International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 624-629 www.ijaem.net ISSN: 2395-5252

A Comprehensive Analysis of Kernel Management

Guna R Ivan herald. W Mithunvinayak H, Dr.M. Sujithra M.C.A, M.Phil., PhD, Dr.A.D. Chitra M.C.A, M.Phil., PhD,

2nd Year,M.Sc Software systems(Integrated) Coimbatore Institute of Technology,Coimbatore.
Assistant Professor, Department of Data Science, Coimbatore Institute of Technology, Coimbatore
Assistant Professor, Department of Software Systems, Coimbatore Institute of Technology, Coimbatore

---

Date of Submission: 15-11-2020 Date of Acceptance: 30-11-2020

ABSTRACT: This paper presents a new architecture of operating system kernel. The new architecture discussed is based on modularity concept and consideration of the shortcomings of traditional kernel, and the operating system kernel is divided into three independent modules — executing module, policy module and monitoring module. Policy module determines the policy of process scheduling separated from traditional kernel. Monitoring module is responsible for monitoring processes, handling system error and detecting of important data. These three modules work independently and communicate with each other by interrupts to ensure the safety of CPU time, important system data structures and user data. As a result, these three independent modules improve the expandability and simplify maintenance of the operating system kernel and enhance the robustness of the system. In this paper, techniques' implementation is given and important data structures are defined. Finally, potential problems of this architecture are also discussed.

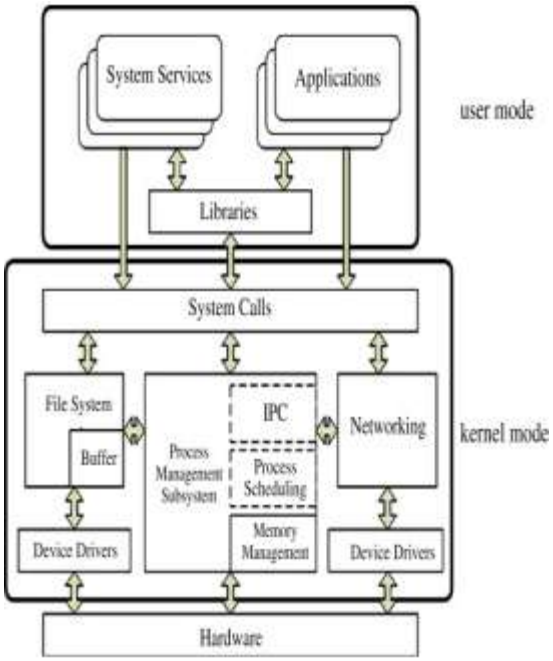
- **Previous** article in issue
- **Next** article in issue

Keywords:
Operating system
Kernel
Modularity
System safety

I. INTRODUCTION:

1.1 Classic kernel architecture:

Operating system implements two basic functions below,It must be simply use as extension machine and it must manage and distribute all kinds of resources reasonably as the manager of the computer system. In addition, some operating system also takes charge of the computer system's safety and provides application-specific services, such as networking, graphic interface and so on Linux and Windows family operating systems that are based on NT are most typical operating systems at the moment, but Linux focus on executing efficiency while Windows family pay more attention to convenient use. As to kernel architecture, Linux is a monolithic kernel operating system, and the whole kernel is very compact. Maintaining for this type of kernel is difficult, and the kernel takes up more memory space in running time. The architecture of Linux is shown in .

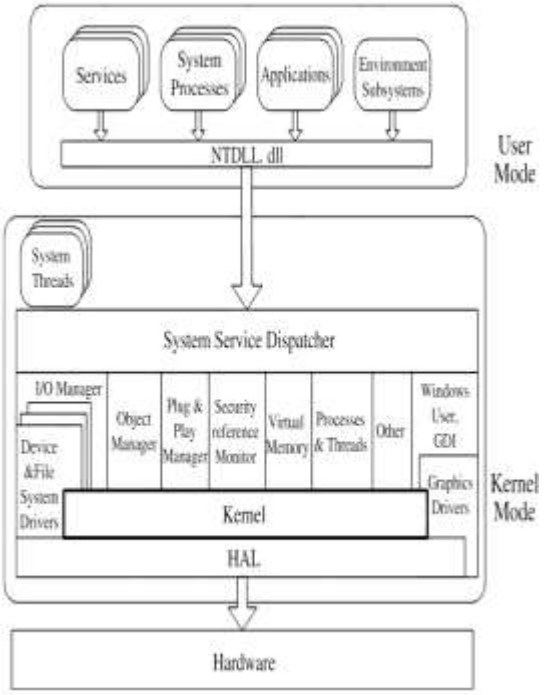


Download :Download full-size image

Fig.1: Architecture of Linux.

On the other side, Windows is a microkernel operating system. As a result, it makes many operating system functions modularity and is easy for maintaining and extension for operating system. However, the system running efficiency is lower than monolithic kernel. It is worth noting that the graphics drivers of Windows operating system are directly running on the hardware, thus Windows have better graphics capabilities, which makes up for the lower running efficiency. The architecture of Windows operating system is shown in

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 624-629 www.ijaem.net ISSN: 2395-5252



Download :Download full-size image

Fig.2.Architecture of Windows NT.

Actually, it is difficult to distinguish between monolithic kernel and microkernel because there is one trend of kernel developmentthe services provided by operating system that has a monolithic kernel are more modular, that is to say, this kind of operating system allow dynamic loading, while the services provided by operating system that has a microkernel are more integrated into kernel. So, the kernel architectures of almost all the current operating systems are hybrid kernels that are between monolithic kernels and microkernels. Linux and Windows is the case. Besides, there are peculiar kernels (such as nanokernels and exokernels) that will not be discussed in this paper.

1.2 Existing problems of current kernel

No matter for monolithic kernel, microkernel, or hybrid kernel, there is only one kernel in the operating system. This kernel not only monopolizes the rights of assigning all the system resources, but also holds the actual executing rights and safety monitoring. So the existing problems of current operating system kernel architecture are as follows at least: firstly, to some degree, single program (even if it is operating system) monopolizes all the software or hardware resources of system. Secondly, operating system is also a program that often makes mistakes, such as system error, system halt, BSOD and so on. If an error occurs in the kernel level, the whole system would crash. Finally, some applications can make use of some mechanisms in the operating system to capture the CPU time to execute all kinds of malicious or destructive operations, such as tampering data, monopolizing resources, even causing the system crash or user data missing.

To fully utilize the system resource, the operating system kernel's monopolization of the system resources must be changed, and the kernel error or bugs must be decreased and not causing the system crash. So, we need new operating system kernel architecture.

II. NEW ARCHITECTURE OF OPERATING SYSTEM KERNEL

In allusion to the problems of single kernel described above, we construct a new kernel architecture using modularity concepts and methods.

2.1Construct the new kernel

Policy for assigning the CPU time is separated from the traditional kernel, and a new kernel module is constructed, which works as policy module; then the monitoring and safety mechanisms are separated from the traditional kernel, combined with some exception handling mechanisms. Therefore another new kernel module is constructed, which can be named monitoring module.

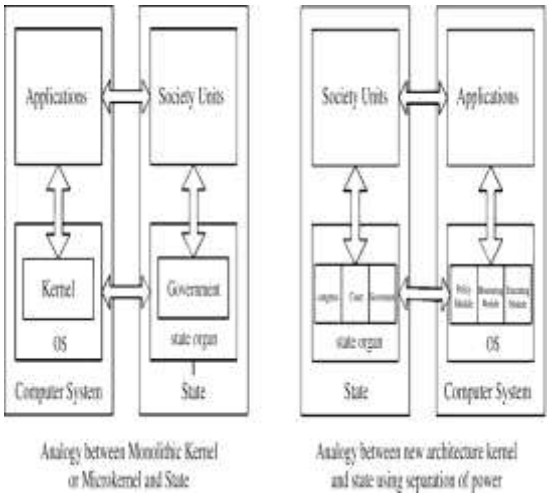
The CPU time is the most important resource in the computer system. All the computer system functions are implemented based on it. So it is very crucial for the system to make it independent and reasonably assign. Thus we construct policy module as a single kernel module. We expect that system can maintain in the stable basis not only when errors occur in applications and system services level but also in the traditional kernel level. Therefore we construct monitoring module as a single module. Besides some safety and monitoring mechanisms are added to the monitoring module, the remaining kernel is still responsible for the process control (creating process, switching process and IPC), memory allocation, I/O control and so on. Thus it is the third kernel, executing module.

Here we get the modular kernel architecture, but it is different from that in the hybrid kernel or microkernel. The modularity of hybrid kernel or microkernel is modularity in functions. That is to say, the services have the modularity characteristic but in one single kernel which does not have modularity characteristic; while the modularity of kernel described above

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 624-629 www.ijaem.net ISSN: 2395-5252

consists of three separated kernel modules. It is real modularity in kernel level.

If we regard the computer system as state, we can make an interesting analogy shown in Fig. 3. From the diagram, we can get that the architecture of new kernel that has three separated kernel module is very similar to state organizations that is based on the Separation of Power Theory.



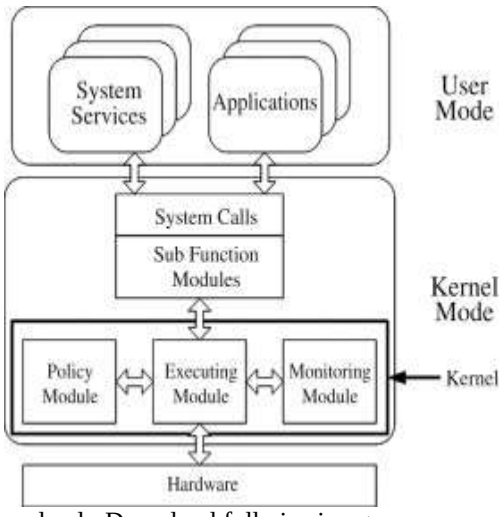
Download : Download full-size image

Fig.3. Analogy between computer system and state.

2.2.Communication between the three kernel modules

Interrupt mechanism is universally adopted in modern computer systems. In traditional kernel, process scheduling is implemented by timer interrupt and active call for functions are used for process scheduling. The functions of monitoring and safety in traditional kernel work in a similar way.

To unify the interfaces between the three kernel modules, the interrupt mechanism is employed for communication among the three modules. When executing module need to schedule the processes, no matter the routine preemption for CPU time or active call for process scheduling, executing module transmit the scheduling request to policy module through specific interrupt. Policy module decides which process to be scheduled according to given policy. Then executing module controls the process control submodule to implement process switching using the value returned by interrupt. Thus, communication is completed between executing module and policy module. The procedure of communication between executing module and monitoring module is similar to this. So we can get new kernel architecture of operating system shown inFig.4.



Download : Download full-size image

Fig.4. Kernel architecture of operating system based on modularity concept.

Detailed communication mechanism, necessary techniques and mechanisms, and important data structures of implementation are also discussed.

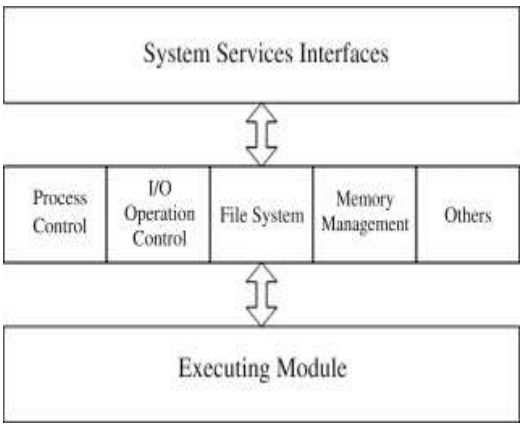
III. THE THREE KERNEL MODULES

As shown inFig.4, the kernel operating system is divided into three independence kernel modules. The executing module is the controlling core of functions and management. Policy module responds to the process scheduling request passed by interrupt, and decides which process can gain the CPU time; finally executing module completes the process switching. When there is an error, no matter from applications or executing module, the executing module will initiates an interrupt and being responded by monitoring module. Monitoring module will then make a variety of operations according to error types to maintain system safety. In addition, monitoring module detects the important system data structure and file system to protect system and user data.

3.1.Executing module

The remaining part of traditional kernel after separating process scheduling policy, monitoring of system and detecting functions, is the executing module. It implements process control, memory management, I/O operation control, file system management, networking control and so on. It must also complete the actual control of process switching, because the value returned by policy module is just referred to task ID. The structure of executing module is shown inFig.5.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 624-629 www.ijaem.net ISSN: 2395-5252



Download : Download full-size image

Fig.5. Structure of executing module.

The communication between executing module with policy or monitoring module will be discussed in detail in the descriptions of policy module and monitoring module. The reason why timer interrupt is used as the unique communication method is that policy module needs search for the right task from all the present tasks regularly, and monitoring module also makes use of this interrupt to implement the detecting for system important data structure and file system periodically.

3.2.Policy module

Executing module communicates with policy module through interrupt. What it needs from the returned value is just the ID of task that will gain the CPU time. Then according to this ID it completes the process switching. So policy module needs to select the right task or process that will obtains CPU from all the current tasks. Therefore, the following information must be confirmed:

•The interrupt that executing module communicates with policy;
•All the present tasks in the system;
•Task that will gain the rights to use CPU time.

Firstly, interrupt must be set:set_gate(&idt[n], &task_policy). Here,nis the interrupt number of timer interrupt;idt[]is the global interrupt description table;task_policy()is the executing function in policy module. Considering the communication between executing module and monitoring module also uses timer interrupt, so more common interrupts can be set:set_gate (&idt[n], &kernel_module). Here, kernel_ module() is a function used to implement the communication between executing module and specific module (policy module or monitoring module).

By default, the kernel_module()interrupt handling calls the task_policy()function in the policy module.kernel_module()can also call the executing function in the monitoring module through proper method, thus detecting function can be implemented. The detailed method is discussed in the section “Monitoring Module”.

Based on the interrupt, appropriate parameters need to be passed to handling to implement the different functions, such as process scheduling, error handling and system detecting. For a simple and unified interface used to communicate between kernel modules, a data structure must be introduced:

```
struct address_struct{
task_struct (*task_addr)[];
void (*function_addr)();
detect_struct *detect_addr;
};
```

This data structure is called as address structure, and all the members of this structure are address. So, task_struct(*task)[]is a pointer that points to task array;void (*function_addr)()is a pointer that points to the error function or process; detect_struct (*detect_addr)is a pointer that points the detecting address structure. This structure is discussed in detail in section “Monitoring Module”. The address of the address_structis the parameter of timer interrupt, and will be passed to the handling of int n. Interrupt handling decides to call policy module or monitoring module according to the value of address_struct. So the initial value of address_struct is one of the set{NULL, NULL, NULL}. If policy module is called, the value of task_addris not NULL, while if monitoring module is called, the value of function_addr or detect_addris not NULL. Thus, unified interface is constructed for the communication: timer interrupt int n, handling kernel_module(), and address structure address_struct.

task_policy()is the executing in policy module that is used to find the most reasonable task who will gain the CPU time. So task_policy()is the implementation of the process scheduling algorithm. According to the specific applications and scheduling goals, task_policy()can implement all kinds of scheduling algorithms, such as First-Come, First-Served, Shortest Job Fist, Round-Robin, Multiple Queues, Priority Scheduling, and even other customized algorithm. Therefore, process scheduling policy is separated from the

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 624-629 www.ijaem.net ISSN: 2395-5252

actual process switching. So, task_policy()can be implemented as follows:

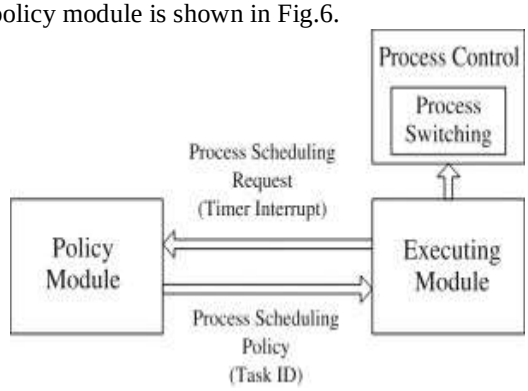
```
int task_policy(address_struct *addr){

//the number or ID of task will gain the CPU time.

int task_number;

//task is the pointer that points to the task array.

task_struct *task;

//get the address of task array from address structure.

 task=addr- >task_addr;

/*the implement of the scheduling algorithm */

//return the task number or ID to kernel_module()

Return task_number;

}
```

task_policy() returns the task number to kernel_module(), then kernel_module() calls the switch_task()function in the executing module to complete the process switching. So the parameters that switch_task()needs are the address of task array and the number of task which will obtain the CPU time.

Besides the implementation of process switch using timer interrupt, executing module and the subfunction modules can request the active process scheduling. In this case, we use int nin the request scheduling function. So request scheduling function schedule()can be defined as follows:

```
void schedule(void){

…

_asm{

…

int n;

…

}

…

switch_task(&task[], task_number);

}
```

The communication between executing module and policy module is shown in Fig.6.



Download : Download full-size image

Fig. 6.The communication between executing module and policy module.

3.3.Monitoring module

There are two functions for monitoring module: error handling and system detecting periodically. Errors discussed here is the fatal error that can endanger the system, such as hardware error, kernel running error, system services error, and illegal operation error of application. Other common bugs of applications should be captured and handled by the application itself. Thus, the structure of operating system can be simple, and it is convenient to design and extend system.

As described above, monitoring module communicates with executing module through int n and address structure address_struct. When a fatal error occurs in applications or system services, executing module captures the pointer (usually is the address) of the error programs, then the pointer is passed to monitoring module through timer interrupt int n. When the executing function error_handle()receives the pointer, it can terminate or restart the programs. If a fatal error occurs in kernel module, monitoring module can restart the executing module, which means initiating the number 0 task or restarting the operating system. Thus, system is protected from death loop. Of course, before all the error handling operations, necessary actions must be done to ensure the safety of important system and user data.

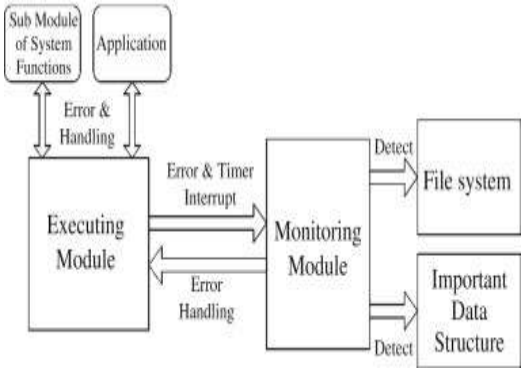
As for detecting periodically of monitoring module, some special method is implemented using timer interrupt int nand address structure address_struct. In Linux, the system timer clock is defined as 10 ms. So every 10 ms,int nis called, and executing module communicates with policy module once. Since monitoring module can use timer interrupt to implement system detecting, a counter that names module_counter is set. The initial value of module_counteris 0, when it is less

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 624-629 www.ijaem.net ISSN: 2395-5252

than 50, and the value of task_addr in the address_struct is notNULL, interrupt handling calls the task_policy(address_struct *)to implement the process scheduling. If the value of module_counter is 100 (that is 1s), interrupt handling calls the system_detecting(address_struct *)to implement the detecting for important system data structure and file system.

The important data structures that monitoring module detects includes: task array, buffer head, and description tables and so on. These data structures relate to safety and continuous running of system directly.

As for file system detecting, disk parameters, partition parameters and file directory are detected by system_detecting(). Besides the operations of file attributions and writing or reading file in large range, special function must be set to ensure the system safety. The communication between executing module and monitoring module is shown inFig.7.



Download :Download full-size image

Fig.7.The communication between executing module and monitoring module.

IV. CONCLUSIONS

In this paper, the kernel is divided into three independent modules according to their functions. This architecture has modularity to confirm the concept of modularity programming which is convenient for design, extension and implement of system. In addition, the communication interface between modules is simple and unified because it is implemented by timer interrupt and addresses structure. Moreover, this architecture intimately concerns with safety and protection of user data.

The main potential problem is the executing efficiency of system because of the extra communication overhead of three kernel modules. Possible improvements include: debugging is added to error handling in monitoring module, thus the programs can be debugged besides terminated or restarted.

REFERENCES

[1]. [1]Andrew S. Tanenbaum**Modern Operating System**(2008) pp. 3–6
Google Scholar
[2]. William Stallings, Operating Systems, Internaland Design Principles, 9th Edition, DorlingKindersley Pvt. Ltd., 2018