UNIVERSIDADE FEDERAL DE SANTA CATARINA

CAMPUS TRINDADE

INE-DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

INE5410 - PROGRAMAÇÃO CONCORRENTE

Alunos:

João Victor Cabral Machado, Pedro Alfeu Wolff Lemos

22204113 **22200373**

Relatório do Trabalho 2

Florianópolis

2023

Sumário

1. Introdução

O programa em questão é um simulador de uma nova atração que se chama Ixfera, que se assemelha muito com a Sphere que tem em Los Angeles. A ixfera tem três experiências distintas, que são feitas para certas faixas etárias: experiência A para crianças de 4 a 11 anos, experiência B para adolescentes de 12 a 18 anos e a experiência C para adultos acima de 19 anos.

As pessoas esperam em uma fila única para entrar na atração, e a partir do momento que a primeira pessoa entrar, a Ixfera vai iniciar a atração para a sua respectiva faixa etária. Enquanto a Ixfera está fazendo a experiência de uma certa faixa etária, outras pessoas da mesma faixa etária podem entrar na Ixfera, contando que elas estejam em sequência na fila.

Cada pessoa é representada por uma thread, e mecanismos de sincronização são essenciais para garantir que as regras da Ixfera sejam cumpridas. Além disso, o número de pessoas, vagas na atração, tempo de permanência, intervalo máximo entre chegadas, semente para geração de números aleatórios e unidade de tempo para a simulação são todas dadas pelo usuário via linha de comando.

A abordagem adotada neste trabalho mostra a importância da alocação correta dos recursos, a aplicação dos semáforos e mutexes para sincronizar as operações e o controle das etapas da simulação.

No decorrer desse relatório, iremos explicar a estrutura do código e os conceitos de programação concorrente que utilizamos para realizar o projeto.

2. Classe Client

A classe Client modela o comportamento de um cliente na simulação da atração. Ao ser inicializado como uma thread, cada cliente é colocado em uma fila de entrada, aguardando a sua vez. Uma vez liberado para participar da sessão, o cliente entra na fila específica da sessão e aguarda a liberação para iniciar sua experiência. Durante a simulação, o cliente registra o tempo de início e término da sua participação, calculando assim o tempo total de espera. Este valor é então adicionado à lista de relógios apropriada, dependendo da faixa etária do cliente. A classe assegura a sincronização apropriada entre as threads, utilizando semáforos e bloqueios, e imprime mensagens indicando quando o cliente sai da Ixfera, contribuindo para a análise do tempo médio de espera e a conclusão da simulação.

3. Classe Session

A classe Session é a responsável por modelar o funcionamento de uma sessão na simulação da atração Ixfera. Cada instância da classe representa uma sessão específica, definida pela faixa etária dos participantes. Ao ser inicializada como uma thread, a sessão é caracterizada pela sua capacidade de acomodação, determinada pelo usuário. Durante a execução, a sessão permanece em um estado pronto por um período de tempo correspondente à variável permanência, indicando a duração da experiência. Quando a sessão está pronta, os clientes são admitidos em uma fila específica da sessão. A classe garante que, durante a participação dos clientes, o tempo de permanência seja respeitado e que a sessão seja pausada quando necessário. O tempo total de participação de cada cliente é registrado, contribuindo para a análise do tempo médio de espera. Ao final de cada sessão, a classe imprime mensagens indicando a pausa da experiência e incrementa o contador global. A sincronização entre as threads é realizada com a utilização de semáforos e acontece dentro dessa classe, mas iremos falar mais dela no próximo item.

4. Sincronização

A sincronização no código é alcançada por meio de semáforos, bloqueios e condições. O queue_enter semaphore controla a entrada única de clientes na fila,

enquanto o print_semaphore e final_print_semaphorew coordenam a impressão de mensagens no console. O lock e mutex_flag garantem exclusão mútua e evitam condições de corrida ao acessar estruturas de dados compartilhadas e variáveis de controle. A condição item_no_buffer notifica a fila de clientes quando é seguro prosseguir. Além disso, o clock_semaphore e mutex_flag garantem que a medição do tempo de espera ocorra apenas quando a session está pronta. Essa combinação de mecanismos de sincronização assegura uma execução ordenada e segura das operações críticas, contribuindo para a consistência e precisão dos resultados na simulação.

5. Conclusão

Em conclusão, nosso grupo adquiriu uma série de conhecimentos e experiências valiosas. Aprendemos a importância da programação concorrente e como semáforos e mutexes são usados para fazer a sincronização das threads de maneira eficaz. Esta experiência destacou a complexidade de situações do mundo real e como as soluções podem ser adaptadas para lidar com desafios.

Em segundo lugar, a execução deste projeto evidenciou a relevância da estrutura e documentação meticulosa do código. Ao adotarmos uma organização sólida e inserimos comentários claros, conseguimos manter o código de forma legível, compreensível e robusta.

Por fim, a simulação proporcionou uma compreensão mais profunda da interação entre as diversas partes de um sistema complexo, proporcionando uma visão mais aprofundada dos desafios enfrentados na tomada de decisões e na coordenação de tarefas em ambientes concorrentes. Este projeto nos apresentou uma oportunidade valiosa de aprendizado, ressaltando a importância da aplicação prática dos nossos conhecimentos.