①

2,0 1- O processo existe S.O mas ainda não está pronto para execução, é feita as checagens no que diz respeito a espaço, se é um processo permitido, etc ✓

2- O S.O aloca um espaço de memória p/ o processo e ele ingressa na fila de READY na memória ✓

3- O processo sai da fila e para executar em modo kernel. O escalonador de curto-prazo estará agindo para selecionar o processo na memória e colocá-lo em estado RUNNING seguindo alguma política de escalonamento.

4. O processo é preemptado e volta p/ fila de READY por algum motivo (Fim da fatia de tempo, em detrimento de um processo com maior prioridade, etc). O escalonador de curto-prazo estará agindo aqui.

5. O processo sai de user-Running e é desviado p/ modo Kernel, onde terá todos privilégios de instruções e de acesso à memória (Ex. Uma SVC)...

6. O processo retorna ao modo usuário após ter sido tratada alguma operação que dependia de uma chamada ao sistema (Retorno de uma Syscall).

7. O processo passa de Kernel-Running para bloqueado, onde aguardará até que um evento solicitado ocorra. (Ex: Requisição I/O)

8. O evento que ele esperava ocorreu e ele volta p/ fila READY.

9. Situação em que a memória está cheia de processos BLOCKED e o escalonador de médio prazo usa o conceito de memória virtual p/ levar um processo da memória p/ o disco. O processo aguardará o evento em disco.

10. O processo bloqueado em disco ainda espera o evento mas em memória.

11. O evento de um processo bloqueado em disco ocorreu mas ele ainda não pode voltar p/ quem de READY. Ficará READY em disco.

13. O processo fica READY em memória. pq...

12. O processo fica READY em disco porque um

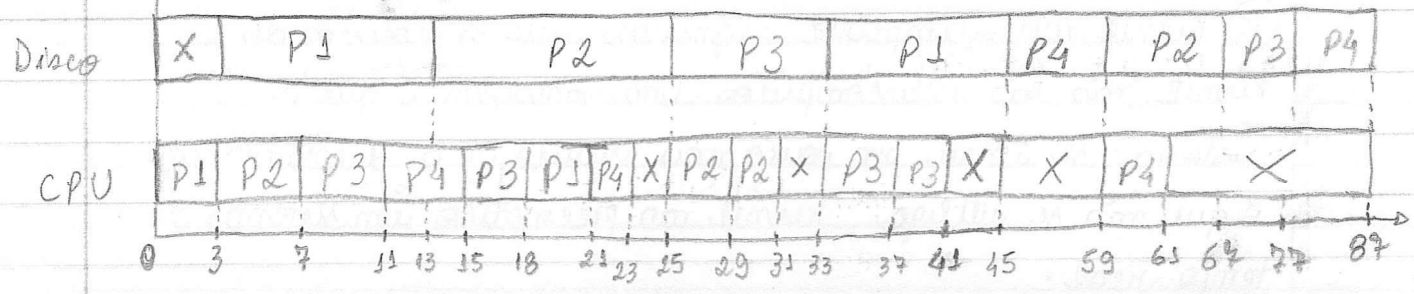
Leonardo F Menezes 20121013 93

processo com prioridade maior chegou e não tem espaço p/ ficar na fila de prontos da memória, por exemplo.

14- O processo passa de um estado de execução para ZOMBIE, onde deixará de existir, contudo, ainda terá registros no processo tabela por algum motivo (Ex.: Terminação normal, anormal, deadlock, etc) Geralmente ele deixa esse registro p/ que seu pai possa recolher.

2) Um Kernel não preemptivo significa que a execução de código de Kernel não pode ser interrompida. Por ser uma abordagem simples, dispensa que áreas da memória precisem ser protegidas contra acesso simultâneo (locks) em maquinas com apenas uma CPU. Entretanto, uma implementação pode gerar uma latência muito alta, diminuindo o tempo de resposta do sistema e consequentemente fazendo com que o sistema não seja capaz de executar processos de tempo real.

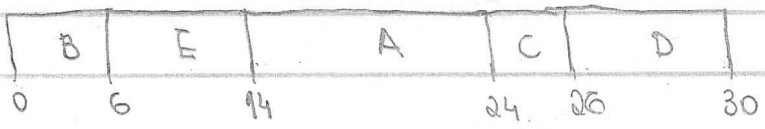
1,5 ③


Disco
X
P1
P2
P3
P1
P4
P2
P3
P4
CPU
P1
P2
P3
P4
P3
P1
P4
X
P2
P2
X
P3
P3
X
X
P4
X
0
3
7
11
13
15
18
21
23
25
29
31
33
37
41
45
59
61
64
77
87

2.0

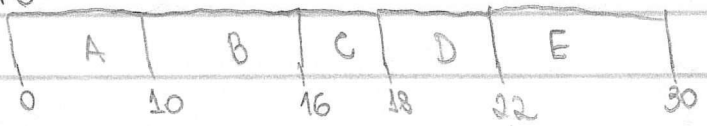
(4) 10, 6, 2, 4, 8 (tempo) — tempo de chegada em 0 para todos os processos.
3, 5, 2, 1, 4 (prioridade)

a) prioridade:



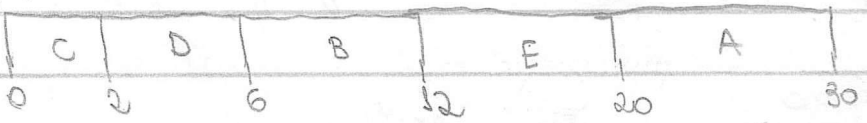
$$\text{turnaround médio} = \frac{\overset{(A)}{24} + \overset{(B)}{6} + \overset{(C)}{26} + \overset{(D)}{30} + \overset{(E)}{14}}{5} = 20 \text{ u}$$

b) FIFO:



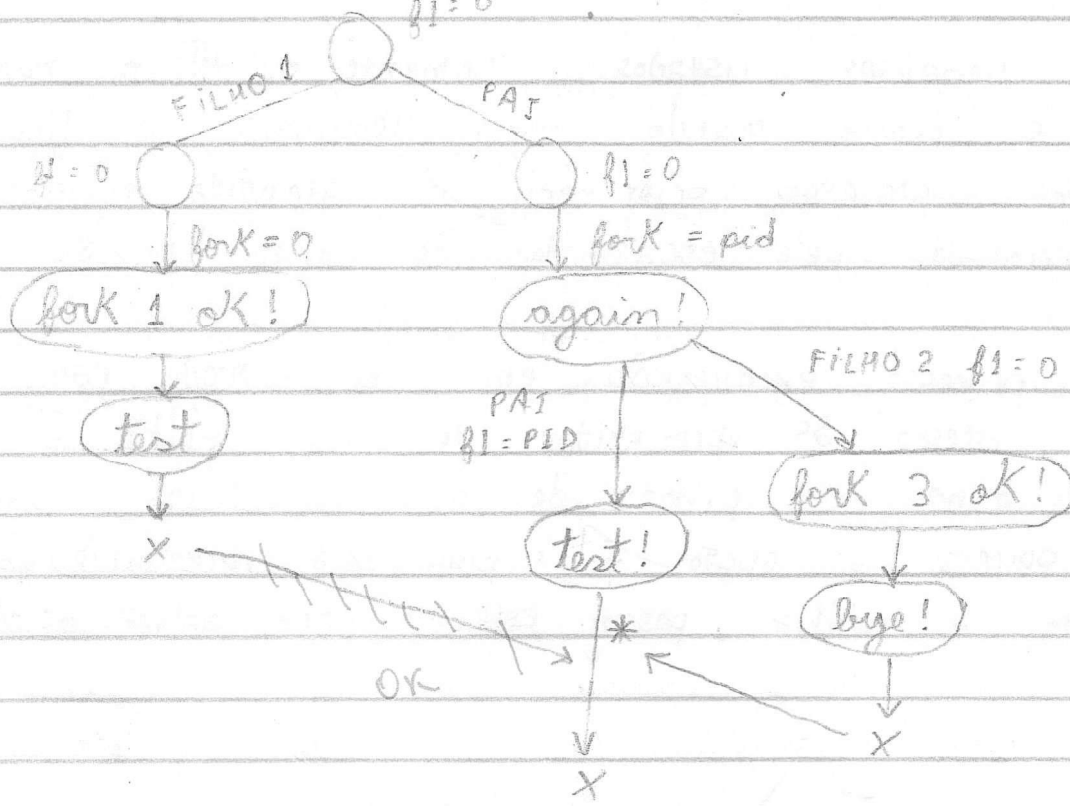
$$\text{turnaround médio} = \frac{\overset{(A)}{10} + \overset{(B)}{16} + \overset{(C)}{18} + \overset{(D)}{22} + \overset{(E)}{30}}{5} = 19,2 \text{ u}$$

c) SRTF:



$$\text{turnaround médio} = \frac{\overset{(A)}{30} + \overset{(B)}{12} + \overset{(C)}{2} + \overset{(D)}{6} + \overset{(E)}{20}}{5} = 14 \text{ u}$$

5)



* O pai vai sair do wait assim que um filho morrer, independente de qual for (filho 1 ou filho 2). Caso o outro filho ainda esteja executando, se tornará órfão e será adotado pelo init.

6) A) Apesar da I não estar completamente precisa se considerado o overhead gerado por um quantum tão menor que a duração média dos processos, a IV. está completamente errada uma vez que ela implicaria que com fatias de tempo suficientemente pequenas poderíamos obter velocidades de CPU burst arbitrariamente altas. Muito bom!

B) Dos impactos listados, somente o III é plausível e ocorre quando uma requisição de disco de um processo prioritário é atendida e este é selecionado pelo escalonador de curto prazo.

C) Um processo executando em user-mode não teria acesso às protegidas de sua pilha e muito menos às pilhas de outros processos, logo, está correta a opção I, pois será necessário garantir que a pilha possui espaço para salvar o contexto.