8
Referências Bibliográficas

ADYA, A. et al. Cooperative task management without manual stack management. In: **USENIX Annual Technical Conference**. Berkeley: [s.n.], 2002. p. 289–302. 1, 2.1, 7.1

AGHA, G. **Actors: a Model of Concurrent Computation in Distributed Systems**. [S.l.]: MIT Press, 1986. 7.1

AGHA, G. et al. Abstraction and modularity mechanisms for concurrent computing. **IEEE Parallel and Distributed Technology: Systems and Applications**, v. 1, n. 2, p. 3–14, 1993. 3.2.2

ALua: asynchronous distributed programming in Lua. Abr 2004. `http://alua.inf.puc-rio.br`. Acesso em: Jun 2009. 1.1, 2.3

ANANDA, A.; TAY, B.; KOH, E. A survey of asynchronous remote procedure calls. **SIGOPS Operating Systems Review**, v. 26, n. 2, p. 92–109, 1992. 3.1

ANDREWS, G. **Foundations of Multithreaded, Parallel, and Distributed Programming**. [S.l.]: Addison Wesley, 2000. 1, 3.2, 5.1.3

ARBAB, F.; HERMAN, I.; SPILLING, P. An overview of Manifold and its implementation. **Concurrency: Practice and Experience**, v. 5, n. 1, p. 23–70, 1993. 7.1

ARMSTRONG, J. et al. **Concurrent Programming in Erlang**. [S.l.]: Prentice Hall, 1996. 2.1, 4.1

BACKUS, J. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. **Communications of the ACM**, ACM, v. 21, n. 8, p. 613–641, Aug 1978. 7.1

BARNES, G. A method for implementing lock-free shared-data structures. In: **5th ACM Symposium on Parallel Algorithms and Architectures**. New York, NY, USA: ACM, 1993. p. 261–270. ISBN 0-89791-599-2. 2.2

BEHREN, R. von et al. Capriccio: scalable threads for internet services. In: **19th ACM Symposium on Operating Systems Principles**. [S.l.]: ACM Press, 2003. p. 268–281. 1

Reliable distributed computing with the Isis toolkit. In: BIRMAN, K.; RENESSEE, R. van (Ed.). [S.l.]: IEEE Computer Society Press, 1994. p. 68–78. 3.1

BOLTON, F. **Pure CORBA**. [S.l.]: SAMS, 2002. 2

BRIOT, J.-P. Actalk: A framework for object-oriented concurrent programming - design and experience. In: BAHSOUN, J.-P. et al. (Ed.). **Object-Oriented Parallel and Distributed Programming**. [S.l.]: Hermès Science Publications, Paris, France, 1999. p. 209–231. 3.2.2

BRIOT, J.-P.; GUERRAOUI, R.; LÖHR, K.-P. Concurrency and distribution in object-oriented programming. **ACM Computing Surveys**, ACM, New York, NY, USA, v. 30, n. 3, p. 291–329, 1998. ISSN 0360-0300. 7.1

BUTENHOF, D. **Programming with POSIX Threads**. [S.l.]: Addison-Wesley, 1997. 5.1.2

CARRIERO, N.; GELERNTER, D. How to write parallel programs: a guide to the perplexed. **ACM Computing Surveys**, v. 21, n. 3, p. 323–357, set. 1989. 3

CHATZIMPARMPAS, E. **Concurrent Oriented Programing in Lua**. Dissertação (Mestrado) — Royal Institute of Technology, Stockholm, Sweden, 2007. 4.1

DABEK, F. et al. Event-driven programming for robust software. In: **ACM Special Interest Group on Operating Systems (SIGPOS)**. [S.l.]: ACM Press, 2002. p. 186–189. 1

DISCOLO, A. et al. **Lock Free Data Structures Using STM in Haskell**. [S.l.]: Springer, 2006. 65–80 p. (Lecture Notes in Computer Science, v. 3945). 2.2

EUGSTER, P.; GUERRAOUI, R.; DAMM, C. On objects and events. In: **Conference on Object-Oriented Programming, Systems, Languages, and Applications**. [S.l.: s.n.], 2001. p. 254–269. 7.1

FIGUEIREDO, L. **A POSIX library for Lua**. Out 2000. `http://www.tecgraf.puc-rio.br/~lhf/ftp/lua/`. Acesso em: Mai 2009. 4.1.1

FISCHER, J.; MAJUMDAR, R.; MILLSTEIN, T. Tasks: language support for event-driven programming. In: **ACM Special Interest Group on Programming Languages (SIGPLAN)**. New York, NY, USA: ACM Press, 2007. p. 134–143. 1, 2.1

FRØLUND, S. **Coordinating Distributed Objects: An Actor-Based Approach to Synchronization**. [S.l.]: The MIT Press, 1996. 3.2.2, 3.2.2

GELERNTER, D.; CARRIERO, N. Coordination languages and their significance. **Communications of the ACM**, v. 35, n. 2, p. 97–107, 1992. 3, 3.2

GOETZ, B. et al. **Java Concurrency in Practice**. [S.l.]: Addison Wesley, 2006. 2, 2.2

GUERRA, J. **Helper Threads: Building blocks for non-blocking libraries**. Mar 2006. `http://helper-threads.luaforge.net/`. Acesso em: Mai 2009. 4.1.1

HALLER, P.; ODERSKY, M. Actors that unify threads and events. In: **Coordination Models and Languages**. [S.l.]: Springer, 2007. (Lecture Notes in Computer Science, v. 4467), p. 171–190. 7.1

HALLER, P.; ODERSKY, M. Scala actors: Unifying thread-based and event-based programming. **Theoretical Computer Science**, v. 410, n. 2–3, p. 202–220, 2008. 7.1

HOARE, C. Monitors: an operating system structuring concept. **Communications of the ACM**, ACM Press, New York, NY, USA, v. 17, n. 10, p. 549–557, 1974. ISSN 0001-0782. 3.2.1

IERUSALIMSCHY, R. **Programming in Lua**. second. [S.l.]: Lua.org, 2006. 2.3

INTEL. **Threading Building Blocks**. 2005. `http://www.threadingbuildingblocks.org/`. Acesso em: Mai 2009. 5.1.2

JOHANSSON, E.; SAGONAS, K.; WILHELMSSON, J. Heap architectures for concurrent languages using message passing. In: **3rd International Symposium on Memory Management**. New York, NY, USA: ACM, 2002. p. 88–99. ISBN 1-58113-539-4. 2.1, 5, 5.3

KAUPPI, A. **Lua Lanes: Multithreading in Lua**. Abr 2007. `http://luaforge.net/projects/lanes/`. Acesso em: Jun 2009. 4.1.1

LEAL, M.; RODRIGUEZ, N.; IERUSALIMSCHY, R. LuaTS - A Reactive Event-Driven Tuple Space. **Journal of Universal Computer Science**, v. 9, n. 8, p. 730–744, ago. 2003. 1.1, 3.1

LEE, E. The problem with threads. **IEEE Computer**, v. 39, n. 5, p. 33–42, 2006. 1, 2, 2.1, 7

LIMA, M. **O Modelo de Contextos Aninhados para Documentos Multimídia: Definição e Implementação**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 1992. 3.2.1

MAGEE, J.; DULAY, N.; KRAMER, J. A constructive development environment for parallel and distributed programs. **IEE/IOP/BCS Distributed Systems Engineering**, v. 1, n. 5, 1994. 7.1

MAIA, R.; CERQUEIRA, R.; COSME, R. OiL: An object request broker in the Lua language. In: **Tools Session of the Brazilian Simposium on Computer Networks (SBRC)**. Curitiba, PR: [s.n.], 2006. 4.1

MAIA, R.; CERQUEIRA, R.; KON, F. A middleware for experimentation on dynamic adaptation. In: **4th Workshop on Adaptive and Reflective Middleware**. Grenoble, France: [s.n.], 2005. 4.1

MICHAEL, M.; SCOTT, M. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In: **15th ACM Symposium on Principles of Distributed Domputing**. New York, NY, USA: ACM, 1996. p. 267–275. ISBN 0-89791-800-2. 2.2

MILLER, M.; TRIBBLE, E.; SHAPIRO, J. Concurrency among strangers – Programming in E as plan coordination. In: **Symposium on Trustworthy Global Computing (European Joint Conference on Theory and Practice of Software)**. [S.l.: s.n.], 2005. LNCS 3705. 4.2

MOURA, A.; IERUSALIMSCHY, R. Revisiting coroutines. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM, New York, NY, USA, v. 31, n. 2, p. 1–31, 2009. ISSN 0164-0925. 2.3

Moura, A.; RODRIGUEZ, N.; IERUSALIMSCHY, R. Coroutines in Lua. **Journal of Universal Computer Science**, v. 10, n. 7, p. 910–925, jul. 2004. 2.3

NEHAB, D. **Multithreading support for the Lua language**. 2004. `http://www.cs.princeton.edu/~diego/professional/luathread/`. Acesso em: Jun 2009. 4.1.1

ODERSKY, M.; SPOON, L.; VENNERS, B. **Programming in Scala**. 1st. ed. [S.l.]: Artima Press, 2008. 7.1

OUSTERHOUT, J. **Why threads are a bad idea (for most purposes)**. 1996. Invited talk at the 1996 USENIX Technical Conference. 1, 2, 2.1, 2.3, 4.2, 7, 7.1

PAPADOPOULOS, G.; ARBAB, F. Coordination models and languages. In: **Advances in Computers**. [S.l.]: Academic Press, 1998. v. 46, p. 329–400. 3, 7.1

REINDEERS, J. **Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism**. [S.l.]: O'Reilly, 2007. 2.2

RIVEILL, M. Synchronising shared objects. **Distributed Systems Engineering Journal**, v. 2, n. 2, p. 112–125, Jun 1995. 3.2.2

RODRIGUEZ, N.; ROSSETTO, S. Integrating remote invocations with asynchronism and cooperative multitasking. **Parallel Processing Letters**, v. 18, n. 1, p. 71–85, 2008. 1.1, 3.1, 3.1.1

ROSSETTO, S.; RODRIGUEZ, N.; IERUSALIMSCHY, R. Abstrações para o desenvolvimento de aplicações distribuídas em ambientes com mobilidade. In: **VIII Simpósio Brasileiro de Linguagens de Programação**. [S.l.: s.n.], 2004. p. 143–156. 1.1, 3.1

SEWELL, P. et al. Acute: High-level programming language design for distributed computation. **Journal of Functional Programming**, v. 17, n. 4–5, p. 547–612, Jul 2007. 7.1

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. **Operating System Concepts**. 8th. ed. [S.l.]: Wiley, 2008. 1

SKYRME, A. **Um modelo alternativo para programação concorrente em Lua**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 2007. 4.1.1, 4.2

SUNSHINE-HILL, B. **Pluto Library**. Jul 2004. `http://lua-users.org/wiki/PlutoLibrary`. Acesso em: Mai 2009. 4.2

SUSSMAN, G.; STEELE, G. Scheme: A Interpreter for Extended Lambda Calculus. **Higher-Order and Symbolic Computation**, v. 11, n. 4, p. 405–439, 1998. 2.1

TANENBAUM, A.; RENESSE, R. van. A critique of the remote procedure call paradigm. In: **European Teleinformatics Conference**. Vienna: [s.n.], 1988. p. 775–783. 3.1

URURAHY, C.; RODRIGUEZ, N. ALua: An event-driven communication mechanism for parallel and distributed programming. In: **Parallel and Distributed Computing and Systems**. Fort Lauderdale, Florida: [s.n.], 1999. 1.1, 2.3

VARELA, C.; AGHA, G. Programming dynamically reconfigurable open systems with SALSA. **ACM SIGPLAN Notices**, v. 36, n. 12, p. 20–34, 2001. 7.1

von Behren, R.; CONDIT, J.; BREWER, E. Why events are a bad idea (for high-concurrency servers). In: USENIX. **9th Workshop on Hot Topics in Operating Systems (HotOS IX)**. Lihue, Hawaii, 2003. 1, 2.1, 7.1

WELSH, M.; CULLER, D.; BREWER, E. SEDA: an architecture for well-conditioned, scalable internet services. In: **18th Symposium on Operating Systems Principles (SOSP)**. Banff, Canada: ACM, 2001. p. 230–243. 7.1

WIKSTROM, C. Distributed programming in Erlang. In: **1st International Symposium on Parallel Symbolic Computation**. Linz, Austria: [s.n.], 1994. 2.1

XAVANTE Web Server. Fev 2004. `http://www.keplerproject.org/xavante/`. Acesso em: Jun 2009. 4.1, 6

A
API do ALua 6.0

alua.create(config [, callback])

Esta função é responsável por iniciar um novo daemon, o qual irá esperar por conexões dos processos. O parâmetro `config` é uma tabela com os campos `addr` e `port`, que contêm, respectivamente, o endereço e a porta TCP para estabelecimento de conexão. `callback` é uma função que será chamada quando o daemon for criado.

alua.daemonid

Variável que contém o identificador do daemon em que o processo está conectado.

alua.exit([code])
alua.exit(procs [, code])

Está função possui duas construções. Na primeira, ela termina o próprio processo, tendo o parâmetro `code` (opcional) como código de retorno do programa. Na segunda forma, a função requisita o término de outros processos. O parâmetro `procs` é um array de identificadores de processos para terminar. `code` com o qual os processos devem terminar.

alua.id

Variável que contém o identificador do próprio processo.

alua.link(daemons [, callback])

A função `link` é resposável por criar uma rede de daemons para que os processos em daemons diferentes possam se comunicar. `daemons` é um array de identificadores de daemons a serem conectados. Como `link` age de forma assíncrona, a função `callback` é chamada para informar o resultado da operação.

`alua.loop()`

Executa o loop de eventos do ALua. `loop` verifica se há algum evento vindo do daemon ou de outro processo, e o executa. Esta função não retorna, por isso deve ser chamada após toda a inicialização do processo.

`alua.open(daemon [, callback])`

Função utilizada pelos processos estabelecer a conexão com o daemon. `daemon` é o identificador do daemon a qual se deseja conectar. O identificador possui informações suficientes para que o processo estabeleça uma conexão TCP. `callback` é uma função (opcional) que é chamada para informar o resultado da operação, dado que `open` atua de forma assíncrona.

`alua.send(destination, message [, callback])`

Esta função envia a mensagem `message` a um processo de forma assíncrona. `destination` contém o identificador do processo para o qual a mensagem deve ser enviada. `callback` é uma função (opcional) que recebe o resultado da operação.

`alua.spawn(param [, callback])`

Dispara novos processos ALua. Caso `param` seja um número, os processos serão criados de forma balanceada pelos daemons que estão ao alcance do processo disparador, ou seja, a quantidade de processos é dividida entre os daemons. `param` também pode ser um array contendo nomes arbitrários que serão os identificadores dos novos processos — também serão balanceados entre os daemons. Como última forma de criação, `param` pode ser uma tabela contendo os identificadores dos daemons como chave e o valor pode ser um número ou um array de identificadores. Os valores são enviados aos respectivos daemons que devem criar o número requisitado de processos. A função `callback` pode ser utilizada para recuperar o retorno da criação dos novos processos.

`alua.tostring(value)`

Função utilitário simples responsável por codificar valores de Lua em string. No entanto, esta função é simples, não tendo suporte à codificação de funções, co-rotinas e *userdata*.

B
Medição de Memória dos Processos Lua

A tabela B.1 apresenta o resultado detalhado da medição de memória dos processos Lua (em kilobytes) que foi apresentada no capítulo 5. Esses valores foram obtidos por meio do programa `pmap` que mostra o mapeamento da memória virtual de um programa, dado o seu PID, bem como o consumo total.

As bibliotecas *rawsend*, *ccr* e *alua* foram desenvolvidas por nós para, respectivamente, controle de envio no canal TCP devido à concorrência dos processos Lua, controle da fila de mensagens e disparo das threads, e controle do disparo de processos do sistema operacional. *core* faz parte do módulo LuaSocket e *uuid* é uma biblioteca usada por *ccr* para gerar identificadores únicos internos. *libtbb* e *libtbbmalloc* fazem parte da biblioteca Threading Building Blocks. *launcher* é o nosso programa C customizado e responsável por iniciar o script de criação de processos Lua. As demais bibliotecas são incluídas pelo sistema ou pelo compilador GCC/G++ como dependências. As regiões *anonymous* são áreas de memória alocadas dinâmicamente pelo programa ou biblitecas, e *stack* é o espaço atual alocado para pilha.

Mapeamento	1 Proc	100 Procs	1.000 Procs	10.000 Procs
[anonymous]	4	4	4	4
rawsend.so	12	12	12	12
rawsend.so	4	4	4	4
libtbb.so.2	92	92	92	92
libtbb.so.2	4	4	4	4
libtbbmalloc.so.2	12	12	12	12
libtbbmalloc.so.2	4	4	4	4
core.so	36	36	36	36
core.so	4	4	4	4
ccr.so	32	32	32	32
ccr.so	4	4	4	4
uuid.so	40	40	40	40
uuid.so	4	4	4	4
alua.so	4	4	4	4
alua.so	4	4	4	4

ld-2.7.so	108	108	108	108
ld-2.7.so	4	4	4	4
ld-2.7.so	4	4	4	4
libc-2.7.so	1356	1356	1356	1356
libc-2.7.so	8	8	8	8
libc-2.7.so	4	4	4	4
[anonymous]	12	12	12	12
libm-2.7.so	156	156	156	156
libm-2.7.so	4	4	4	4
libm-2.7.so	4	4	4	4
libdl-2.7.so	12	12	12	12
libdl-2.7.so	4	4	4	4
libdl-2.7.so	4	4	4	4
libpthread-2.7.so	84	84	84	84
libpthread-2.7.so	4	4	4	4
libpthread-2.7.so	4	4	4	4
[anonymous]	8	8	8	8
librt-2.7.so	28	28	28	28
librt-2.7.so	4	4	4	4
librt-2.7.so	4	4	4	4
libgcc_s-4.1.2-20070925.so.1	44	44	44	44
libgcc_s-4.1.2-20070925.so.1	4	4	4	4
libstdc++.so.6.0.8	896	896	896	896
libstdc++.so.6.0.8	16	16	16	16
libstdc++.so.6.0.8	4	4	4	4
[anonymous]	24	24	24	24
launcher	128	128	128	128
launcher	4	4	4	4
[anonymous]	264	2376	22304	220468
[anonymous]	1028	1028	1028	7564
[anonymous]	8	8	8	8
[stack]	84	84	84	84
Total	**4580**	**6692**	**26620**	**231320**

Tabela B.1: Medição detalhada de memória dos processos Lua, em kilobytes.