Iniciado em	Sunday, 24 Apr 2022, 22:30
Estado	Finalizada
Concluída em	Sunday, 24 Apr 2022, 22:46
Tempo empregado	15 minutos 48 segundos
Avaliar	**8,7** de um máximo de 10,0(**87**%)

Questão 1

Correto

Atingiu 4,0 de 4,0

A respeito da linguagem C++ em suas versões posteriores à 11, pode-se afirmar que:

1. Classes [podem] ser declaradas dentro de outras classes.
2. Namespaces [podem] ser declarados dentro de outros namespaces.
3. Namespaces [podem] ter partes em diferentes unidades de compilação.
4. Objetos complexos, que possuem construtores, [podem] ser alocados dinamicamente com o operador new.
5. Objetos complexos, que possuem construtores, [podem] ser alocados estaticamente.
6. Objetos globais [podem] ter construtores, [uma vez que são] invocados antes da função main().
7. Objetos da biblioteca padrão como *cout* precisam ser explicitamente criados porque [nem todos os programas precisam dele].
8. Destrutores de objetos globais [são invocados] após o retorno da função main(), pois [eles ainda podem manipular tais objetos].
9. A palavra reservada *class* [é] equivalente a palavra reservada *struct*, à exceção da visibilidade, que é privada na primeira e pública na última.
10. O uso de cláusulas *friend* [nem sempre] quebra o encapsulamento preconizado pelo paradigma de programação orientado a objetos.
11. Atributos declarados como *static const* cujos tipos são derivados dos nativos da máquina [não geram] código quando compilados.
12. Templates de classes do tipo *Traits* [não geram] código quando compilados e servem para descrever as características de [estruturas de dados e de algoritmos], principalmente no paradigma de programação intitulado [Generic Programming].
13. Construtores protegidos tornam uma classe [uma unidade de reuso por herança, sem definir um tipo].
14. Enumerações declaradas como *enum class* podem definir operadores de conversão [que substituiriam] os pares de *typedef* e *enum* presentes no EPOS.
15. Herança múltipla [pode ser muito útil para manter correspondência com o domínio], já herança privada [pode ser usada para estabelecer relações de reuso sem estabelecer relações de tipo].
16. Métodos declarados como *const* [não podem] alterar atributos do objeto.

Questão 2

Parcialmente correto

Atingiu 0,7 de 2,0

Considere o seguinte programa escrito para o sistema operacional OpenEPOS:

```
#include <utility/ostream.h>
using namespace EPOS;
OStream cout;
int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Compile e execute o programa usando a versão atual do repositório INE5424-2022-1 do EPOS configurada para a arquitetura RISCV e modo *library* e então utilize as ferramentas ***nm***, ***objdump***, ***readelf*** e ***gdb*** do *cross-compiling toolchain* do EPOS para obter informações similares às obtidas nos tutoriais "Hello World" e "Where is my variable" com as ferramentas do LINUX.

Em seguida, responda as questões abaixo:

1. Utilizando-se a ferramenta ***objdump*** e ***readelf*** para inspecionar os headers do executável resultante do processo de compilação, vemos que o programa tem [ ] segmentos, sendo que o segmento de código está localizado no endereço [0x80000000] , tem tamanho [ ] bytes e será alinhado pelo SO em [ ] bytes, e o segmento de dados está localizado no endereço [ ] , tem tamanho [480] bytes e será alinhado pelo SO em [32] bytes. O segmento de código conterá as seções: .text e [.init] e o segmento de dados as seções: .data e [.bss] .
2. Com a ferramenta ***nm***, podemos ver que o entry point está setado para o símbolo [_start] e a função main() está localizada em [ ] . O entry point do programa está setado em [ ] . Podemos também observar a forma como o compilador fez o *mangling* dos nomes C++. Por exemplo, o símbolo EPOS::S::Thread::dispatch(EPOS::S::Thread*, EPOS::S::Thread*) foi *mangled* como [ ] . Podemos também identificar em qual seção ELF cada símbolo foi alocado, sendo que símbolos marcados com o flag T pertencem à seção .text, com o flag [ ] pertencem à seção .data, com o flag [ ] pertencem à seção .bss, com o flag [ ] pertencem à seção .rodata.
3. Com a ferramenta ***gdb***, a qual tem sua invocação simplificada no EPOS com o comando "make APPLICATION=hello debug", podemos identificar que, na entrada da função main(), o stack pointer está apontando para [ ] e que os flags da CPU mostram interrupções [habilitadas] .

Questão 3

Correto

Atingiu 4,0 de 4,0

A respeito da versão didática inicial do OpenEPOS que estamos usando, responda às seguintes questões:

1. O uso de estruturas de dados [dinamicamente] ✓ alocadas permite um melhor aproveitamento dos recursos da máquina, principalmente se a implementação de filas não apresentar todo o overhead das implementações tradicionais. Os sistemas UNIX-like, incluindo o LINUX e o BSD, utilizam estruturas de tamanho [fixo (arrays) super-dimensionados] ✓ para o mesmo propósito, [desperdiçando] ✓ recursos que poderiam ser utilizados pelas aplicações.
2. As pilhas das threads [não crescem] ✓ dinamicamente como a heap.
3. A possibilidade de criar-se threads a partir de funções com qualquer assinatura [implica em sofisticadas manipulações das pilhas de tais threads] ✓ .
4. A solução de Lamport ao problema do Jantar dos Filósofos proposto por Djikstra é livre de deadlocks [sempre] ✓ .
5. A invocação de *exit()* quando a função *main()* chama *return* é feita [pelo código de inicialização C (CRT0) em tempo de execução] ✓ .
6. Threads são criadas por default em estado [READY] ✓ .
7. Thread::join() é um [método] ✓ que faz com que a *thread* referenciada pelo ponteiro [_running] ✓ espere pelo término da *thread* referenciada pelo ponteiro [this] ✓ .
8. Quando um objeto de sistema, como Thread e Semaphore, é deletado, [o destrutor é invocado e então a memória é desalocada] ✓ .
9. Um [Semaphore] ✓ pode ser utilizado tanto para sincronizar seções críticas quanto para contar recursos compartilhados, [Mutex] ✓ não.
10. O estado de uma Thread é um atributo *volatile* porque [é manipulado fora da semântica sequencial assumida pelo compilador] ✓ .
11. A reunião de todas as funções [init()] ✓ em um único segmento objetiva liberar a memória por elas ocupada após a inicialização do sistema.
12. Para inserir-se um objeto de sistema em múltiplas filas ao mesmo tempo, é necessário que este [possua múltiplos atributos do tipo Queue::Element] ✓ .
13. Thread::lock() [desabilita interrupções] ✓ e portanto basta para evitar preempções em plataformas [monoprocessadas] ✓ .

◄ OSDI with EPOS: Introduction

Seguir para...

OSDI with EPOS: Blocking Synchronization ►