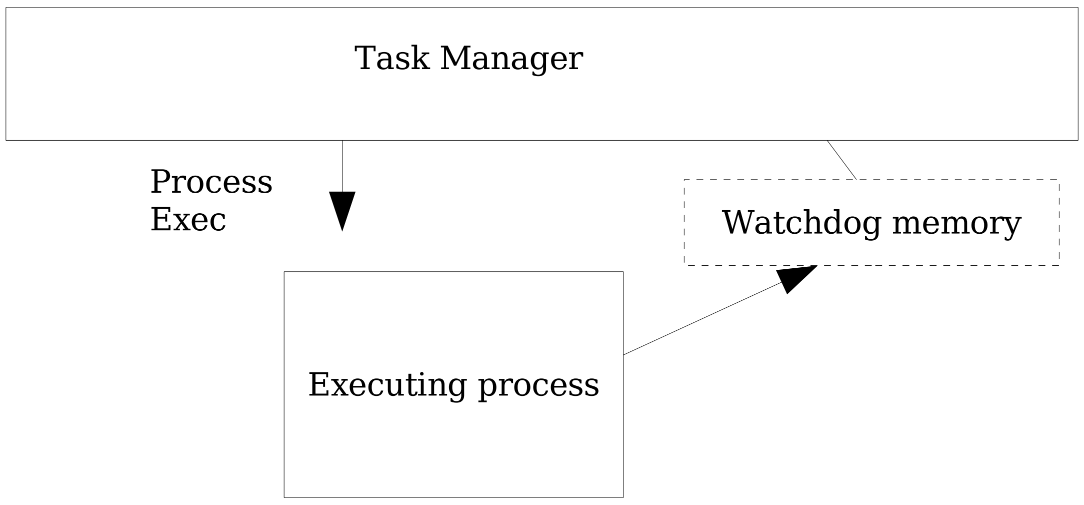
For a negotiated group, when a TM decides that a group needs to be activated, and there are no other TMs that it has detected, it will immediately start the group. If other TMs are available, it will broadcast a message bidding for the group. The response from the other TM(s) may be:

- Acknowledgement that the group can be run on that host.
- A message indicating that the group is already running on the other host, in which case this host will allow that group to continue to run on the other TM (the group is marked as belonging to that host), and the TM will discontinue negotiating for that group.
- A bid for that group. To settle the bid, an ID value is compared (each TM ID is the first IP address available to that host used for broadcasting), and the lowest value wins the bid. If this host is the winner, it will send another bid message to the remote host. If this host is not the winner, it will send a message acquiescing to the remote TM for that group (a later enhancement may be to have *preferrred* hosts for certain groups, which can influence the bidding).
- Silence.

To finalise the bid process, the TM must receive a positive acknowledgement from every available TM. If an acknowledgement is not received from every TM, then another bid is sent, and the negotiation restarted. If then a postive acknowledgement is still not received from every TM, the group is started anyway.

Process Monitoring

In order to gain a higher degree of process monitoring, an optional software watchdog is available. This watchdog is an active heartbeat from the process being monitored back to the TM, indicating that the process has performed some level of self-checking, and is reporting correct operation status (insofar as the internal state of the process can be self-checked).



In the event of a process not providing this heartbeat within a certain timeframe, the TM will kill that process and assume that the process has failed. The watchdog is implemented as a library call, and will only be operational when the process is executing as a child of the TM (and the process is flagged as requiring a watchdog). If the watchdog is configured and running, the process will also check if the TM has exited unexpectedly (by getting its parent PID), and if so, will terminate itself.
The watchdog is implemented as a incrementing value in a shared mmap'ed file. The process optionally can set a process state into the watchdog area so that in the event of a crash or other problem, the TM can log some meaningful data relating to the start of the process at the time of the problem.
Processes can also temporarily suspend the watchdog for debugging purposes.

Service and Host Monitoring.

Interested processes can register with the local TM and receive monitor messages about the state of selected groups, hosts and services. For example, if an IPC library has a connection to a particular host, the library can request to be informed when the host fails (for whatever reason). Thus, in the instance when the CPU fails or is hot swapped, and the connections are not gracefully terminated, the IPC library can avoid relying on the perhaps lengthy connection timeout, and instead terminate the connection quickly.

Processes can also monitor services, and be informed when services are available in the services directory, or are removed, or are moved to a different host.

TM Execution

To initiate proceedings, the TM can be started with an initial process name to be executed. This can be a utility (*addservice*) which reads a file and installs various process groups and services into the TM for execution, or it can be a more intelligent manager that dynamically installs or removes process groups according to external requests.
The TM interacts with the processes it executes via a library API. Processes can install new processes or process groups, install or remove services, or monitor existing services, groups or hosts.

Results

The TM has been in use in the field for over 18 months. It has proved very effective as a framework for delivery of robust services in an embedded system. Process restart and management is likely the most useful feature of the TM, providing very fast detection of failed or hung processes, fast restart of processes, and effective monitoring and reporting of process events. The services directory is an effective way of managing and discovering services across a cluster of CPUs, and when coupled with the process management, is a very fast method of service migration.

Conclusion

The Netdevices Task Manager has been described, showing how processes are installed, managed, and monitored across a cluster of CPUs.
The Services Directory feature has also been described, showing how processes can install and discover different services across the cluster, and when processes exit or are terminated, then clients can quickly rehome to a new service instance
The TM has proven to be an effective framework for robust and reliable process management and service delivery, whilst remaining within the standard Linux environment.