Programação Pthreads (Prioridades e Afinidades)

Dr. Osmar Marchi dos Santos

Definir prioridades

- As prioridades podem ser definidas juntos com a fila de escalonamento associada ao processo

- Para isso, as seguintes funções são utilizadas:

  - int pthread_setschedparam(pthread_t thread, int policy, const struct sched_param *param);
  - int pthread_getschedparam(pthread_t thread, int *policy, struct sched_param *param);

- Onde a estrutura sched_param é definida:
  - struct sched_param {
  - int sched_priority; /* Scheduling priority */
  - };

Definir prioridades

- As seguintes políticas de escalonamento são definidas:
  - SCHED_FIFO:
    - FIFO com prioridades
  - SCHED_RR:
    - RR com prioridades
  - SCHED_OTHER:
    - Política normal do sistema (normalmente RR e outros mecanismos para otimizar o sistema do ponto de vista do usuário)

Definir prioridades

- Em Linux, é possível ler as prioridades max e min do escalonador da seguinte forma:
  - #include <sched.h>
  - int sched_get_priority_max(int policy);
  - int sched_get_priority_min(int policy);

Definir afinidades

- Afinidade consiste em especificar para o Sistema Operacional, qual o processador em que a thread deve ser alocada
- Isso possibilita gerenciar da melhor forma o uso dos processadores de acordo com as necessidades da aplicação
- Em Linux, a seguinte função é definida para pthreads:
  - #define _GNU_SOURCE /* See feature_test_macros(7) */
  - #include <pthread.h>
  - int pthread_setaffinity_np(pthread_t thread, size_t cpusetsize, const cpu_set_t *cpuset);
  - int pthread_getaffinity_np(pthread_t thread, size_t cpusetsize, cpu_set_t *cpuset);

Definir afinidades

- Porém, para definir corretamente as afinidades, é necessário manipular a estrutura cpu_set_t (que representa o conjunto de processadores do sistema) para definir o processador para execução da thread

- Em Linux, as principais macros incluem:
  - #define _GNU_SOURCE /* See feature_test_macros(7) */
  - #include <sched.h>
  - void CPU_ZERO(cpu_set_t *set);
  - void CPU_SET(int cpu, cpu_set_t *set);
  - void CPU_CLR(int cpu, cpu_set_t *set);
  - int CPU_ISSET(int cpu, cpu_set_t *set);
  - int CPU_COUNT(cpu_set_t *set);

Definir afinidades

- CPU_ZERO: Limpa o conjunto
- CPU_SET: Adiciona uma CPU ao conjunto
- CPU_CLR: Remove uma CPU do conjunto
- CPU_ISSET: Verifica se a CPU está no conjunto
- CPU_COUNT: Retorna a quantidade de CPUs no conjunto

Obter a quantidade de processadores

- As macros anteriores permitem manipular a estrutura cpu_set_t, mas não permitem obter do sistema a quantidade de processadores existentes

- Uma das maneiras para obter isso é utilizar a função sysconf (que obtém dados de alguma configuração do sistema, em particular usando a opção _SC_NPROCESSORS_ONLN de processadores atualmente disponíveis)

  - #include <unistd.h>
  - long sysconf(int name) // definição
  - long procs = sysconf(_SC_NPROCESSORS_ONLN); // uso

Exercícios

- Criar um conjunto de 5 threads que executam um loop por 2 segundos (forçar execução, sem uso de sleeps). Definir prioridades e afinidades com as funções discutidas.
- O quê acontece se associarmos mais de uma CPU e escalonar a thread naquele conjunto?
- Com prioridades iguais, o quê acontece com o código desenvolvido?
- Com prioridades diferentes e políticas diferentes, o quê muda?