Referências Bibliográficas

[1] LAMPSON, B. W. On reliable and extendible operating systems. In: *Proceedings of the Second NATO Conference on Techniques in Software Engineering*. [S.l.: s.n.], 1969. 1, 2.1.1, 2.2, 2.2.1

[2] ENGLER, D. R.; KAASHOEK, M. F. Exterminate all operating system abstractions. In: USENIX. *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*. [S.l.], 1995. p. 78–83. ISBN 0818670819. 1, 2.2.1

[3] BERSHAD, B. N. et al. SPIN — an extensible microkernel for application-specific operating system services. *ACM SIGOPS Operating Systems Review*, ACM, v. 29, n. 1, p. 74–77, 1995. ISSN 0163-5980. 1, 2.1.1, 2.2, 2.2.1

[4] SMALL, C.; SELTZER, M. *VINO: An Integrated Platform for Operating System and Database Research*. [S.l.], 1994. 1, 2.2, 2.2.1, 2.2.1

[5] HUNT, G. C.; LARUS, J. R. Singularity: Rethinking the software stack. *ACM SIGOPS Operating Systems Review*, ACM, v. 41, n. 2, p. 37–49, 2007. ISSN 0163-5980. 1, 2.1.1, 2.2

[6] SAVAGE, S.; BERSHAD, B. N. Issues in the design of an extensible operating system. In: *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*. [S.l.: s.n.], 1994. 1, 2.2, 2.2.1, 2.2.1, 2.2.1

[7] OUSTERHOUT, J. K. Scripting: Higher-level programming for the 21st century. *IEEE Computer*, v. 31, n. 3, p. 23–30, 1998. 1, 2.3

[8] IERUSALIMSCHY, R.; FIGUEIREDO, L. H. de; FILHO, W. C. Lua — an extensible extension language. *Software: Practice and Experience*, John Wiley & Sons, v. 26, n. 6, p. 635–652, 1996. 1, 2.3.1

[9] VIEIRA NETO, L. *Repensando o Conceito de Sistema Operacional*. dez. 2008. Projeto Final de Graduação em Engenharia de Computação, Departamento de Informática, PUC-Rio. 1

[10] HANSEN, P. B. *Operating System Principles*. [S.l.]: Prentice Hall, 1973. 2.1

[11] TANENBAUM, A. S.; WOODHULL, A. S. *Operating Systems Design and Implementation*. Third. [S.l.]: Prentice Hall, 2006. 2.1

[12] STALLINGS, W. *Operating Systems: Internals and Design Principles*. [S.l.]: Prentice Hall, 2008. 2.1

[13] ENGLER, D. R.; KAASHOEK, M. F.; O'TOOLE, J. W. Exokernel: An operating system architecture for application-level resource management. In: ACM. *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. [S.l.], 1995. p. 251–266. ISBN 0897917154. 2.1.1, 2.2, 2.2.1, 3.3

[14] DIJKSTRA, E. W. The structure of the "THE"-multiprogramming system. In: ACM. *Proceedings of the first ACM symposium on Operating System Principles*. [S.l.], 1967. p. 10–1. 2.2, 2.2.1

[15] ENGLER, D. R. *The Exokernel Operating System Architecture*. Tese (Doutorado) — MIT, 1998. 2.2

[16] SELTZER, M. I. et al. Issues in extensible operating systems. *Computer Science Technical Report TR-18-97*, Harvard University, 1997. 2.2, 2.2.1

[17] CAMPBELL, R. H.; TAN, S. M. μchoices: an object-oriented multimedia operating system. In: USENIX. *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*. [S.l.], 1995. p. 90–94. 2.2, 2.2.1, 3.3

[18] MAHESHWARI, U. Extensible operating systems. *Area Exam Report*, MIT, 1994. 2.2.1

[19] MONTZ, A. B. et al. Scout: A communications-oriented operating system. In: USENIX. *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*. [S.l.], 1995. p. 58–61. 2.2.1

[20] MOCHEL, P.; MURPHY, M. *Sysfs — The Filesystem for Exporting Kernel Objects*. 2009. `http://kernel.org/doc/Documentation/filesystems/sysfs.txt`. 2.2.1

[21] THE NETBSD FOUNDATION. *NetBSD Kernel Developer's Manual*. [S.l.]. 2.2.1

[22] ESPOSITO, D. *Windows 2000 Registry: Latest Features and APIs Provide the Power to Customize and Extend Your Apps*. 2000. `http://msdn.microsoft.com/en-us/magazine/bb985037.aspx`. 2.2.1

[23] BARHAM, P. et al. Xen and the art of virtualization. In: ACM. *Proceedings of the nineteenth ACM symposium on Operating systems principles*. [S.l.], 2003. p. 164–177. 2.2.1

[24] LEFKOWITZ, G. *Extending Vs. Embedding — There Is Only One Correct Decision*. 2003. `http://twistedmatrix.com/users/glyph/rant/extendit.html`. 2.3.1

[25] MUHAMMAD, H.; IERUSALIMSCHY, R. C APIs in extension and extensible languages. *Journal of Universal Computer Science*, v. 13, n. 6, p. 839–853, 2007. 2.3.1

[26] IERUSALIMSCHY, R.; FIGUEIREDO, L. H. de; CELES, W. The evolution of Lua. In: ACM. *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*. [S.l.], 2007. p. 2–26. 2.3.1

[27] IERUSALIMSCHY, R. *Programming in Lua*. [S.l.]: Lua.org, 2006. 2.3.1

[28] INTERNATIONAL STANDARDIZATION ORGANIZATION. *ISO/IEC 9899:1999(E); Programming languages — C*. Second. [S.l.], 1999. 2.3.1

[29] CONWAY, C. L.; EDWARDS, S. A. NDL: a domain-specific language for device drivers. In: ACM. *ACM SIGPLAN Notices*. [S.l.], 2004. v. 39, n. 7, p. 30–36. 3.2

[30] GRAF, A. *PacketScript — a Lua Scripting Engine for in-Kernel Packet Processing*. Dissertação (Mestrado) — Computer Science Department, University of Basel, jul. 2010. 4.3

[31] PALLIPADI, V.; STARIKOVSKIY, A. The ondemand governor. In: *Proceedings of the Linux Symposium*. [S.l.: s.n.], 2006. v. 2, p. 215–230. (document), 4.3.1, 4.3, 4.3.1, 4.3.1, 4.4

[32] BALLESTEROS, F. J. et al. Using interpreted compositecalls to improve operating system services. *Software Practice and Experience*, v. 30, n. 6, p. 589–615, 2000. 4.3.2, 4.3.2

[33] PATINO, M. et al. Compositecalls: A design pattern for efficient and flexible client-server interaction. In: *Proceedings of PloP*. [S.l.: s.n.], 1999. 4.3.2, 4.3.2

[34] BALLESTEROS, F. et al. Batching: A design pattern for efficient and flexible client/server interaction. *Transactions on Pattern Languages of Programming I*, Springer, p. 48–66, 2009. 4.3.2

[35] BALLESTEROS, F. et al. Object orientation in off++ a distributed adaptable μkernel. In: *Proceedings of the ECOOP Workshop on Object Orientation and Operating Systems*. [S.l.: s.n.], 1999. p. 49–53. 4.3.2

A
Lunatik Kernel Programming Interface (KPI)

Neste apêndice apresentaremos a interface de programação de *kernel* que desenvolvemos para Lunatik, a qual chamamos de Lunatik *KPI (Kernel Programming Interface)*. A Lunatik KPI fornece suporte para auxiliar desenvolvedores a adaptarem partes do *kernel* para serem scriptadas usando Lua.

Apresentaremos, nesse apêndice, apenas a KPI desenvolvida para NetBSD. A implementação para Linux, por ter sido a primeira a ser desenvolvida, é uma versão simplificada da KPI para NetBSD.

A KPI para NetBSD será apresentadas sob a forma de páginas de manual UNIX, contendo as seguintes subseções:

Nome: nome da interface de programação que será abordada na página de manual, seguido de uma breve descrição sobre ela.

Sinopse: lista dos protótipos das funções disponíveis na interface e dos arquivos de cabeçalho que são necessários para a utilização delas.

Descrição: descrição do funcionamento das funções presentes na interface.

A.1
Lunatik KPI para NetBSD

Nesta seção, apresentaremos o manual da Lunatik KPI para NetBSD.

NOME

Lunatik KPI — interface de programação de kernel para Lunatik.

SINOPSE

```
#include <lua/lua.h>
#include <sys/lunatik.h>

lunatik_State *
lunatik_newstate(lua_Alloc allocf, void *ud, void *mutex,
```

```
    lunatik_lock_t lockf, lunatik_unlock_t unlockf);

void
lunatik_close(lunatik_State *Lk);

inline void
lunatik_lock(lunatik_State *Lk);

inline void
lunatik_unlock(lunatik_State *Lk);

inline void *
lunatik_alloc(lunatik_State *Lk, size_t size);

inline void
lunatik_free(lunatik_State *Lk, void *ptr, size_t size);

inline lua_State *
lunatik_getluastate(lunatik_State *Lk);

int
lunatik_runstring(lunatik_State *Lk, const char *str,
    char **resultp, size_t *size_resultp);

int
lunatik_runcfunction(lunatik_State *Lk, lua_CFunction cfunction,
    void *lightuserdata, char **resultp, size_t *size_resultp);

int
lunatik_registerstate(lunatik_State *Lk, const char *name);

#include <sys/kmem.h>
#include <sys/mutex.h>

lunatik_State *lunatikL_newstate(km_flag_t km_flags,
    kmutex_t *kmutex, kmutex_type_t kmutex_type);
```

DESCRIÇÃO

Lunatik é uma infra-estrutura para scripting de kernel usando Lua. Este manual descreve a interface de programação de kernel dessa infra-estrutura, a Lunatik KPI. Esta interface é um dos componentes da infra-estrutura Lunatik, a qual fornece suporte para que partes do kernel sejam adaptadas para serem scriptadas.

lunatik_newstate() cria um novo estado Lunatik. Um estado Lunatik é um estado Lua encapsulado por um mecanismo de sincronização. Em caso de sucesso essa função retorna o novo estado criado. Caso não tenha memória suficiente disponível no sistema, ela retorna `NULL`. Essa função toma os seguintes argumentos:

allocf ponteiro para função de alocação de memória. Esta função é utilizada para toda alocação de memória feita pelo Lunatik para esse estado e, conseqüentemente, por toda alocação de memória feita por Lua para o estado Lua encapsulado no estado Lunatik.

allocud ponteiro opaco passado para a função de alocação de memória, referenciada por `allocf`, a cada chamada.

mutex ponteiro opaco passado para as funções referenciadas por `lockf` e `unlockf`. Esse ponteiro é tipicamente usado como descritor de exclusão mútua para o mecanismo de sincronização. Desta forma, as funções referenciadas por `lockf` e `unlockf` são chamadas sobre um `mutex` para, respectivamente, bloqueá-lo ou desbloqueá-lo.

lockf ponteiro para a função de bloqueio do mecanismo de sincronização. Esta função toma como argumento o ponteiro `mutex`. Ela deve bloquear caso tenha sido chamada anteriormente sem que a função referenciada pelo argumento `unlockf` tenha sido chamada em seguida sobre o mesmo parâmetro `mutex`. Caso contrário, ela deve simplesmente retornar. Essa função deve obedecer a seguinte definição de tipo: `typedef void (*lunatik_lock_t)(void *)`.

unlockf ponteiro para função de desbloqueio do mecanismo de sincronização. Esta função toma como argumento o ponteiro `mutex`. Ela deve liberar as chamadas bloqueadas à função referenciada pelo argumento `lockf`. Caso existam uma ou mais chamadas pendentes à função de bloqueio sobre um dado `mutex` (ou seja, bloqueadas), uma (e somente uma) das chamadas pendentes deve ser li-

berada. Essa função deve obedecer a seguinte definição de tipo: `typedef void (*lunatik_unlock_t)(void *)`.

lunatik_close() encerra um estado Lunatik. Toda a memória alocada para esse estado, incluindo a memória utilizada diretamente por Lua, é liberada utilizando a função apontada por `allocf` (passado como parâmetro na criação do estado). Essa função toma o seguinte argumento:

Lk descritor do estado Lunatik que se deseja encerrar.

lunatik_runstring() executa um script Lua contido em uma string. Essa função toma os seguintes argumentos:

Lk descritor do estado Lunatik onde deseja-se executar o script Lua.

str string terminada em `'\0'` contendo o script Lua a ser executado.

resultp ponteiro para string passado como referência para receber o resultado retornado pelo script Lua. O valor de retorno será convertido para uma string. Caso não se deseje receber o resultado da execução, deve-se passar o valor `NULL` através deste parâmetro. Em caso de erro de compilação ou de execução do script, a string retornada conterá a mensagem de erro. A string contendo o resultado é alocada dinamicamente utilizando o alocador do estado `Lk`. É responsabilidade do cliente da interface desalocar essa string. Para facilitar esse processo, pode-se usar a função `lunatik_free`.

size_resultp ponteiro passado como referência para receber o tamanho da string de resultado.

lunatik_runcfunction() executa uma função C em um estado Lua passando como parâmetro um `light userdata`. Essa função equivale as chamadas das funções `lua_pushcfunction`, `lua_pushlightuserdata` e `lua_pcall` em seqüência. Essa função toma os seguintes argumentos:

Lk descritor do estado Lunatik onde deseja-se executar a função C.

cfunction função C que será executado no estado Lua. Essa função deve obedecer o tipo `lua_CFunction`. e deve aguardar um único parâmetro do tipo `light userdata` na pilha virtual de Lua. Ela pode retornar no máximo um único valor. Neste caso, o valor será retirado da pilha virtual e retornado através do parâmetro `resultp`. Além disso, essa função pode utilizar irrestritamente a API C de Lua.

lightuserdata parâmetro que será passado para a função `cfunction` através da pilha virtual de Lua.

resultp ponteiro para string passado como referência para receber o resultado retornado pela função C. O valor de retorno será convertido para uma string. Caso não se deseje receber o resultado da execução, deve-se passar o valor `NULL` através deste parâmetro. Em caso de erro de execução da função C, a string retornada conterá a mensagem de erro. A string contendo o resultado é alocada dinamicamente utilizando o alocador do estado `Lk`. É responsabilidade do cliente da interface desalocar essa string. Para facilitar esse processo, pode-se usar a função `lunatik_free`.

size_resultp ponteiro passado como referência para receber o tamanho da string de resultado.

lunatik_lock() bloqueia o acesso a um estado Lunatik através da função `lockf` registrada no estado. Após o retorno desta função, a exclusão mútua ao estado Lua encapsulado no estado Lunatik `Lk` é garantida (caso a função `lockf` tenha sido implementada corretamente). Essa função toma o seguinte argumento:

Lk descritor do estado Lunatik que se deseja bloquear.

lunatik_unlock() desbloqueia o acesso a um estado Lunatik através da função `unlockf` registrada no estado. Essa função toma o seguinte argumento:

Lk descritor do estado Lunatik que se deseja bloquear.

lunatik_alloc() aloca memória utilizando a função alocadora passada na criação do estado Lunatik. Em caso de sucesso, retorna um ponteiro para início do espaço de memória alocado. Caso não tenha memória disponível no sistema, retorna `NULL`. Essa função toma os seguintes argumentos:

Lk descritor do estado Lunatik.

size tamanho do espaço de memória a ser alocado.

lunatik_free() libera memória utilizando a função alocadora passada na criação do estado Lunatik. Essa função toma os seguintes argumentos:

Lk descritor do estado Lunatik.

ptr ponteiro para o início do espaço de memória que se deseja liberar.

size tamanho do espaço de memória a ser liberado.

lunatik_getluastate() obtém o estado Lua encapsulado em um estado Lunatik. Essa função toma o seguinte argumento:

Lk descritor do estado Lunatik.

lunatik_newstate() cria um novo estado Lunatik. Um estado Lunatik é um estado Lua encapsulado por um mecanismo de sincronização. Em caso de sucesso essa função retorna o novo estado criado. Caso não tenha memória suficiente disponível no sistema, ela retorna `NULL`. Essa função toma os seguintes argumentos:

lunatikL_newstate() cria um novo estado Lunatik usando o alocador de memória `kmem(9)` e o mecanismo de exclusão mútua `mutex(9)`. Essa função toma os seguintes argumentos:

km_flags flags que serão passadas para as funções e de liberação de memória, respectivamente, `kmem_alloc` e `kmem_free`.

kmutex ponteiro para o descritor de mutex. Esse descritor deve ser previamente inicializado. É também função do cliente dessa função, destruir o mutex depois do encerramento do estado Lunatik retornado.

kmutex_type tipo do mutex apontado pelo parâmetro `kmutex`. Este argumento é utilizado para determinar as funções que serão utilizadas para bloquear e desbloquear o estado Lunatik, como por exemplo funções `spin` (espera ocupada).