DESIGN AND IMPLEMENT THE CORE MODULE IN A EVENT PROCESSING SYSTEM

1. Introduction

Assume that you are in a development team where it needs a new design for a user space system that enables an administration system supports registering user space handlers for lower layer component/device events. The role of the software is a generic manager implemented as a daemon on a Linux system and listen to the events happen when a new component/device is initialized or a component or device is removed from the system.

You need to propose and provide a prototype implementation of the generic manager as a daemon. For simplicity, we use the daemon DEA to denote the work. The component is illustrated as pink component in the whole system design as the following diagram.

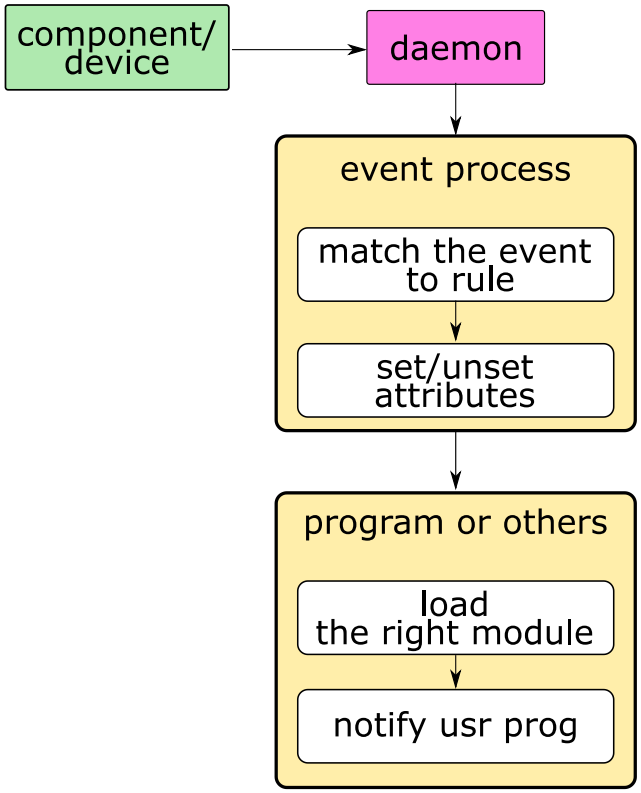

Figure 1 Daemon DEA component in the system design

Although the illustration provides an overview of all system components, the work in this project only focuses on the pink daemon.

2. Daemon description:

Daemon components:

Daemon DAE is designed with a core inside, which is simply a structure that keeps the list of events to make them available for later processing by the event processor. In general, the speed of event processing may be different from the speed of event generation. This model creates a fatal risk to system integrity since we may lost an unknown important event; and hence the daemon with a listing structure comes to help guarantee the correct work of the whole system.

The structure of the list of events is freely designed by your team in order to achieve different qualities of service:

- At a minimum requirement, a list can be implemented as a queue where all events are treated as having the same priority.
- To enhance the feature, you may provide the design of a multi-level queue associated with different priorities.
- It is not limited by the queue since we can provide another enhancements, i.e. the supporting of structured events. An example is the GUI support for board management, where a resize event of the base window component will have a group of structured events associated with it (UI box, button, label, etc.).

You can come up (at least) with the basic feature as a queue of event.

Daemon operations:

The daemon DAE supports two main operations: write() and read() to generate and retrieve the event, respectively. In general, the daemon needs to support contemporary operation in which it is unknown the order of the operations and there may be many operations called at the same time.

The data passed to these operations is a simple tuple:

```
class/struct Event {
  int event_id;
};
```

You can get more support for other fields, but it is not mandatory. Some other suggested fields include:

```
string event_name;
string event_handler;
```

The component and operation of the daemon DAE can be illustrated in the following diagram.

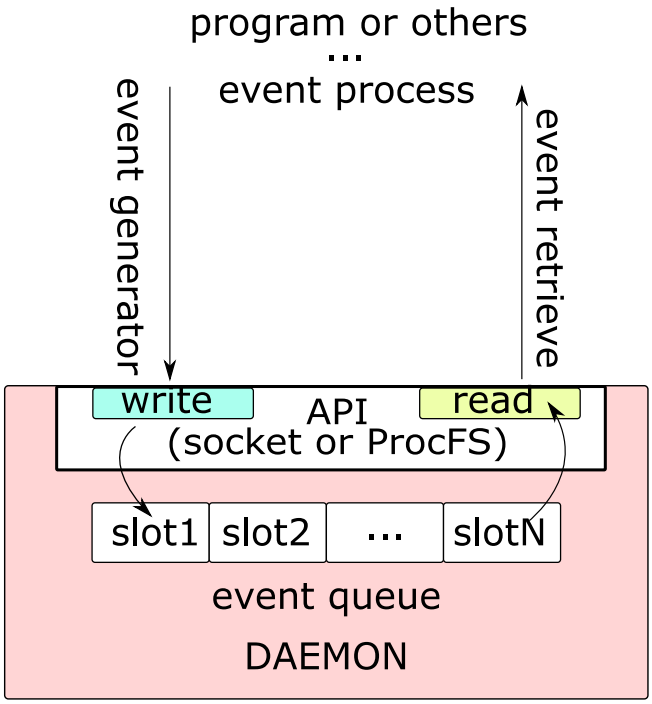

Figure 2 Daemon structure

A near exact example of the event queue is the log buffer in the previous lab, where the operations are replaced by the read() and write() routines with the event passing as data. (In the previous work, they are flushlog() and wrlog, respectively.)

Daemon implementation:

In a legacy design, a common way to use the daemon DAE on Linux systems was to let it sends events through an IPC communication channel, i.e., a socket, which would perform further component-specific actions. For example, the daemon DAE would notify other software running on the system that the new hardware had arrived by issuing a broadcast message on a higher layer of the system to all interested programs.

By the middle of 201x, that design had been deprecated by most Linux systems. The new design supports:

- Daemon provides low-level access to system allow program to enumerate event notification during early boot process where all system devices are initializing and the user space has not been initialized yet.
- The communication framework allows programs to communicate with each other, securely, reliably, and with a high-level programming interface. It needs to remind that there is a tradeoff design in socket where it has initially fixed number of connections at the same time.
- The daemon sits on top of other kernel interfaces and provides a high-level interface to a system feature through a kernel space system call without preemptive risk.
- The daemon provides a high-level interface for upper layer configuration and setup and is accessible via a communication bus.

To achieve these features, you can implement it as a kernel module and allow the interface through file system entry, i.e. procfs. For example, you can support the two basic operations:

/proc/your_module/write

/proc/your_module/read

which can be used to generate and retrieve event, for example:

echo -n “event_id” > /proc/your_module/write

cat /proc/your_module/read

For the legacy implementation, you should provide a basic library support the interface for testing. As an example, you need have 2 programs:

client_read : to get an event with your predefined communication channel

client_write <event_id> : to produce an event submit to deamon through your defined communication channel.

Daemon application:

The designed system can be applied to implement software management for a specific board product. Assume you need to provide a user interface through a graphic (GUI) application or web-based UI to configure your product board. It needs to keep track all the event generated by the system devices to update the status to upper layer program or the user activity through UI (user interface) to do some setting/modification to configure the device register.

The daemon DAE is sitting in between the system UI and the device management library.

3. Evaluation

Although the new design has some advantages over the legacy design, it may have complicated technical implementation. Whatever design you choose, it needs to be a feasible solution first. The evaluation of the work simply has 10% mark on the complexity of the design while it put 80% on the feasibility solution. Please make sure to have a protect solution for concurrent operation since it is a very fundamental feature of any deamon implementation.

The rest 10% of the evaluation is reserved for your design on the operation if you can support more arguments instead of event id and you can prove the superior of your design to the upper event processing and program component. But in this case, you need an example of how to use your design in the next component and what its innovation is.

4. Submission:

You need to submit the following item in your compressed folder:

- A simple document describing your design should include at least a list of your system components and instructions on how to use the interface.
- a prototype code that is a proof-of-concept in which it can be demonstrated for the two basic read/write operations.