UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS OPERACIONAIS II

P5: Partitioned Multicore Scheduling with Migration

Gustavo Tarciso da Silva
Lucas Mayr de Athayde
Teo Haeser Gallarza

PROFESSOR

Antonio Augusto Medeiros Frohlich

Florianópolis
Dezembro de 2020

Sumário

1 Introdução

Para o escalonador particionado, foi adaptado o escalonador Shortest Remaining Time First, implementado nas entregas anteriores, onde foi criado uma lista para cada hart do processador e as threads foram inseridas nessa lista, onde cada thread era executada no core respectivo da lista.

Houveram tentativas de implementar a migração de threads de uma fila para a outra, porém, não conseguimos uma implementação funcional. A seguir serão descritas as nossas tentativas de implementar a migração de acordo com o que acreditávamos ser os trechos de código correto a ser desenvolvido no EPOS.

Foram criados testes para demonstrar o funcionamento do escalonador particionado, e um teste que seria utilizado para verificar o funcionamento da migração. O arquivo readme.txt possui mais informações sobre os testes.

2 Problemas encontrados

2.1 Primeira Proposta

Sobrecarga do método queue() do escalonador.

Segundo a documentação do método priority() em thread.cc, percebemos que uma solução para a migração seria a sobrecarga do método queue() do nosso escalonador implementado no P4, onde o novo método retornaria o valor da queue para onde o recurso deveria ser migrado.

```
static unsigned int queue() { return ++_next_queue %= CPU::cores(); }
```

Além disso, ao nos depararmos com PANIC!, decidimos que além do método queue(), era necessário sobrecarregar o método current_queue() para refletir a mudança de queues. Tentamos criar variáveis locais para o controle das queues sem sorte.

2.2 Segunda Proposta

Ainda se debruçando sobre o método priority() encontrado em thread.cc, é possível notar que o método recebe um critério e a partir dele e de seu .queue() respectivo, então uma possibilidade para se resolver o problema seria criar um novo critério para essa Thread, e a partir disso, conseguir atualizar a queue da Thraed em questão. Esse novo critério teria uma fila diferente, baseado em algum critério, e assim, por meio da implementação de priority(), faria com que a thread fosse migrada para outro core.

A migração ficaria ao encargo de

```
_link.rank(c);
```

que atualiza o rank e, consequentemente, a queue do _link, então sendo apenas necessário reescalonar o cpu que teve uma thread retirada ou inserida.

2.3 Interrupção

Essas implementações que se utilizando do método priority() seriam bem satisfatórias pois se utilizando de priority() para atualizar a queue, já será feita a parte de re-escalonamento da CPU onde será atualizada, o que faltaria apenas uma parte da implementação para a execução correta da migração, o tratamento correto de interrupção por parte da cpu.

O trecho de código onde se mostra importante essa implementação é em reschedule(unsigned int cpu), onde o reschedule é enviado a outro cpu, por meio de IC::ipi(cpu, IC::INT_RESCHEDULER), e então é necessário dentro da parte de interrupções implementar uma função que, ao ser

chamada, apenas chama a função reeschedule() para o próprio cpu, assim colocando a thread corretamente no escalonamente daquela fila da cpu.