PUCRS - Escola Politécnica
Disciplina: Sistemas Operacionais - 2022/2 - Trabalho Prático - Projeto Concorrente
Prof. Fernando Luís Dotti

1. Antecedentes

Na fase anterior introduzimos o escalonamento. Entretanto o funcionamento do sistema ainda era sequencial. O sistema tinha um modelo de dois estados: **running e ready.** Ou seja, durante o IO um processo não ia para o estado bloqueado.

2. Concorrência

Nesta fase veremos como um sistema sobrepõe tempos de dispositivo de entrada e saída com tempos de CPU de processos diferentes. Os dispositivos são entidades concorrentes à CPU. Teremos um modelo de três estados para os processos: **running, ready, e blocked**.

Quando um processo solicita uma operação a um dispositivo, este não pode esperar em CPU que a operação do dispositivo fique pronta (os dispositivos são muito lentos, a cpu muito rápida e cara) pois assim estaríamos sub-utilizando a CPU.

Para que isto seja equacionado, toda vez que um processo pede uma operação (chamada de sistema) que envolva qualquer dispositivo, o pedido deve ser despachado para o dispositivo correspondente e o processo solicitante deve ficar bloqueado até que o dispositivo complete a operação. Enquanto isso, outros processos utilizam a CPU. Assim temos aproveitamento de tempo.

Anteriormente, as operações IN e OUT acontecem sincronamente, com o processo esperando na CPU. **Nesta Fase esta sincronia é quebrada: uma operação IN ou OUT gera uma requisição e a coloca na fila do dispositivo, o processo requerente é bloqueado (salva estado da CPU no PCB, passa para o estado bloqueado e entra numa fila de bloqueados). Um processo pronto é então escolhido para executar, passando para running** (restaura estado do respectivo PCB na CPU, libera CPU para executar). Neste momento há paralelismo entre a CPU e o dispositivo. Para modelar isto no nosso sistema necessitaremos multithreading.

Quando o dispositivo termina a operação solicitada, ele interrompe a CPU. Para tal, deve-se criar mais uma interrupção com código e rotina próprias. O dispositivo escreve em uma variável da CPU (criada para isso), sinalizando a interrupção. Esta interrupção sinaliza que o pedido de um determinado processo ficou pronto. Este processo, que estava bloqueado, deve migrar para a fila de prontos de forma a poder ser escalonado. O processo que estava executando e foi interrompido pelo dispositivo pode, depois da execução da rotina de tratamento, retomar sua execução e acabar sua fatia de tempo.

Devido ao escalonamento da fila de prontos, em algum momento o processo desbloqueado vai ser escalonado e continuar de onde parou, mas agora com a operação de IN ou OUT concluída. O processo esperou pela conclusão no estado bloqueado.

Como visto, as operações de IN e OUT leem um valor para a memória, ou escrevem um valor da memória. **Consideramos aqui que existe acesso direto aa memória (DMA)**. Ou seja, o dispositivo tem a capacidade de acessar a posição informada de memória, na operação de IN ou OUT, utilizando o barramento e sem envolver a CPU. **Em uma operação de IN, isto significa que o valor lido pelo dispositivo é escrito na posição específica de memória do processo que pediu a leitura.** Posteriormente quando o processo é escalonado, o valor solicitado encontra-se na sua memória, na posição informada. Desta forma o processo pode proceder de forma transparente para a próxima operação (apontada normalmente no PC no processo).

Como descrito, **o dispositivo é *concorrente* com a CPU**. O dispositivo executa os pedidos de IO **enquanto** a CPU executa instruções de outros processos. **Isto implica em um design multithreaded. O dispositivo é uma thread. A CPU é outra thread.**

3. SO operante e reativo todo tempo Interação do Usuário

Você deve implementar um shell para aceitar comandos para o sistema concorrentemente com a execução do mesmo.
Ou seja: o sistema operacional aceita continuamente a criação de novos processos enquanto executa os já submetidos. Como qualquer SO, você coloca o sistema no ar e “dispara" programas. Os programas passam a ser executados imediatamente, enquanto você pode pedir a criação de outros. Está completo o sistema.

4. Sugestões

Os elementos acima indicam que, no mínimo, os seguintes elementos são concorrentes, e por isso necessitam threads para sua modelagem:

(i) O shell fica em loop eterno aceitando pedidos de criação de programas, e submete ao SO, como você já fez.

(ii) considere a CPU como uma thread em loop eterno: fica aguardando (enquanto o escalonador seta o estado de um processo nas variáveis da CPU), é liberada (através de um mecanismo de sincronização como um semáforo), executa até a fatia de tempo acabar (sinalizado por interrupção do relógio), ou acontecer uma chamada de sistema pelo processo rodando (IN/OUT), ou um retorno de IO de outro processo acontecer (uma interrupção do IO). Em qualquer caso, a CPU dispara a rotina de tratamento respectiva e volta a aguardar (no semáforo) que outro processo esteja configurado para executar na CPU.

(iii) considere o dispositivo (ou console) como uma thread que fica em loop eterno esperando pedidos em uma fila de pedidos (papel de consumidor modelo produtor/consumidor), recebe e trata, então gera interrupção na CPU para sinalizar que o pedido ficou pronto. O tratamento de IN e OUT é exatamente o mesmo já feito, entretanto agora é feito por esta thread e não mais na mesma linha de execução da CPU.

O esquema abaixo pode ajudar sua compreensão.