1) Multiprogramação é um conceito de desenvolvimento de sistemas Operacionais, no qual vários processos são executados "simultaneamente" em uma CPU. Com isso é possível otimizar o uso dos recursos do sistema e diminuir o tempo médio de resposta dos processos, uma vez que a maioria dos processos executam operações de entrada e saída, portanto para que a CPU não fique ociosa esperando o dispositivo solicitado responder, outros processo assume a posse da CPU,

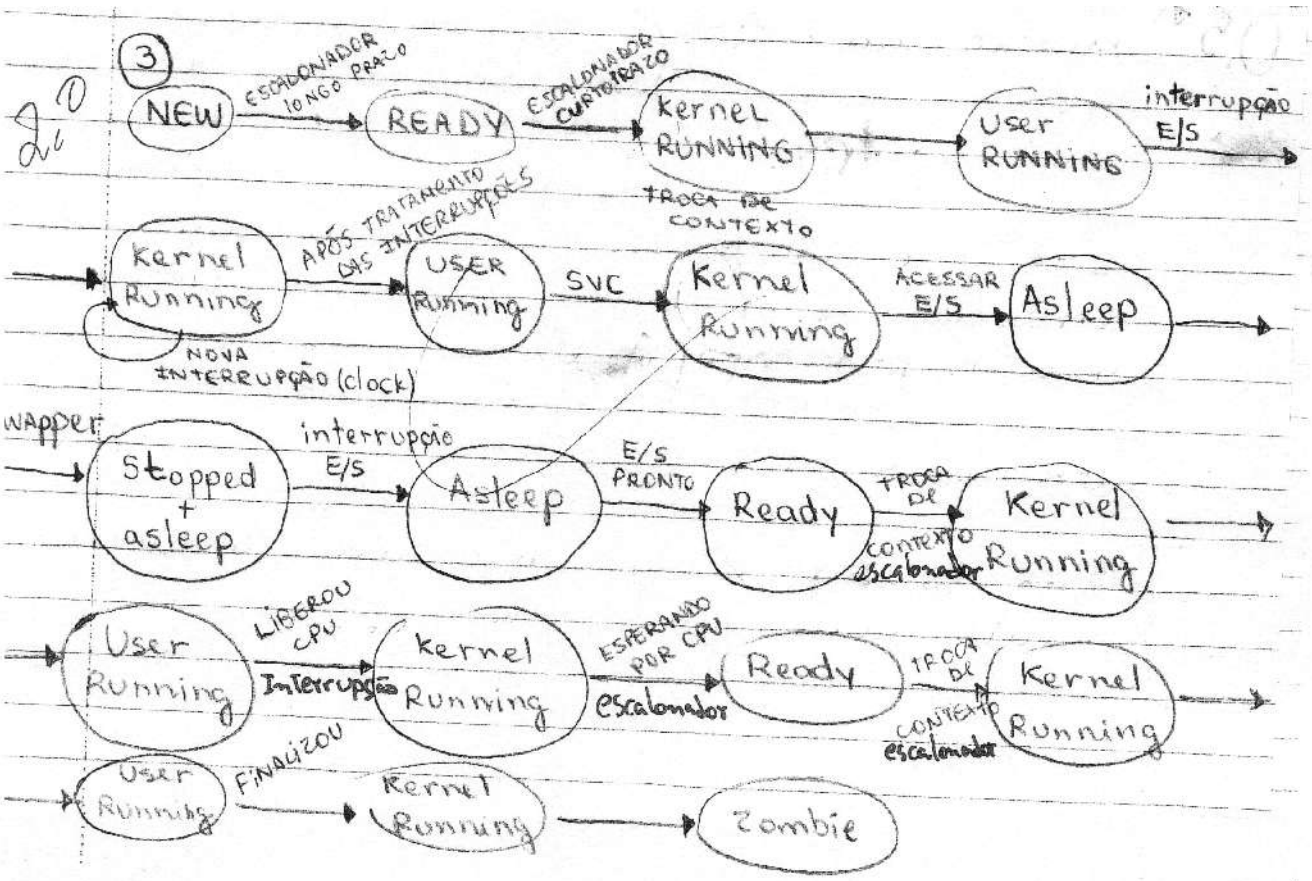
Para que essa troca seja viável é necessário implementar mecanismos de interrupção, onde o dispositivo que foi solicitado pela informar que está pronto para operar, e que o processo que ficou no aguardo deste dispositivo pode voltar a executar.

2)

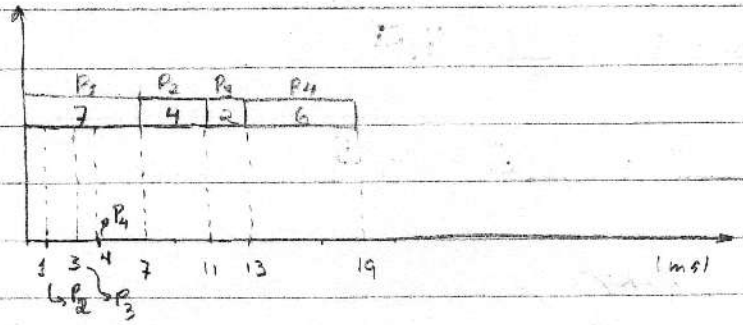
O escalonador tradicional do UNIX possui Kernel não preemptivo ou seja, o processo não perde a posse da CPU em Kernel mode mesmo que o quantum acabe. ~~Isso garante o tempo de resposta~~ do processo deixando o sistema mais confiável e estável. Porém se um processo toma posse da CPU e outro mais prioritário chega na fila de prontos, há inversão de prioridades.

a) À medida que o processo usa CPU, o valor do campo p_CPU do proc struct aumenta e consequentemente aumenta o valor do campo p_usrpri ao passo que o p_CPU dos processos na fila ~~em~~ ready diminuí a uma taxa mais rápida. Sendo assim, a cada vez que o p_pri (campo que o escalonador utiliza para escolher quem deve tomar posse da CPU. Quanto menor p_pri, mais prioritário é o processo) é recalculado até que ocorra troca de contexto. Processos I/O bound possuem CPU burst pequeno e utilizam pouca CPU, logo p_CPU (usado no cálculo de p_pri) é pequeno consequentemente, mais prioritários. Pelo mesmo motivo o escalonador tradicional do UNIX não favorece processos CPU bound pois estes possuem p_CPU grande logo p_pri grande.

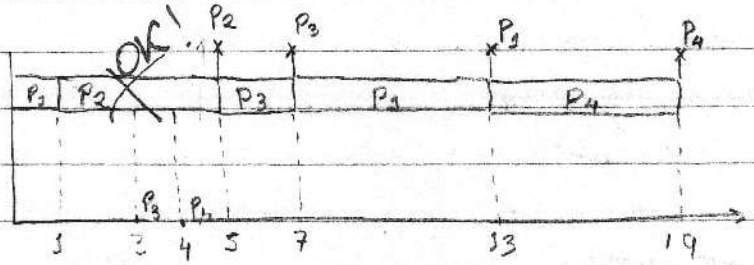
B) Processos de tempo real são processos que dependem de um tempo de resposta rápido para garantir o funcionamento correto do sistema. Logo, precisam ~~ter~~ prioridade alta para tomar posse da CPU e executar o mais rápido possível, o que não é garantido pelo escalonador tradicional do UNIX que possui Kernel não preemptivo. Por exemplo, se um processo grande tomar posse da CPU, e um processo de tempo real chega na fila de prontos ele deve esperar o processo terminar a execução, isso aumenta o tempo de resposta comprometendo a integridade do sistema.


3
NEW
ESCALONADOR LONGO PRAZO
READY
ESCALONADOR CURTO PRAZO
Kernel RUNNING
User RUNNING
interrupção E/S
Kernel Running
APÓS TRATAMENTO DAS INTERRUPÇÕES
USER Running
TROCA DE CONTEXTO
SVC
Kernel Running
ACESSAR E/S
Asleep
NOVA INTERRUPÇÃO (clock)
WAPPER
Stopped + asleep
interrupção E/S
Asleep
E/S PRONTO
Ready
TROCA DE CONTEXTO escalonador
Kernel Running
User Running
LIBEROU CPU
Interrupção
kernel Running
ESPERANDO POR CPU
escalonador
Ready
TROCA DE CONTEXTO escalonador
Kernel Running
User Running
FINALIZOU
Kernel Running
Zombie

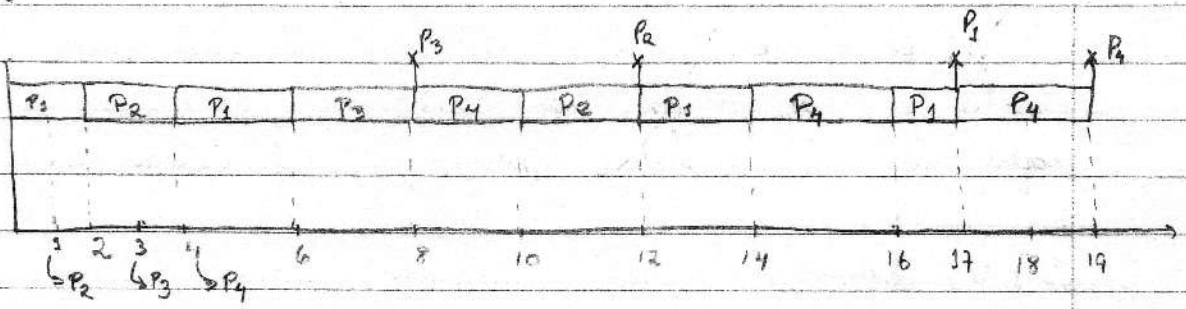
4) FIFO



SRTF



RR

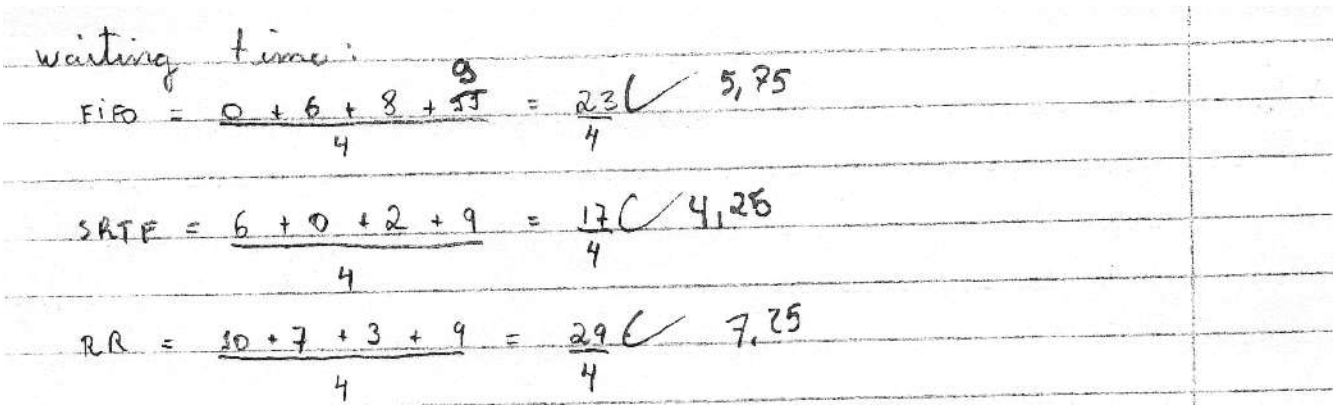


turnaround:

FIFO: $\frac{7 + 10 + 10 + 15}{4} = \frac{42}{4} = 10,5$

SRTF: $\frac{13 + 4 + 4 + 15}{4} = \frac{36}{4} = 9$

RR: $\frac{17 + 11 + 5 + 15}{4} = \frac{48}{4} = 12$



5)
Vantagens: Prioridade aos I/O bound pois pode ficar nas filas de maior prioridade com quanto menor (melhor tmp de resposta); CPU bound de baixa prioridade (não têm requisitos de tmp de resposta) podem rodar com pouco overhead nas filas de menor prioridade e quantum maior; se kernel preemptivo , pode ser usada para processos de tmp real (na fila mais prioritária).
Desvantagens: Starvation; processos são dinâmicos... podem ser I/O bound em um momento, e CPU/bound em outro momento... não serão bem atendidos se não puderem “transitar” entre as filas.

6 a) Falso

Um processador não pode ser tirado a força de um processo e vice-versa o contrário.

O escalonamento preemptivo é capaz de desviar o fluxo de execução de um processo, partindo que ele perca a posse da CPU e seu estado de execução seja alterado.

b) Verdadeiro

Um sistema de tempo real só pode existir caso o Kernel seja preemptivo, pois os processo de tempo real exigem altíssima prioridade e devem ser executados imediatamente ou com espera muito pequena. Além do Kernel preemptivo, precisa de um escalonador que permita preempção.

c) Falso

Há necessidade de ser um sistema preemptivo, mas como os sistemas de tempo compartilhado geralmente são de uso geral e os processos de usuário não têm necessidade de tempo tão curto de resposta, o Kernel não precisa necessariamente ser preemptivo.

d) Verdadeiro

Como o processo não pode sofrer preempção, se chegar na fila de prontos um processo mais prioritário que ele, se novo processo (+ prioritário) terá que aguardar o atual (- prioritário) terminar sua execução até que possa tomar posse da CPU.