Padronização POSIX *x* Sistemas Operacionais de Tempo-Real: Uma Análise Comparativa

Edgard de Faria Corrêa
edgard@inf.ufsc.br

Luis Fernando Friedrich, Dr. Eng.
lff@inf.ufsc.br

Universidade Federal de Santa Catarina (UFSC)
Centro Tecnológico - Departamento de Informática e Estatística (CTC-INE)
Curso de Pós-Graduação em Ciências da Computação (CPGCC)
Caixa Postal 476 - 88.040-970 Florianópolis - Santa Catarina - Brasil
Fone: +55(48)231-9738 Fax: +55(48)231-9566

Resumo

Diversas áreas de pesquisa buscam atualmente conquistar a padronização em suas respectivas tecnologias. Em Sistemas Operacionais o desejo de padronização deve-se, principalmente, à questão da portabilidade, possibilitando a independência dos programas em relações aos sistemas operacionais e às máquinas onde são executados. No âmbito do tempo-real isso se torna mais crucial pelo fato da necessidade de se ter previsibilidade, ou seja, a garantia de que determinado evento ou tarefa aconteça dentro de um intervalo de tempo especificado ou até um certo tempo limite.

Muitas vezes Sistemas Operacionais para Tempo-Real apresentam características específicas de uma determinada aplicação e, portanto, são ainda mais difíceis de serem padronizados, levando a uma maior complexidade e diversidade de soluções para cada aspecto a ser considerado. Em geral, tem-se como objetivo determinante, a padronização do mecanismo utilizado para garantir as características de tempo-real. Uma questão a ser levantada é a possibilidade de se garantir a existência de um padrão de direito e de fato, e se isso garantirá a portabilidade e a previsibilidade.

O presente artigo analisa os diversos aspectos propostos no padrão POSIX.4 (1003.1b) e POSIX.4a (1003.1c), tais como: comunicação entre processos; compartilhamento de memória; sincronização; escalonamento e threads, traçando também um comparativo com sistemas operacionais de tempo-real conhecidos, tanto aqueles considerados comerciais quanto os acadêmicos.

1. Introdução

A busca de uma padronização representa uma tentativa de se estabelecer uma maior interação entre os diversos componentes de um determinado sistema. Na atualidade, isso vem ocorrendo em diversas áreas do conhecimento.

Especificamente, neste artigo, trataremos da área de sistemas operacionais, especialmente aqueles para ambientes distribuídos e de tempo-real. Como fatores motivadores poderíamos citar o desejo de portabilidade das aplicações, permitindo independência de hardware e de sistema operacional. Em ambientes distribuídos de tempo-real acrescenta-se a necessidade de previsibilidade.

A atuação dos sistemas operacionais de tempo-real abrange as áreas funcionais de gerenciamento de processo; sincronização / comunicação entre processos; gerenciamento de memória; mecanismos de E/S e threads.

Inicialmente comentamos as diferenças e características de Sistemas Convencionais e de Tempo-Real, e das classificações do último em Soft e Hard. Na seção 2 descrevemos as funções da padronização POSIX para tempo-real. Em seguida, na seção 3, apresentamos uma visão de sistemas operacionais conhecidos, divididos em três classes: comerciais com núcleos pequenos, rápidos e proprietários; sistemas convencionais comerciais com extensões para tempo-real; e os sistemas operacionais de pesquisa ou acadêmicos. Na seção 4 fazemos um comparativo entre os sistemas apresentados na seção 3 e o padrão POSIX discutido na seção 2.

1.1 Sistemas Convencionais x Sistemas de Tempo-Real: Características e Diferenças

Os Sistemas Operacionais de Tempo-Real apresentam como característica fundamental a interação com o ambiente, onde este determina o período, o tempo de início ou o prazo máximo ("deadline") para a realização de uma tarefa. Essas tarefas devem ser executadas de acordo com a previsão estimada, considerando o pior caso, que corresponde ao pico de carga aliado ao número máximo de falhas suportáveis pelo sistema.

Os Sistemas Convencionais por outro lado tentam garantir o acesso aos recursos de forma justa, o que pode não garantir a previsibilidade necessária para aplicações de tempo-real, pois esses sistemas trabalham com previsão para respostas válidas em um prazo aceitável, segundo condições "normais".

Assim, temos que considerar alguns pontos quando tratamos de sistemas operacionais de tempo-real, pois teremos necessariamente múltiplas tarefas que devem ser executadas dentro do tempo estipulado e seguindo a ordem de prioridade.

- Em sistemas operacionais convencionais as tarefas geralmente são atendidas pela ordem de chegada, o que pode deixar uma tarefa com maior prioridade bloqueada por uma de prioridade mais baixa.
- Deve-se ter a garantia de atendimento aos eventos esporádicos (alarmes, por exemplo) bem como a eventos que acontecem de forma periódica (controladores).
- O sistema operacional deve atender a previsibilidade que o ambiente impõe, como por exemplo a qualidade de serviço que algumas aplicações multimídia exigem.

1.2 Sistemas de Tempo-Real Soft e Hard (Soft Real Time e Hard Real Time)

Dentre os Sistemas de Tempo-Real tem-se ainda a distinção na classificação, de acordo com o nível de comprometimento do sistema, quando não há o cumprimento das prerrogativas temporais.

Os sistemas onde as conseqüências de uma falha, devido ao tempo, comprometem o sistema, equipamentos ou até mesmo vidas humanas, são conhecidos como "Hard Real Time Systems" ou Sistemas de Tempo-Real *Hard*. Exemplos destes sistemas são equipamentos médicos/hospitalares, fabris (caldeiras, controladores, etc.), freios de automóveis.

Aqueles sistemas onde não é necessária uma garantia de tempo de resposta tão rígida, pois o atraso não será tão comprometedor, são conhecidos como "Soft Real Time Systems" ou Sistemas de Tempo-Real *Soft*. Exemplos deste tipo são os sistemas de processamento bancário e aplicações multimídia.

2. A Padronização POSIX

POSIX ("Portable Operating System Interface") é o conjunto de documentos produzidos pela IEEE e adotado pelo ANSI e ISO, que abrangem desde a interface básica de sistemas operacionais até questões de administração ou segurança.

O padrão POSIX.1 trata das operações comuns de funções de sistema que tornam portáveis as aplicações na mudança de um sistema operacional para outro, necessitando apenas a recompilação das mesmas.

As extensões para tempo-real estão presentes no padrão POSIX.4 (redenominado como POSIX.1b) [Gallmeister 95], tratando de assuntos como: filas de mensagens; semáforos; memória compartilhada; sinais; escalonamento prioritário; relógios de tempo-real (com granularidade da ordem de nanosegundos); notificações de eventos assíncronos; E/S síncrona e assíncrona. Threads são abordadas no padrão POSIX.4a (POSIX.1c) [Nichols 96].

O POSIX.4 tem como única parte obrigatória algumas adições à base do POSIX.1 na questão de mecanismos de sinais. Alguns sinais adicionais de tempo-real, bem como escalonamento prioritário, passagem de mensagens, semáforos, mecanismos de entrada e saída, gerenciamento de memória são definidos como padrão de especificação opcional. O detalhamento das partes obrigatórias e das partes opcionais é mostrado no anexo "A".

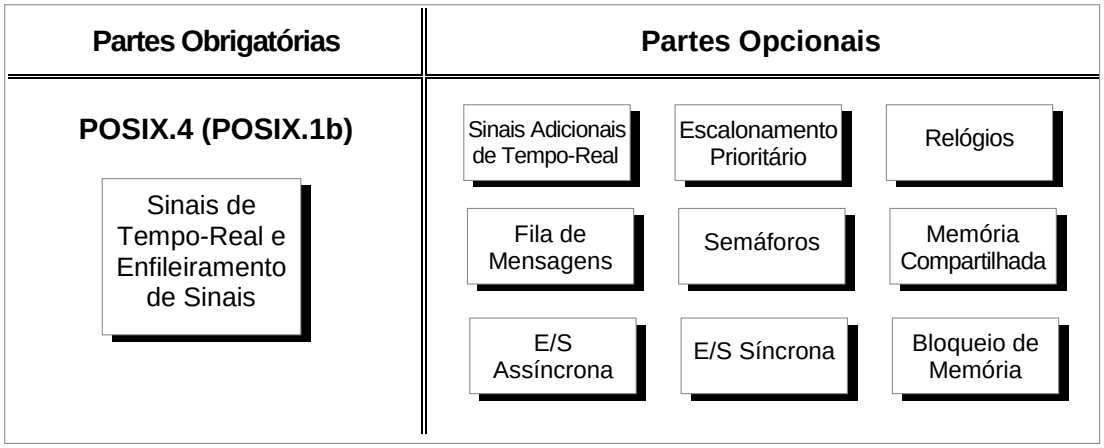

Figura 1: Partes obrigatórias e opcionais no POSIX.4

Threads é o modelo padronizado para dividir programas em sub-tarefas, cuja execução pode ser intercalada ou paralela. PThreads corresponde ao padrão POSIX da IEEE para threads. O anexo "B" apresenta uma breve descrição das threads do POSIX.4a (POSIX.1c).

3. Sistemas de Tempo-Real

De acordo com [Ramamritham 94] os sistemas operacionais para tempo-real são classificados em três categorias gerais: sistemas comerciais com núcleos pequenos, rápidos e proprietários; sistemas comerciais que foram desenvolvidos através de sistemas convencionais acrescidos de extensões para tempo-real; e aqueles que tiveram seus núcleos desenvolvidos através de pesquisas acadêmicas, os sistemas operacionais acadêmicos.

3.1 Sistemas Comerciais com Núcleos Pequenos, Rápidos e Proprietários

Os sistemas comerciais com núcleos proprietários, pequenos e rápidos geralmente são desenvolvidos para serem utilizados embutidos em pequenos sistemas, onde a execução rápida e previsível deve ser garantida.

Para garantir a rapidez e previsibilidade, o núcleo apresenta como requisitos: pequeno tamanho (o que implica em funcionalidade mínima); resposta rápida a interrupções externas; intervalos mínimos de tempo onde as interrupções ficam desabilitadas; e prover gerenciamento de memória para partições fixas e variáveis.

De forma a negociar as requisições temporais o núcleo deve: manter "clock" tempo-real; prover um mecanismo de escalonamento prioritário; prover alarmes especiais e "timeouts"; suportar disciplinas de enfileiramento de tempo-real, tais como, realizar primeiro as tarefas com deadline mais próxima; e prover primitivas para retardar processamento por um tempo fixo e determinado.

Como exemplo desse tipo de sistema podemos citar o QNX [http 97], que segue muitas das especificações do padrão POSIX.4 e POSIX.4a, mas devido as especificidades das aplicações foge em muitos pontos a essa padronização.

3.2 Sistemas Convencionais Comerciais com Extensões de Tempo-Real

As funcionalidades de tempo-real acrescidas ao sistemas convencionais para que estes respondam aos requisitos temporais enfocam basicamente "timers", escalonamento prioritário, memória compartilhada, semáforos, comunicação entre processos, notificações de eventos assíncronos, bloqueio/proteção de memória, E/S síncrona e assíncrona e threads.

As vantagens em se desenvolver novas funcionalidades a partir de núcleos já existentes são basicamente a facilidade de portabilidade e o fato da interface, geralmente, já ser conhecida. Os principais problemas são: overhead; latência excessiva na resposta a interrupções; dúvidas em relação a capacidade de preempção do núcleo; e o fato das filas internas serem FIFO.

Exemplo desse tipo de sistemas é o Chorus System [Gien 95] [Abrossimov 96] para Tempo-Real, que apresenta configurabilidade de micronúcleo flexível para hardware e software específico. A funcionalidade do software pode ser customizada para a necessidade exata da aplicação, selecionando as funções de áreas, tais como: gerenciamento de memória; comunicação entre processos; e políticas de escalonamento. Essa capacidade de customização permite adaptar a configuração do sistema operacional para uma variedade de plataformas de hardware.

A modularidade e escalabilidade da arquitetura do micronúcleo permite suportar o padrão POSIX ou características de sistema de tempo-real "hard" embutidos que possuam fina granularidade e determinismo. Apresenta conformidade para garantir a implementação do padrão POSIX, mas acrescenta primitivas adicionais nas áreas não padronizadas, como Interfaces de Sistemas de Programação (SPIs) que suportam componentes de sistemas operacionais "plug and play". O sistema é modularizado, permitindo que chamadas de sistemas sejam enviadas a interfaces de programação das aplicações (APIs), dentre as quais as de tempo-real, com padrão POSIX e também para APIs multithreads. Além das abstrações de processos e threads o CHORUS trabalha com "actors", que são módulos de unidades de aplicações.

3.3 Sistemas Acadêmicos

Sistemas operacionais de tempo-real desenvolvidos através de pesquisa permitem o desenvolvimento de modelos de processo de tempo-real que podem responder aos diversos requisitos necessários, independendo do tamanho do núcleo e das características dos sistemas convencionais.

Um exemplo que representa esta categoria de sistemas é o *Spring* [Teo 95] [Humphrey 95] [Stankovic 91], desenvolvido na Universidade de Massachusetts (EUA).

4. Comparação entre o Padrão e o Real

A grande distinção entre o padrão proposto e os sistemas de tempo-real do "mundo real" deve-se em grande parte a especificidades dos problemas a serem resolvidos, ocasionando na escolha de soluções específicas às aplicações.

Verifica-se esse fato em todos os tipos de sistema abordados na seção 3. Os sistemas comerciais com núcleo prioritário necessitam muitas vezes evitar determinados itens da padronização para garantir o tamanho e rapidez necessários a suas aplicações. No caso dos sistemas com extensões para tempo-real, o problema se apresenta principalmente devido as características já existentes nos sistemas convencionais que os originaram. Como exemplo, podemos citar: escalonamento para condições "normais" utilizando o caso médio e não o pior caso; seleção de recursos por demanda; escalonamento de CPU independente e alocação de recursos podendo causar bloqueio ilimitado. Mesmo os sistemas acadêmicos, podem muitas vezes não seguir as padronização, para que sejam atendidas as garantias de, por exemplo, "hard" deadlines, ou evitar que seja introduzido algum tipo de indeterminismo ou imprevisibilidade.

Uma visão da falta de padronização nos sistemas do "mundo real" pode ser percebida quando analisamos as primitivas dentro das áreas funcionais de tempo-real.

4.1 Gerenciamento de Processos

Na área de gerenciamento de processos, temos as primitivas de sinais de tempo-real, de escalonamento prioritário e os relógios ("clocks" e "timers").

- *Sinais*: O QNX define operações de sinalização de acordo com o POSIX, adicionando porém, oito sinais proprietários que não podem ser ignorados ou capturados, enquanto o Spring utiliza mecanismo de sinalização distinto da padronização POSIX.

- *Escalonamento*: O CHORUS realiza o escalonamento a nível de threads. O Spring não utiliza o conceito de prioridade, baseando o escalonamento na requisição do recurso. Já o QNX segue o modelo POSIX com algoritmos de escalonamento FIFO, round-robin e adaptativo.

- *"Clocks" e "Timers"*: O Spring e o QNX seguem o padrão POSIX, inclusive com granularidade da ordem de nanosegundos. O primeiro, porém, utiliza um mecanismo diferente para sinalizar expiração de tempo ("timeout").

4.2 Sincronização / Comunicação entre Processos

Para sincronização e comunicação entre processos, em tempo-real, são utilizadas primitivas de passagem de mensagens, semáforos e memória compartilhada.

- *Passagem de Mensagens*: a diferença fundamental entre o QNX e o POSIX em termos de filas de mensagens é que no QNX as mensagens são atreladas aos processos e bloqueantes, enquanto no POSIX as filas de mensagens existem independentes dos processos que as utilizam.

• *Semáforos*: enquanto o QNX segue o padrão POSIX, o CHORUS utiliza semáforos também em threads e realiza sincronização também com mutexes e eventos, além de semáforos. O POSIX utiliza semáforos contadores para sincronização que não existem no Spring.

• *Memória Compartilhada*: O CHORUS apresenta arquitetura modular, onde determinados módulos seguem o padrão para tempo-real. O QNX também segue as diretrizes do padrão. O Spring, entretanto, diferencia-se por não permitir a criação e o gerenciamento de memória compartilhada de forma dinâmica. A criação e alocação da memória é feita em tempo de compilação.

4.3 Gerenciamento de Memória

O gerenciamento de memória, em geral, é feito através do bloqueio de memória, podendo ser bloqueado uma extensão de endereço de memória ou o processo inteiro.

• *Bloqueio de Memória*: No Spring todos os processo tem suas páginas pré-alocadas em tempo de criação, e mantidas residentes na memória. O QNX permite a alocação em tempo de criação e em tempo de execução.

4.4 Mecanismos de Entrada/Saída

Os mecanismos de entrada e saída previstos no padrão POSIX para tempo-real são síncronos, assíncronos, priorizados ou não. Em quase todos os sistemas estudados existe a implementação de sinais de "timeout" para indicar que não houve sucesso na requisição de um recurso após a expiração do tempo estipulado.

4.5 Threads

O POSIX considera threads em dois níveis, um a nível de sistema (as threads são escalonadas em relação a todas as outras) e outro a nível de processo (as threads são escalonadas em relação as threads do mesmo processo).

O mecanismos de Threads é utilizado por todos os três sistemas abordados, existindo porém diferenças em sua utilização.

O QNX não suporta muitas threads do POSIX, mas suporta as que permitem criar/destruir/associar/separar/cancelar threads, bem como as de sincronização (mutex, variáveis condicionais, escalonamento e sinais) estão presentes no núcleo.

O CHORUS define mutexes a nível de "actors" e não de threads, apresentando também threads para semáforos.

O Spring define políticas de escalonamento para threads, ao contrário do POSIX, que implementa apenas interfaces de escalonamento.

5. Conclusão

As necessidades apresentadas pelas aplicações de tempo-real, geralmente específicas, determinam características de projeto que ocasionam em soluções proprietárias, fugindo das propostas de padronização existentes.

Procuramos apresentar dentre as diversas razões para existência de sistemas que não seguem a padronização, aquelas que julgamos mais relevantes e que apresentavam documentação mais significativa. O que nos leva a crer que a padronização nos moldes em que se encontra atualmente dificilmente conseguirá algum avanço. Isso se explica pelo fato da padronização

abranger uma área bastante ampla, especificando, no entanto, apenas uma pequena parte como obrigatória.

Como solução poderia-se pensar em um estudo mais detalhado e aprofundado nas características e soluções dos sistemas existentes, de modo a resultar em uma padronização mais efetiva e eficiente.

6. Referências

[Gallmeister 95] Gallmeister, B. O., "POSIX.4: Programming for the Real World". O'Reilly & Associates, 1995.

[Nichols 96] Nichols, B., Buttlar D. e Farrel J., "Pthreads Programming". O'Reilly & Associates, 1996.

[Ramamritham 94] Ramamritham K. e Stankovic J. A., "Scheduling Algoritms and Operating Systems Support for Real-Time Systems". Proceedings of the IEEE, vol. 82, n. 1, p. 61-65, Jan 94

[http 97] "QNX/Neutrino Microkernel". Disponível pela Internet via www: <URL: http://www.qnx.com/docs/neutrino/sysarch/>, Jun 97.

[Gien 95] Gien, M., "Evolution of the CHORUS Open Microkernel Architecture: the STREAM Project". V IEEE Workshop on Future Trends in Distributed Computing Systems, Cheju Island, Ago 95.

[Abrossimov 96] Abrossimov, V. et. al, "STREAM-v2 Kernel Architecture and API Specificacion Version 1.0". Esprit Project Documentation, STREAM Technical Committee, Mar 96.

[Teo 95] Teo, M. S. E., "A Preliminary Look at Spring and POSIX.4". Spring Project Documentation, University of Massachusetts, Amherst, MA, 1995.

[Humphrey 95] Humphrey, M., Wallace, G. e Stankovic, J., "Kernel-Level Threads for Dynamic, Hard Real-Time Environments". Spring Project Documentation, University of Massachusetts, Amherst, MA, 1995.

[Stankovic 91] Stankovic J. A. e Ramamritham K., "The Spring Kernel: A New Paradigm for Real Time Systems". IEEE Software, vol. 8, n. 3, p. 62-72, Mai 91

Apêndice A

Primitivas do Padrão POSIX.4 (POSIX.1b)

Nome Opcional	Funcionalidade
Gerenciamento de Processos:	sinais; escalonamento; "clocks" e "timers".
Não Opcional	Enfileiramento de sinais; sinais tempo-real. (SA_SIGINFO, SIGRTMIN, SIGRTMAX)
_POSIX_REALTIME_SIGNALS	Funções adicionais de sinais • *sigwaitinfo* • *sigtimedwait* • *sigqueue*
_POSIX_PRIORITY_SCHEDULING	Controle de escalonamento de processo • *sched_setparam* • *sched_getparam* • *sched_setscheduler* • *sched_getscheduler* • *sched_yield*, • *sched_rr_get_interval* • *sched_get_priority_max* • *sched_get_priority_min*
_POSIX_TIMERS	"Clocks" e "Timers" • *clock_settime* • *clock_gettime* • *clock_getres* • *timer_create* • *timer_delete* • *timer_settime* • *timer_gettime* • *timer_getoverrun* • *nanosleep*
Sincronização/Comunicação InterProcessos:	passagem de mensagem; semáforo e memória compartilhada
_POSIX_MESSAGE_PASSING	Filas de Mensagens • *mq_open* • *mq_close* • *mq_unlink* • *mq_send* • *mq_receive* • *mq_notify* • *mq_setattr* • *mq_getattr*
_POSIX_SEMAPHORES	Semáforos contadores • *sem_init* • *sem_destroy* • *sem_unlink* • *sem_open* • *sem_close* • *sem_wait* • *sem_trywait* • *sem_post* • *sem_getvalue*
_POSIX_SHARED_MEMORY_OBJECTS	Memória compartilhada • *mmap* • *munmap* • *shm_unlink* • *shm_open* • *shm_close* • *ftruncate*
Entrada e Saída:	E/S priorizada, síncrona e assíncrona.
_POSIX_ASYNCHRONOUS_IO	Entrada/Saída assíncrona • *aio_read* • *aio_write* • *lio_listio* • *aio_suspend* • *aio_cancel* • *aio_error* • *aio_return* • *aio_fsync* (*) (*): se e somente se _POSIX_SYNCHRONIZED_IO
_POSIX_PRIORITIZED_IO	E/S assíncrona priorizada: modifica a ordem da fila de E/S.
_POSIX_SYNCHRONOUS_IO	Garantia de que os dados de um arquivo serão sempre os dados atualizados no disco e não de memória temporária. • *fdatasync* • *msync* (**) • *aio_fsync* (***) • adicionais para *open* e *fcntl* (**) se e somente se _POSIX_MAPPED_FILES (***) se e somente se _POSIX_ASYNCHRONOUS_IO
_POSIX_FSYNC	Garante que um arquivo no disco não estará desatualizado.
_POSIX_MAPPED_FILES	Arquivos mapeados na memória • *mmap* • *munmap* • *ftruncate* • *msync* (#) (#) se e somente se _POSIX_SYNCHRONOUS_IO
Gerenciamento de Memória:	bloqueio e proteção de memória.
_POSIX_MEMLOCK	Bloqueia toda a memória para evitar paginamento/swapping • *mlockall* • *munlockall*
_POSIX_MEMORY_RANGE	Bloqueia uma faixa de memória • *mlock* • *munlockall*
_POSIX_MEMORY_PROTETION	Habilidade de estabelecer proteção de memória • *mprotect*

Tabela 1: Funções do padrão POSIX.4 (POSIX.1b)

Apêndice B

Primitivas do Padrão POSIX.4a (POSIX.1c)

SINCRONIZAÇÃO

• pthread_join	suspende a thread até que outra termine
• pthread_once	garante que a inicialização de rotinas seja feita uma única vez

MUTEXES

• pthread_mutexattr_destroy	destrói um atributo de objeto mutex
• pthread_mutexattr_getprioceiling	obtém a prioridade máxima do atributo
• pthread_mutexattr_getprotocol	obtém o protocolo do atributo
• pthread_mutexattr_getpshared	obtém situação do processo compartilhado do atributo de objeto
• pthread_mutexattr_init	inicializa um atributo
• pthread_mutexattr_setprioceiling	estabelece prioridade máxima do atributo
• pthread_mutexattr_setprotocol	estabelece protocolo do atributo: herança, protegido, nenhum.
• pthread_mutexattr_setpshared	estabelece situação do processo: compartilhado ou privado.
• pthread_mutex_destroy	destrói um mutex
• pthread_mutex_init	inicializa um mutex com os atributos especificados no objeto
• pthread_mutex_lock	bloqueia um mutex, suspendendo a thread caso ele já esteja bloqueado
• pthread_mutex_trylock	tenta bloquear um mutex, sem suspender a thread
• pthread_mutex_unlock	desbloqueia mutex, libera thread suspensa (se houver), conforme prioridade

VARIÁVEIS CONDICIONAIS

• pthread_condattr_destroy	destrói um atributo de objeto variável condicional
• pthread_condattr_getpshared	obtém situação do processo compartilhado do atributo de objeto
• pthread_condattr_init	inicializa um atributo
• pthread_condattr_setpshared	estabelece situação do processo: compartilhado ou privado.
• pthread_cond_broadcast	desbloqueia todas as threads que esperam um variável condicional
• pthread_cond_destroy	destrói uma variável condicional
• pthread_cond_init	inicializa variável condicional com atributos especificados no objeto
• pthread_cond_signal	desbloqueia thread bloqueada, conforme prioridade de escalonamento
• pthread_cond_wait	desbloqueia mutex, suspende thread até sinalização à variável condicional
• pthread_cond_timedwait	desbloqueia mutex, suspende thread até sinalização à variável condicional ou expiração do tempo especificado

BLOQUEIO LEITURA/ESCRITA

• pthread_rdwr_init_np	inicializa bloqueio leitura/escrita
• pthread_rdwr_rlock_np	bloqueia leitura
• pthread_rdwr_wlock_np	bloqueia escrita
• pthread_rdwr_runlock_np	desbloqueia leitura
• pthread_rdwr_wunlock_np	desbloqueia escrita

Tabela 2: Funções do padrão POSIX.4a (POSIX.1c)

GERENCIAMENTO DE THREADS

• pthread_atfork	declara procedimentos chamados antes e depois da chamada "fork"
• pthread_attr_destroy	destrói um atributo de objeto thread
• pthread_attr_init	inicializa um atributo de objeto thread
• pthread_cancel	cancela thread específica
• pthread_create	cria uma thread com os atributos especificados no objeto
• pthread_detach	marca uma estrutura de dados interna da thread para deleção
• pthread_equal	compara uma thread com outra
• pthread_exit	termina a chamada de uma thread, retornando valor especificado
• pthread_getspecific	obtém os dados e a chave associada com a thread específica
• pthread_setspecific	estabelece os dados e a chave associada com a thread específica
• pthread_key_create	gera chave única de thread específica, visível a todas as demais no processo
• pthread_key_delete	deleta chave de uma thread específica
• pthread_setcancelstate	estabelece o estado de cancelamento: habilitado/desabilitado
• pthread_setcanceltype	estabelece se o estado de cancelamento pode ser mudado durante a execução da thread
• pthread_testcancel	requer que qualquer pedido de cancelamento pendente seja enviado a thread que solicitou o pedido
• pthread_self	obtém a thread que interage com a thread chamada
• pthread_attr_getdetachstate	obtém a situação em relação ao estado de isolamento de uma thread
• pthread_attr_setdetachstate	ajusta o estado de isolamento de uma thread: isolada ou associada
• pthread_attr_getstackaddr	obtém o endereço da pilha da thread
• pthread_attr_getstacksize	obtém o tamanho da pilha da thread
• pthread_attr_setstackaddr	estabelece o endereço de pilha da thread
• pthread_attr_setstacksize	estabelece o tamanho da pilha da thread
• pthread_cleanup_pop	remove uma rotina do topo da pilha de remoção da thread
• pthread_cleanup_push	coloca uma rotina no topo da pilha de remoção da thread

ESCALONAMENTO

• pthread_attr_getinheritsched	obtém a situação de herança de escalonamento de uma thread
• pthread_attr_getschedparam	obtém parâmetros associados com a política de escalonamento
• pthread_attr_getshedpolicy	obtém a situação da política de escalonamento
• pthread_attr_getscope	obtém a situação do escopo do escalonamento
• pthread_attr_setinheritsched	obtém a situação
• pthread_attr_setschedparam	estabelece parâmetros associados com a política de escalonamento
• pthread_attr_setshedpolicy	estabelece a situação da política de escalonamento
• pthread_getschedparam	obtém as políticas e parâmetros de escalonamento
• pthread_setschedparam	estabelece as políticas e parâmetros de escalonamento
• pthread_attr_setscope	estabelece a situação do escopo do escalonamento

SINAIS

• pthread_kill	envia sinal para terminar uma thread específica
• pthread_sigmask	examina ou modifica a máscara do sinal

Tabela 2: Funções do padrão POSIX.4a (POSIX.1c) - continuação