Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

DCC641 FSPD — EP 1

Pool dinâmica de threads

Alexander Thomas Mol Holmquist

21 de maio de 2022

1 Introdução

Uma *pool de threads* é composta essencialmente por

1. Um número *mínimo* de threads
2. Um número *máximo* de threads
3. Uma política de escalonamento de tarefas
4. Uma política de alocação de desalocação de threads

Este trabalho é uma implementação de uma pool de threads, acompanhada por esta breve explicação de seu funcionamento. Não fazemos análise de desempenho, mas notamos que o programa escala quase perfeitamente quando as threads são mantidas ocupadas, e de acordo com $min\_threads$ e $max\_threads$. Foi interessante variar esses valores e as tarefas dadas como entrada para o programa, e ver como a pool de threads reage. A Seção 2 apresentará a solução que este trabalho implementa para cada um dos pontos acima.

2 Implementação

Esta seção está dividida de acordo com a enumeração apresentada na Seção 1. *pool* é o nome da classe que implementa o funcionamento da thread mestre. Em alto nível, a thread mestre e as threads trabalhadoras acessam a fila de tarefas através do esquema clássico produtor-consumidor (veja arquivo `lib/main/task/task.cpp`).

2.1 Número mínimo de threads

Ao ser iniciada, *pool* já acorda $min\_threads$ e as dispara para trabalho. Cada thread então executa uma espera ocupada por tarefas. Uma thread só sai dessa espera ocupada sob uma das condições abaixo:

- Recebe descritor de tarefas especial EOW
- O número de threads esperando por tarefas é maior que o número mínimo de threads

Como, por definição, o número de threads esperando por tarefas é menor ou igual ao número de threads ativas, tem-se que o número de threads no programa nunca pode cair abaixo de $min\_threads$, a não ser que o descritor de tarefas especial EOW seja passado, indicando a terminação do programa. O descritor EOW deve ser o único descritor que tem o valor 0 para pid e ms.

2.2 Número máximo de threads

Depois de iniciada a pool de threads, a única função que cria threads se chama `pool::dispatch_thread`. Essa função é chamada no enlace principal da pool de threads `pool::run`. Ela cria uma thread se e somente se as seguintes condições são satisfeitas:

- O número de threads esperando por tarefas é 0. Isto é, todas as threads atuais estão ocupadas.

- Existe alguma thread na fila `thread::thread::free_threads`

O *set* `thread::thread::free_threads` tem tamanho máximo $max\_threads$. Isso é garantido, porque os dois momentos em que se insere elementos nesse conjunto são:

- Ao iniciar a pool, insere-se exatamente $max\_threads$ identificadores de threads
- Quando uma thread termina seu trabalho, seu identificador é inserido (e é removido quando começa a trabalhar)

Note que `thread::thread::free_threads` só é acessado pelos métodos da classe `thread::thread`, sempre com a guarda de um mutex.

2.3 Política de escalonamento de tarefas

Descritores de tarefa são criados à medida que o programa recebe entradas na entrada padrão. Descritores criados são então inseridos em uma fila de tarefas global `task::TQ::QUEUE`. O tamanho máximo dessa fila é 40, como requerido pela especificação. Note que essa fila só é acessada pelas funções `consume_task` e `produce_task` do namespace `task`, sempre com a guarda de um mutex. Essas duas funções implementam o paradigma produtor-consumidor.

2.4 Política de alocação e desalocação de threads

Threads são alocadas de acordo com o processo descrito na Seção 2.2.

A Instruções de uso

A interface de linha de comando do programa segue a especificação. Isto é, a usage básica é:

```
<program> <min_threads> <max_threads>
```

Existe um parâmetro adicional permitido *-d* que pode ser utilizado para imprimir logs que ajudam com depuração. Digite *make build* para gerar o executável *prog* no diretório principal. Digite *make clean* para deletar todos os arquivos gerados com *make build*.

O zip entregado também contém o arquivo `load_gen.sh`, que é um gerador de carga. Um exemplo de uso deste gerador é dado abaixo.

```
./load_gen.sh 10 1000 10 | ./prog 4 40
```

O funcionamento do gerador de carga é trivial, e pode ser verificado lendo o código do script.

B Detalhes da máquina local

- **Sistema operacional:** Ubuntu 20.04.3 LTS.
- **CPU:** Intel(R) Core(TM) i7-1065G7 @ 1.30GHz-3.90GHz. 1 processador físico, 4 núcleos, 8 threads virtuais.
- **RAM:** 16GB tecnologia DDR4 a 3200MHz.