3
Sistema Operacional Scriptável

Sistema operacional scriptável é a nossa proposta de modelo de projeto de sistema operacional com o objetivo de aumentar a sua flexibilidade e facilidade de desenvolvimento, através da junção de extensibilidade de sistema operacional e linguagens de script. A idéia central dessa abordagem é permitir o desenvolvimento e a extensão do sistema operacional de forma semelhante com que aplicativos de usuário são desenvolvidos e estendidos utilizando linguagens de script, tanto estendendo quanto embarcando o interpretador da linguagem. Desta forma, um sistema operacional scriptável é também um sistema operacional extensível que utiliza uma linguagem de script para escrever as extensões. A seguir descreveremos a caracterização de um sistema operacional scriptável baseados nos mesmos pontos de projeto e implementação descritos para sistemas operacionais extensíveis.

3.1
Projeto e Implementação

Nesta seção descreveremos as principais características de projeto e implementação de sistemas operacionais scriptáveis.

Mutabilidade

Sistema operacionais scriptáveis assumem dois modos de mutabilidade. Linguagens de script muitas vezes são utilizadas como simples linguagens para a configuração de um programa através do preenchimento de parâmetros. Isto faz com que um sistema operacional scriptável seja ao mesmo tempo um sistema reconfigurável e extensível, sob o ponto de vista da mutabilidade, pois ele possibilita ao mesmo tempo a adição de novo código ao sistema e a simples reconfiguração de parâmetros. Desta forma, a linguagem de script poderia substituir ou se integrar a mecanismos de reconfiguração de parâmetros como *sysfs* e *sysctl*.

Localização

Scripts podem ser usados tanto para estender o próprio *kernel* quanto para estender porções do sistema operacional executadas fora do *kernel*. Contudo, acomodar extensões no *kernel* agrega vantagens em flexibilidade e desempenho, como argumentado na seção 2.2.1, o que chamamos de *scripting* de *kernel* de sistema operacional e este é o foco da nossa implementação.

Proteção

A proteção dos scripts é feita por *software*, utilizando o isolamento provido pelo interpretador da linguagem (e.g., gerenciamento automático de memória). Parte da proteção pode ser feita também por *hardware*, como por exemplo preempção baseada em *threads* de *kernel* para garantir que o sistema não terá problemas de vivacidade (*liveness*). Entretanto, existem partes do *kernel* que não podem ser tratadas dentro de *threads* (e.g., tratadores de interrupção); neste caso, utilizamos a autorização para garantir que apenas scripts confiáveis possam comprometer a vivacidade do sistema. Outro recurso que pode ser utilizado para garantir a corretude é o oferecimento de interfaces sem estado (*stateless*) para garantir que scripts não violem os protocolos de uso das interfaces, seja acidentalmente ou propositalmente.

Duração

A duração dos scripts é condicionada à duração da instância do interpretador da linguagem onde o script é executado. O interpretador da linguagem de script, por sua vez, pode ser instanciado por aplicação, recurso ou *kernel*. Além disso, podemos ter o estado das instâncias salvo entre execuções do *kernel*. Assim, scripts podem ter a duração condicionada a aplicação, recurso, *kernel* ou permanente.

Granularidade

Sistemas operacionais scriptáveis oferecem granularidade procedural limitada. Esse tipo de sistema permite a modificação ou adição de código no sistema operacional através de ligações entre as extensões, implementadas em linguagem de script, e o sistema operacional, implementado em linguagem de sistema. Através dessas ligações é possível definir funções para serem chamadas pelo sistema operacional no lugar das implementações originais.

Arbitragem

Podemos misturar os três tipos de arbitragem (proibição, decisão separada e multiplexação) em um sistema operacional scriptável. A proibição e multiplexação podem ser implementadas através da API exportada para os scripts, utilizando mecanismos de sincronização nas interfaces exportadas. A segmentação pode ser implementada utilizando várias instâncias ou estados do interpretador da linguagem, diferenciando estados globais e locais.

Programabilidade

A programabilidade em um sistema scriptável é dada obviamente por uma linguagem de extensão, mais precisamente, uma linguagem de script. Além disso, a linguagem de script pode servir como uma ferramenta para a criação de linguagens de domínio específico para diferentes domínios de extensão de um sistema operacional. Desta forma, a programabilidade de um sistema scriptável pode ser considerada a combinação entre a programabilidade dada por uma linguagem de extensão e uma linguagem de domínio específico.

Autorização

A autorização de um sistema operacional scriptável pode ser feita combinando-se diversos níveis de privilégios, segmentando a autorização ao uso de scripts dentro do kernel a diversos domínios de segurança separados por estados diferentes do interpretador. Desta forma, temos um controle mais granular da autorização, permitindo que grupos de usuários específicos tenham acesso a estados específicos do interpretador.

3.2 Casos de Uso

Elaboramos alguns casos de uso que esperamos tratar com *scripting* de *kernel* de sistema operacional, tanto casos embutir quanto de estender o interpretador Lua. Nesta seção, descreveremos alguns deles.

Escalonador de Processos

A idéia desse caso de uso é possibilitar que usuários definam novos algoritmos para o controle global do escalonamento de processos ou que criem diferentes politicas de escalonamento para conjuntos independentes de processos ou *threads*, em tempo de execução. Isto poderia, por exemplo, auxiliar na implementação de qualidade de serviço *(QoS)* em processos. Esse caso utiliza a abordagem de embutir o interpretador.

Filtro de Pacotes

O propósito desse caso é possibilitar a criação de regras mais elaboradas para o processamento e filtragem do tráfego de pacotes de rede, em vez de usar simples tabelas de regras. Por exemplo, um usuário poderia criar sua própria implementação para inspeção de pacotes ou para tradução de endereços de rede *(NAT)*. Esse caso utiliza a abordagem de embutir o interpretador.

Gerenciamento de Energia

O propósito desse caso é possibilitar que usuários definam os seus próprios métodos para o gerenciamento do consumo de energia no sistema. Por exemplo, usuários poderiam definir algoritmos para escalonar a freqüência e voltagem da unidade central de processamento *(CPU)* para poupar energia. Nós exploramos esta idéia em mais detalhes na seção 4.3. Esse caso utiliza a abordagem de embutir o interpretador.

Drivers de Dispositivo

Existem algumas iniciativas de utilização de linguagens seguras e de domínio específico para escrever *drivers* de dispositivos, como NDL (29). A idéia desse caso é utilizar uma linguagem de scripting tanto para fazer a configuração dos drivers (normalmente utilizando tabelas de parâmetros) quanto para implementar as funções necessárias para o funcionamento adequado do dispositivo e criar drivers especializados por aplicação (e.g., comprimir dados durante o acesso ao disco). Esse caso utiliza tanto a abordagem de embutir e quanto a de estender o interpretador.

Banco de Dados

O propósito desse caso é utilizar scripts para, por exemplo, filtrar as buscas em arquivos de *backend* em sistemas de gerenciamento de banco de dados, evitando realizar sucessivas chamadas de sistema para processar o filtro em nível de usuário. Esse caso utiliza a abordagem de estender o interpretador, pois o sistema de gerenciamento de banco de dados faria as chamadas ao script (de forma semelhante a uma chamada de sistemas).

Servidor Web

O propósito desse caso é utilizar scripts para implementar a pilha ou parte da pilha de protocolo HTTP de um servidor *web* dentro do kernel, também poupando diversas trocas de contexto. Esse caso utiliza a abordagem de estender o interpretador, pois o aplicativo faria explicitamente as chamadas

ao script (também de forma semelhante a uma chamada de sistemas). Isso poderia ser feito, por exemplo, através do *scripting* de BSD Sockets.

3.3 Trabalhos Relacionados

Nesta seção descrevemos alguns trabalhos relacionados à nossa proposta de sistema operacional scriptável e *scripting* de *kernel*. Embora existam muitas abordagens para sistemas operacionais extensíveis e usos de recursos de linguagem de programação no desenvolvimento de sistemas operacionais, abordaremos nesta seção apenas trabalhos que utilizam formas semelhantes à da nossa proposta, ou seja, utilizam interpretadores de linguagens com características de linguagens de script dentro do *kernel*.

Exokernel (13) é uma arquitetura de sistema operacional extensível desenvolvido pelo MIT que é baseada na idéia que o sistema operacional não deve prover abstrações, ou seja, não deve desempenhar o papel de máquina virtual. Em vez de prover abstrações, o sistema operacional deve ser exclusivamente responsável por gerenciar recursos de forma segura, atuando como um multiplexador do *hardware*. Para atingir esse objetivo, as implementações de Exokernel desenvolvidas pelo MIT utilizam linguagens de domínio específico para possibilitar que os usuários manipulem o disco e a rede. Os códigos escritos nessas linguagens são carregados pelos usuários e interpretados dentro do kernel. A principal diferença desta abordagem para a nossa é a utilização de linguagens de domínio específico e limitadas pelo Exokernel. O *scripting* de *kernel* poderia ser utilizado nesse caso para a construção das linguagens de domínio especifico, facilitando e homogenizando o desenvolvimento e manutenção de código.

Outro sistema que utiliza interpretador embutido no kernel é μChoices (17), um sistema operacional extensível baseado em microkernel. A idéia principal do μChoices é permitir uso de scripts escritos em Tcl para a criação de chamadas de sistemas customizadas para atender a necessidades específicas de aplicações. As chamadas de sistema customizadas são definidas através de scripts de usuário carregados no *kernel* do sistema operacional. Esses scripts definem as chamadas utilizando chamadas de sistema já existentes no sistema operacional, evitando assim o excesso de trocas de contexto para se realizar um lote de chamadas. A principal diferença desta abordagem para a nossa é que μChoices se baseia exclusivamente em prover *scripting* de *kernel* por embutir o interpretador Tcl no *kernel*. Enquanto Lunatik, a nossa implementação de *scripting* de *kernel*, provê *scripting* tanto por embutir quanto por estender o interpretador Lua.

Além desses exemplos de trabalhos relacionados, não é incomum o uso de linguagens de domínio específico dentro de *kernels* de sistema operacional, como por exemplo para a construção de filtro de pacotes ou acesso ao controle avançado de energia (*ACPI — Advanced Configuration and Power Interface*). Sistemas operacionais scriptáveis possibilitam a substituição dessas diversas linguagens de domínio específico por uma única linguagem de scripting, provendo um ambiente mais uniforme de desenvolvimento e mais fácil de se manter, no lugar de várias implementações de linguagens *ad-hoc*.