Revisar envio do teste: Semana 6 - Atividade Avaliativa

Usuário	LIZIS BIANCA DA SILVA SANTOS
Curso	Sistemas Computacionais - COM210 - Turma 008
Teste	Semana 6 - Atividade Avaliativa
Iniciado	13/09/23 19:20
Enviado	13/09/23 19:29
Data de vencimento	15/09/23 05:00
Status	Completada
Resultado da tentativa	10 em 10 pontos
Tempo decorrido	8 minutos
Instruções	Olá, estudante! 1. Para responder a esta atividade, selecione a(s) alternativa(s) que você considerar correta(s); 2. Após selecionar a resposta correta em todas as questões, vá até o fim da página e pressione "Enviar teste". 3. A cada tentativa, você receberá um conjunto diferente de questões. Pronto! Sua atividade já está registrada no AVA.
Resultados exibidos	Todas as respostas, Respostas enviadas, Respostas corretas, Comentários, Perguntas respondidas incorretamente

Pergunta 1 1,43 em 1,43 pontos

As últimas quatro funções de API relacionadas são usadas para gerenciar arquivos mapeados em memória. Para mapear arquivos, você deve primeiro criar um objeto de mapeamento de arquivo usando uma função de API. Essa função retorna um identificador para um objeto de mapa de arquivos e, opcionalmente, registra um nome no sistema de arquivos para que outro processo possa usá-lo.

Assinale a alternativa correta que corresponde à função API citada.

Resposta Selecionada: a. CreateFileMapping cria um objeto, mapeia um arquivo e lhe designa (opcionalmente) um nome.

Respostas:
- a. CreateFileMapping cria um objeto, mapeia um arquivo e lhe designa (opcionalmente) um nome.
- b. OpenFileMapping abre um objeto de mapeamento de arquivo previamente criado.
- c. MapViewOfFile remove um arquivo mapeado do espaço de endereço.
- d. MapViewOfFile é uma chamada que determina o mapeamento e estabelecimento do arquivamento.
- e. UnmapViewOfFile mapeia (parte de) um arquivo para o espaço de endereço.

Comentário da resposta: **JUSTIFICATIVA**
CreateFileMapping está correta, pois é a chamada API do Windows 7 para a criação de um objeto, mapeia um arquivo e lhe designa (opcionalmente) um nome.
MapViewOfFile está incorreta, pois é utilizado para mapear (parte de) um arquivo para o espaço de endereço de um determinado processo de chamada.
UnmapViewOfFile está incorreta, pois remove (desmapeia) um arquivo mapeado do espaço de endereço de um dado processo de chamada.
MapViewOfFile está incorreta, pois mapeia (parte de) um arquivo para o espaço de endereço e não é usado para remover nada.
OpenFileMapping está incorreta, pois é utilizada para abrir um objeto de mapeamento de arquivo previamente criado.

Pergunta 2 1,43 em 1,43 pontos

De acordo com a relação entre a linguagem-alvo e a linguagem-fonte, a tradução pode ser categorizada de duas maneiras. Em relação a essa definição, faz-se a seguinte afirmação: em situações em que a linguagem-fonte é fundamentalmente uma representação simbólica de uma linguagem de máquina numérica, o tradutor se chama _____, e a linguagem-fonte se chama _____. Já em situações em que o tradutor é chamado de compilador, irá ocorrer em situações em que a linguagem-fonte é tida como alto nível, como _____, e a linguagem-alvo é uma linguagem de máquina numérica.
Preencha as lacunas escolhendo a alternativa correta.

Resposta Selecionada: b. *assembler* (montador), linguagem *assembly*, JAVA e linguagem C.

Respostas:
- a. *assembler* (montador), JAVA e linguagem C, linguagem *assembly*.
- b. *assembler* (montador), linguagem *assembly*, JAVA e linguagem C.
- c. JAVA e linguagem C, *assembler* (montador), linguagem *assembly*.
- d. JAVA e linguagem C, *assembler* (montador), linguagem C.
- e. linguagem assembly, *assembler* (montador), JAVA e linguagem C.

Comentário da resposta: **JUSTIFICATIVA**
A primeira lacuna é completada pelo termo "assembler (montador)", pois, quando a linguagem-fonte é, basicamente, uma representação simbólica para uma linguagem de máquina numérica, o tradutor é denominado *assembler* (montador).
A segunda lacuna é completada pelo termo "linguagem assembly", pois uma representação simbólica da linguagem de máquina numérica é chamada de linguagem-fonte. Nesse caso, a linguagem *assembly* é o tradutor apropriado para o processo.
A terceira lacuna é completada pelo termo "JAVA e linguagem C", pois essas são linguagens de alto nível, por serem linguagens que podem ser lidas e escritas facilmente por seres humanos.

Pergunta 3 1,43 em 1,43 pontos

Existem ligeiras diferenças que podem ser verificadas nas notações para definição de macros em diferentes *assemblers*, mas todos querem as mesmas partes básicas em uma definição de macro. Sendo assim, algo comum entre os diferentes *assemblers*:

I. O texto do corpo da macro.
II. Um cabeçalho que chama uma outra macro criada no passado.
III. A pseudoinstrução que marca o início da macro.

Assinale a alternativa correta sobre os ponteiros:

Resposta Selecionada: d. I, apenas.

Respostas:
- a. I e III, apenas.
- b. III, apenas.
- c. II e III, apenas.
- d. I, apenas.
- e. I e II, apenas.

Comentário da resposta: A afirmativa I está correta, pois as partes básicas de uma instrução de macro são: o texto do corpo da macro, a pseudoinstrução que marca o final da macro e um cabeçalho da macro, que dá o nome da macro que está sendo definida. Sendo assim, a afirmativa correta é "I, apenas". Já as afirmações II e III estão erradas, pois têm sentido oposto ao que foi afirmado na questão.

Pergunta 4 1,43 em 1,43 pontos

A linguagem *assembly* é comumente referida como linguagem *assembler* e é considerada uma linguagem de programação de baixo nível. A linguagem *assembly* pura é uma linguagem em que cada declaração produz apenas uma instrução de máquina. Ou seja, há uma correspondência um-para-um entre instruções de máquina e instruções em um montador.

Com base nas informações, marque a alternativa correta.

Resposta Selecionada: d. Ao contrário da programação em linguagem de máquina (binária ou hexadecimal), a linguagem *assembly* é mais fácil de programar.

Respostas:
- a. As abreviaturas para adição, subtração, multiplicação e divisão são ADD, SUB, MUL, DIV, as mesmas são usadas em linguagem de máquina.
- b. Para programadores de linguagem *assembly*, nenhum conhecimento de nomes de símbolos é necessário, porque o montador os traduz para instruir a máquina.
- c. O programador em linguagem de máquina pode optar por usar o valor numérico do endereço, que é mais comumente usado.
- d. Ao contrário da programação em linguagem de máquina (binária ou hexadecimal), a linguagem *assembly* é mais fácil de programar.
- e. O uso de nomes simbólicos e endereços simbólicos em vez de binários ou hexadecimais não faz diferença significativa para a linguagem.

Comentário da resposta: **JUSTIFICATIVA**
A linguagem de montagem é uma linguagem de transição mais fácil de programar e, portanto, amplamente utilizada por programadores comparada à linguagem de máquina (binária ou hexadecimal).
Há um impacto significativo em relação a diferentes linguagens, no uso de nomes e endereços simbólicos, pois algumas linguagens possuem sua maneira particular de definição de tais símbolos.

As abreviaturas ADD, SUB, MUL e DIV são, na verdade, de linguagem *assembly* e não de linguagem de máquina.
Na linguagem *assembly*, não há uma tradução por parte do montador de forma a instruir a máquina, devendo o programador possuir conhecimentos da simbologia da linguagem para realizar isso.

O programador em linguagem de máquina não tem outra opção a não ser utilizar o valor numérico do endereço, ao contrário da linguagem de montagem, em que se pode utilizar nomes simbólicos em vez de valor numérico.

Pergunta 5 1,43 em 1,43 pontos

Programas escritos em linguagem *assembly* devem ser traduzidos em códigos de máquina antes que possam ser interpretados pela máquina. Esse processo de traduzir programas em códigos de máquina é conhecido como linguagem *assembly*, que é um nível inferior de linguagem de programação. As pessoas normalmente usam níveis mais altos de programação ao escrever programas que requerem funcionalidades mais complexas.

Diante do contexto, observe as afirmativas a seguir.

I) A linguagem-fonte e a linguagem-alvo podem ser definidas em diferentes níveis de complexidade.
II) Alguns processadores executam diretamente programas escritos no idioma de origem sem a necessidade de tradução para o idioma de destino.
III) A tradução é efetuada quando há um processador (*hardware* ou intérprete) disponível para o idioma de chegada, mas não para o idioma de origem.
IV) A conversão ocorre em um programa equivalente chamado programa-objeto ou programa binário executável após a conclusão da tradução.

Está correto o que se afirma em:

Resposta Selecionada: c. I, II, III e IV.

Respostas:
- a. I e III, apenas.
- b. II e III, apenas.
- c. I, II, III e IV.
- d. I e II, apenas.
- e. I, apenas.

Comentário da resposta: **JUSTIFICATIVA**
A afirmativa I está correta, pois os diferentes níveis são determinados pelos detalhes da linguagem do detalhe. Quanto mais detalhes são necessários, mais baixo nível é a linguagem, quanto menos detalhes, mais alto nível.
A afirmativa II está correta, pois o programa-fonte não precisa ser traduzido em linguagem-alvo, já que os computadores possuem processadores capazes de executar programas escritos em linguagem-fonte.
A afirmativa III está correta, pois processadores, seja um interpretador ou *hardware*, são necessários para uma tradução da linguagem-alvo, entretanto não há disponibilidade para linguagem-fonte.
A afirmativa IV está correta, pois, após a conclusão da tradução, não há execução direta do programa original na linguagem-fonte e, sim, conversão para um programa binário executável.

Pergunta 6 1,43 em 1,43 pontos

A simplicidade do sistema de arquivos UNIx, do inglês Unix File System (UFS), é o motivo de sua popularidade entre o público. As principais funções do sistema são *creat* e *open*, que possuem a capacidade de criação de novos arquivos no sistema de arquivos.

Com base nessa afirmativa, assinale a alternativa correta.

Resposta Selecionada: a. Creat (name, mode) é uma chamada utilizada para especificar o modo de proteção.

Respostas:
- a. Creat (name, mode) é uma chamada utilizada para especificar o modo de proteção.
- b. open(name, mode) é a chamada que apaga um arquivo (admitindo que há só um *link* para ele).
- c. Unlink (name) é uma chamada que abre ou cria um arquivo e retorna um descritor de arquivo.
- d. write(fd, buffer, count) é chamado e utilizado para a leitura *count bytes* para o *buffer*.
- e. read(fd, buffer, count) é chamado e utilizado para a escritura *count bytes* do *buffer*.

Comentário da resposta: **JUSTIFICATIVA**
Creat (name, mode) está correto, pois esse comando indica que um novo arquivo deve ser criado no sistema operacional, sendo que, caso haja um arquivo, ele será substituído, havendo uma proteção de gravação que exige permissão para tal criação.
Unlink (name) está incorreto, pois a função unlink() deve remover um *link* para um arquivo. Se a função nomear um *link* simbólico, unlink() removerá o *link* simbólico nomeado e não afetará nenhum arquivo ou diretório nomeado pelo conteúdo do *link* simbólico.
open (name, mode) está incorreto, pois a função open() recebe dois parâmetros elementares para a manipulação de arquivos, que é o "file_name", que inclui a extensão do arquivo e assume que ele está no diretório de trabalho atual; e o "mode", que é um parâmetro opcional que define o método de abertura do arquivo, abre ou cria um arquivo e retorna um descritor de arquivo.
read(fd, buffer, count) está incorreto, pois read() tenta ler para contar *bytes* do descritor de arquivo fd no *buffer*. Em arquivos que oferecem suporte à busca, a operação de leitura tem início no deslocamento do arquivo atual, e o deslocamento do arquivo é incrementado pelo número de *bytes* lidos.
write(fd, buffer, count) está incorreto, pois write() escreve para contar *bytes* do *buffer* apontado para o arquivo referenciado pelo descritor de arquivo fd. O número de *bytes* gravado pode ser menor do que o contador se, por exemplo, houver espaço insuficiente no meio físico subjacente ou o limite de recurso RLIMIT_FSIZE for encontrado.

Pergunta 7 1,42 em 1,42 pontos

Unix é um sistema operacional amplamente usado em todas as formas de sistemas de computação, como *desktop*, *laptop* e servidores. O padrão UNix para *threads* é chamado de pthreads e é definido pelo POSix (P1003.1C). Ele contém chamadas para gerenciar e sincronizar *threads*, mas não define se são gerenciadas pelo *kernel* ou totalmente no espaço do usuário.

Diante disso, assinale a alternativa que indica a função mencionada no enunciado.

Resposta Selecionada: d. pthread_mutex_unlock Desbloqueia um *mutex*.

Respostas:
- a. pthread_cond_wait Libera um *thread* que está esperando em uma variável de condição.
- b. pthread_cond_init Destrói uma variável de condição.
- c. pthread_cond_signal Espera em uma variável de condição.
- d. pthread_mutex_unlock Desbloqueia um *mutex*.
- e. pthread_cond_destroy Cria uma variável de condição.

Comentário da resposta: **JUSTIFICATIVA**
pthread_mutex_unlock está correta, pois o objeto *mutex* referenciado por *mutex* deve ser bloqueado chamando pthread_mutex_lock (). Caso o *mutex* se encontre bloqueado, o *thread* de chamada deve bloquear até que o *mutex* esteja disponível.
pthread_cond_init está incorreta, pois é uma chamada utilizada para criar uma variável de condição.
pthread_cond_destroy está incorreta, pois é uma chamada para destruir uma variável de condição.
pthread_cond_wait está incorreta, pois espera uma variável de condição.
pthread_cond_signal está incorreta, pois é chamada para liberar um *thread* e espera para uma variável de condição.