Projeto 1. Resolvendo problemas de concorrência e sincronização usando Semáforos e Monitores

1. Objetivos

- Desenvolver aplicações concorrentes que acessam recursos compartilhados.
- Compreender os mecanismos de semáforos e monitores para sincronização.

2. Materiais

- Distribuição Linux/Unix
- Ambiente de desenvolvimento para C/C++.
- Bibliotecas de programação: *pthreads, semaphore* e outras.

3. Descrição

O Professor Campiolo reservou uma sala de estudos em grupos para os alunos da disciplina de Sistemas Operacionais. Os alunos de SO podem usar essa sala acompanhados de um estudante-monitor.

No entanto, há duas restrições para uso da sala:

1. deve-se ter um estudante-monitor para cada **X** alunos de SO (X é o número de estudantes por grupo).
2. estudantes-monitores podem deixar a sala se a restrição (1) for mantida.
3. Alunos de SO podem entrar na sala se a restrição (1) for mantida.

Os estudantes-monitores podem tentar deixar a sala a qualquer momento, mas não conseguirão se a restrição (1) não for mantida. Os alunos que chegarem para usar a sala, devem entrar imediatamente se a restrição (1) se mantém.

O professor Campiolo abrirá a sala para o início da entrada e avisará quando não poderá mais entrar alunos e estudantes-monitores. Quando a sala estiver vazia, o professor Campiolo fechará a sala.

Considerações:

- tipos de threads: Professor, Alunos de SO, estudantes-monitores.
- Professor executa as ações: **`abrirSala`**, **`avisarAlunos`**, **`avisarEstudantesMonitores`**, **`fecharSala`**.
- Alunos de SO executam as ações: **`entrarSala`**, **`sairSala`**, **`estudar`**.
- Estudantes-monitores executam as ações: **`entrarSala`**, **`sairSala`**, **`supervisionarAlunos`**.
- A turma de Sistemas Operacionais tem **N** Alunos. Cada grupo tem **X** alunos. Há **K** estudantes-monitores. Use constantes/variáveis no código para testes com diferentes valores.
- exiba mensagens para mostrar as ações, por exemplo, *alunoSO_1 entra na sala*, *estudanteMonitor_1 saiu da sala*, *professor abre a sala*, *estudanteMonitor_2 esperando para sair, alunoSO esperando para entrar*, e assim por diante.

Faça a implementação usando semáforos ou variáveis de condição e mutex (alternativa a monitores em C) para simular e controlar o comportamento das entidades: professor, alunos de SO e estudantes-monitores.

4. Instruções para entrega

A entrega deve ser realizada no Moodle até a data limite especificada no sistema, respeitando as seguintes observações:

- Entregar o código-fonte, *Makefile* e *README*.
- Inclua em todos os arquivos de código-fonte um cabeçalho com a funcionalidade, autor(es) e data.

- Adicione comentários antes dos nomes das funções descrevendo a finalidade e os parâmetros de entrada e saída. Adicione comentários nos principais trechos de códigos do programa.
- Enviar um único arquivo compactado (**tar.gz** ou **zip**).
- A atividade pode ser realizada por grupos de no máximo **três** integrantes.

5. Critérios de avaliação

Os critérios para pontuação são:

- Código-fonte com Makefile e README: 10%.
- Documentação do código: 20%
- Solução correta: 70%

A solução estará correta se não ocorrerem impasses ou inanicação, erros de execução e atender aos requisitos do problema.