ERAD SP 2013

IV Escola Regional de
Alto Desempenho de São Paulo
São Carlos / SP

Minicurso

Programando para múltiplos processadores: Pthreads, OpenMP e MPI

Liria Matsumoto Sato (EP/USP) e
Hélio Crestana Guardia (DC/UFSCar)

POSIX Threads

POSIX Threads, ou *Pthreads*, é um padrão POSIX para a programação com *threads*.

POSIX 1003.1é uma família de padrões com foco no sistema Unix.

Extensões ao padrão 1003.1 para suporte a tempo real:

IEEE Std 1003.1b-1993 Realtime Extension
IEEE Std 1003.1c-1995 Threads
IEEE Std 1003.1d-1999 Additional Realtime Extensions
IEEE Std 1003.1j-2000 Advanced Realtime Extensions
IEEE Std 1003.1q-2000 Tracing

Posix Threads: pthreads API

API Pthreads apresenta 3 grupos de funções:

- Funções para **gerenciar** *threads*
  - Tratam da criação e manipulação de *threads* (criar e destruir) e do ajuste de seus atributos (*joinable*, *scheduling*, etc.)
- Funções para **sincronizar** a execução de *threads* e bloquear o acesso a recursos
  - ***Mutexes***: manipulam um mecanismo de controle de exclusão mútua, mutex, e seus atributos.
  - ***Condition variables***: incluem funções para criar, destruir, esperar e tratar sinais baseada em valores de variáveis.
  - Outras: (*sem)*, *barrier*, *rw_locks*, *spins*, *keys*, ...
- Funções para gerenciar o **escalonamento** de *threads*

Posix Threads: pthreads API

pthread_attr_destroy(), pthread_attr_getdetachstate(), pthread_attr_getguardsize(), pthread_attr_getinheritsched(), pthread_attr_getschedparam(), pthread_attr_getschedpolicy(), pthread_attr_getscope(), pthread_attr_getstack(), pthread_attr_getstackaddr(), pthread_attr_getstacksize(), pthread_attr_init(), pthread_attr_setdetachstate(), pthread_attr_setguardsize(), pthread_attr_setinheritsched(), pthread_attr_setschedparam(), pthread_attr_setschedpolicy(), pthread_attr_setstack(), pthread_attr_setstackaddr(), pthread_attr_setstacksize()

pthread_barrier_init(), pthread_barrier_wait(), pthread_barrier_destroy()
pthread_barrierattr_destroy(), pthread_barrierattr_getpshared(), pthread_barrierattr_init(), pthread_barrierattr_setpshared()

pthread_cleanup_pop(), pthread_cleanup_push()

pthread_cond_init(), pthread_cond_signal(), pthread_cond_broadcast(), pthread_cond_wait(), pthread_cond_timedwait(), pthread_cond_destroy()

pthread_condattr_init(), pthread_condattr_getclock(), pthread_condattr_setclock(), pthread_condattr_getpshared(), pthread_condattr_setpshared(), pthread_condattr_destroy()

pthread_create(), ***pthread_exit***(), ***pthread_join***(), ***pthread_kill***(), pthread_equal(), pthread_once(), pthread_sigmask(), ***pthread_self***(), pthread_detach()

pthread_getconcurrency(), pthread_getcpuclockid(), pthread_getschedparam(), pthread_getspecific()

pthread_key_create(), pthread_key_delete()

pthread_atfork()

pthread_mutex_init(), pthread_mutex_unlock(), pthread_mutex_lock(), pthread_mutex_timedlock(), pthread_mutex_trylock(), pthread_mutex_setprioceiling(), pthread_mutex_getprioceiling(), pthread_mutex_destroy()

pthread_mutexattr_init(), pthread_mutexattr_getprioceiling(), pthread_mutexattr_getprotocol(), pthread_mutexattr_getpshared(), pthread_mutexattr_gettype(), pthread_mutexattr_setprioceiling(), pthread_mutexattr_setprotocol(), pthread_mutexattr_setpshared(), pthread_mutexattr_settype(), pthread_mutexattr_destroy()

pthread_rwlock_init(), pthread_rwlock_rdlock(), pthread_rwlock_timedrdlock(), pthread_rwlock_tryrdlock(), pthread_rwlock_wrlock(), pthread_rwlock_timedwrlock(), pthread_rwlock_trywrlock(), pthread_rwlock_unlock(), pthread_rwlock_destroy()

pthread_rwlockattr_init(), pthread_rwlockattr_destroy(), pthread_rwlockattr_getpshared(), pthread_rwlockattr_setpshared()

pthread_cancel(), pthread_testcancel(), pthread_setcancelstate(), pthread_setcanceltype()

pthread_setconcurrency(), pthread_setschedparam(), pthread_setschedprio(), pthread_setspecific()

pthread_spin_init(), pthread_spin_lock(), pthread_spin_trylock(), pthread_spin_unlock(), pthread_spin_destroy()

Posix Threads: pthreads API

Compilando com gcc:

// prog.c
#include <pthread.h>
...

 # man gcc
/pthread

-pthreads
Add support for multithreading using the POSIX threads library. This option sets flags for both the preprocessor and linker. This option does not affect the thread safety of object code produced by the compiler or that of libraries supplied with it.

-pthread
This is a synonym for -pthreads.

#gcc prog.c –o prog –pthread
#gcc prog.c –o prog –pthreads
#gcc prog.c –o prog -lpthread

Criação de *Threads*

int ***pthread_create*** *(pthread_t * thread,*
*const pthread_attr_t * attr,*
*void * (*start_routine)(void *),*
*void *arg);*

Ex: *result =* ***pthread_create****(&th, NULL, função, NULL);*

- Quando um programa é iniciado com ***exec***, uma única *thread* é criada (***initial thread*** ou ***main thread***).
- *Threads* adicionais são criadas com ***pthread_create***
- ***ptrhead_create( ):*** cria uma nova *thread*, especificando a **função** que deve ser executada
- Semelhante à combinação de *fork* e *exec*, com espaço de endereçamento compartilhado
- Retorna um *thread id* em *thread*
- Se ***attr*** é NULL, usa parâmetros *default*

Threads: atributos

Ao invés de usar valores *default*, estrutura ***pthread_attr_t*** pode ser ajustada e passada como parâmetro para a criação de *threads*.

Atributos: *detachstate, schedpolicy, schedparam, inheritsched* e *scope*

Detachstate:

- PTHREAD_CREATE_JOINABLE (*default*), ou PTHREAD_CREATE_DETACHED
- O parâmetro ***joinable*** permite que outras *threads* esperem pela conclusão de uma *thread*, identificando a condição de saída da *thread* concluída.
- Quando ***detached***, recursos de uma *thread* são liberados imediatamente em sua conclusão e *pthread_join* não pode ser usado para sincronização (espera pelo fim)
- *pthread_detach( )* permite ajustar estado para ***detached*** em tempo de execução.

Schedpolicy:

- Seleciona a política de escalonamento para a *thread*
- SCHED_OTHER: escalonamento **normal**, não tempo-real, é a política *default*
- SCHED_RR: *realtime*, *round-robin*
- SCHED_FIFO: *realtime*, *first-in first-out*
- Políticas SCHED_RR e SCHED_FIFO são permitidas apenas para processos com privilégios de superusuário
- Política de escalonamento pode ser alterada em tempo de execução com o comando *pthread_setschedparam( )*

Threads: atributos

Schedparam

- Contém parâmetros de escalonamento: **prioridade** (***default=0***).
- É relevante apenas para as políticas **SCHED_RR** e **SCHED_FIFO**.
- Pode ser alterada em tempo de execução com o comando *pthread_setschedparam( )*

Inheritsched

- Indica se a política e a prioridade de escalonamento são definidas pelos parâmetros, ou herdadas da *main thread*
- **PTHREAD_EXPLICIT_SCHED** e **PTHREAD_INHERIT_SCHED** *(default)*

Scope

- Define o âmbito de contenção (concorrência) da *thread* pelo uso de CPU
- **PTHREAD_SCOPE_SYSTEM**: *thread* compete pela CPU com todos os outros processos sendo executados no sistema (valor *defautl*)
- **PTHREAD_SCOPE_PROCESS**: *thread* compete apenas com outras *threads* do mesmo processo

Threads: atributos

- *int* ***pthread_attr_init****(pthread_attr_t *attr);*
- *int* ***pthread_attr_destroy****(pthread_attr_t *attr);*

- *int* ***pthread_attr_setdetachstate****(pthread_attr_t *attr, int detachstate);*
- *int* ***pthread_attr_getdetachstate****(const pthread_attr_t *attr, int *detachstate);*
- *int* ***pthread_attr_setschedpolicy****(pthread_attr_t *attr, int policy);*
- *int* ***pthread_attr_getschedpolicy****(const pthread_attr_t *attr, int *policy);*
- *int* ***pthread_attr_setschedparam****(pthread_attr_t *attr, const struct sched_param *param);*
- *int* ***pthread_attr_getschedparam****(const pthread_attr_t *attr, struct sched_param *param);*
- *int* ***pthread_attr_setinheritsched****(pthread_attr_t *attr, int inherit);*
- *int* ***pthread_attr_getinheritsched****(const pthread_attr_t *attr, int *inherit);*
- *int* ***pthread_attr_setscope****(pthread_attr_t *attr, int scope);*
- *int* ***pthread_attr_getscope****(const pthread_attr_t *attr, int *scope);*

- O ajuste de atributos de *threads* é realizado preenchendo um objeto ***pthread_attr_t*** passado como argumento na função de criação da *threadt pthread_create( ).*
- *pthread_attr_init( )* inicia um objeto ***pthread_attr_t*** e o preenche com valores *default*.

- Consulta de atributos da thread corrente pode ser feita com a função pthread_getattr_np():
- *int* ***pthread_getattr_np****(pthread_t thread, pthread_attr_t *attr);*
- *pthread_getattr_np()* inicia estrutura de parâmetros referenciada com atributos da *thread* corrente.

Término de *Threads*

void pthread_exit(void * return_value);

- ***pthread_exit( )***: termina uma *thread*, que também pode ser encerrada com o retorno da função executada.
- Semelhante ao retorno da função *main( )*
- Uso de *return( )* na função da *thread* (*start_routine*) é equivalente a *pthread_exit( )*
- Se a *thread* corrente é a última, processo termina
- Uso de *return( )* na função *main( )* (*initial thread*) é equivalente a chamar a função *exit()*
- **Atenção**: função *exit( )* termina o processo e todas as suas *threads!*

Detach e Join

- Comando ***join*** é executado para esperar o fim de uma *thread*, de maneira semelhante a ***wait( )/waitpid( )*** para processos.
- Identificador da *thread* é especificado; **não** é possível esperar por **qualquer** *thread*
- *Threads* podem ser ***detached*** ou ***joinable*** **(*default* POSIX)**

int ***pthread_join*** *(pthread_t thread, void ** status);*

int ***pthread_tryjoin_np*** *(pthread_t thread, void **retval);*

int ***pthread_timedjoin_np*** *(pthread_t thread, void **retval, const struct timespec *abstime);*

*The pthread_tryjoin_np() function performs a nonblocking join with the thread thread, returning the exit status of the thread in *retval. If thread has not yet terminated, then instead of blocking, as is done by pthread_join(3), the call returns an error.*

The pthread_timedjoin_np() function performs a join-with-timeout. If thread has not yet terminated, then the call blocks until a maximum time, specified in abstime. If the timeout expires before thread terminates, the call returns an error. The abstime argument is a structure of the following form, specifying an absolute time measured since the Epoch (see time(2)):

```
struct timespec {
    time_t tv_sec;     /* seconds */
    long   tv_nsec;    /* nanoseconds */
};
```

Detach e Join

int ***pthread_join****(pthread_t thread, void ** status);*
int ***pthread_tryjoin_np****(pthread_t thread, void **retval);*
int ***pthread_timedjoin_np****(pthread_t thread, void **retval, const struct timespec *abstime);*

- Atributo na criação determina se *thread* é ***joinable*** ou ***detached***:
  - Declarar variável ***pthread_attr_t***
  - Iniciar variável com o comando ***pthread_attr_init( )***
  - Ajustar atributo ***detached status*** com ***pthread_attr_setdetachstate( )***
  - Ao final, liberar a variável com ***pthread_attr_destroy( )***

int pthread_detach(pthread_t thread);

- Função ***pthread_detach( )*** pode ser usada para mudar o *status* de uma *thread* para *detached*, mesmo se criada *joinable*
- Quando uma *joinable thread* termina, seu **identificador** e ***status* de saída** são mantidos até que outra *thread* execute *pthread_join*.
- ***pthread_detach( )*** faz com que recursos de uma *thread* sejam imediatamente liberados em sua conclusão

Passagem de argumentos

int ***pthread_create*** *(pthread_t * thread,*
*const pthread_attr_t * attr,*
*void * (*start_routine)(void *),*
*void *arg);*

- Tipo **void* usado no argumento da função da *thread* pode ser qualquer coisa.
- É preciso apenas que o argumento seja usado considerando o tipo que foi passado.
- Não é qualquer tipo de parâmetro que pode ser tratado como um ponteiro, contudo.
- No caso de estrutura de dados, é preciso passar o seu endereço.

Retorno de dados

void ***pthread_exit****(void *retval);*

- ***pthread_exit( )*** termina a execução de uma *thread*
- Valor de retorno (retval) pode ser consulado por outra *thread* usando *pthread_join( )*
- Tipo ponteiro permite passagem de diferentes formatos de dados

Threads: Sincronização

- *Threads* possuem diversos mecanismos de sincronização:
  - ***Joins***: fazem com que uma *thread* espere até que outra complete (termine)
  - ***Semaphores***: sincroniza *threads* em função de valor de contador
  - ***Mutexes***: bloqueios para preservar **seções críticas** ou obter **acesso exclusivo** a recursos
  - ***Condition variables***: usado junto com *mutex*, permite bloqueio em função de uma condição definida no programa.
  - ***Barriers***: realiza o bloqueio de um grupo de *threads* até que todas executem a operação da barreira.
  - ***Reader Writer locks***: fornece bloqueios escritor/escritor e escritor/leitor.
  - ***Thead Specific***: *pthread_key_t ...*
  - ***Spin***: *pthread_spinlock_t ...*

Ex: produtor / consumidor

- Comunicação feita através de buffer circular em memória compartilhada (variável global *_buffer*)
- Sincronização via semáforos
- Exclusão mútua com *mutex*

```
#define LEN 128
#define MAX_ITEMS 4
#define N_CONS    4

// semáforos
sem_t s_prod, s_cons;

// mutex
pthread_mutex_t _mutex_lock;

int _fim=0;

// buffer compartilhado
int _buffer[MAX_ITEMS];
int _in, _out;
int _item_available;
```

```
int main (int argc, char *argv[])
{
      pthread_t prod_t, cons_t[N_CONS]; int result, t; char err_msg[LEN];

      // inicia semáforos
      if (sem_init(&s_cons, 0, 0) == -1) {
        strerror_r(errno,err_msg,LEN); printf("Erro em sem_init: %s\n",err_msg);
        exit(1);
      }
      if (sem_init(&s_prod, 0, MAX_ITEMS) == -1) {
        strerror_r(errno,err_msg,LEN); printf("Erro em sem_init: %s\n",err_msg);
        exit(1);
      }
      // inicia mutex
      pthread_mutex_init(&_mutex_lock, NULL);

      // cria produtor
      result = pthread_create(&prod_t, NULL, producer, NULL);
      if (result) {
        strerror_r(result,err_msg,LEN); printf("Erro criando produtor: %s\n",err_msg);
        exit(0);
      }
      // cria consumidores
      for(t=0;t<N_CONS;t++) {
        result = pthread_create(&(cons_t[t]), NULL, consumer, (void *)t);
        if (result) {
              strerror_r(result,err_msg,LEN); printf("Erro criando consumidor %d: %s\n",t,err_msg);
              exit(0);
        }
      }
...
```

```
...
    // criar condição para fim de programa (_fim=1)?

    // espera threads terminarem (ou não?)
    result = pthread_join(prod_t, NULL);
    if (result) {
      strerror_r(result,err_msg,LEN);
      fprintf(stderr,"Erro em pthread_join: %s\n",err_msg);
    }
    for(t=0;t<N_CONS;t++) {
      result = pthread_join(cons_t[t], NULL);
      if (result) {
            strerror_r(result,err_msg,LEN);
            fprintf(stderr,"Erro em pthread_join: %s\n",err_msg);
      }
    }

    pthread_mutex_destroy(&_mutex_lock);

    sem_destroy(&s_prod);
    sem_destroy(&s_cons);

    pthread_exit(NULL);
    // aqui, como outras threads já terminaram, poderia apenas usar return…
}
```

```
void *
producer (void *producer_arg)
{
      int item;

      srandom(getpid());

      while (!_fim) {

        item=produce_item();

        // espera bloqueante no semáforo que indica espaço no buffer
        sem_wait(&s_prod);

        // bloqueia mutex
        pthread_mutex_lock(&_mutex_lock);

              insert_into_queue(item);
              _item_available++;

        // desbloqueia mutex
        pthread_mutex_unlock(&_mutex_lock);

        // sinaliza liberação da condição para consumidor
        sem_post(&s_cons);
      }
      pthread_exit(NULL);
}
```

```
void *
consumer (void *consumer_arg)
{
      int item;
      int id=(int)consumer_arg;

      while (!_fim) {

        // espera bloqueante no semáforo que indica itens produzidos
        sem_wait(&s_cons);

        // bloqueia mutex
        pthread_mutex_lock(&_mutex_lock);

              item = extract_from_queue();
              _item_available--;

        // desbloqueia mutex
        pthread_mutex_unlock(&_mutex_lock);

        // sinaliza liberação da condição para produtor
        sem_post(&s_prod);

        process_item(id,item);
      }
      pthread_exit(NULL);
}
```