POSIX Threads

Laboratório de Sistemas Operacionais

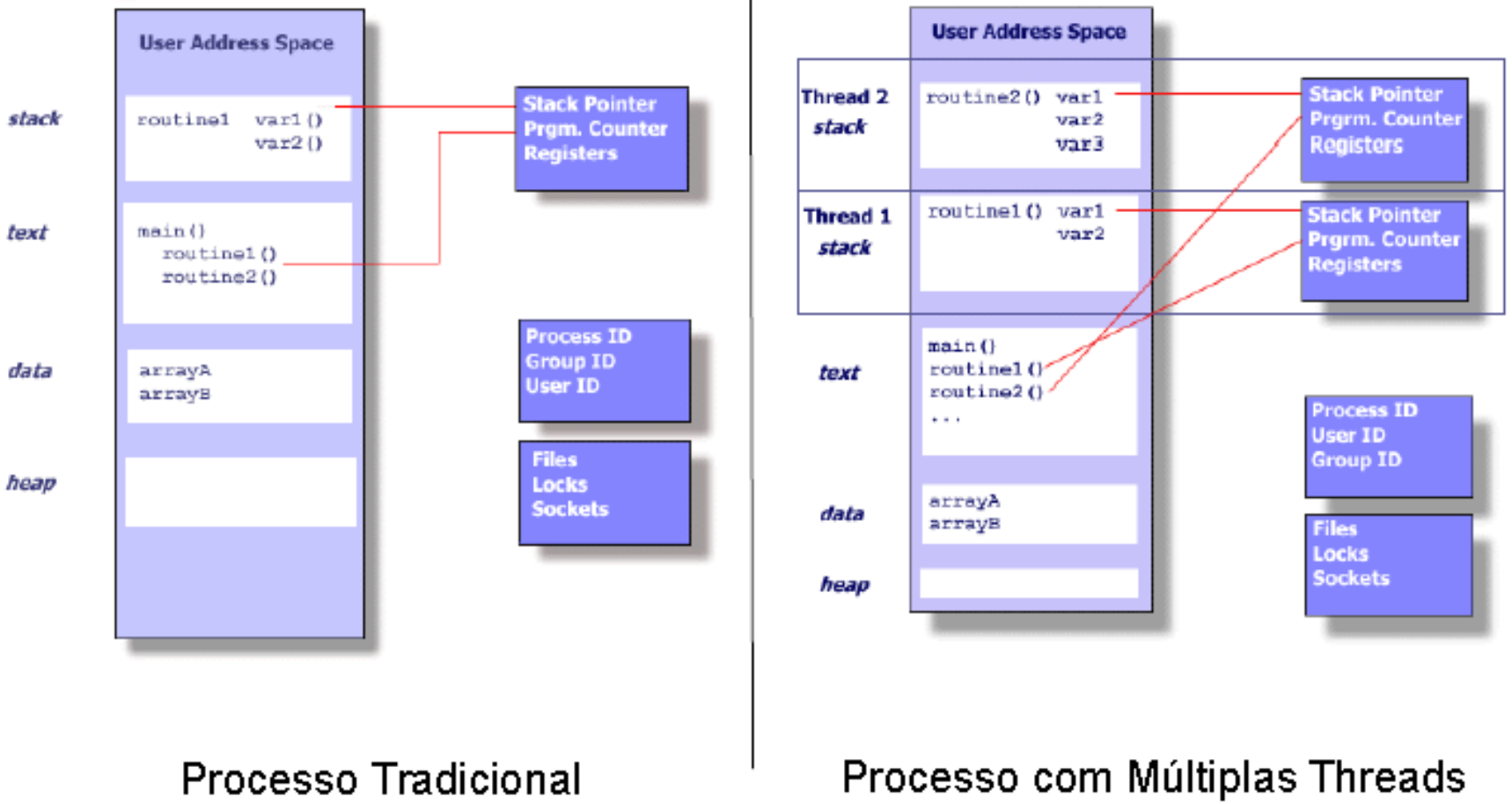
Prof. MSc. João Tavares

Introdução

- Thread → corresponde a um fluxo independente de instruções dentro de um processo
  - Representam partes do programa que podem executar de forma concorrente
  - Permite explorar o paralelismo de E/S e em arquiteturas multiprocessadas

- Implementação
  – Duplicam partes essenciais de um processo.
  – Podem ser gerenciadas
    * pelo SO (nível de sistema),
    * por bibliotecas (nível de usuário); ou
    * por soluções híbridas

Thread X Processo



POSIX Threads

- PTHREAD → padrão POSIX para interface de programação com threads
  - ANSI/IEEE POSIX 1003.1 - 1995 standard
  - Disponível em diversos Unix

- Implementada com uma Biblioteca C
  - Protótipos de funções definidas em pthread.h
    * Todas as funções da API recebem o prefixo *pthread_*
  - Programa deve ser ligado à biblioteca pthread

  Ex.: *compilação usando gcc*

  $ gcc myprog.c -o progname **-l pthread**

POSIX Threads

- A API Pthreads inclui funções para:
  - Gerenciamento de ciclo de vida de threads
    * Criação, terminação, escalonamento
  - Sincronização entre threads
    * Mutex, semáforos, variáveis condicionais, barreiras
  - Gerenciamento de sinais
    * máscara de sinais de thread, envio de sinal a uma thread do mesmo processo
  - Outros...

Programa C com Pthreads

- Inicialmente o main() contém uma única thread
- Todas outras threads devem ser explicitamente criadas pelo programador
  - Em pthreads usa-se a função pthread_create()
  - Pode ser chamado várias vezes no programa
- Cada thread POSIX recebe um ID do tipo pthread_t
  - pthread_self() → retorna o ID da thread chamadora
  - pthread_equal() → usada para comparar IDs
    * Apesar de em algumas implementações, pthread_t ser um número inteiro, isso não é sempre garantido

Criando Threads

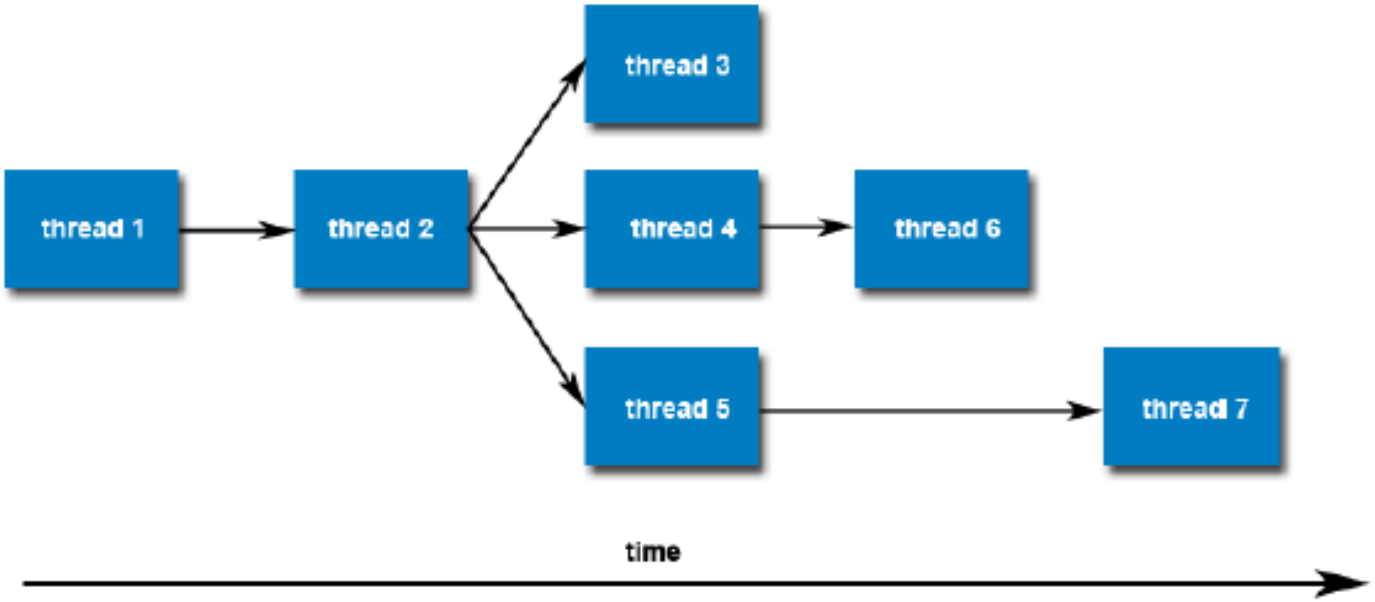
API

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

- Descrição
  - Cria uma nova thread e dispara sua execução
  - Parâmetros:
  * *thread* → receberá o ID único da thread recém criada
  * *attr* → contém atributos de criação da thread
    - Usa-se NULL para criação com atributos padrão
  * *start_routine* → função definida no programa que servirá de ponto de entrada da thread
    - Recebe um parametro void* e retorna um void*
  * arg → parâmetro que será passado à função thread

Criando Threads

- Threads podem criar outras threads



- Após a criação de uma thread, como saber quando ela será escalonada?

Exemplo de Criação de Thread

- *Ex.: thrhello.c*

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

#define NUM_THREADS 3
#define NUM_MSG 5
#define SLEEP_TIME 1

void* PrintHello(void *param) {
    int i;
    char *who;
    who = (char*) param;
    for (i=0; i<NUM_MSG; i++) {
        sleep(SLEEP_TIME);
        printf("Hello World! It's me, %s!\n", who);
    }
    pthread_exit(NULL);
}
```

Exemplo de Criação de Thread

- *Ex.: thrhello.c (continuação)*

```
int main(int argc, char *argv[]) {
    pthread_t tids[NUM_THREADS];
    int n;
    int te;
    char* names[NUM_THREADS] = { "moe", "larry", "curly" };

    for (n=0; n<NUM_THREADS; n++) {
        te = pthread_create(&tids[n],
                            NULL,
                            &PrintHello,
                            (void *) names[n]);
        if (te) {
            errno = te;
            perror("Falha na criação da thread");
            exit(EXIT_FAILURE);
        }
    }
    pthread_exit(NULL);
}
```

Exemplo de Criação de Thread

- Compilar e ligar com a biblioteca pthread

  $ gcc -Wall thrhello.c -o thrhello **-l pthread**

- Executar

  $ ./thrhello

  Hello World! It's me, larry!
  Hello World! It's me, moe!
  Hello World! It's me, curly!
  Hello World! It's me, moe!
  Hello World! It's me, curly!
  Hello World! It's me, larry!
  ...

Atributos de Threads

- Os atributos de uma thread parametrizam como o sistema deverá gerenciar a execução da thread
  Ex.: politica de escalonamento, tamanho da pilha

- São especificados quando da criação da thread através do tipo pthread_attr_t
  - Implementação exata do tipo pode variar, portanto ele é manipulado unicamente através de funções

- Alguns podem ser alterados durante a execução da thread com chamadas específicas
  - Ex.: pthread_setschedprio()

Gerenciando atributos das Threads

API

```
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_getattrname();
int pthread_attr_setattrname();
```

- Descrição
  - init → Aloca armazenamento e atribui valores padrão para os atributos em *attr*
  - destroy → Libera recursos alocados previamente para armazenar atributos em *attr*
  - get/set → Consultam/modificam atributos específicos
    * *detachstate*, *schedpolicy*, *schedprio*, *stack*, *scope* etc.

- Exemplo detalhado → man pthread_attr_init

Terminação de Threads

- Uma thread termina quando:
  - A função que representa seu ponto de entrada (start_routine) retorna
  - A thread chama explicitamente pthread_exit()
  - Outra thread solicita o seu cancelamento usando pthread_cancel()

- Por padrão, um processo com threads termina quando main() termina!
  - Chamada implícita a exit() no final do main()
  - Para modificar esse comportamento, main() deve chamar pthread_exit() explicitamente.

Terminando Threads

API

```
int pthread_exit(void *retval);

int pthread_cancel(pthread_t thread);
```

- Descrição
  - exit → finaliza a thread chamadora, retornado retval como resultado da computação da thread
  - cancel → solicita o cancelamento de outra thread
    * Morte da thread cancelada não é imediata/sincrona!
    * Terminação ocorrerá somente quando a thread alvo atingir um ponto de cancelamento
      - Efetivamente, no ponto de cancelamento, a thread testa se alguém solicitou sua terminação e então pede para encerrar seu processamento

Threads destacadas

- Resultado da computação de threads...
  - Nem sempre uma thread produz um resultado final
  - Outras vezes o resultado é retornado através da memória que já é compartilhada
- Solução Pthreads → dois tipos diferentes de threads
  - Joinable (juntáveis) → resultado da thread será guardado até que seja coletado por outra thread
    * POSIX especifica que esse é o comportamento padrão, mas isso não é estritamente respeitado em todas as implementações
  - Detached (destacadas) → sistema é autorizado a desalocar todos os recursos reservados a thread tão logo ela termine

Threads destacadas

- O tipo JOINABLE ou DETACHED pode ser especificado com um atributo de criação da thread
  - Usar função pthread_attr_setdetachstate()
  - POSIX define que JOINABLE seria o padrão se omitido
    * Essa recomendação não é estritamente respeitado em todas as implementações

- pthread_detach() → permite transformar uma thread joinable em destacada após sua criação
  - A transformação é irreversível

Threads destacadas

```
pthread_t tid;
int err;
pthread_attr_t attr;
...
pthread_attr_init(&attr);
err = pthread_attr_setdetachstate(&attr,
                                  PTHREAD_CREATE_DETACHED);
if (err != 0) {
    errno = err; perror("Falha ao modificar attributo")
    exit(EXIT_FAILURE);
}
...
err = pthread_create(&tid,
                     &attr,
                     &PrintHello,
                     (void *) names[n]);
...
pthread_attr_destroy(&attr);
```

Aguardando Término das Threads

API

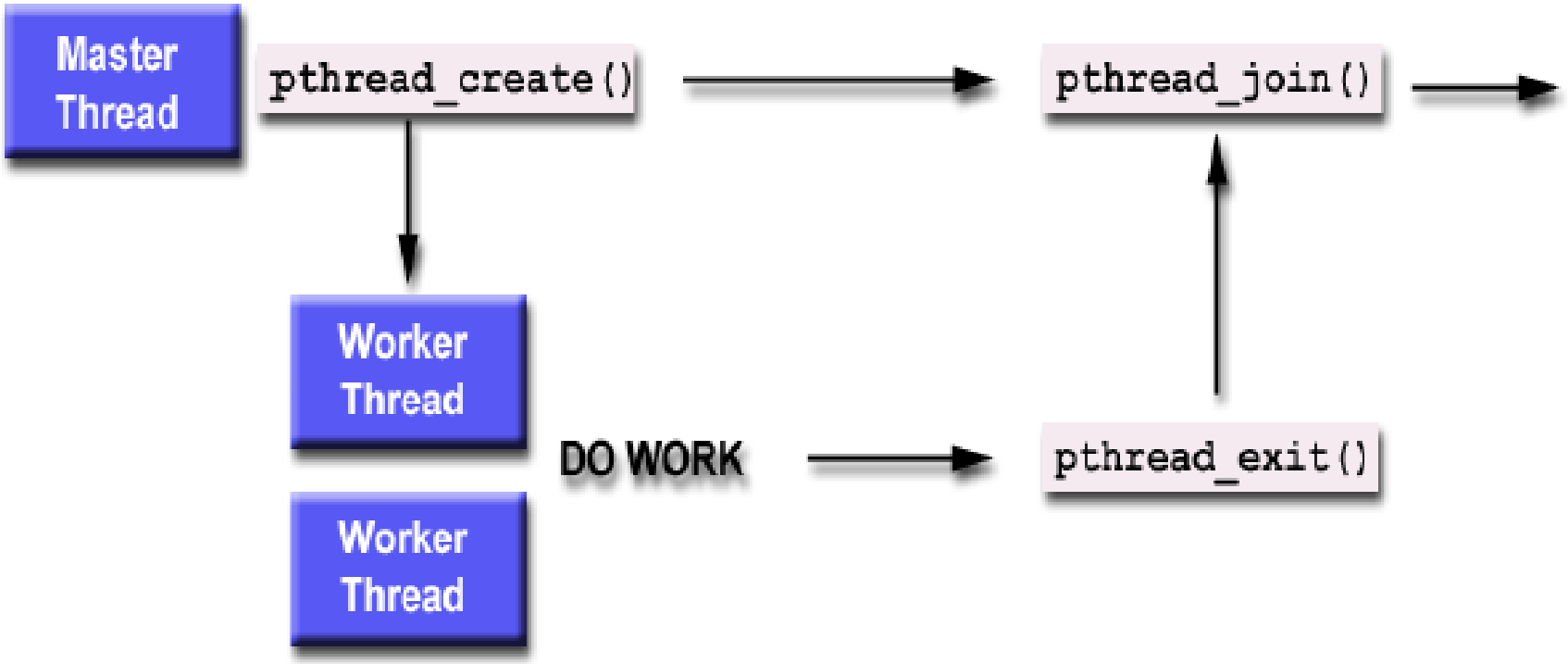
```
int pthread_join(pthread_t thread, void **retval);
```

- Descrição
  - Aguarda o término da execução de uma thread e coleta o resultado final de sua computação
  * thread → ID da thread aguardada
  * retval → endereço de variável onde o resultado da thread será armazenado

- Thread aguardada deve ser do tipo *joinable*

- Pode ser realizada uma única vez para cada thread

Aguardando Término das Threads



Exemplo 2

- *Ex.: join, attributos e parâmetros da thread*

```c
/* Includes omitidos, consulte as man pages */

#define NUM_THREADS 3
#define ROUNDS 10000000;

struct WorkUnit {
        int id;
        int n_runs;
        double result;
};

void *BusyWork(void *param) {
   int i;
   struct WorkUnit *wu = param;
   printf("Starting thread #%d\n", wu->id);
   wu->result = 0.0;
   for (i=0; i<wu->n_runs; i++) wu->result += random();
   pthread_exit((void *) wu);
}
```

Exemplo 2

- *Continuação...*

```
int main(int argc, char *argv[]) {
   pthread_t thread[NUM_THREADS];
   struct WorkUnit wunits[NUM_THREADS];
   struct WorkUnit *w;
   pthread_attr_t attr;
   int rc, t;
   struct WorkUnit *status;

   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr,
                               PTHREAD_CREATE_JOINABLE);
   for(t=0;t<NUM_THREADS;t++) {
      w = &wunits[t];
      w->id = t; w->n_runs = ROUNDS;
      printf("Creating thread %d\n", t);
      rc = pthread_create(&thread[t], &attr, &BusyWork, w);
      if (rc) {
        errno = rc; perror("pthread_create()");
        exit(EXIT_FAILURE);
      }
```

Exemplo 2

- *Continuação...*

```
    } /* for: create */

    pthread_attr_destroy(&attr);

    for(t=0;t<NUM_THREADS;t++) {
        rc = pthread_join(thread[t], (void *) &status);
        if (rc) {
          errno = rc; perror("pthread_join()");
          exit(EXIT_FAILURE);
        }
        printf("Thread #%d produced result=%f\n",
                t, status->result);
    } /* for: join */

    pthread_exit(NULL);

} /* main() */
```

Sincronização entre Threads

API

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
•int pthread_mutex_lock(pthread_mutex_t *mutex);
•int pthread_mutex_trylock(pthread_mutex_t *mutex);
•int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- Descrição

- init → inicializa uma trava do tipo mutex (exclusão mútua) para uso com as pthreads
- destroy → libera recursos previamente alocados para um mutex
- lock → bloqueia até que mutex esteja livre e então obtém o lock
  * trylock → alternativa não bloqueante
- unlock → libera mutex

Sincronização entre Threads

- API bastante rica suportando diversos modelos de sincronização:
  - Mutex
  * Mais informações → man pthread_mutex_init
  - Variáveis condicionais
  * Mais informações → man pthread_cond_init
  - RW locks
  * Mais informações → man pthread_rwlock_init
  - Barreiras
  * Mais informações → man pthread_barrier_init
  - Semáforos
  * Mais informacoes → man sem_init

Threads x fork() x exec()

- Em um exec()...
  - Todas as threads do processo são mortas
  - Código da thread não existe no novo programa!

- Em um fork()...
  - Apenas a thread chamadora é clonada
  - Necessário atenção especial para evitar deadlocks
    * Filho herda cópia da memória do pai
    * Incluindo variáveis de lock com seu estado atual!

Threads x Sinais

- Cada thread tem máscara de sinais própria
  - Configurável através de pthread_sigmask()

- Porém, as disposições de sinais são características globais do processo
  - Sinais entre threads de um mesmo processo...
    * Enviado com pthread_kill()
    * Sinal será tratado pelo no contexto da thread destino
  - Quando um sinal de outro processo chega...
    * Uma thread que não tenha bloqueado aquele sinal é parada para executar o tratador do sinal
    * Demais threads permanecem em execução!

Leituras complementares

- STEVENS, W.R. Advanced Programming in the UNIX Environment. 2nd. Ed., Addison Wesley, 2005.

- Man pages
  - cada uma das funções abordadas
  - pthreads(7) → provê visão geral da API

- Tutorial Online sobre PThreads
  https://computing.llnl.gov/tutorials/pthreads/

- Livro: Advanced Linux Programming
  http://www.advancedlinuxprogramming.com/alp-folder

Referências Bibliográficas

- Material originalmente elaborado por Prof. Cristiano Costa. Material autorizado e cedido pelo autor. Revisado e atualizado por Prof. Luciano Cavalheiro e posteriormente pelo Prof. João Tavares.