Iniciado em	Sunday, 15 May 2022, 17:57
Estado	Finalizada
Concluída em	Sunday, 15 May 2022, 17:59
Tempo empregado	2 minutos 9 segundos
Avaliar	**10,0** de um máximo de 10,0(**100**%)

Questão 1

Correto

Atingiu 1,5 de 1,5

1. A instanciação de objetos dentro de métodos sem o uso do operador *new* implica
   na liberação da memória alocada após a saída do método ✓ .
2. Para o uso de tratadores de interrupção reentrantes é ideal que o método
   utilize apenas variáveis de escopo local ✓ .
3. Uma fila relativa é caracterizada por
   lidar com ordenação de elementos baseada na diferença entre o rank do elemento e dos elementos adjacentes ✓ .

Questão 2

Correto

Atingiu 6,0 de 6,0

1. Na versão anterior do EPOS (referente a preempção) o tratador de interrupções do alarme é programado de tal maneira que caso alguém crie um *Alarm* com um *handler* que contenha um *loop* infinito, o *handler*
   execute indefinidamente e trave o sistema ✓ .
2. O Traits LIFE_SPAN ✓ do EPOS é utilizado, entre outras coisas, para ajustar a resolução de algumas variáveis em função do tempo máximo de execução previsto para o sistema ✓ . Caso a configuração desejada não possa ser implementada pelo EPOS para a arquitetura selecionada, um erro de compilação ✓ é gerado como resultado dos tipos de dados pertinentes terem sido mapeados em void ✓ . Este mecanismo é implementado na classe RTC_Common ✓ .
3. O EPOS usa o TSC ✓ para gerar *ticks* e contar tempo. A partir disso é possível gerenciar alarmes, e neste contexto, se um alarme for criado com tempo menor ou igual a metade de um tick ✓ ele é disparado imediatamente pelo construtor.
4. Analisando a implementação atual do *Alarm::handler()* é possível ✓ que um rank tenha valor negativo quanto múltiplos alarmes são criados para o mesmo tick ✓ .
5. Na implementação reentrante de *Alarm::handler()*, a principal mudança, além da eliminação de variáveis estáticas ✓ , é a postergação da invocação do *handler* até a liberação do lock ✓ . Esta versão impede a postergação indefinida da execução das tarefas de aplicação, mas não impede que uma invocação de *Alarm::handler()* bloqueie indefinidamente a thread que estava executando quando a interrupção ocorreu ✓ . Este problema não acontece quando o handler do alarme é um semáforo, pois a thread associada somente será escalonada se tiver prioridade mais alta do que a atual ✓ .
6. A operação dos timers de um sistema como single-shot (ao invés de tick-based)
   é mais eficiente apenas quando reprogramações dinâmicas dos timers não são necessárias ✓ .

Questão 3

Correto

Atingiu 2,5 de 2,5

Para responder às questões abaixo, considere a versão atual do código do EPOS.

1. Utilizando a aplicação *Philosophers' Dinner executando sobre a arquitetura RV32/RISCV/SiFive_E single-core*, altere o código do EPOS para identificar quantas vezes o método *reschedule()* é invocado durante a execução da aplicação. Identifique também quantas das invocações são feitas pelo *time_slicer*. Após analisar os resultados, pode-se afirmar que

   a maioria das invocações é feita pelo time_slicer ✓ .

2. Considere a aplicação abaixo executando sobre a arquitetura *RV32/RISCV/SiFive_E* single-core, responda:

```
#include <process.h>
#include <utility/ostream.h>
#include <time.h>

using namespace EPOS;
OStream cout;

#define WAIT_TIME ???

const unsigned int thread_num = 5;
Thread * t[thread_num];
int code;

int func_a() {
  Delay wait_micro_sec(WAIT_TIME);
  cout << code-- << " " << endl;
  return 0;
}

int func_b() {
  for (unsigned int i = 0; i < thread_num; ++i) {
    t[i] = new Thread(&func_a);
    cout << code++ << " " << endl;
  }
  for (unsigned int i = 0; i < thread_num; ++i) {
    t[i]->join();
    delete t[i];
  }
  return 0;
}

int main()
{
  cout << "Simple Test" << endl;
  Thread * c = new Thread(&func_b);
  c->join();
  delete c;
  cout << "\nThe end!" << endl;
  return 0;
}
```

3. Considerando um tempo de Tick de 10ms. Qual o maior valor possível, em us (1/1000ms), para *WAIT_TIME* a fim de que a saída da aplicação se mantenha como "0 -1 0 -1 0 -1 0 -1 0 -1"? 4999 ✓ . Não modifique nada além do valor de *WAIT_TIME*. Justificativa:

   Isto acontece pois se um alarm é criado com valor menor que a metade de um tick, ele é disparado imediatamente pelo construtor ✓ .

◄ OSDI with EPOS: Timing and Alarms

Seguir para...

E5b: OSDI with EPOS: Hardware Performance Monitor (coding) ►