Towards an Operating System for the Campus

Pengfei Yuan, Yao Guo, and Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
School of Electronics Engineering and Computer Science, Peking University
{yuanpf12, yaoguo, cherry}@sei.pku.edu.cn

ABSTRACT

Almost every computing device runs an operating system, which is responsible for managing different resources on the device and providing higher-level programming abstractions. This paper proposes CampusOS, an operating system which is responsible for managing networked resources on university campuses, including data of students, teachers, courses, organizations, and even data generated from users' computing devices. CampusOS provides flexible support for campus application development with SDKs consisting of campus-related APIs. CampusOS features and SDK APIs can also be extended by developers easily. We discuss the design of CampusOS, as well as its challenges.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*software libraries*; D.4 [**Operating Systems**]: Organization and Design

General Terms

Design, Management

Keywords

Internet operating system, Middleware, Campus application development, Internetware

1. INTRODUCTION

Operating systems are running on almost all modern computing devices, such as PCs, smartphones, embedded systems and even sophisticated sensors. The key functionalities of an operating system are managing various kinds of resources (CPU, storage, peripherals, etc.) and providing programming and runtime support.

With the prevalence of the Internet, many computers are connected together, forming a grid or cloud, in order to support more complex applications. If we treat a cluster, a data center or even the whole Internet as a computer, we can envision that many applications are "actually" running on the network of computers. Consequently, many researchers have proposed various operating systems for cloud or even the Internet. For example, Tim O'Reilly proposed his vision of an Internet Operating System [7]. At the same time, many other operating systems have been proposed for cloud computing [10], networked home appliances [3], commercial buildings [2], and even connected city blocks [4].

In contrast, this paper focuses on the evolvement of applications and services running on closely-connected communities such as a university campus, especially mobile applications targeting community members and activities. Developing applications for communities like university campuses requires a lot of trivial work. A developer would typically need to figure out how to retrieve and manage users' personal information, schedules, social relations, etc. Large amount of time is spent on these lower-level details, instead of focusing on implementing and improving the key functionalities of the applications.

For a closely-connected community such as a university or a similar organization, different applications have much in common at the lower level. Providing higher-level APIs for commonly used lower-level features can improve the development process significantly. For example, identifying a specific place (such as a building on campus) with a smartphone requires access to the sensors on the device and the campus building layout information, as well as a positioning algorithm. As many campus applications will require the same identification feature, we can provide common APIs or services for outdoor (or even indoor) location identification, such that all location-based campus applications do not need to reinvent the wheel.

In order to support the existing (and emerging) applications on campus, we argue that an operating system is needed to provide sufficient support during application development, deployment, execution and maintenance. We design CampusOS as an open, community-oriented operating system for university campuses. Just like traditional operating systems, CampusOS provides programming interfaces, libraries, and runtime management of applications, although at a higher level. Many components of CampusOS depend on traditional operating systems like Windows and Linux, as well as networked resources in the cloud and on the Internet.

Many people may wonder why we need another operating system as many features provided by CampusOS can be implemented separately. We illustrate the necessity of an operating system, as well as the contributions of CampusOS, from the following aspects.

- CampusOS can provide support for students who are interested in building personalized applications and excited to share them with friends.
- CampusOS can provide support for courseware and course-related applications with course-specific customizations.
- CampusOS can help manage different communities on campus, such as student union, student clubs, interest groups, sports teams, etc.
- CampusOS can also serve as a research platform where researchers can distribute applications for experimental purposes and collect user information with permission.

CampusOS belongs to a larger ongoing project, in which we are constructing a prototype of an Internet operating system, which is a development and runtime supporting environment for Internetware [5, 6].

2. CAMPUSOS DESIGN

We adopt a hierarchical architecture in CampusOS, which is similar to traditional operating systems. In this section, we first present our design goal to make CampusOS an open and extensible ecosystem. Then we describe the specific design of CampusOS.

2.1 Goal

CampusOS is intended to serve as an open platform for campus applications, which means that not only can developers publish new applications to CampusOS, but they can also add new features to CampusOS. CampusOS is designed with the principle of extensibility in mind, such that new features can be easily added into it. When some developers come up with interesting new features, they can choose to share the features with other developers by adding them to CampusOS. With more and more features added into CampusOS, more complex campus applications can be built, which will in turn attract more users and more developers, thus forming an ecosystem.

Besides its open and extensible characteristics, CampusOS supports multiple computing platforms to better adapt to the development and deployment of campus applications, including traditional PCs, smartphones, tablet computers and web browsers. In the meanwhile, CampusOS should protect the security and privacy of users' data against attackers and malicious campus applications.

2.2 Architecture

Figure 1 shows the hierarchical development architecture of CampusOS. The bottom layer is the CampusOS Core and the underlying implementations of the whole system. This layer corresponds to the development of the CampusOS system. The middle layer consists of service drivers on the server side and the CampusOS SDK and service daemons on the client side. This layer corresponds to the development and extension of CampusOS features. The top layer is based on the CampusOS SDK, including CampusStore and campus applications. The development of various campus applications belongs to this layer.

Just like SDKs in traditional operating systems, the CampusOS SDK is the key component that provides development support for campus applications. To support multiple

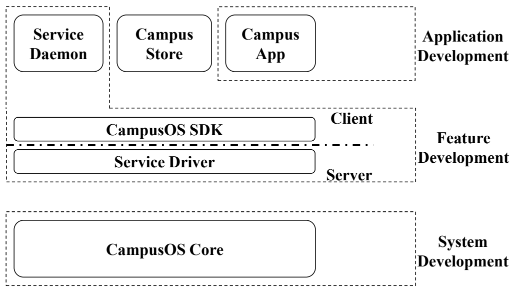

Figure 1: The development architecture of CampusOS

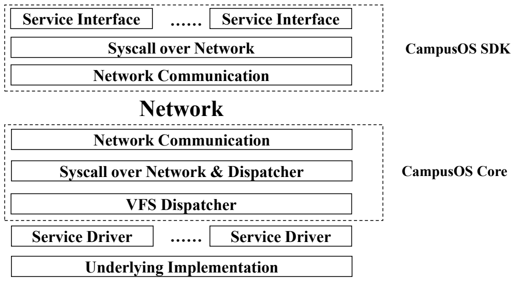

Figure 2: The runtime architecture of CampusOS

computing platforms in CampusOS, the CampusOS SDK supports multiple programming languages, such as C, C++, Java, C#, JavaScript, etc.

For better extensibility, we adopt two abstractions that are originated from traditional operating systems and use them in CampusOS. One is the system call (syscall) and the other is the virtual file system (VFS).

Figure 2 shows the hierarchical runtime architecture of CampusOS. It illustrates the relation between the CampusOS SDK and the CampusOS Core. The top layer consists of various service interfaces, which provide campus applications with APIs for accessing various features. The service interfaces invoke CampusOS syscalls, which are encoded and passed to the CampusOS Core via network.

The CampusOS Core running on the server side accepts syscalls, decodes and dispatches them accordingly. The VFS layer maps service interfaces to their corresponding service drivers. This mapping actually correlates feature invocations in campus applications to feature implementations in CampusOS.

2.3 Feature Extension

In our design, CampusOS features are attached to VFS nodes and accessed via syscalls. In this way, we achieve the principle of extensibility. On the one hand, we provide uniform interfaces for accessing different features in the CampusOS SDK. On the other hand, we provide a general approach to organizing different CampusOS features. In short, syscall is the interface for VFS manipulation, i.e., actual CampusOS feature invocation.

Since all developers can contribute features to CampusOS, we design four different approaches to service driver implementation, where syscalls are handled to extend CampusOS features, to better adapt to the goal of extensibility. Different approaches correspond to different levels of isolation between the service drivers and the CampusOS Core.

1. Service drivers can be statically linked into the CampusOS Core.
2. Service drivers can be dynamic libraries and loaded by the CampusOS Core at startup.
3. Service drivers can be separate processes running on the same server as the CampusOS Core.
4. Service drivers can also be hosted on separate virtual machines or servers.

In the first three approaches, the service drivers share the same runtime environment with the CampusOS Core, so the service drivers should be audited and tested before actual deployment. In Approaches 1 and 2, the service drivers share the same virtual address space with the CampusOS Core, so the services drivers should be tested more carefully to avoid crashes, which will directly affect the CampusOS Core. In the first approach, which has the lowest level of isolation, service drivers can even access internal data structures of the CampusOS Core. This approach is only used in a limited number of features, such as session management. In our design, the last approach, which has the highest level of isolation, should be the main approach in which community developers contribute features to CampusOS.

Based on the architecture of CampusOS shown in Figures 1 and 2, we summarize three patterns for feature implementation, which cover most common scenarios where features of CampusOS are extended.

1. Features encapsulate existing services provided by the third party, such as cloud storage service, social network service, etc. This pattern is illustrated in Figure 3(a) for cloud storage service.
2. Features involve data related to users, such as users' location information. This pattern is illustrated in Figure 3(b) for location service.
3. Features involve public data, such as campus news or notifications. This pattern is illustrated in Figure 3(c) for campus news service.

The difference between Figures 3(b) and 3(c) is that the service daemon in Figure 3(b) runs on every CampusOS user's device and the crawler in Figure 3(c) only runs on one server.

2.4 Security and Privacy Protection

In our design, we take a variety of measures to protect the security of CampusOS and the privacy of user information.

Firstly, all the encoded syscall data transmitted over network is tunneled through Secure Socket Layer/Transport Layer Security (SSL/TLS). So user data is protected against man-in-the-middle attack during transmission. User data stored on the server is encrypted and can only be directly accessed by the CampusOS Core and trusted service drivers, i.e., service drivers that have lower levels of isolation to the CampusOS Core.

We also design an access control mechanism, which is based on application-specific passwords, for other service drivers and campus applications to access user data with permission. The authorization process works as the user grants a subset of his/her privileges to a password and delivers the password to a campus application. This process can be applied when campus applications are installed from CampusStore. The authentication process works reversely as the campus application logs into CampusOS with the password and exercises the subset of privileges granted to the password on behalf of the user via certain service drivers.

More sophisticated mechanisms for protecting cooperative privacy and colocation privacy are currently under research. They can provide more fine-grained privacy protection than this basic access control mechanism.

3. DISCUSSIONS AND RELATED WORK

In this section, we discuss challenges beyond our design, relations and differences between CampusOS and traditional operating systems and middleware, and related work.

3.1 Challenges

One major challenge in CampusOS involves its open and extensible characteristics. Since many features in CampusOS will be contributed by application developers instead of system developers, how to test these features on CampusOS without service downtime or broken functionalities is critical and very challenging.

As there are already many campus applications developed and running, we need to consider how to migrate these existing services and applications to CampusOS. Because CampusOS is designed as an open community project, user and developer involvement is critical to its success. How to promote CampusOS in order to increase community involvement can also become an imminent challenge once the initial development and deployment are completed.

3.2 OS Evolution

Since the first working operating system was developed on IBM mainframes, many operating systems have been built in the last 50 years. However, ever since Windows and Linux dominated the market, the main functionalities and structures of traditional operating systems running on single machines have kept unchanged for many years.

As we enter the Internet era, many people have proposed to build an operating system for the "Internet as a computer" concept. O'Reilly has proposed the idea of Internet Operating System in 2010, arguing that an Internet OS should contain new information subsystems and provide higher levels of abstraction [7]. Although we have not seen a working operating system for the entire Internet, we have witnessed various approaches to developing operating systems for software running on the Internet. For example, TransOS [10] is proposed to manage the cloud from the viewpoint of transparent computing. Urban OS [4] is proposed as a software platform for accelerating the development and deployment of urban technology and devices. HomeOS [3] provides a centralized and holistic control of devices in the home. CleanOS [9] uses trusted, cloud-based services to limit exposure of sensitive data on Android. BOSS [2] provides a set of operating system services which supports applications on top of the distributed physical resources present in large commercial buildings.

These "new" operating systems still provide basic key functionalities similar to traditional operating systems, i.e., managing various resources and providing services to applications and/or users. In this sense, CampusOS is an operating system serving the campus community, managing campus-related data and resources, while providing development and runtime support for campus applications.

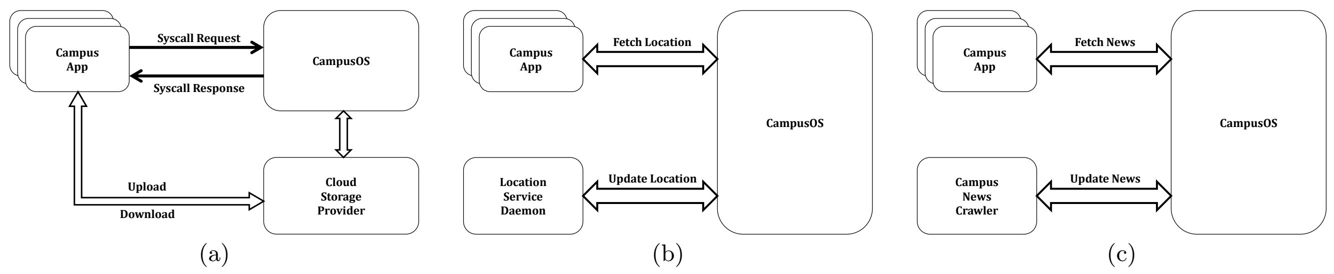

Figure 3: Patterns for CampusOS features

3.3 CampusOS vs. Middleware

Traditionally, middleware [1] is based on the operating system kernel, providing a glue layer between the operating system and applications. Although the boundary between operating system and middleware is to some extent arbitrary, one could view CampusOS as a middleware system when its dependence on traditional operating systems is taken into consideration.

However, from a higher-level view, CampusOS is more like an operating system, because its management of campus data and services can be compared to the management of various resources in traditional operating systems.

3.4 Centralized vs. Decentralized

The Internet is a huge decentralized network. Managing data on the Internet in a centralized way is not possible. However, data on university campuses is quite limited compared with that on the Internet, so centralized management is feasible and will save much time and effort for campus application developers and users. Furthermore, we can also manage decentralized Internet services, such as cloud storage and social network, in a centralized manner.

In CampusOS, centralized management of data and services conforms to the UNIX philosophy [8], like a monolithic kernel. The modularity is achieved via the VFS abstraction and the unified syscall interface. The hierarchical architecture also provides good separation and transparency between campus applications and the underlying implementations of the whole system.

4. CONCLUDING REMARKS

In this paper, we argue that an operating system is needed for applications targeting communities like university campuses. We present our design of CampusOS, which aims at providing an ecosystem that supports campus application development, deployment and maintenance. In our design, CampusOS is not restricted to being used in only one university. It can span multiple universities and provide a universal computing platform for general-purpose campus applications. Operating systems similar to CampusOS can be developed for organizations other than universities as well.

With the development and prosperity of Internetware, current operating systems and middleware systems should collaborate to provide a common infrastructure to support the development, deployment, execution and maintenance of various Internet-based applications. While an Internet OS that could manage all computers and computing environments connected to the Internet may be too ambitious, CampusOS can serve as one tiny case within such a big goal.

ACKNOWLEDGMENT

This work is supported by the National Basic Research Program of China (973) under Grant No. 2009CB320703, the High-Tech Research and Development Program of China under Grant No. 2011AA01A202, and the National Natural Science Foundation of China under Grants No. 61103026, 61121063, U1201252.

References

[1] Bernstein, P. A. Middleware: a model for distributed system services. *Commun. ACM 39*, 2 (Feb. 1996), 86–98.

[2] Dawson-Haggerty, S., Krioukov, A., Taneja, J., Karandikar, S., Fierro, G., Kitaev, N., and Culler, D. BOSS: building operating system services. In NSDI'13, pp. 443–458.

[3] Dixon, C., Mahajan, R., Agarwal, S., Brush, A. J., Lee, B., Saroiu, S., and Bahl, P. An operating system for the home. In NSDI'12, pp. 25–25.

[4] Living PlanIT SA. The Urban Operating System. `http://living-planit.com/UOS_overview.htm`.

[5] Mei, H., and Guo, Y. Network-oriented operating systems: status and challenges. *Scientia Sinica Informationis 43*, 3 (2013), 303–321. in Chinese.

[6] Mei, H., Huang, G., and Xie, T. Internetware: A software paradigm for internet computing. *Computer 45*, 6 (2012), 26–31.

[7] O'Reilly, T. The State of the Internet Operating System, 2010. `http://radar.oreilly.com/2010/03/state-of-internet-operating-system.html`.

[8] Raymond, E. S. *The Art of UNIX Programming.* Pearson Education, 2003.

[9] Tang, Y., Ames, P., Bhamidipati, S., Bijlani, A., Geambasu, R., and Sarda, N. CleanOS: limiting mobile data exposure with idle eviction. In OSDI'12, pp. 77–91.

[10] Zhang, Y., and Zhou, Y. TransOS: a transparent computing-based operating system for the cloud. *IJCC 1*, 4 (2012), 287–301.