# Btw o que está a azul não são respostas obviamente

# Teste 2014-15

Parte A)

1)

A) e b) Pergunta cringe nao vai sair trust

c) also cringe, like what, its asking what a interaction environment does oh well also pretty sure we didn’t talk about this

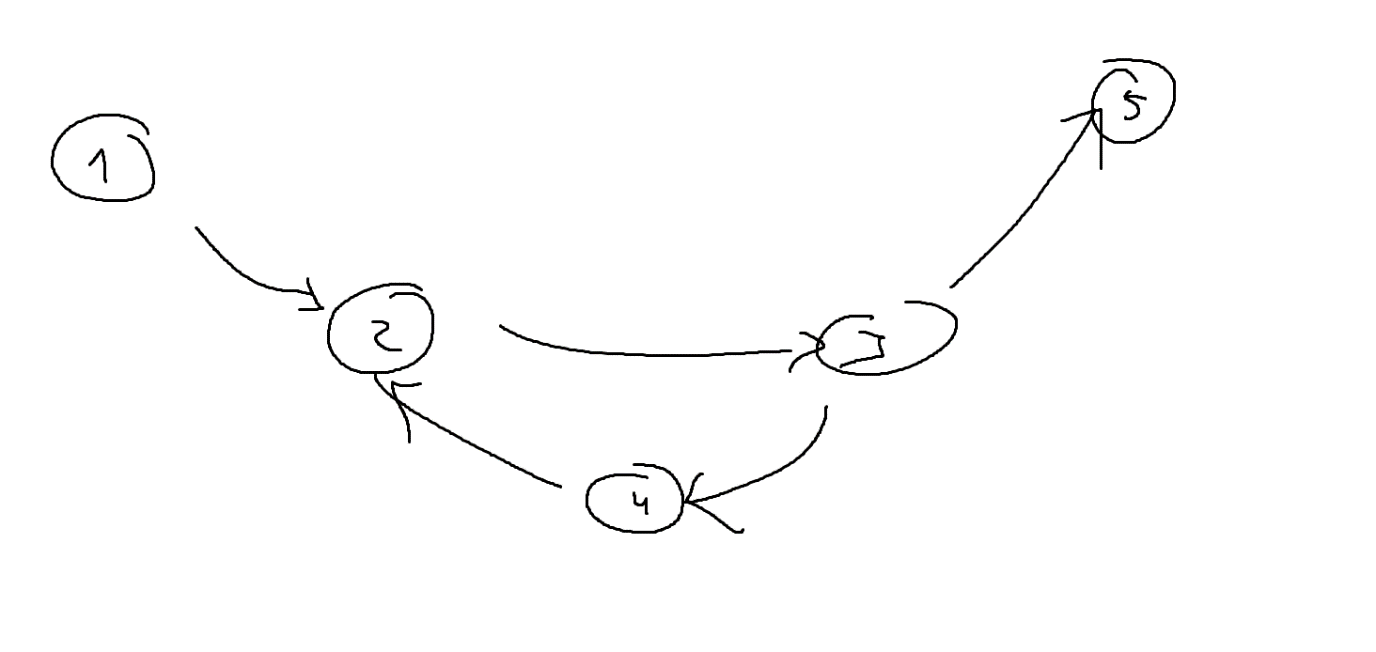
O ambiente de interacção com o utilizador tal como o nome sugere é o um ambiente que permite o utilizador interagir com o sistema operativo com ajuda de programas.

2)

Um programa é um conjunto de instruções que têm de ser realizadas para completar algum objetivo, um processo é uma instância da execução de um programa.

3)

a) a política de scheduling é classificada como Preemptive scheduling quando age sobre um processo que transaciona de um estado “running” ou “waiting” para um estado “ready”, non-preemptive é quando age sobre um processo que transaciona do estado “running” para “waiting” ou quando o processo termina durante “running”.

b) bro que teste cringe

1 -> new, o processo é criado

2 -> ready, o processo está pronto para ser corrido pelo cpu

3 -> running, o processo está a ser corrido pelo cpu

4 -> waiting, o processo está á espera de I/O ou wait()

5 -> terminated, o processo terminou (erro ou exit normal)

c) FCFS -> First come first served, é uma politica de prioridade estática, onde a ordem de chegada é a ordem de processamento, e SRTF -> shortest remaining time first onde a ordem de processamento depende do que precisa menos tempo para ser processado naquele momento, isto é, o processo pode ser trocado a meio caso outro que necessite de menos tempo apareça.

4)

a) SSD, HDD e usb pens? I dont fucking know, disquetes prontos.

b) Armazenar informação que se encontra na memória real caso esta se encontra cheia, normalmente armazena memória de programas que não estão a ser processados para libertar memória para outros.

c) O sistema de ficheiros é a estrutura e os métodos usados para organizar toda a informação de ficheiros que são armazenados no sistema operativo, sem um seria impossível distinguir onde um programa acaba e outro começa, existem vários tipos como nfts, ext4, fat32.

Parte B

1)

A região partilhada são todas as variáveis com a keyword “shared”, ou seja, o buffer de comunicação, “buff”, o valor de “access”, o valor de “bFull” e o valor “bEmpty”. A região crítica seriam as funções insertValOnCommBuff(&buff, val); no produto e retrieveValFromCommBuff(&buff, &val); no consumidor pois ambas estão a aceder á região partilhada.

2)

bFull bloqueia os produtores quando o buffer de comunicação está cheio, ou seja já existem K elementos no buffer, o bEmpty bloqueia os consumidores quando o buffer de comunicação está vazio, pois estes não seria capazes de retirar um valor.

3)

Na inicialização o val de bFull estará inicializado ao valor K, número de elementos que o buffer pode ter, e o val de bEmpty estará inicializado a 0.

4)

InsertValOnCommBuff:

semDown(bFull);

semDown(access);

putVal (buff, val);

semUp(access);

semUp(bEmpty);

return;

retrieveValFromCommBuff:

semDown(bEmpty);

semDown(access);

getVal (buff, val);

semUp(access);

semUp(bFull);

return;

# Teste 2016-2017

1

A)

O filho executa as linhas de códigos 3, 4, 5, 10 e o pai executa as linhas de código 3, 6, 7, 8, 10.

B)

ABC ou BAC são possíveis.

C)

Um processo é uma instância de um programa que está a ser corrido, enquanto um thread faz parte de um processo, o que significa que um processo pode ter vários threads. Um processo tem a abilidade de criar processos filhos mas todos os processos são isolados, isto é não partilham memória, ao contrário de vários threads de um processo que partilham a memória.

D)

Mesma merda que está em cima na 3b 2014-15.

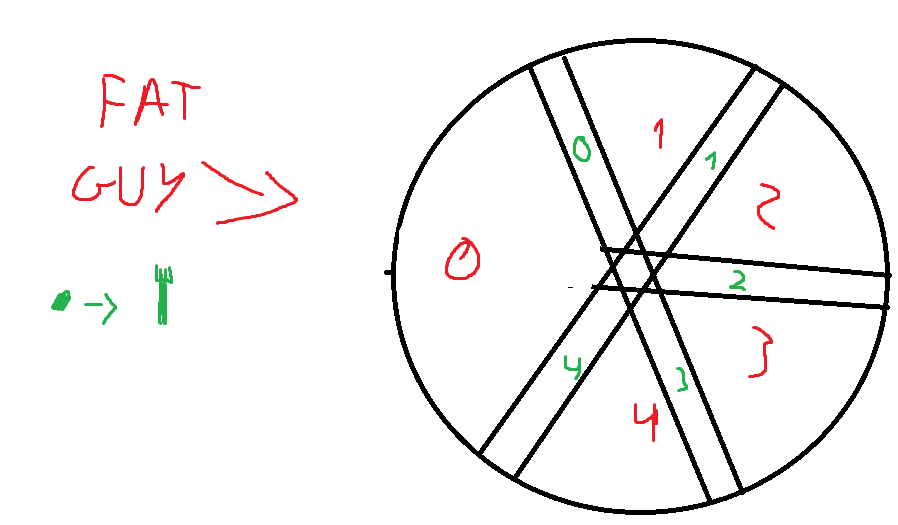
e)

3 -> running, 6 -> running/waiting, 7 -> waiting, 8 -> ready/running/waiting, 10 -> terminated.

Not sure about this one

2)

A) isto aparece nos videos but just draw a diagram and its easy

´

Left(int f){

Return f;

}

Right(int f){

Return (f+4)%5;

}

b)

Todos os filosofos podem pegar o garfo á sua esquerda, isto cria um deadlock quando vão pegar no garfo á sua direita, pois já não estão disponíveis.

c)

void filosofo(int f) {

while(true) {

think();

//getForks

If(f == 4) {

forks[right(f)].down();

forks[left(f)].down();

} else {

forks[left(f)].down();

forks[right(f)].down();

}

eat();

//putForks()

forks[left(f)].up();

forks[right(f)].up();

}

}

// explicação -> adicionando assimetria na solução resolve o deadlock, neste caso o filosofo quatro vai começar por tentar apanhar o garfo a direita isto origina três outcomes, se falhar quer dizer que o zero já apanhou o garfo da esquerda e prende o quarto filosofo no garfo da direita, isto permite que o filosofo 3 apanhe o garfo a direita dele e comece a comer, caso o filosofo quatro apanhe o garfo á direita, ele tenta de seguida apanhar o da esquerda, se falhar quer dizer que o filosofo três já apanhou o garfo da direita e já está a comer, se conseguir o garfo da esquerda, o filosofo quatro começa a comer. Não é uma solução ideal porque só haverá situações onde só há um filosofo a comer mas sem recursos de sincronização é o melhor que se pode fazer acho eu.

3)

a)

É possível, caso o P2 se apoderar dos dois R2 e 1 dos R3, ele pode terminar libertando assim 2 R1, 3 R2 e 1 R3, de seguida P1 teria de apoderar-se de 2 R1 para poder terminar e libertar 5 R1, 1 R2 e 4 R3, por fim P3 poderia apoderar-se do que necessitava, 3R1 e 1 R2 para terminar.

b)

Caso isto aconteça, um deadlock é guarantido, porque não existe recursos suficientes para acabar um dos processos para libertar recursos, a tabela de necessidade e disponibilidade é a seguinte

|  |  |  |
| --- | --- | --- |
| R1 | R2 | R3 |
| 0 | 1 | 3 |

Disponibilidade

|  |  |  |  |
| --- | --- | --- | --- |
|  | R1 | R2 | R3 |
| P1 | 2 | 0 | 0 |
| P2 | 0 | 2 | 1 |
| P3 | 3 | 0 | 0 |

Necessidade

void print\_avail\_resources(void){

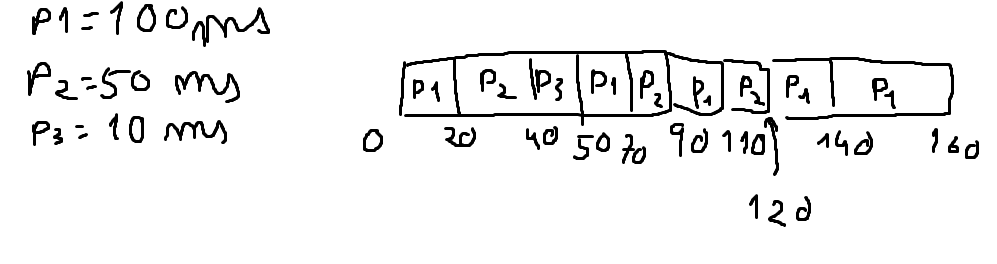
semDown(s\_mutex); // secalhar é s\_mutex.down();

printf(“%d R1 disponíveis; %d R2 disponíveis; %d R3 disponíveis;\n”, info.avail\_r1, info.avail\_r2, info.avail\_r3);

semUp(s\_mutex); // secalhar é s\_mutex.up();

4)

a)



b)

((160-100)+(120-50)+(50-10))/3 = 56.67 ms

c)

A vantagem de FCSF é que é fácil de implementar, a desvantagem é que processos que em outros casos demorariam pouco tempo, podem ter o seu tempo de espera aumentado simplesmente porque chegaram mais tarde do que um processo que vai demorar muito tempo a acabar, entretanto o Round Robin exibe um grau de igualdade independentemente do processo e apresenta um tempo médio de espera menor q ue FCSF, a desvantagem é que a sua eficiência é determinada pelo o valor do time quantum, caso este seja demasiado grande, o algoritmo seria igual a FCSF, caso seja muito pequeno, a sobrecarga criada será maior, logo é necessário determinar o time quantum óptimo, por outro lado o Round Robin também sofre em termos de performance caso os processos tenham tempos de execução iguais.

O round robin é melhor para aplicações interactivas pois guarante que nenhuma das aplicações fique irresponsivo, pois dá prioridade igual a todos os processos, enquanto o FCFS é mais indicado para aplicações de cálculo intensivos para evitar que operação seja parada a meio e que a memória física esteja sempre a ler e a escreve na memoria virtual (processo lento).

5

1 byte = 8 bits

a)

GB

8 bits por pagina, 1 bit para cada endereço, logo são duas paginas i think not sure.

b)

Não, o processo não pode alterar o conteúdo da tabela de página, pois caso fosse possível permitiria que qualquer processo fosse capaz de aceder a qualquer região de memória, que seria um risco de segurança.

c)

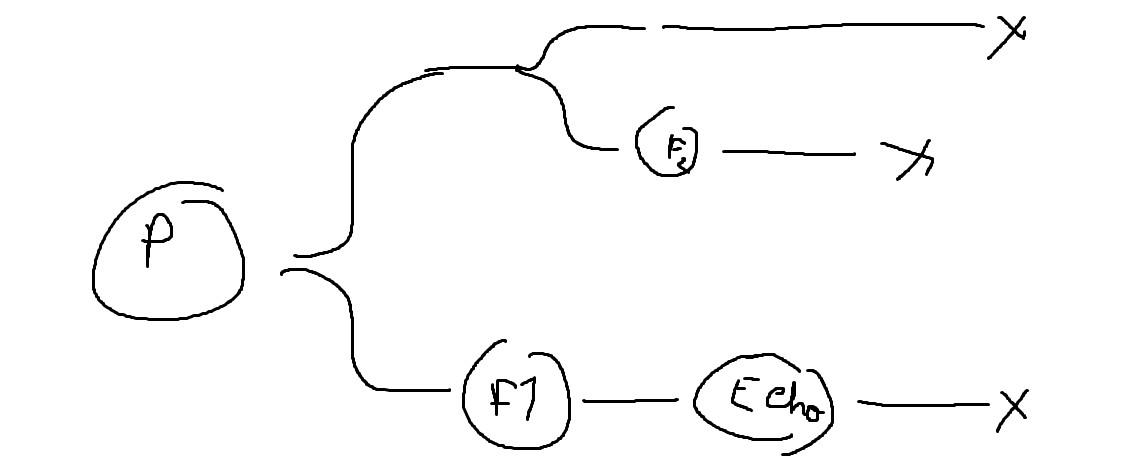
Libertar memória física que não tem sido usada recentemente, isto é feito por copiar a informação para a memória virtual, também oferece um grau de segurança pois os endereços usados pelos processos não são endereços da memória física e permite que vários processos acedem à mesma zona de memória.

# Teste 2018-2019

1)

a)

4 Processos, o processo inicial, o primeiro filho, o segundo filho, e o processo “Echo”.



b)

Pai – 6,7,8,9,10; F1 – 7,11,12,13,14; F2 – 8,9,10.

c)

ABBCCD (echo), AD (echo) BCBC // pergunta estupida, ainda há mais 3 ou 4 variações

d)

Não sei estou cansado, tem haver com os estados de running, ready e escalonamento.

e)

O fork() cria um novo processo, com uma cópia do código original, e retorna 0 para a cópia ( o filho ) e o pid do filho no pai, o execlp substitui o processo atual pelo processo especificado nos argumentos, no caso anterior o filho 1 foi substituído pelo o processo do echo.

f) ???????????????????????????????????????

2)

Três das quatro threads irão executar o código e irão ficar á espera na linha seis, a este ponto o count está com o valor 3, quando o quarto thread correr o código o count irá incrementar e a comparação na linha 4 será verdadeira logo a barrier.up será executada, após uma das threads irá sair da linha 6 e dará barrier.up deixando outro process que está na linha 6 passar e assim sucessivamente até todas as threads passarem a linha 7. O valor do semáforo barrier no final será 1 porque recebeu 5 Ups() e 4 downs().

b)

A barreira não irá funcionar após a primeira passagem, porque a barreira irá acabar com o valor de 1, que permite que qualquer processo passe a linha 6 e incremente a barreira na linha 7, assim nenhum dos processos ficará sincronizado na linha 6.

c)

codebefore()

mutex.down()

count++;

if count == n{

for i=1..n{

barrier[i].up()

count = 0;

mutex.up()

Barrier[J].down()

Codeafter()

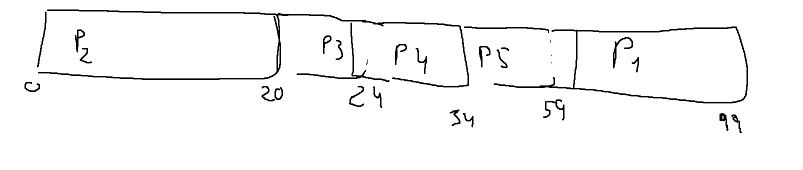
Este código seria ligeiramente diferente em cada processo pois onde está J no Barrier[J].down teria que ser um valor de 1 a n que identificaria o processo no array de semáforos de barreira.

d)

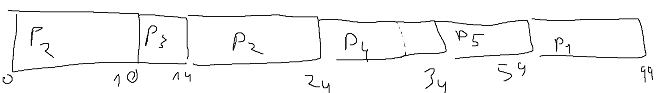
A diferença principal é que o Wait e Signal não têm memória associada, devido a isto o Signal pode-se perder caso não exista um processo a fazer Wait, isto não acontece nos semáforos devido a memória associada, logo um Up() irá ter sempre o efeito de incrementar o semáforo.

3) ????????????????????????????????

4)

a)  
sem preempção 

com preempção



b)

Este algoritmo é ótimo do ponto de vista do tempo médio de espera de um conjunto de processos, e garante sempre que os processo com menos sobrecarga são realizados primeiro.

5)

a)

GA

b)

I think it’s the same as 2016-2017 5) b or at least is similar, like blah blah cant edit table, so it cant access other regions, also process cant write to virtual memory only when it goes idle does the physical memory write the information to virtual memory.

c)

Yes.

d)

It’s the same as 2016-2017 5) c