Problema de Comunicação entre Processos aplicado em RuPaul's Drag Race

Ítalo Eduardo Dias Frota
(18/0019279)

9 de maio de 2021

Universidade de Brasília - Instituto de Ciências Exatas
Departamento de Ciência da Computação - CIC0202 - Programação Concorrente
2020.2 - Turma 01A - Professor Eduardo A. P. Alchieri
Prédio CIC/EST - Campus Universitário Darcy Ribeiro
Asa Norte 70919-970 Brasília, DF
180019279@aluno.unb.br

Resumo

O projeto foi desenvolvido para a disciplina de Programação Concorrente da Universidade de Brasília - (UnB) ministrada no semestre 2020/2. O presente relatório tem como objetivo apresentar um algoritmo para solucionar um problema de comunicação entre processos através de uma memória compartilhada.

Palavras-chave: *Programação Concorrente. UnB. Threads. Condição de Corrida.*

1 Introdução

Projetos com a aplicação dos conceitos de *multithreading* expandem as possibilidades de solução para problemas computacionais, entretanto, podem implicar aumento de complexidade comparados com aplicações sequenciais. Para o bom gerenciamento de múltiplas *threads* é imprescindível analisar possíveis gargalos na sincronização dos processos e compartilhamento de memória, visando a eficiência do projeto.

Aqui, foi proposta uma solução para um problema de comunicação entre processos baseado no *reality show* competitivo *RuPaul's Drag Race*. Este relatório apresenta a elaboração da situação e a descrição do algoritmo de solução.

2 Formalização do Problema Proposto

2.1 Inspiração

O *Drag Race* é uma franquia de *reality shows* do gênero competição, atualmente presente em diversos países. O programa original, *RuPaul's Drag Race* foi ideali-

zado pelo apresentador estadunidense RuPaul Andre Charles, mais conhecido apenas como RuPaul, o nome da sua persona *drag queen*.

O programa seleciona candidatos de qualquer orientação e/ou condição sexuais, basta apenas que assumam um personagem artístico. A fórmula do programa consiste em competições semanais envolvendo gincanas e provas orientados pelo apresentador, onde as *drag queens* têm suas habilidades em costura, maquiagem, canto, dança, talento e personalidade testadas. A eliminação é progressiva, logo, em cada episódio uma competidora é eliminada, ao sobrarem 3 participantes é realizada a final da temporada onde a vencedora é coroada.

No formato original, cada episódio possui um mini-desafio e um desafio principal que podem ser individuais em grupo. Os mini-desafios costumam proporcionar vantagens para as campeãs no desafio principal. Após o cumprimento das provas, as competidoras desfilam figurinos inspirados em uma temática e são avaliadas pelo painel de jurados de acordo com o desempenho daquela semana.

Após o julgamento, uma *drag queen* é consagrada como a vencedora do desafio semanal e as participantes com o pior desempenho vão para a berlinda. Para tentar se salvar, elas devem dublar uma música escolhida pela produção para demonstrar seu talento e força de vontade. Com base nas performances, RuPaul decide quem continua no programa e quem é eliminada.

2.2 Descrição do Problema

Ao analisar o formato do *reality show*, é possível apontar algumas semelhanças com mecanismos de sincronização utilizados em programação concorrente. Partindo disso, foi eloborado o problema.

No primeiro episódio da nova temporada de *RuPaul's Drag Race*, temos 16 novas competidoras em busca do título de "*America's Next Drag Superstar*". Nesta semana, as participantes foram desafiadas a confeccionar figurinos inspirados em professores do Departamento de Ciência da Computação da Universidade de Brasília.

A *workroom* possui um número x de espelhos, o que define a quantidade de *queens* que podem se preparar ao mesmo tempo, ninguém vai para a passarela enquanto todas as adversárias não estiverem prontas.

Uma vez que todas estão preparadas, elas seguem para o *backstage*. Aqui, elas esperam a permissão para desfilar, uma de cada vez.

Assim que todas mostraram o figurino na passarela, é hora do julgamento. A RuPaul consagra uma vencedora e coloca na berlinda as duas competidoras com o pior desempenho da semana.

As *queens* dublam e então uma é escolhida para dar adeus à competição.

3 Descrição do Algoritmo Desenvolvido para Solução do Problema Proposto

São criadas as seguintes *threads*:

- *rupaul* - para o apresentador do programa;

- *queen[X]* - para as 16 participantes da temporada.

Para os mecanismos de sincronização foram utilizados apenas semáforos pois foi a técnica que enfrentei mais dificuldades em utilizar ao longo da disciplina, então precisava me capacitar mais. Os semáforos utilizados são os seguintes:

- *rupaul_sem* - semáforo do apresentador, possui valor inicial 0 e funciona como uma variável de condição;
- *critica_sem* - semáforo para as críticas do apresentador, possui valor inicial 0 e funciona como uma variável de condição;
- *espelho_sem* - semáforo para os espelhos que as competidoras usam para se arrumar, possui valor inicial mínimo 1 e define quantas *drag queens* podem se montar ao mesmo tempo;
- *passarela_sem* - semáforo para a passarela, possui valor inicial 1 e funciona como um *mutex*;
- *lipsync_sem* - semáforo para a dublagem, possui valor inicial 0 e funciona como uma variável de condição;
- *veredito_sem* - semáforo para o veredito do apresentador, possui valor inicial 0 e funciona como uma variável de condição;
- *pronta_sem* - semáforo das competidoras no *backstage*, possui valor inicial 0 e funciona como uma variável de condição.

3.1 Rotina das *Drag Queens*

O ciclo de execução de uma *thread* de competidora, genericamente, funciona da seguinte forma:

1. Espera a vez de se montar através do semáforo do espelho (*sem_wait*);
2. Se monta (*sleep*);
3. Se todas as queens estiverem montadas, acorda a RuPaul através do semáforo do apresentador (*sem_post*);
4. Espera para seguir para o *backstage* através do semáforo que indica que está pronta (*sem_wait*);
5. Espera para desfilar através do semáforo da passarela (*sem_wait*);
6. Desfila (*sleep*);
7. Libera a passarela para outra *queen* (*sem_post*);
8. Se não está na berlinda, a *thread* é finalizada;
9. Espera para dublar através do semáforo de *lipsync* (*sem_wait*);

10. Dubla (*sleep*);
11. Acorda a RuPaul para o veredito final através do semáforo de veredito (*sem_post*);

```
if(gravando){

  // As queens se montam
  sem_wait(&espelho_sem);       // semáforo para usar o espelho
    queens_montadas++;
    printf(COLOR_YELLOW "A queen %d terminou de se montar.\n" RESET, id);
    sleep(1);                   // se monta
    if(queens_montadas ==  QTD_QUEENS){
      sem_post(&rupaul_sem);    // se todas as queens estiverem prontas, acorda a RuPaul
    }
  sem_post(&espelho_sem);       // libera o espelho

  sem_wait(&pronta_sem);       // espera permissão para ir pro backstage

  // As queens desfilam, uma de cada vez
  sem_wait(&passarela_sem);
    queens_desfilaram++;
    sleep(1);                  // desfila
    if(queens_desfilaram == QTD_QUEENS){      // Quando todas as queens desfilam, a RuPaul avalia
      sem_post(&critica_sem);
    }
    printf(COLOR_CYAN "A queen %d desfilou na passarela.\n" RESET, id);
  sem_post(&passarela_sem);    // libera a passarela

  sem_wait(&lipsync_sem);      // espera para dublar
    printf(COLOR_CYAN "A queen %d está dublando.\n" RESET, id);
    sleep(1);                  // dubla
    sem_post(&veredito_sem);     // libera a decisão final

}
```

Figura 1: Código da thread das competidoras.

3.2 Rotina da RuPaul

Já o ciclo de execução da *thread* do apresentador se baseia na seguinte rotina:

1. Espera para começar a apresentar o programa através do semáforo da RuPaul (*sem_wait*);
2. Apresenta (*sleep*);
3. Libera as queens para o *backstage* através do semáforo que indica que elas estão prontas (*sem_post*);
4. Espera para criticar as *queens* (*sem_wait*);
5. Avalia e critica (*sleep*);
6. Escolhe pseudorandômicamente a vencedora e as possíveis eliminadas;
7. Finaliza as *threads* das competidoras que foram salvas (*pthread_cancel*);
8. Libera as possíveis eliminadas para a dublagem através do semáforo de *lipsync* (*sem_post*);

9. Espera a dublagem para dar o veredito final através do semáforo de veredito (*sem_wait*);

10. Dá o veredito e encerra o episódio;

```
while(gravando){

  sem_wait(&rupaul_sem);            // espera para apresentar o programa
    queens_montadas = 0;            // zera o contador de queens montadas
    // APRESENTA O PROGRAMA
    sleep(3);

    for(i = 0; i < QTD_QUEENS; i++)
      sem_post(&pronta_sem);        // libera as queens para o backstage

  sem_wait(&critica_sem);           // espera para criticar as queens
    sleep(1);
    printf(COLOR_GREEN "\n          ########## Todas as queens desfilaram ##########\n\n"RESET);
    sleep(2);

    printf(COLOR_GREEN "\n          ########## RuPaul está avaliando as competidoras ##########\n"RESET);

    // escolha pseudorandomica das queens para o julgamento
    x = rand() %QTD_QUEENS-1;
    y = rand() %QTD_QUEENS-1;
    z = rand() %QTD_QUEENS-1;

    for(i = 0; i < QTD_QUEENS; i++){
      if((i ≠ y) && (i ≠ z)){
        pthread_cancel(queen[i]);         // encerra as threads das queens que não vão dublar
      }
    }
    // AVALIA AS QUEENS

    // APRESENTA O PROGRAMA

    for(i = 0; i < 2; i++)
      sem_post(&lipsync_sem);         // libera as queens para dublar

    sem_wait(&veredito_sem);            // espera para dar o veredito final

    // escolha pseudorandômica de qual queen vai sair
    srand(time(NULL));
    int rnd = rand();
    int result = (rnd > RAND_MAX/2) ? y : z;
    // ESCOLHE E ELIMINA UMA PARTICIPANTE

    gravando =  false;            // encerra o episódio
}
```

Figura 2: Generalização do código da thread da RuPaul.

3.3 Código na íntegra

O código completo, assim como as instruções de execução estão disponíveis no seguinte repositório:
`https://github.com/titofrota/pc-dragrace` e no arquivo compactado enviado pela plataforma Aprender.

4 Conclusão

Foi possível simular o formato de um típico episódio de *RuPaul's Drag Race*, embora o projeto apresente uma versão adaptada, seria possível replicar o funcionamento original tranquilamente com as técnicas ministradas na disciplina. Uma possível

evolução do trabalho poderia ser simular uma temporada inteira, até obter a vencedora da final.

Através das técnicas de programação concorrente é possível implementar soluções para os mais diversos problemas que envolvam múltiplas *threads*, expandindo a gama de aplicações de situações até mesmo do mundo real. Além disso, como visto, é possível utilizar semáforos no lugar de *locks* e variáveis de condição, o que demonstra o poder de tal artifício.

5 Bibliografia

Referências

1. "POSIX Threads Programming", `https://hpc-tutorials.llnl.gov/posix/` . Acesso em maio de 2021 .

2. "RuPaul's Drag Race", `https://pt.wikipedia.org/wiki/RuPaul's_Drag_Race` . Acesso em maio de 2021

3. ALCHIERI, Eduardo. "Semáforos". 20 slides, `https://cic.unb.br/~alchieri/disciplinas/graduacao/pc/semaforos.pdf` . Acesso em maio de 2021 .