MAC0422 –Prova 3
6 de Dezembro de 2018
Prof. Alan Durham

Esta prova contém 66 afirmações, você deve selecionar 20 destas que estão ERRADAS (há mais erradas). Sua folha de resposta deve conter as questões erradas NA ORDEM com uma justificativa do porquê estão erradas. A justificativa deve ter apenas uma sentença (curta, não deve ter mais que 3 linhas). Respostas sem justificativa não serão consideradas. Cada resposta vale 1 ponto. Não serão consideradas mais que 20 respostas.

1. No Minix existem apenas 3 maneiras dos processos se comunicarem, todas síncronas: arquivos, *pipes* e mensagens.
2. Na versão que utilizamos do Minix o SO é dividido em 4 camadas, sendo que as duas inferiores compartilham o espaço de endereçamento e nas outras os processos rodam com memorias independentes.
3. Em Minix, chamadas de sistemas não são realmente "chamadas". A biblioteca do sistema transforma a chamada de procedimentos no envio de mensagens. Como estas mensagens são assíncronas, o programa pode continuar processando enquanto aguarda o resultado da chamada, melhorando o paralelismo potencial.
4. O funcionamento do Minix com envio de mensagens torna o sistema menos seguro.
5. As duas maneiras de se criar um processo em Minix são a chamada de sistema `` `fork()` `` e a chamada de sistema `` `execve()` ``.
6. As rotinas `` `malloc()` `` e 'free()' do C têm implementação trivial em Minix, basta converter sempre a primeira e a segunda em uma chamada de sistema 'brk()'.
7. A chamada 'open()' para abertura de arquivos poderia ser eliminada sem perda de eficiência. Neste caso utilizaríamos direto o nome do arquivo. A utilidade do comendo é mais documentação, já que o comando 'close()' é necessário.
8. Para implementarmos um pipe entre dois programas na Shell, redirecionando diretamente a saída padrão de um programa para a entrada padrão de outro (tipo 'ls | more') , só utilizamos uma chamada
9. Tabelas de processo são essenciais para o multi-processamento. Elas guardam todas as informações de um processo e permitem o restauro de seu estado quando ele voltar a executar.
10. No Minix existem várias tabelas de processo. Uma no Kernel, e outras nos servidores. Cada tabela contem do dados necessários para funcionamento do serviço (Process Server, File Server, System Task). Com isso algumas informações são armazenadas em mais de uma tabela/.
11. Um processo pode receber vários mensagens enquanto está bloqueado ou pronto, sem executar. Porém a implementação do Minix garante que não é necessário implementar um buffer de mensagens pendentes associada a cada processo.
12. A maneira de implementar resposta a interrupções no kernel e sinais nos processos é semelhante: existe uma tabela de endereços de rotinas de tratamento. Estas rotinas precisam lidar com o problema de salvar o contexto do processamento que estava em curso.
13. Em muitos sistemas operacionais, processos que trabalham em conjunto compartilham alguma memória em comum. O uso compartilhado dessa memória cria "condições de corrida", o que implica que o uso deste recurso precisa ser coordenado. As regiões do programa que acessam a memória comum são chamadas de "regiões criticas".
14. As condições para uma boa solução para o problema da exclusão mútua são: (i) Só um processo deve entrar na região crítica de cada vez.,(ii) um processo não pode esperar por tempo arbitrariamente entrar em sua região critica, não deve ser feita nenhuma hipótese sobre a velocidade relativa dos processos..
15. 'Hard links" não podem ser utilizados em alguns casos onde "soft links" podem.

16. Na arquitetura DMA (abreviação de "Disk Management Access") podemos aumentar a eficiência de proteção no sistema operacional, pois podemos locar um bit de acesso nas instruções de entrada e saída quando queremos permitir acesso apenas pelo kernel.
17. O código abaixo resolve o problema dos filósofos comilões:

```
#define N 5
Philosopher(i) {
     int I;
     think();
     take_chopstick(i);
     take_chopstick((i+1) % N);
     eat();
     put_chopstick(i);
     put_chopstick(i+1);
}
```

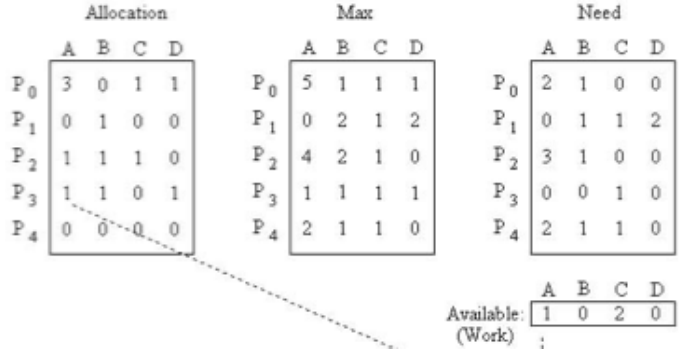
18. Existe uma equivalência entre semáforos, monitores e mensagens. Qualquer um destes esquemas pode ser implementado usando o outro.
19. São 3 os níveis de escalonamento de processos:
    i. alto nível onde se decide quais processos entram na disputa por recursos;
    ii. nível médio onde se decide qual o próximo processo a rodar
    iii. baixo nível onde são escalonados processos de tempo real
20. Em sistemas não preemptivos, o relógio é desligado e o processo decide quando deve ceder a CPU. Isso pode gerar o travamento do sistema.
21. No sistema de *multi-level feeback queues*, existem várias filas de prioridade e cada processo é alocado a uma no início de sua execução. Desta maneira a alocação desta prioridade não é essencial para o bom desempenho do sistema pois o sistema se adapta ao comportamento do processo.
22. Um processo alocado a uma fila de menos prioridade pode demorar mais para executar do que um processo em uma fila de mais prioridade.
23. O sistema FAT de arquivos tem acesso aleatório muito lento, mesmo para arquivos pequenos, devido ao overhead para percorrer a lista ligada.
24. O utilitário fsck faz várias checagens no sistema. Em particular verifica o numero de referencias a determinado I-node em diretórios. Isso é importante para impedir remoção prematura de arquivos.
25. Interleaving é uma técnica de alocação de setores no disco que visa aumentar a eficiência em leituras de blocos consecutivos. Não é implementada no nível de software independente de dispositivo.
26. Processos como editores de texto devem ter alta prioridade.
27. O pseudo-código abaixo soluciona o problema de exclusão mútua para dois processos

```
                                   Int p1QuerEntrar = FALSE;
                                   Int p2QuerEntrar = FALSE;
     Processo 1                                Processo2
while (TRUE)(                          while (TRUE){
    p1querEntrar - TRUE;                 p2querEntrar - TRUE;
    while (p2QuerEntrar){                while (p1QuerEntrar){
       p1QuerEntrar = FALSE;                p2QuerEntrar = FALSE;
       wait(random());                      wait(random());
       p1QuerEntrar = TRUE;                 p2QuerEntrar = TRUE;
   }                                     }
   ...< região críticia>...              ...< região críticia>.......      .
    pqQuerEntrar = FALSE;                p2QuerEntrar = FALSE;
}                                          }
```

28. A troca do sistema FAT pelo de I-nodes em um SO não deve piorar significativamente a performance de leitura seqüencial de arquivos.
29. O pseudo código abaixo resolve o problema de exclusão mútua PARA QUALQUER NÚMERO DE PROCESSOS.

```
binary_semaphore mutex = 1;
While (TRUE){
      P(mutex);
      ......<região critica>.....
     V(mutex);
.....<região não critica>.....
}
```

30. Monitores promovem melhor organização do código de exclusão mutua, mas precisam ser implementados no compilador/interpretador da linguagem. Já semáforos têm uso mais livre e podem também ser oferecidos pelo Sistema Operacional
31. O Algoritmo do banqueiro, utiliza uma metáfora de empréstimos e pagamentos para evitar a ocorrência de deadlocks. Porém, deve ser associado a um algoritmo de tratamento de deadlocks caso ocorra um estado inseguro.
32. Drivers de dispositivos são softwares que precisam conhecer os detalhes de funcionamento dos dispositivos, oferecendo uma visão mais abstrata para os softwares independentes de dispositivo.
33. O algoritmo de escalonamento de disco "menor tempo de busca antes" aumenta a performance de saída dos discos. Alem disso é justo, apesar de sua complexidade.
34. Criação de buffers para, utilizando o princípio da localidade, diminuir o temo de acesso a requisições seguidas ao mesmo setor do disco **não** é função dos drivers.
35. O estado abaixo é seguro:



36. A organização do Minix é bem diferente da do Unix tradicional. No primeiro temos 4 camadas, sendo 3 dedicadas ao sistema operacional. Cada camada têm processos que se comunicam por mensagens. Já nu Unix o código do kernel é mapeado na memória do processo e a comunicação é por chamada de funções/procedimentos.
37. Num canal oculto (covert channel), o software servidor mantém uma porta não declarada com comunicação direta ao software malicioso. Este esquema pode ser controlado encapsulando as comunicações do processo servidor
38. Na estratégia de "working set" todos os programas tem um número fixo de páginas residentes, isso promove uma maior justiça na alocação de espaço.
39. A estratégia de "working set" determina uma "janela" de acessos. O working set consiste das páginas acessadas nesta janela.
40. A estratégia de tabela de páginas invertidas garante um overhead de espaço fixo para o uso de memória virtual, independente do número de processos.
41. Em operações de leitura, o usuário deve fornecer uma área de escrita para o sistema operacional. Se isso não for feito fica difícil prever o overhead de espaço necessário para os sistema de arquivos funcionar. Esta área é independente do buffers do sistema de arquivos.
42. Threads são linhas de processamento que podem compartilhar memória com outras threads. O uso de threads requer muito cuidado devido ao compartilhamento dos dados da pilha de kernel.
43. A grande vantagem de threads, alem do compartilhamento de memória, é que a troca de contexto entre threads do mesmo processo é mais rápida do que a troca de contexto entre processos.

44. Threads podem tanto ser oferecidas pelo sistema operacional (kernel level thread) como simuladas por sistemas de execução (user level threads). Neste último caso a troca de contexto é ainda mais rápida.
45. O modelo OSI consiste em 7 camadas para processamento de comunicação. Cada camada executa funções bem específicas como roteamento, particionamento das mensagens, encriptação. estabelecimento de conexão e escalonamento dos processos que aguardam a comunicação.
46. Carda camada do OSI deve acrescentar mais dados à mensagem original
47. O Protocolo TCP/IP é parcialmente aderente ao padrão OSI, pois junta as duas camadas inferiores e as três camadas superiores.
48. Existem várias classes de endereços IP, cada uma com um tamanho distinto de prefixo. Classes distintas são utilizada para redes de tamanhos distintos. Alem disso alguns intervalos de endereços foram reservados para redes privadas, podendo ser livremente utilizados
49. A camada de transporte do TCP/IP oferece dois tipos de serviços UDP e TCP, A principal diferença entre ambos é que um deles oferece serviço de encriptação e outro não. (Untranslated Delivery Protocol e Translated Criptographed Protocol)
50. Em ambientes de virtualização os Hipervisores de tipo 1 rodam diretamente os sistemas operacionais. Para que seja Possível isolar um si stema de outro isso é feito por emulação de hardware.
51. Hipervisores podem utilizar as técnicas de tradução de operações ou de emulação de instruções que geram *traps*. No primeiro caso é necessário pré-processar o código do sistema convidado.
52. Ambientes de virtualização contam, a partir de 2006, com acréscimos na funcionalidade dos hardwares Intel para dar melhor suporte à execução de sistemas operacionais convidados.
53. Ao criar uma nova chamada de sistemas no Minix, precisamos fazer alterações em todas as camadas do SO
54. O system task faz parte do espaço de memória do kernel, mas possui interface similar à de um driver para receber requisições dos servidores como o Process Manager.
55. O process manager faz parte do espaço de memória do kernel, mas possui interface similar à de um driver para receber requisições dos servidores como o system task.
56. O primeiro processo de usuário a ser executado em sistemas *nix é o init. Todos os outros são criados a partir dele usando a chamada de sistemas fork.
57. Num fork, o processo filho ganha cópias de todos os dados que o processo original possuía, incluindo cópias dos arquivos abertos por eles.
58. O sistema de arquivos é uma parte do SO cuja modificação é crítica: caso o sistema de arquivos não possa ler mais os arquivos armazenados, o sistema pode entrar em falha catastrófica.
59. Sistemas de arquivo FAT-32 são comumente utilizados em pen-drives devido à simplicidade do formato e por ele ter caches write-through.
60. Para implementar a remoção automática de arquivos temporários no Minix, basta remover os arquivos cujos I-nodes têm o tipo especial I_TEMPORARY durante o término de um processo no sistema de arquivos (o que só deve ser feito se aquele processo for o último a apontar para o arquivo temporário).
61. O driver de disco precisa administrar vários detalhes baixo-nível relativos ao hardware do disco, tais como posição da cabeça de leitura, velocidade de rotação, setores a serem lidos, e buffers de blocos lidos.
62. O utilitário de terminal kill, apesar do nome, não serve para matar processos: ele permite enviar sinais para processos, dentre os quais SIGTERM, que encerra o programa gentilmente mas pode ser tratado, e SIGKILL, que não pode ser tratado e mata o programa sempre.
63. Existe um processo no Minix que tem prioridade mais baixa que todos os outros. Ele contém uma rotina de baixo nível que faz o SO rodar num modo de baixa de consumo de energia.
64. Matrizes em C são armazenadas por linha, enquanto em Fortran são armazenadas por coluna. Ao programar, é muito importante observar o armazenamento das matrizes e a forma como elas serão percorridas para garantir o maior proveito possível da localidade dos dados.
65. Processamento em batch não é mais útil nos tempos modernos, pois sistemas multi-usuários já dão conta de todos os casos de uso de batch.
66. Processadores modernos contém uma hierarquia de caches que, quanto mais perto do processador, podem armazenar mais dados .

MAC0422 –Sistemas Operacionais: Prova 3
RESPOSTAS

NOME:_____
NÚMERO USP:_____

COLOQUE NESTA FOLHA A LISTA DAS QUESTÕES ERRADAs COM JUTIFICATIVA. MANTENHA A ORDEM ORIGINAL E INDIQUE OS NÜMEROS. ENTREGUE APENAS ESTA FOLHA.

Verso:

}