An Aspect Oriented Approach to Introduce Aspects in the Operating System

Jatin Arora
Chitkara University
Rajpura, Punjab, India

Pavneet Kaur
Chitkara University
Rajpura, Punjab, India

ABSTRACT
Software systems are very inflexible towards modification of already existing functionalities such as security, dynamic re-configurability, robustness etc. In such functionalities if need arises for any enhancements then it affects large fractions of the code. Thus results in difficult to implement. Such functional enhancements in any component of the system that affect large fractions of the program code, are often called crosscutting concerns. Such cross-cutting concerns can be solved by the new emerging extension to object oriented paradigm i.e. Aspect Oriented Programming (AOP). The main idea in AOP is the programmer's ability to affect the execution of core code by writing aspects. Aspects are pieces of code that are run before, or after core function for which aspect is written. The quantification part means that programmer can define points in the main program. Aspects should affect the main program by using some definition language that is usually a declarative one. The obliviousness means that the affected code does not need to know anything about aspects.

General Terms
Operating system, point-cut, cross cutting concerns, Join points, aspects, advice.

Keywords
Dependency injection, AOP, cross cutting concerns, Nachos, logs, security, authentication.

1. INTRODUCTION
We ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to download the template, and replace the content with your own material. Computers devices can perform operation in fraction of seconds and without any error which normal human require his lifetime to do. But computers as such are useless without the software to run on them. And all the software's are useless too without the Operating system which is the important part of computer system. Operating system is responsible for all the other programs to run on the computer. Various programming strategies have been applied for the development of operating system. Originally operating system was developed using assembly language or C language. Using C programming in the development of Operating system caused many problems, because C is a procedural language. Programming practices were not modular in the early computer's age and they weren't easily modifiable, but with the advent of modern computers, modular configuration is possible[5]. The performance of an OS is not as important as it was in the past because now day's computers are more advanced and powerful. The priority now is given to security and stability of OS rather than on its processing time and memory usage. Moreover, modern operating systems are highly complex to develop them in low-level programming languages, so a solution proposed to this problem is to use a high-level language.

Many new high level languages such as C++ and other object oriented programming languages have been used for improving the development of OS[3]. Various research operating systems have been developed using high level language but still, operating systems face problem with modularity. Complex interactions due to dependencies between the modules of the system threats module's boundaries and make them highly susceptible to errors. Security, authentication checks, exception handling, statistics, dynamic re-configurability are the major modules of any operating systems but if any modifications are required in these modules then it effects on the large fraction of code which is difficult to implement.

Problems related to dependency and understandability of these modules such as security, authentication etc are of highest concern. Program code of these modules are scattered almost in every module of the program so in order to modify these modules, we can't ensure that a change will require an effort proportional to the amount of code involved when key concerns lack locality. Such functional enhancements in any module of the OS that affect large fractions of the program code, are often called crosscutting concerns. These cross-cutting concerns can be handled by the new emerging extension to object oriented paradigm that is Aspect Oriented Programming (AOP)[1]. AOP deals with separation of concern in software development. Also, the significant feature of AOP is that the cohesion and modularity of a system increases the understandability and easing the maintenance burden.

Furthermore, in the era of the frequent changing specifications of customer, software needs to be modified, updated or even changed from time to time. To meet these modification requirements, developers of the software or even those programmers who are not familiar with that particular software have to meet this task. In either case, they need to read and understand the source code and fulfill the required changes. So, readability and understandability of the software is mandatory otherwise false conclusions will be made and applied which again leads to erroneous software. This results in lack of software quality which is the most important attribute of software development process[4]. Therefore, in order to maintain software quality software design should be built in such a way so as to make them easily understandable, testable, alterable, and preferably stable.

To achieve software quality, AOP is applied to an OS kernel to develop various aspects that can be used to introduce new features in the operating systems. This is achieved by using

the AspectJ compiler, which is an addition to standard Java compiler that makes it possible to use aspects, the modules defining crosscutting concerns, with Java programming language. In the proposed work with the inclusion of aspects, functionality of OS will be enhanced without affecting the entire OS code. This is done by designing code that will enclose all the major modules into aspects. Now any modification if required can be applied to aspects rather than whole OS code. Thus, it overcomes the limitation of scattered code, and provides flexibility for enhancing the core functionalities. The performance comparison of aspect implementation Vs existing OS implementation in java, is done on the basis of no of line of code added, execution time , number of locations affected in the cross cutting code. Also dependency among the various cross cutting modules is being migrated to aspects which in turn will lead to the reusability and the ease in understandability of the main code.

The operating system used for this work is Nachos and it is designed for research purposes. It is a rather small and simple operating system but still has all the important features of a real operating system.

2. ASPECT-ORIENTED PROGRAMMING APPROACH

Aspect-oriented programming is based on the concept of separation of cross cutting concerns[6]. This term refers to the division of a system into modules containing common program features or behaviors, often called concerns. Without this, it might be hard for a developer making a change in program code to identify specific concerns if they are scattered across multiple modules or possibly mixed with code implementing other concerns. The more a programming paradigm supports separation of concerns, the less that concerns are dependent and scattered across programs, which is especially important from the point of view of the maintenance and evolution of software. Separation of concerns is supported by all modern programming paradigms, such as OOP through the decomposition and composition mechanisms. Although the idea of separation of crosscutting concerns appeared much earlier with subject-oriented programming and adaptive programming, the aspect oriented programming paradigm proposed hereby is much more practical. AOP works because of the mechanisms for orthogonal isolation of so-called cross cutting concerns into separate modules. As a result, the program code created is more reusable and less tangled. This code consists of two parts: base code that is more purpose specific and additional code that is represented by aspects. An aspect contains advice, additional code executed at a join point, a particular point in the execution of a program, such as method invocation, or field access. An executable program is produced through the process of merging both code parts, often referred to as weaving aspects into classes. Weaving rules or point cuts are written in aspects or in the base code, depending on the implementation technology; they specify rules that join points need to satisfy. Overall, this mechanism allows developers to concentrate on business logic and cross-cutting concerns separately.

The main idea in AOP is the programmer's ability to affect the execution of core code by writing aspects. Aspects are pieces of code that are run before, after or instead of some core function. The quantification part means that the programmer can define points in the main program that aspects should affect by using some definition language that is usually a declarative one. The obliviousness means that the affected code does not need to know anything about aspects.

For example logging is a good example of using aspects. To log all function calls the programmer simply needs to define a logging aspect that is executed before and after each function call in the program. This logging aspect then writes the name of the called function to the log file. In this way the logging code is neatly separated in its own module and the logged program does not even need to know that it is being logged.

3. NACHOS

Nachos is research operating system which is developed by University of California in Berkeley[7]. Originally Nachos was written in C++ language; however, Java version is used for the present work. Nachos simulate the real operating system in the sense that it has all the important features as real operating system does for e.g. Nachos includes interrupts handling, virtual memory management and process management. The difference which lies between Nachos and a real operating system are that Nachos runs as a UNIX process, which is not the case with real operating system as it runs on hardware machine itself.

Nachos is simple and smaller OS to study and maintain as nachos does not run directly on hardware, it runs as a UNIX process so it is free from the task of performing I/O which is the work of real OS[8]. Thus, Researchers can easily perform experiments in Nachos as now they don't have to switch between development environment and the Nachos. In Nachos user programs are written in binary format which can be used with MIPS compiler and make Nachos work as real OS. MIPS simulator executes MIPS instructions in user programs as real MIPS CPU executes taking into account concept of looping, fetching MIPS instructions and executing them using simulated machine memory and registers which acts as data structures in nachos program.

In this work Nachos 5.0 j version is used. Nachos java version is more useful and popular than Nachos C++ version as Java is much simpler than C++. Because Java is type-safe language while C++ is not type-safe. Java is portable. Also, In Java, low level errors cannot occur or if even occur is catch by exception handling mechanism while this feature is not supported in C++ language.

Nachos is more advanced and powerful than the other simulated OS in the following ways:

1) In nachos, the program which runs on it is required to be compiled in MIPS, without interrupting the working of host OS i.e. cross-compiler is needed only to run an instructional OS (which is Nachos) while in other instructional OS such as OS/161 cross-compiler is needed for two purposes, one is for the programs which run under simulated OS and one the host OS itself[9].

2) There is another major difference between Nachos and other instructional operating system which is other simulated OS are written in C/ C++ Language while Nachos is present in C, C++ as well as Java language.

3) In Nachos one can absolutely simulate the concept of network of workstations, by connecting a UNIX socket to all the Nachos machines which are running UNIX process. Thus threads on one machine can communicate with others using simulated networks.

4) In Nachos timer is responsible for deterministic simulation. It increments the clock whenever instruction is executed by user programs or whenever interrupts occurs.

5) Machine-dependent interface Layer of Nachos provides hardware simulation which is hidden from user.

Software Industry is growing at a fast pace. To meet the daily changing requirements of customers, software companies have to develop smart software's which are able to compete with the market values. Due to this huge target, software are made complex day by day which in turn badly effects understandability, simplicity and maintainability like quality attributes of software. A product is known by its quality, if quality degrades then the product has no utility. So, software developers should develop software which is easy to understand as well as maintained. Understandability is an important attribute of software, as it is rightly said that one cannot maintain or modify software that one doesn't understand. If maintainer is not able to understand what developer has developed, then no fruitful work can be achieved. Thus understandability enhances the maintainability of software which can be achieved if developer sticks to the simplicity of software.

Software system which is considered in this work is Operating System. It is so complex that it cannot be managed by dividing it into sub-modules. Even it has problem with modularity also[3]. Interdependencies among modules results in disappearance of their boundaries and so it is difficult to debug and understand. The major components which are highly affected from this drawback are security, error management and logging. Coding modules that are used to implement these components are dispersed throughout the whole software due to which these components are categorized as cross cutting concern. These concerns don't follow the mechanism of dividing the system into modular units as their main goal is to provide functionality to the various modules so their code cannot be modularized into single unit [16].

Cross cutting concerns are difficult to implement as their coding and placement required lot of effort because there is no fix location of their placement as well if any modification is to be applied to them then every occurrence of these concerns need to be changed which is error prone as they are dispersed throughout the system which makes this task complicated and time consuming [15]. Due to lack of modularization and inter dependencies associated with these concerns, a solution is being demanded. In present work, solution to this problem is proposed which is adoption of Aspect Oriented Programming. AOP deals with such code handling and provides a mechanism that can be applied on such modules. It introduces the concept of aspects which are powerful units, in the sense that they have capability to centralize the scattered or distributed functionality. To execute AOP, AspectJ language is used which is open-sourced and is an extension to already built Java. AspectJ offers many advantages as it is based on java and implements features like aspects. By using AspectJ one can easily program without much burden of tangled and complicated code. Also software developed with AspectJ is easy to debug, document, reusable, refactor, maintainable and modifiable.

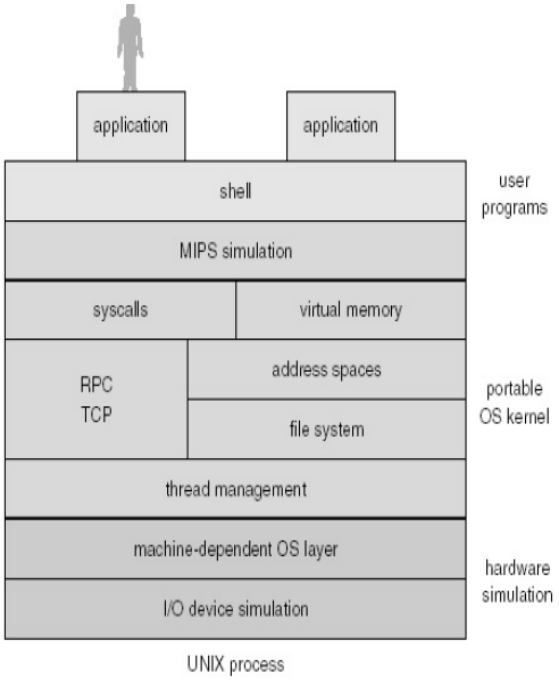

Figure 1: How the major pieces in Nachos fit together

3.1 Architecture of NachOS

In modern personal computers, operating system is the most important piece of software. It is the link between computer hardware and computer software. Operating system allows other programs to use computer resources and peripherals in a hardware independent way, without worrying about the low-level implementation details [19]. Most modern operating systems also allow execution of multiple programs at the same time while making this process completely transparent to the application programmer. A virtual operating system[9] accomplishes the same tasks as a regular operating system, only it doesn't run on a real hardware but rather on a virtual computer often called the virtual machine. Nachos simulate a real CPU along with memory management, interrupt handling and hardware devices. Nachos.machine of nachos includes the following simulations [8].

3.1.1 Boot Process

On running, nachos invokes nachos.machine.Machine.main file. Nachos devices like timer, interrupt handler, serial console, and processor are initialized by Machine.main class.passes control to the Auto Grader which will create a Nachos kernel and start the OS.

3.1.2 Nachos Hardware Devices

The Nachos machine simulation includes several hardware devices. Nachos.machine.machine provides access to the hardware devices which are listed below:

- Machine.interrupt()
- Machine.timer()
- Machine.console()
- Machine.networkLink()

3.1.3. Interrupt Handler

Interrupts are handled by Interrupt class of nachos.machine which maintains an event queue with a clock. The clock is maintained in software and ticks are executed depending on the following criteria:

- One tick is implemented for a MIPS instruction. With one tick of MIPS simulator, clock is advanced by one tick.
- Ten ticks are executed to re-enable the interrupts.

After any tick, interrupt checks for pending interrupts also. Device event handler services these pending events. It is to be noticed that device event handlers which are part of hardware simulation is used for this purpose and not software interrupt handlers. Important methods which are accessible to other hardware simulation devices are:

schedule() takes two arguments- time, device event handler and then accordingly schedules the specified time with respect to the specified handler.

tick() consists of a Boolean value(i.e. either 1 or 10 ticks). Depending upon Nachos status of user or kernel mode clock is advanced 1 or 10 tick.

setStatus() is called when interrupt is enabled from disabled mode and for each user instruction

Processor.run() is executed. checkIfDue() calls event handlers for due interrupts. tick() is responsible for calling these handlers.enable() and disable() interrupts are invoked by simulators of Interrupt class. Other hardware devices of Nachos rely on interrupt device only.[18]

3.1.4. Timer

Real-time clock is simulated by Timer class of Nachos, which at regular intervals generates interrupts. Machine.timer() is executed to generate event driven interrupt which is applied to implement Timer. getTime() and setInterruptHandler() are the two operations which Timer class supports. From the initializing of the Nachos getTime() is responsible for counting and returning the number of ticks till Nachos halts. Stats.TimerTicks ticks occurs when setInterruptHandler() is invoked by Timer class which sets the timer interrupt handler. Preemption is responsible using Timer.

3.1.5. Serial Console

Out of three classes which Nachos provides for I/O devices, serial console is considered as the simplest one. SerialConsole class specifies the serial console which simulates the working of serial port. Machine.console() returns the serial console of a machine. Further, serial console consists of unblocking read and write primitives.

Read operation checks and returns the data byte if it is to be returned, else it returns -1. On arrival of another data byte, receive interrupt is executed. But as only one data byte is handled at a time therefore interrupts for two received bytes cannot be generated without having one read operation in between them. Transmission of data byte by a write operation takes place and control is immediately returned after one write. Now a send interrupt takes place if required. No two writes can occur without having read in between them. If such situation occurs then data which is actually transmitted is undefined.

3.1.6. Simulated Disk

Static data is stored in a Disk. It works on a simple concept of consisting of a surface which is divided into tracks and tracks are further divided into sectors. Sectors can be read or write asynchronously one at a time by the operating system. Disk interrupts is generated when a request is fulfilled. Disk or Unix file saves the data of simulated disk.

3.1.7. Network Link

Network Link class is responsible for communication of different instances of Nachos over a communication channel. Machine.networkLink() returns an instance of this class. Network Link sends and receives messages in the form of packets at a time whereas Serial Console send and receives bytes at a time. Other than this difference no major dissimilarity is found in Network Link interface and Serial Console interface.

Packet class in network link creates instances of packets. getLinkAddress() provides a link address which is a unique number associated with each link on the network. Packet comprises of two field - one is header in which the link address of source machine who is sending the packet is enclosed and second is information of no of data bytes in the packet is mentioned. Network hardware only analyzes the header of the packet and not the data bytes. Header also contains the destination link address where packet is to be dropped, so a specific link who is selected to transfer the required packet transmits the packet to header destination link address.

Working of rest of the NetworkLink interface is similar to SerialConsole interface i.e. now the receive() is invoked by a kernel to check the arrival of a message. If packet has not arrived then it returns null otherwise it generates interrupt. Kernel can further transmit packet using send() or it waits for send interrupt.

4. ASPECTJ

AspectJ is an aspect-oriented extension of the Java programming language[13]. It resulted from research into aspect-oriented programming at Xerox Parc in the 80s and 90s and saw its first release in 1998. It is now being developed as part of the Eclipse project. Of the several different implementations of aspect-oriented ideas, AspectJ is, as of this writing, by far the most popular, both in industry and in academia.

AspectJ in AOP allows programmers to have the advantage of modularization for cross cutting concerns that are present in almost every part of software[2]. In OOPs like C++ or Java, class is considered as modular unit. Similarly in AOP aspects provide the same functionality to the cross cutting concerns which provides functionality to more than one class. Program in AspectJ is compiled with its compiler and is then run with a runtime library.

One of the goals of the AspectJ project is for it to function as a large scale software engineering experiment to validate the ideas of aspect-oriented programming in real-world contexts[5]. Consequently, its design has been driven by the

desire to develop a large and active developer community by making the language easy to learn for current Java programmers and by making it easy to incorporate elements of AspectJ into extant Java systems. As such, AspectJ is a strict extension to Java: every valid Java program is a valid AspectJ program. Furthermore, AspectJ compiles to normal Java byte code that can be executed in a standard JVM, not requiring a specialized runtime environment. AspectJ extends Java with a new top-level construct: the aspect. The aspect is AspectJ's unit of modularization for crosscutting concerns. A concern whose implementation, in Java, was inevitably scattered across multiple classes or methods, entangled with the implementations of other concerns, should, in AspectJ, be neatly encapsulated within an aspect.

AspectJ is an asymmetric aspect-oriented language in that it distinguishes between core and crosscutting concerns, specifying them differently. Core concerns continue to be implemented in pure Java, modularized within classes and methods. Their implementation is referred to as the base program. Crosscutting concerns are implemented in aspects, using an extended syntax of Java. The aspects and base program are composed together to produce the complete program. The features AspectJ provides for implementing crosscutting concerns in aspects can be classified into two groups: dynamic crosscutting features and static crosscutting features. The dynamic crosscutting features are those that implement crosscutting concerns by modifying the runtime behavior of a program; static crosscutting features modify the static type structure of a program.

The following sections will provide a brief introduction to these AspectJ features.

4.1 Dynamic Crosscutting

An aspect is analogous to a class in many ways. Like a class, it can have methods and fields[14]. It can extend another class or aspect and can itself be extended. It can be concrete or abstract. An aspect, however, may also contain several special AspectJ constructs: point-cuts, advice, and intertype declarations. The first two implement dynamic crosscutting, and is discussed in this section; the latter implements static crosscutting and are discussed below.

The dynamic crosscutting features of AspectJ are those that implement crosscutting concerns by means of modifying the dynamic behavior of the program. The nature of these features can be illustrated by analogy to the observer pattern. Conceptually, an aspect may be considered an observer, with the execution of the whole program the subject. The aspect observes the execution of the whole program, and at particular points within the execution, modifies the behavior of the program by executing new code. The points at which new code can be injected are called join points, and the code that is injected is called advice. A point-cut is a pattern that selects join points of interest, and every piece of advice has an associated pointcut.

To actually implement AspectJ in this fashion would be terribly inefficient. It would also require special VM support (which would conflict with AspectJ's goal of easy adoption by Java developers). Instead of a literal implementation of aspects as observers, aspects and base program are composed statically in a form of partial evaluation. This is known as weaving. A join point shadow is the static counterpart of a join point. Or, equivalently, a join point is a particular execution of a join point shadow. The weaver inserts instructions at join point shadows to execute the advice that would apply to the corresponding join points. Since a single join point shadow may correspond to an arbitrary number of join points, and since not entire join points may be matched by a particular pointcut, the weaver often needs to add a runtime check to the code inserted at the join point shadow. This is known as a dynamic residue. If the dynamic residue specifically tests the applicability of advice at a given join point, it is called an advice guard.

4.2. Join Points

Join points are the most fundamental of the concepts AspectJ adds to Java. A join point is a particular point in the execution of a program, a specific runtime event. An aspect-oriented language's join point model defines what runtime events are exposed as join points. In AspectJ's case, the following events are exposed as join points:

- method call and execution
- constructor call and execution
- field get and set
- class initialization
- object initialization and pre-initialization
- exception handling
- advice execution

Not every possible join point is exposed. These particular events have been chosen because they are relatively stable in the face of compiler optimizations and some code refactoring. Other potential join points, such as entry into a loop or other control flow structure, are much more volatile in the face of such code transformations and so are not exposed. It is important to realize that a join point is not an atomic point, but rather a region of execution. A join point has a beginning, it has an end, and it can contain other join points.

4.3. Pointcuts

A pointcut is a pattern that matches join points[14]. A pointcut may also specify some context that should be exposed to advice at a join point—the target object or the arguments of a method call join point, for example. Pointcuts are specified by the programmer in the pointcut definition language, whose syntax is distinct from that for the rest of AspectJ. A pointcut is either a primitive pointcut or a compound expression composed of other pointcuts and Boolean operators.

Primitive pointcuts can be classified into three groups: those that match join points by their kind; those that match join points based on their static context; and those that match join points based on their dynamic context. The first two groups can be matched statically, while matching of the third may require dynamic residues.

4.4. Advice

"Is a method-like construct that contains additional behavior to be added at the matched joint point". The advice is what is inserted into the join points. The advice is used to express the cross-cutting actions that must happen within the method body at the matched join point. Advice generally represents a fragment of control and data that must be added to the body of an existing method. There are three kinds of advices: before advice, after advice, and around advice. Before advice: A before advice executes its body before executing the body of

the matched join point. After advice: An after advice executes after executing the body of the matched join point. Around advice: An around advice body surrounds the match join point. Around advice may change the execution of the matched join point, or may even replace it."

4.5. Aspects

Aspects, as described at the beginning of this chapter, are the basic modular units of crosscutting concerns, and are very similar to classes in many ways. Aspect is the part of code describing how point-cuts and advices should be combined together. Here this is referred to as a point cut definition; it defines a certain join point and "cuts" into the classes to get references to certain variables inside.

4.6. Weaver

The weaver is a sort of compiler. It takes the advices and inserts the advices at the appropriate Point cuts and creates the additional code needed. It "weaves" the source-code and advices together with the aspects as the template."

5. WORKING MODEL

The working model of the work is represented by the flow diagram in figure 2. The cross cutting concerns are implemented by using Aspect Oriented concepts as well as by using Object oriented paradigm i.e Java. The comparison can be done on the basis of LOC, Execution time, reconfiguration.

6. CONCLUSION

After much research Nachos operating system was found suitable with the requirements of the work to be done. Nachos java version is used in which AspectJ can be embedded. Nachos is not only simple but it also provides ease to programmers to modify and implement its code. With the use of aspects in Nachos, the tangled code is shifted from the main module into the aspect module thus making the design and implementation of OS clearer and understandable. The maintainability of Operating system is also enhanced in the AOP implemented code as the design consists of separated concerns and the required changes are applied only to the particular aspect that is associated with module to be maintained. While adding aspects in the system, its severity and tangling of the code should be taken into consideration. If necessity arises to include aspects then only it should be made. Unnecessary inclusion of aspects in the system should be avoided because work required for it is entirely wasted due to unwanted weaving process. Also another point which should be focused while using such paradigm is that there are more source code files if an application is implemented by AOP. The programmer of the code has to do more efforts by frequent switching among the aspect files. This makes it a bit difficult to understand the design and program control flow. But this is overcome by the fact that the understandability of the core code is very high even when the new and extended features are encapsulated into the aspects

7. FUTURE SCOPE

It is a known fact that it is complicated and difficult to develop a new and good software system. So, research has to be conducted out on aspect-oriented paradigms and compilers too. In future, work can be done upon reducing the number of source code files, if an application is implemented by purely aspect oriented paradigm. As of now the programmer has to do more efforts by frequent switching among the aspect files. This makes it a bit difficult to understand the design and program control flow.

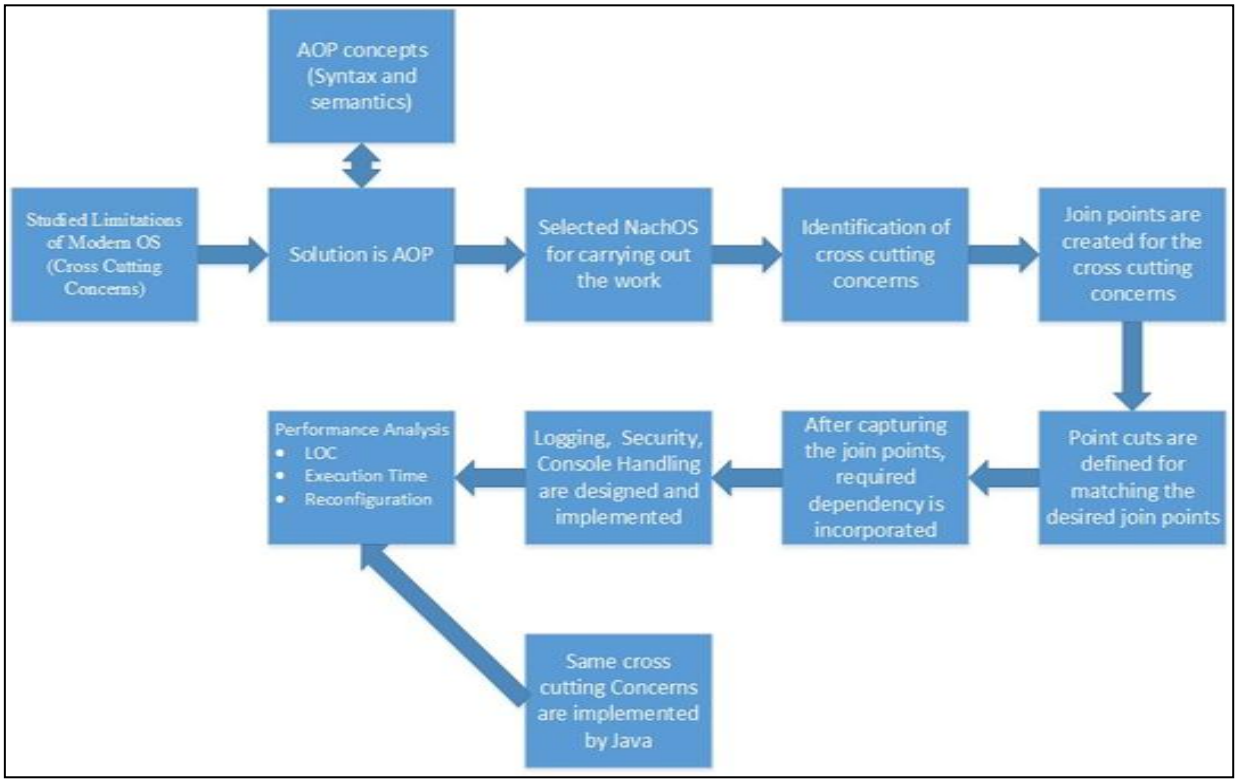

Fig 2 Working model of Nachos using AOP

7. REFERENCES

[1] Hanenberg, S., Kleinschmager, S., and Walter, M. 2009 "Does Aspect-Oriented Programming Increase the Development Speed for Crosscutting Code", Third International Symposium on Empirical Software Engineering and Measurement, IEEE, Feb 2009

[2] Zhang, Y., et.all, 2009 "Implementing and Testing Producer-Consumer Problem using AOP", Fifth International Conference on Information Assurance and Security, IEEE, 2009

[3] Coady, Y., Kiczales, G., and Feeley, M. 2000 "Exploring an Aspect-Oriented Approach to Operating System Code", September 2000.

[4] Rashid, A., Cottenier, T., Greenwood, P. and Chitchyan, R. 2010 "Aspect-Oriented Software Development in Practice", Computer Society IEEE, Feb 2010.

[5] Murphy, G.C., Walker, R. J., and Baniassad, E. L.A. 1999. "Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-Oriented Programming". IEEE transactions on software engineering, vol. 25, no. 4, July 1999

[6] Murphy, G., and Schwanninger, C. 2006. "Aspect-Oriented Programming", Computer Society, IEEE, 2006

[7] Christopher, W. A., Procter, S. J., and Anderson, T. E. 2005 "The NachOS Instructional Operating System", 2005.

[8] Narten, T. 1995. "A road map through NachOS", 1995

[9] Niu, J. 2003. "NachOS Overview", Operating Systems, CS-CCNY, Fall, September 2003

[10] Chiba, S., and Ishikawa, R. 2005. "Aspect-Oriented Programming Beyond Dependency Injection", Springer-Verlag Berlin Heidelberg 2005

[11] Brichau, J., et all. 2006 "A Model Curriculum for Aspect-Oriented Software Development", IEEE Software, December 2006

[12] Anderson, C. L., and Nguyen, M. 2005. "A survey of contemporary Instructional Operating System for use in Undergraduate Courses". JCSC 21, Oct 2005

[13] Laddad, R. 2003. " AspectJ in Action- Practical Aspect-Oriented Programming", Manning Publications co. 2003

[14] Gradecki, J.D. 2003. "Mastering AspectJ-Aspect Oriented Programming in Java", Wiley Publishing, Inc, 2003

[15] Lieu, W. L., Lung, C. H., and Ajilla, S. "Impact of Aspect Oriented Programming on Software Performance: A Case Study of Leader/Followers and Half-Sync.

[16] Lamping, J., and Kiczales, J. 1993. "The Need for Customizable Operating systems", IEEE, 1993

[17] Ceccato, M., and Kessler, F. B. 2007. "Migrating Object Oriented code to Aspect Oriented Programming", Software Maintenance, ICSM International Conference, IEEE, 2007