Laboratório de Sistemas Operacionais Embarcados (ECOS13)

Prof Otávio Gomes (otavio.gomes@unifei.edu.br)

Roteiro 01

Ponteiro de função e Ponteiro para void

1) A partir da análise do código-fonte disponível no arquivo ***ECOS13_Lab01_1_CircBuffer.c***, temos:

- Uma estrutura de processo, que define um processo com um nome (**processName**) e um tempo (**time**).
- Um buffer circular, que utiliza um vetor estático (buffer) de tamanho definido por **BUFFERSIZE** para armazenar os processos. As posições **start** e **end** controlam o início e o fim do buffer (índices).
- Procedimentos de adição e de remoção de processos, através das funções **addProc** e **removeProc**.
- O código verifica se o buffer está cheio antes de adicionar um novo processo e se está vazio antes de remover um processo.
- A remoção de processos simplesmente avança o ponteiro start, sem tratar o processo removido.

2) A partir das informações apresentadas e do código-fonte fornecido, faça:

a) Implemente as funções **isBufferFull** e **isBufferEmpty** que retornam um booleano indicando o estado do buffer. Use essas funções nas verificações prévias nas funções **addProc** e **removeProc**.

b) Implemente uma função **printBuffer** que lê o buffer e imprime os detalhes de cada processo, como **nome** e **tempo**, e a posição atual dos ponteiros **start** e **end**.

3) A partir da análise do código-fonte disponível no arquivo ***ECOS13_Lab01_2_FuncPtr.c***, temos:

- Uma estrutura de Processo, onde cada processo possui um nome, um tempo e um ponteiro para função (**func**). Permite que cada processo tenha uma função específica associada a ser chamada pelo ponteiro.
- Um buffer circular similar ao primeiro arquivo, mas cada entrada no buffer agora também inclui um ponteiro para função. Procedimentos de adição e de remoção de processos.
- A implementação permite que ações específicas sejam executadas ao remover processos do buffer, o que aumenta a flexibilidade do sistema.

4) A partir das informações apresentadas e do código-fonte fornecido, faça:

a) Implemente a utilização do retorno das funções de processo. A estratégia consistirá em adaptar a função **removeProc** para que ela aja com base no valor retornado pelas funções dos processos. Assumindo que o retorno 0 indica a conclusão do processo e qualquer outro valor indica que o processo deve ser reagendado. Considere que:

- Se a função retorna **0**, processo será concluído:

  ```
  free(buffer[start].name); // Libera a memória do nome do processo
  ```

- Caso o valor de retorno esteja entre **1 e 14**, este valor representará o número de vezes que a função deverá ser executada novamente.
- Caso seja **15** a função será executada indefinidamente.

b) Implemente a função **execProc**, que utilizará alguns comandos da função **removeProc** e fornecerá mais flexibilidade para a execução do código. A função **execProc** executará a função. A retirada da função do buffer circular e análise de seu retorno à fila de execução será feita pela função **removeProc**.

c) A partir das modificações anteriores, adicione o parâmetro *priority* na estrutura dos processos e crie uma função que **priorityBuffer**, que ordena os elementos do buffer circular de acordo com a prioridade (crescente).

Possíveis abordagens:

- Copiar para um Array Temporário, Ordenar e Copiar de Volta
- Ordenação In-Place com Algoritmo Adaptado
- Uso de Estruturas Auxiliares (árvore binária de busca, uma heap ou uma lista ligada)