Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Trabalho Extraclasse

Identificação da Disciplina

Código da Disciplina	Nome da Disciplina	Turma	Professor	Período
117935	Programação Concorrente	A	Jeremias Moreira Gomes	2019/2

Objetivo do Trabalho Extraclasse

O objetivo deste trabalho é exercitar os conceitos programação concorrente aprendidos na disciplina, por meio de um trabalho a ser realizado fora da sala de aula.

Detalhes Acerca das Aulas Práticas

0.1. Restrições deste Trabalho

Durante a escrita de quaisquer códigos, tentar não utilizar bibliotecas exóticas que não sejam as disponíveis no UNIX como biblioteca padrão. Se o fizer (pois ajuda na realização de testes, por exemplo), lembrar que no momento da correção o sistema padrão a ser utilizado é similar a uma instalação padrão das máquinas do LINF, e nada além do código será instalado apenas para uma correção individual. Então o uso de uma `<gtest.h>`, por exemplo, pode acabar fazendo com que o aluno perca pontos, porque seu código não pôde ser testado, em virtude da falta desta no computador de testes.

O atraso da entrega da atividade, incorre no desconto de 0.5 pontos a cada 60 minutos de atraso, em relação ao prazo de entrega da atividade e da nota final do aluno nesta atividade.

Esta atividade pode ser feita em dupla.

0.2. Por onde será a entrega o Trabalho?

O trabalho deverá ser entregues via Aprender (https://aprender.ead.unb.br), na disciplina de Programação Concorrente.

Na disciplina haverá uma atividade chamada “Trabalho Extraclasse”, para submissão do trabalho.

0.3. O que deverá ser entregue, referente ao trabalho?

Deverão ser entregues o código de resolução do questionário, e um relatório relacionado a resolução do trabalho.

0.4. Como entregar o trabalho?

A submissão das atividades deverá ser feita em um arquivo único comprimido do tipo zip (`Zip archive data, at least v1.0 to extract`), contendo um diretório com o relatório e os códigos elaborados. Além disso, para garantir a integridade do conteúdo entregue, o nome do arquivo comprimido deverá possuir duas informações (além da extensão `.zip`):

- A matrícula do aluno (ou dos alunos separadas por hífen, em caso de dupla).
- O *hash* md5 do arquivo `.zip`.

Para gerar o md5 do arquivo comprimido, utilize o comando `md5sum` do Linux e em seguida faça o renomeamento utilizando o *hash* coletado. Exemplo:

```
[6189] j3r3mias@tardis:aula-01-processos|master > zip -r aaa.zip atividade-01/
  adding: atividade-01/ (stored 0%)
  adding: atividade-01/relatorio.docx (stored 0%)
  adding: atividade-01/01-hello-3-fork.c (deflated 34%)
  adding: atividade-01/03-exemplos.c (deflated 47%)
  adding: atividade-01/02-arvore.c (deflated 35%)
  adding: atividade-01/02-pid_t.c (deflated 39%)
  adding: atividade-01/01-hello-fork.c (deflated 21%)
  adding: atividade-01/03-processos-e-ordens.c (deflated 55%)
[6190] j3r3mias@tardis:aula-01-processos|master > ls -lha aaa.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 2,5K set  4 23:26 aaa.zip
[6191] j3r3mias@tardis:aula-01-processos|master > md5sum aaa.zip
fe36b6799eae103a464cbc4857fce404  aaa.zip
[6192] j3r3mias@tardis:aula-01-processos|master > mv aaa.zip 160068444-fe36b6799eae103a464cbc4857fce404.zip
[6193] j3r3mias@tardis:aula-01-processos|master > ls -lha 160068444-fe36b6799eae103a464cbc4857fce404.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 2,5K set  4 23:26 160068444-fe36b6799eae103a464cbc4857fce404.zip
[6194] j3r3mias@tardis:aula-01-processos|master > md5sum 160068444-fe36b6799eae103a464cbc4857fce404.zip
fe36b6799eae103a464cbc4857fce404  160068444-fe36b6799eae103a464cbc4857fce404.zip
[6195] j3r3mias@tardis:aula-01-processos|master >
```

0.5. Apresentação

Além da entrega do trabalho, via plataforma Moodle, os alunos deverão realizar uma apresentação oral de sua solução.

A data limite para entrega da parte escrita é igual para todos os alunos, porém, para a apresentação, duas aulas foram reservadas para a apresentação. Dessa forma, será realizado um sorteio para apresentação, baseado nos trabalhos entregues, onde cada dupla sorteada terá 10 minutos para apresentar o conteúdo entregue na plataforma.

0.6. Datas Importantes

Entrega da parte escrita: Esse trabalho poderá ser submetido até 10/11/2019 (domingo) - 23:55h.

Apresentações: Os trabalhos serão apresentados nos dias 12/11 e 14/11, por meio de sorteio a ser realizado no início da aula (ou seja, todos deverão estar em condições de apresentar na aula do dia 12/11).

1. Problema de Comunicação Entre Projetos

Cada dupla deverá elaborar um problema de comunicação entre processos que utilize uma solução de memória compartilhada que envolva concorrência e condições de corrida, o qual deverá ser até o dia 24/10/2019. Após aprovado, os alunos deverão elaborar um algoritmo que solucione este problema utilizando os mecanismos de sincronização estudados em aula (*locks*, variáveis condição, semáforos, etc.). Todas as implementações deverão utilizar a biblioteca POSIX Threads.

Como exemplo, segue abaixo, duas sugestões/inspirações para trabalhos envolvendo o uso de mecanismos de sincronização.

1.1. Problema do Banheiro Unissex

1.1.1. Descrição do Problema

No Rock in Rio deste ano, existia apenas um banheiro com n compartimentos com sanitários para uso. O banheiro pode ser utilizado apenas por um gênero por vez. Para fins de simplificação, vamos considerar para o exemplo apenas dois gêneros (1 e 2). Se uma pessoa do tipo 1 chega no banheiro e todos os compartimentos estão livres, ela ocupa um desses compartimentos (por uma espera de tempo aleatório). Se durante o tempo em que a primeira pessoa está no banheiro, chega uma outra pessoa, três situações devem ser avaliadas: se a pessoa é do mesmo gênero e há um compartimento livre, essa pessoa pode utilizar o banheiro; se a pessoa é do mesmo gênero e não há um compartimento livre, essa pessoa deve esperar o compartimento ser liberado; e se a pessoa é de um gênero diferente, ela deve esperar que todas as pessoas que estão utilizando o banheiro libere todos os compartimentos para poder ocupar um compartimento. Dessa forma, gêneros diferentes não utilizam o banheiro ao mesmo tempo.

1.1.2. Descrição (simplificada) da Solução

Identificação de Pontos de Concorrência:

Os recursos a serem gerenciados são banheiros e compartimentos.

Na solução, cada pessoa será representada por uma *thread*, e estas serão todas criadas pela própria função `main`, que irá dispará-las todas no início da execução.

Além disso, cada banheiro será representado por uma fila diferente, a qual o escalonamento ocorrerá por pessoas que estão a mais tempo na fila esperando por sua utilização, independente do gênero.

. . . (e tudo o mais para que o problema e a solução fique o mais claro possível)

Testes

A solução será testada utilizando testes em três casos distintos:

- Um banheiro, um compartimento, um gênero e 100 pessoas.
- Um banheiro, um compartimento, dois gêneros e 100 pessoas.
- Um banheiro, dez compartimentos, dois gêneros e 100 pessoas.
- Três banheiros, dez compartimentos cada, dois gêneros e 1000 pessoas.

1.2. Gerencimento de Jogadores em *Battle Royales*

1.2.1. Descrição do Problema

Battle Royale é um gênero de jogo eletrônico que mistura elementos de exploração, sobrevivência, e procura de equipamentos e de armas, encontrados em um jogo de sobrevivência com a jogabilidade encontrada em um jogo de último sobrevivente. Para acesso a esses jogos, os jogadores se conectam a um servidor online e este realiza o gerenciamento para que uma partida ocorra sempre que n estejam prontos para jogar essa partida. Como há limites nos servidores, sempre que o limite de partidas é atingido, novos jogadores ou jogadores que terminaram sua partida anterior devem esperar por uma nova arena para conseguirem jogar novamente. Além disso, os servidores são diferenciados entre partidas em primeira pessoa e partidas em terceira pessoa.

1.2.2. Descrição (simplificada) da Solução

Identificação de Pontos de Concorrência:

Os recursos a serem gerenciados são arenas do servidor, divididas pelo tipo de cada uma.

Na solução, cada jogador será representado por uma *thread*, e estes serão criados por uma *thread* independente que irá dispará-las de acordo horários pré-definidos ou aleatórios. Além disso, cada jogador sobrevive por um tempo t em sua partida, que após terminada o mesmo decide entre jogar uma nova partida ou sair do jogo.

Para cada tipo de arena será utilizada uma fila diferente, a qual o escalonamento ocorrerá por jogadores que estão a mais tempo na fila, dado o tipo de partida em que o mesmo quer participar.

. . . (e tudo o mais para que o problema e a solução fique o mais claro possível)

Testes

A solução será testada utilizando testes em três casos distintos:

- Uma arena e 100 jogadores.
- Duas arenas de um mesmo tipo e 100 jogadores.
- Duas arenas de tipos diferentes e 400 jogadores.
- Três arenas de tipos diferentes (2, 1) e 1000 jogadores.
- Dez arenas de tipos diferentes (n, m) e 1000 jogadores.

Entrada Para registrar o tempo de login dos jogadores, será utilizado um arquivo de entrada, lido por uma *thread* específica, que irá dispará-las de acordo com suas características de entrada. Esse arquivo contém a identificação do jogador (inteiro), o tipo de partida que ele pretende jogar e o horário em que o jogador conectou-se no servidor com intenção de jogar. Dessa forma, a *thread* responsável irá criar cada um dos jogadores a partir desse arquivo com o formato descrito, cujo exemplo pode ser visto abaixo.

```
3333 1 21
1234 2 30
9336 1 21
...
```

Do exemplo, o jogador 3333 entrou no tempo 21 e pretende jogador uma partida do tipo 1 e o jogador 1234 entrou no tempo 30 e pretende jogar uma partida do tipo 2, por exemplo.