Teste 2021 - 2022

1.
a)

Políticas de Escalonamento do Processador:

- Batch Systems usam Non-Preemptive Scheduling:

FCFS (First Come First Serve) Scheduler :
- O processo mais antigo no Ready queue é o primeiro a ser atendido e executado pelo processador
- Non-Preemptive, sendo que não possui time-out ou preempt.
- Pode ser combinado com um esquema de prioridades sendo que nesse caso preemption pode existir
- Favorece processos com CPU-Bound (Longos ciclos de CPU) sobre I/O-Bound (Pequenos ciclos de CPU)
- Pode resultar no mau uso do processador e dispositivos de I/O

• Shortest Process Next (SPN) Scheduler
- Non-preemptive
- O processo com o menor CPU burst time expectável é selecionado
- FCFS é usado no caso de empate quando mais do que um processo tem o menor CPU burst time esperado

Maximizar o throughput, a quantidade de informação que o sistema pode processar durante um dado período de tempo
Minimizar o tempo de espera e de turnaround
Risco de Starvation para processos mais longos
Requer conhecimento sobre o tempo expectável de um processo a executar
Usado systema batch com long-term-scheduling

- Interactive Systems usam Preemptive Scheduling:

Round robin (RR) Scheduler:
- Uso de Time Slicing ou Time Quantum → É atribuído um tempo máximo a cada processo antes de serem preempted, que irá corresponder ao tempo permitido que um processo pode estar a ser executado no CPU de uma vez.
- O processo mais antigo no Ready queue é o primeiro a ser selecionado para ser executado pelo processador

Não há prioridades, no entanto é possível implementar um esquema de prioridades
- O principal problema é o uso de Time Quantum, sendo complicado de estabelecer qual o tempo que deve ser usado.

Se for usado um tempo quantum muito reduzido é bom porque processos com uma duração muito pequena são completados rapidamente no entanto cada context switching envolve um overhead (Explicação em baixo)
- Favorece processos com CPU-Bound (Longos ciclos de CPU) sobre I/O-Bound (Pequenos ciclos de CPU)
- Pode resultar no mau uso do processador e dispositivos de I/O

Explicação:
Context Switching → Tarefa de guardar o estado atual de o processo em execução e fazer o load do estado guardado de outro processo para execução
Overhead → Recursos adicionais ou operações extras necessárias mas que não contribuem para a execução principal da tarefa

Métricas adequadas para Avaliar o Desempenho de cada Sistema:

- Tempo de Turnaround → Batch Systems deve ser minimizado
- Tempo de Turnaround → corresponde ao intervalo de tempo desde que o processo é submetido até à sua conclusão
- Batch Systems são caracterizados por não possuírem processos que, durante a sua execução, necessitam de interação com um ou mais utilizadores. Logo o Tempo de Turnaround é uma métrica adequada, do ponto de vista de um utilizador, que procura que os processos submetidos, sejam concluídos o mais rapidamente possível.
- Interactive Systems são caracterizados por possuírem processos que, durante a sua execução, necessitam de interação com o(s) utilizador(es).
Logo nem sempre se pretende que estes processos sejam concluídos o mais depressa possível, pois para o fazer, implica que outros processos sejam atrasados de um ponto de vista temporal. Visto que há uma grande componente de interação com os utilizadores é preferível que o Tempo de Resposta seja maximizado e não o tempo de conclusão dos processos.

- Tempo de Espera → Batch e Interactive Systems, deve ser minimizado
- Tempo de Espera → Tempo Total gasto no estado Ready
- Em batch systems é importante reduzir ao máximo o tempo de espera para uma melhor utilização dos recursos e uma maior eficiência no processamento
- Em interactive systems é importante reduzir ao máximo o tempo de espera para proporcionar ao indivíduo uma melhor experiência, visto que os seus pedidos serão atendidos mais rápidos

- Tempo de Resposta → Interactive Systems, deve ser minimizado e o número de processos com um Tempo de Resposta aceitável deve ser maximizado
- Tempo de Resposta → Intervalo de tempo entre a submissão de um pedido e o início da produção de uma resposta
- Em batch systems esta métrica não é tão relevante pois a interação com utilizadores é mínima ou inexistente.
- Em interactive systems é fundamental reduzir o tempo de resposta para os utilizadores percecionarem um atendimento aos seus pedidos de forma rápida e eficaz. No entanto é importante encontrar um balanço entre a redução do tempo de resposta e o aumento do número de processos com um tempo de resposta aceitável, para o caso em que múltiplos utilizadores estejam a interagir com o sistemas, alguns utilizadores tenham uma excelente experiência e outros terem uma má experiência

b) Resolvido em Papel

c)
Tempo de Espera → Tempo Total gasto no estado Ready:

Tempo de Espera do P1 = 13 - 0,4 = 12,6
Tempo de Espera do P2 = 0
Tempo de Espera do P3 = 9

Tempo de Turnaround → corresponde ao intervalo de tempo desde que o processo é submetido até à sua conclusão

Tempo de Turnaround do P1 = 22
Tempo de Turnaround do P2 = 12
Tempo de Turnaround do P3 = 16

2. a)
Quando queremos alocar um processo na memória e estão várias regiões disponíveis podemos usar diversas políticas como:
- First-Fit: Percorre a lista de regiões livres e aloca o processo na primeira região encontrada com espaço suficiente para o processo.
- Next-Fit: Igual ao First-Fit mas começa a percorrer a lista no momento em que parou na última pesquisa.
- Best-Fit: Percorre a lista de regiões livres toda, e aloca o processo na região com o mínimo espaço possível suficiente para o processo.
-Worst-Fit: Percorre a lista de regiões livres toda, e aloca o processo na região com o maior espaço.

b)

Lista das Regiões Ocupadas:
L1: A | 20000 | 2000 | L2
L2: B | 32000 | 2000 | L3
L3: C | 34000 | 4000 | L4
L4: D | 46000 | 6000 | null

Lista das Regiões Livres:
L5: - | 22000 | 10000 | L6
L6: - | 38000 | 8000 | L7
L7: - | 52000 | 28000 | null

i) First-Fit

Lista das Regiões Ocupadas:
L1: A | 20000 | 2000 | L8
L8: E | 22000 | 6000 | L2
L2: B | 32000 | 2000 | L3
L3: C | 34000 | 4000 | L4
L4: D | 46000 | 6000 | null

Lista das Regiões Livres:
L5: - | 28000 | 4000 | L6
L6: - | 38000 | 8000 | L7
L7: - | 52000 | 28000 | null

ii)
Best-Fit

Lista das Regiões Ocupadas:
L1: A | 20000 | 2000 | L2
L2: B | 32000 | 2000 | L3
L3: C | 34000 | 4000 | L8
L8: E | 38000 | 6000 | L4
L4: D | 46000 | 6000 | null

Lista das Regiões Livres:
L5: - | 22000 | 10000 | L6
L6: - | 44000 | 2000 | L7
L7: - | 52000 | 28000 | null

c) Mais do Mesmo

3) a)
Recursos podem ser:
1. preemtable → Quando o recurso pode ser temporariamente suspenso ou interrompido (Processador quando um processo de maior prioridade chega)
2. non-preemtalbe → Quando o recurso não pode ser suspenso ou interrompido antes da sua conclusão natural, ou seja, só termina quando o processo concluir a tarefa pretendida (Acesso à zona de exclusão mútua)

Deadlock Prevention:
Deadlock ocorre quando estas 4 condições são verdade:
- Mutual-Exclusion → apenas um processo pode utilizar um recurso de cada vez
- Hold and Wait → Um processo retém recursos na sua posse enquanto espera por outros recursos se tornem disponíveis
- No preemption → Recursos são non-preemptable, ou seja, não podem ser libertados por outros processos (não podem ser suspensos ou interrompidos por outros processos)
- Circular Wait → Um conjunto de processos no estado wait precisam de existir, de modo a que cada processo esteja a pedir recursos que estejam a ser retidos por outros processos neste conjunto

No deadlock prevention o foco é prevenir que as 4 condições sejam verdade, ou seja tornar pelo menos 1 delas falsa (não ocorrendo por isso situações de deadlock)

Que condições devem ser negadas:
1. (Complicado) Negar a condição de “mutual exclusion” só é possível se os recursos poderem ser partilhados simultaneamente, caso contrário, race conditions podem ocorrer.
2. (Complicado) Negar “non preemption” só é possível se os recursos forem preemtable o que é raro.
3. (Mais Fácil) Negar Hold-Wait pode ser feito
    - Caso um processo faça o pedido de todos os recursos que necessita de um só vez. Está propenso a starvation por isso um aging mechanism deve ser usado
    - Caso um processo liberte os recursos adquiridos quando falha em obter o próximo recurso, podendo tentar a aquisição mais tarde. Está propenso a busy wait e starvation. Para combater o busy wait o processo deve ser bloqueado e acordado só quando o recurso for libertado. Para combater a starvation deve ser usado um aging mechanism
4. (Mais Fácil) Atribuir um id a todos os recursos e forçar que a aquisição dos mesmos seja feita por ordem crescente ou descendente (Pode resultar em starvation)

Deadlock Avoidance:
Menos restritiva em comparação com Deadlock Prevention.
Existe um resource manager que gere a forma como os recursos são distribuídos.
Requer conhecimento prévio do tamanho máximo dos pedidos de recursos feitos pelos processos

Negar a iniciação de um processo caso os seus requisitos em termos de recursos possam levar a um deadlock. O sistema só permite a iniciação de um processo caso consiga garantir a sua finalização.
Negar a alocação de recursos (Banker's Algorithm) a um processo caso este possa provocar de imediato ou em futuras alocações uma situação de deadlock

Deadlock Detection:
O sistema permite que situações de deadlock possam ocorrer
Quando situações de deadlock ocorrem devem ser detectadas e iniciado um procedimento para recuperar do deadlock
O procedimento pode consistir em:

1. Libertar Recursos de um Processo, suspendendo o mesmo
2. Rollback, caso se esteja a guardar o estado de execução dos processos pode libertar recursos temporariamente de um processo e retornar mais tarde ao estado de execução do processo que libertou os recursos
3. Kill Processes

c) Como abordar o problema:
1. Vamos usar Deadlock Prevention ou seja 1 das quatro condições não pode ser verificada
2. Qual das condições deve ser negada?
- Mutual Exclusion → (Não) Os garfos teriam de poder ser partilhados ao mesmo tempo ou um filósofo poderia ter a capacidade de tirar o garfo a outro filósofo, o que não é possível
- Hold and Wait → (Sim mas é má) Cada filósofo pedia os recursos todos de uma vez (agarrava os 2 garfos ao mesmo tempo). Poderia ser implementado mas leva a Starvation
→ (Não) Caso um filósofo não conseguisse adquirir um recurso ele liberta os recursos já adquiridos (Filósofo liberta o garfo esquerdo quando não consegue adquirir o direito). Pode levar a busy wait e starvation.
- No preemption → (Não) Um filósofo teria de ser capaz de interromper outro processo (filósofo), ou seja, um filósofo poderia retirar garfos a outro filósofo
- Circular Wait → (Sim) É atribuído um ID, desde 0 até N-1 a cada filósofo. É atribuído um ID a cada garfo um ID correspondente ao ID do filósofo à sua esquerda. Todos os filósofos pegam no garfo com o menor ID. Isto vai provocar que todos os filósofos desde 0 até N-2 peguem no garfo à sua direita e o filósofo com o ID N-1 vai pegar no garfo à sua esquerda. Deve ser imposto uma ordem pela qual os recursos são adquiridos (ascendente ou descendente). Isto irá provocar uma quebra no ciclo permitindo que todos os filósofos acabem por comer. (Desenhar círculo com os ids dos filósofos e dos garfos para perceber)

Pseudo-Código Resolução negando Circular Wait:

think()
down(fork[min(id, (id+1) % N)]) → Agarra o garfo com o menor id
down(fork[max(id, (id+1) % N)]) → Agarra o garfo com o maior id
eat()
up(fork[min(id, (id+1) % N)])
up(fork[max(id, (id+1) % N)])

Vai provocar uma quebra no ciclo e permitir que o filósofo N-2 consiga comer, levando a que os próximos filósofos consigam comer também, eliminando por isso situações de deadlock. Existem mais implementações possíveis negando a condição do hold-wait em vez do circular wait.
Para uma explicação visual: https://www.youtube.com/watch?v=_ruovgwXyYs

4)
a)
As threads são consideradas como lightweight processes porque:

1. Várias threads independentes podem coexistir dentro de o mesmo processo.
2. Podem ser facilmente criadas e terminadas dentro de um processo.
3 Threads, geralmente estão associadas à execução de uma só função.
4. A comunicação entre threads pode ser feita através da estrutura de dados do processo, que é global do ponto de vista das threads, facilitando assim a comunicação entre as mesmas, por exemplo, através do uso de variáveis partilhadas.
5. As threads que coexistem dentro do mesmo processo partilham o espaço de endereçamento desse processo.

b)

Processo Pai → 1 à 5 da 14 à 21
Processo Filho → 7 à 12
Ambos → 6

O valor do Fork para o processo Pai é o PID do processo Filho
O valor do Fork para o processo Filho é 0

c)
01234
01324
01342
03124
03142

03412

Processo Filho 1 pode imprimir → 1 e 2
Processo Filho 2 pode imprimir → 3 e 4
O 2 vem sempre depois do 1
O 4 vem sempre depois do 2

Exame 2016-2017

Não vou fazer os exercícios que são repetidos!

3.

b) A mensagem 4 é sempre a última. O processo pai tem de esperar sempre pela finalização do processo filho.

c)

Ready → Processo está à espera mas reúne todas as condições necessárias para ser executado pelo processador,
Run → Processo está a ser executado pelo processador
Blocked → Processo encontra-se bloqueado pois está à espera da ocorrência de um evento externo

Dispatch → Processo passa do estado ready para o estado run
Time Out → Programa chegou ao final do seu time quantum, transita do estado run para o estado ready
Preempt → Processo transita do estado run para o estado ready, pois um processo de maior prioridade apodera-se do processador
Event Wait → Um processo está à espera da ocorrência de um evento externo. Transita do estado run para o estado blocked
Event Occurs → O evento externo pelo qual o processo estava à espera. O processo transita do estado blocked para o estado ready

d)
As operações de I/O são bloqueantes ou seja sempre que um processo está à espera de um input de um utilizador ou pretende realizar uma operação de output, o processo entra no estado Blocked.

Caso não consideremos o fprintf como uma operação bloqueante o processo pai pode mesmo assim entrar num estado bloqueante, quando executa a instrução wait (espera que o processo filho termine a sua execução).
O processo pai entra no estado Blocked pois está à espera da ocorrência de um evento externo, a finalização do processo filho. Até esta não acontecer, o processo permanecerá nesse estado pois não reúne todas as condições para poder ser executado pelo processador (estado ready)

O processo pai pode até entrar no estado Suspended-Blocked, sendo realizado o swapped out para a memória secundária para que o mesmo não esteja a ocupar recursos desnecessários do CPU, permitindo assim, a execução de outros processos.

4.
a) Resposta no Papel
b) Resposta no Papel → https://www.youtube.com/watch?v=4wVp97-uqr0

c)
Num sistema de memória virtual paginada é necessário encontrar a melhor política para realizar a substituição de páginas.
Existem diversas políticas que podemos utilizar:

(Pequeno Resumo sobre políticas de substituição de Páginas)

LRU (Least Recently Used):
1. Seleciona a frame que não foi referenciada à mais tempo.
2. Alto custo de implementação e não é muito eficiente

FIFO (First In First Out):
1. Seleciona a frame com a página mais antiga
2. Simples de implementar mas pode não ser a melhor política a utilizar visto que não tem em conta as páginas que foram modificadas ou referenciadas.

NRU (Not Recently Used):
1. De modo a que o Sistema Operativo escolha a melhor página para ser substituída, atribuímos uma classe a cada página.
Existem 4 classes às quais as páginas podem pertencer

Classe 0: Pages → não Referenciadas (REF) e não Modificadas (MOD)
Classe 1: Pages → não Referenciadas (REF) e sim Modificadas (MOD)
Classe 2: Pages → sim Referenciadas (REF) e não Modificadas (MOD)
Classe 3: Pages → sim Referenciadas (REF) e sim Modificadas (MOD)

class	Ref	Mod
0	0	0
1	0	1
2	1	0
3	1	1

O algoritmo é simples de implementar e tem um baixo overhead.

O algoritmo dita que é preferível substituir uma página modificada que não é referenciada à mais tempo.

Sempre que há um page fault o algoritmo escolhe de forma aleatória uma das páginas que se encontram na classe mais baixa não vazia, ou seja, a classe mais baixa que contém páginas. Após essa seleção o Sistema Operativo substituía pela nova página que pretende inserir.

2. Menos exigente do que a política LRU (Least Recently Used) e é relativamente eficiente.

5.
a) Não Sei

b) Não Sei

c) Não Sei

Exame 2017-2018

Não vou fazer os exercícios que são repetidos!

1.
a)
Semáforos Binários
Podem ser inicializados com o valor 0 ou 1
sem1: 0
sem2: 1
sem3: 1
OU
sem1: 1
sem2: 1
sem3: 0

b)
Signal e Wait → Threads
Up e Down → Semáforos

Se o sinal de Signal é emitido e não há nenhuma thread no estado wait, o sinal de Signal é perdido.
Se o Up ocorrer, este não se perde. O valor do semáforo é simplesmente incrementado e fica à espera do Down.
O Wait fica sempre à espera de um Signal futuro, pois qualquer Signal que tenho ocorrido antes da thread entrar no estado wait é simplesmente perdido
O Down, mais uma vez, simplesmente decrementa o valor do semáforo.

c)

```
/* Processo/Thread 1*/
for (int i = 0; i < 3; i++)
{
	mutex_lock(&mutex);
	while(!cond1) // Enquanto a condição 1 não se verificar
	{
		wait vcond1, mutex; // Thread 1 fica no estado wait
	 }
	cond1 = false;
	printf(“A”); fflush (stdout);

	cond3 = true;
	signal vcond3;
	mutex_unlock(&mutex);
}

/* Processo/Thread 2*/
for (int i = 0; i < 3; i++)
{
	mutex_lock(&mutex);
	while(!cond2) // Enquanto a condição 2 não se verificar
	{
		wait vcond2, mutex; // Thread 2 fica no estado wait
	 }
	cond2 = false;

	printf(“B”); fflush (stdout);
	mutex_unlock(&mutex);
}

/* Processo/Thread 3*/
for (int i = 0; i < 3; i++)
{
	mutex_lock(&mutex);
	while(!cond3) // Enquanto a condição 3 não se verificar
	{
		wait vcond3, mutex; // Thread 1 fica no estado 1
	 }
	printf(“C”); fflush (stdout);
	cond3 = false;

	cond1 = true;
	signal vcond1;
	cond2 = true;
	signal vcond2;
```

```
        mutex_unlock(&mutex);
}
```

3.
a)
Arquitetura com organização de memória real O processo está completamente em memória ou na área de swap. Caso o processo esteja em memória este tem de ser armazenado de forma contígua.
A alocação contígua de memória leva a certas consequências:
1. É impossível executar um programa cujo o seu address space seja maior que a memória
2. Swapping Area serve como uma extensão da memória principal quando há falta de espaço um processo pode ser totalmente transferido para lá.

Arquitetura com organização de memória virtual O espaço de endereçamento do processo vai estar dividido em páginas, frames, blocos, segmentos… Estes vão se encontrar distribuídos, pela memória, não necessariamente de forma contígua. Um processo pode estar distribuído pela Swapping Area e pela memória, logo para executar um processo este pode estar parcialmente na memória e na Swapping Area
1. É possível executar um programa cujo o address space é superior ao tamanho da memória
2. Swapping Area serve como uma extensão da memória.

Partições Fixas:
O sistema operativo começa por dividir a memória em:
1. Slices Iguais
2. Slices de diferentes tamanhos

O espaço de endereçamento lógico de um processo é alocado numa Slice de igual ou maior tamanho.
Simples de Implementar
Eficiente → Overhead Reduzido
Existe um número fixo de processos que podem estar na memória. Esse número irá corresponder ao número de Slices. Ou seja 1 Slice = MAX 1 Processo
Fragmentação → A parte da Slice que não foi usada pelo processo é desperdiçada

Partições Variáveis (Dinâmicas):
Toda a memória disponível, inicialmente, é vista como uma partição.
Quando um processo precisa de ser alocado reservamos uma região de memória com espaço suficiente.
Quando o processo termina essa região é libertada
É necessário ter duas listas:
1. Lista de Regiões Livres
2. Lista de Regiões Ocupadas

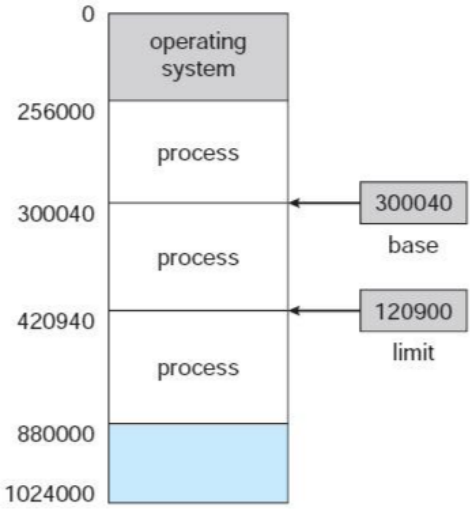
Simples de Implementar
Partições Dinâmicas podem levar a External Fragmentation → O espaço livre é dividido em muitas regiões de memória, obviamente não contíguas, não sendo capazes de alocar

processos. Em certos casos a memória total livre disponível pode ser suficiente para alocar o processo, mas como não é contíguo esse processo não pode ser alocado.
Existe uma forma de combater esta situação, chamada Garbage Collection → Compacta todo o espaço livre, ou seja, pega em todas as regiões livres e transforma-as numa só
Infelizmente a operação Garbage Collection envolve que todos os processos parem.
É também muito complicado desenvolver algoritmos que seja muito eficientes em alocação e libertação de espaço.

b)
i e iii)
Registo Base → Contém o endereço do começo da região de memória física onde o processo vai ser alocado.
(Contexto) O registo base é usado como um rellocation register, ou seja, é adicionado ao endereço lógico produzido pelo CPU transformando-o assim num endereço físico.
Geralmente os endereços lógicos são começados a ser gerados a partir do endereço 0. Os endereços físicos, apesar de começarem também no registo 0, geralmente desde o endereço 0 até ao endereço X, eles estão reservados para o Sistema Operativo, sendo por isso, necessário uma conversão entre endereços lógicos e físicos.
Limit Register → Contém o tamanho do espaço de endereçamento lógico.
(Contexto) Caso o endereço lógico produzido pelo CPU seja maior que o Limit Register, significa que ao fazer a conversão de um endereço lógico para um endereço físico este seria inválido, pois o endereço físico poderia não existir ou aceder a um endereço não válido da memória (exemplo: Endereço do Sistema Operativo).

Nota: Ambos estes endereços são dinâmicos e podem variar consoante o processo a alocar

ii)
Os registos limit register e base register são alterados na função dispatch (processo transita do estado ready para o estado run).
O sistema operativo acede à tabela de processos (PCT → Process Control Table) e transfere para MMU o base o limit register



c) Já fiz num exame anterior. Basicamente é só apresentar a lista de regiões ocupadas e a lista de regiões livres.

4)
a)
Deadlock Prevention → É da responsabilidade dos processos prevenirem a existência de deadlock
Deadlock Avoidance → (Resposta)É da responsabilidade do gestor de recursos prevenir o deadlock

b)

Safe State → Um recurso disponível só é alocado a um processo caso haja, pelo menos uma sequência de futuras alocações que não resultem em deadlock

Unsafe State → Pode não estar em deadlock, mas vai resultar num deadlock em futuras alocações

b) e c) ver a explicação do sor → https://www.youtube.com/watch?v=_EKqQCNYVvs está super bem

Exame 2022-2023

Não vou fazer os exercícios que são repetidos!

Secção Gestão de Memória

1. Função dos base Register P e PT
Se alguém souber meta no discord !!!

2. Explicar o processo de como obter o endereço físico num sistema de memória virtual.
Se alguém souber meta no discord !!!