Especificação Trabalho Prático - Problema de sincronização usando *threads*

Eduardo G. R. Miranda

November 1, 2023

O trabalho prático (TP) deve ser realizado em trios e consiste na implementação da resolução de um **problema de sincronização usando threads**. Tal resolução deve ser implementada utilizando a linguagem de programação "C".

1 Instruções

Conforme mencionado, o TP proposto consiste em solucionar um problema de sincronização.

1.1 Valor

A implementação correta que atenda a todos os requisitos especificados da primeira parte do TP tem **40** pontos.

A implementação correta que atenda a todos os requisitos especificados da segunda parte do TP tem **20** pontos.

1.2 Entrega

A **data máxima de entrega** da primeira etapa é: 14/11/2023 às 23:55.

As apresentações da primeira etapa ocorrerão no dia 14/11/2023 durante a aula.

A **data máxima de entrega** da segunda etapa é: 05/12/2023 às 23:55.

As apresentações da primeira etapa ocorrerão no dia 06/12/2023 durante a aula.

Deverá ser submetido no Sistema Integrado de Gestão de Atividades Acadêmicas um arquivo compactado (.zip ou .tgz) contendo:

- o **código-fonte** do programa
- um **Makefile**[1] (para automatizar com o make a compilação via gcc);

[1] `https://pt.wikibooks.org/wiki/Programar_em_C/Makefiles`

- um breve relatório contendo um resumo do projeto, descrevendo a estrutura geral do código, estruturas importantes, decisões relevantes (ex.: como lidou com ambiguidades na especificação), bugs conhecidos ou problemas (lista dos recursos que não implementou ou que sabe que não estão funcionando). Não inclua a listagem do seu código-fonte no relatório; afinal, você já vai entregar o código fonte!

1.3 Grupo

Trabalho prático em grupo, a ser realizado em trio (três alunos). Exceções devem ser previamente comunicadas e aprovadas pelo professor.

1.4 Sobre o Problema de Sincronização

Seu programa deverá ser implementado em C. Recursos de outras linguagens (C++, Python, etc) não deverão ser utilizados (na dúvida, pergunte ao professor).

Foi inaugurada a primeira casa lotérica da cidadezinha de Leopoldinópolis. Os (poucos) moradores estão empolgados com a possibilidade de poder sacar dinheiro, pagar contas ou apostar na BoboSena sem precisar viajarem até uma outra cidade. Entretanto, a casa lotérica conta com apenas um operador de caixa. Naturalmente, apenas uma pessoa pode ser atendida por vez e para resolver conflitos há determinadas regras no atendimento aos clientes.

1ª ETAPA) SINCRONIZAÇÃO: O **único caixa** da casa lotérica deve ser compartilhado pela vovó **Vanda** (já idosa), mamãe **Maria** (que está grávida), prima **Paula** (pé quebrado) e **Sueli** (uma pessoa comum). Para a prioridade no atendimento aos clientes, a casa lotérica definiu o seguinte conjunto de regras:

- se o caixa estiver liberado, quem chegar primeiro na casa lotérica já pode ser logo atendido;
- caso contrário, se o caixa estiver ocupado, quem chegar depois tem que esperar o caixa ser desocupado.
- se mais de uma pessoa estiver esperando na fila de atendimento do caixa, valem as seguintes precedências:
  - **Grávida** (ou com criança de colo) deve ser atendida antes de **Idoso** ;
  - **Idoso** deve ser atendido antes de pessoa com **Deficiência** ;
  - Pessoa com **Deficiência** deve ser atendida antes de pessoa **Comum**
  - Pessoas **Comuns** só serão atendidas após as pessoas com maior prioridade terem sido atendidas;

  OBS.: pessoas de mesma prioridade são atendidas na ordem de sua chegada.
- quando alguém termina de ser atendido, deve liberar o caixa para a próxima pessoa de maior prioridade, exceto em dois casos:

- para evitar inanição (discutido a seguir);
- quando houver deadlock (i.e.: tiver sido formado um ciclo de prioridades).

Cada uma destas quatro pessoas está associada a uma outra pessoa de igual prioridade, formando um casal: **vovó Vanda** é casada com **vovô Valter** , **mamãe Maria** é casada com **papai Marcos** (que sempre carrega no colo a primeira filha), **prima Paula** está noiva com o **pedreiro Pedro** (que também quebrou o pé) e **Sueli** namora com simplesmente **Silas** (igualmente comum).

Os dois membros de um casal podem ir à casa lotérica separadamente. Os cônjuges entram no esquema de prioridades da seguinte forma: cada cônjuge tem a mesma prioridade da pessoa a quem está associado. Só que, se ambos os membros de um casal quiserem ser atendidos pela caixa da lotérica, um tem que esperar depois do outro (por ordem de chegada entre eles).

Exemplo.: Se Marcos, Paula e Pedro estiverem esperando para serem atendidos e ninguém mais chegar, Marcos usa primeiro pois tem prioridade sobre Paula/Pedro, depois será atendido Pedro ou Paula, dependendo de quem chegou primeiro entre eles (pois têm a mesma prioridade); mas se Vanda chegasse antes de Marcos acabar de ser atendido, ela teria preferência de atendimento logo em seguida a Marcos (antes de Pedro/Paula).

INANIÇÃO: Atente para o fato de que o tratamento de fila com prioridades pode levar à inanição em casos de uso intenso do caixa da lotérica, daí é preciso criar uma regra para resolver o problema. Se uma pessoa de menor prioridade no atendimento deixar de ser atendida por duas vezes em seguida porque outras pessoas com maior prioridade chegaram depois dela mas foram atendidas primeiro, então esta pessoa sofrendo inanição deverá se beneficiar de um "envelhecimento". Gradualmente incremente a prioridade da pessoa em inanição, incrementando sua categoria de prioridade (uma vez a cada dupla frustração de "furarem fila na sua frente") até que eventualmente ela seja atendida e, logo após, volte ao seu patamar original de prioridade de atendimento. Qualquer pessoa com prioridade menor que outras pode sofrer inanição!

[ATENÇÃO] Se você reparar as precedências definidas, vai notar que nesta primeira versão do trabalho ainda não ocorrerá deadlocks. É isso mesmo: nesta primeira etapa do trabalho ainda não ocorrerão deadlocks, apenas eventual inanição.

DEADLOCK: Cada pessoa, como os filósofos daquele famoso problema, dividem seu tempo entre períodos em que fazem outras coisas (estudam, trabalham, descansam) e períodos em que resolvem ir à casa lotérica para realizar alguma operação financeira. Nesta simulação de Leopoldinópolis, o tempo que cada um gasta com outras coisas varia entre 3 e 5 segundos e ser atendido na lotérica leva 1 segundo.

OBS.: os tempos das outras coisas são apenas uma referência, você pode experimentar valores de tempo um pouco diferentes, se for mais adequado aos seus testes. Certifique-se de que em alguns casos esses tempos sejam suficientes para gerar alguns deadlocks de vez em quando, bem como situações que exijam o mecanismo de prevenção de inanição.

Para a entrega da segunda etapa do trabalho deverá ser implementada uma pequena modificação no mecanismo de prioridades. O gerente da casa lotérica foi chamada para ser padrinho no futuro casamento de Pedro e Paula, de maneira que implementou uma nova política de prioridades no atendimento, para agradar seus clientes preferidos: se mais de uma pessoa estiver esperando na fila de atendimento do caixa, valem as seguintes precedências:

- **Grávida** (ou com criança de colo) deve ser atendida antes de **Idoso** ;
- **Idoso** deve ser atendido antes de pessoa com **Deficiência** ;
- Pessoa com **Deficiência** deve ser atendida antes de pessoa **Grávida** (ou com criança de colo);
- Pessoas **Comuns** só serão atendidas após as pessoas com maior prioridade terem sido atendidas;

Agora, é possível ocorrer um **deadlock** (ex.: Maria → Valter → Pedro → Marcos). O gerente da casa lotérica sabe que este problema pode ocorrer mas não quer voltar atrás na modificação de prioridades pois Pedro e Paula são bons clientes. Para evitar uma trava geral no atendimento, o **gerente** da casa lotérica periodicamente (a cada 5 segundos) confere a situação da fila do caixa e, se encontrar o pessoal travado (caixa vazio e alguém de cada categoria de prioridade esperando), deve escolher uma pessoa aleatoriamente e liberá-la para usar o caixa.

1.5 Implementação

Sua tarefa neste trabalho é implementar as pessoas da cidade como **threads** e implementar o caixa como um **monitor** usando **pthreads** , **mutex** e variáveis de **condição**.

Parte da solução requer o uso de uma mutex para o caixa, que servirá como a trava do monitor para as operações que as pessoas podem fazer sobre ele. Além da mutex, você precisará de um conjunto de variáveis de condição para controlar a sincronização do acesso prioritário ao caixa.

O programa deve criar as nove pessoas (threads) e receber como parâmetro o número de vezes que eles vão tentar usar o caixa — afinal, não dá para ficar assistindo eles fazerem isso para sempre. Ao longo da execução o programa deve então mostrar mensagens sobre a evolução do processo. Por exemplo:

```
$ ./loterica 2
Maria está na fila do caixa   {fila:M}
Maria está sendo atendido(a)  {fila:  }
Vanda está na fila do caixa   {fila:V}
Silas está na fila do caixa   {fila:VS}
Pedro está na fila do caixa   {fila:VSP}
Maria vai para casa           {fila:VSP}
Marcos está na fila do caixa  {fila:VSPM}
Gerente detectou deadlock, liberando Pedro para atendimento
Pedro está sendo atendido(a)  {fila:VSM}
Paula está na fila do caixa   {fila:VSMP}
Pedro vai para casa           {fila:VSMP}
Marcos está sendo atendido(a) {fila:VSP}
Gerente detectou inanição, aumentando prioridade de Silas
Marcos vai para casa          {fila:VSP}
Vanda está sendo atendido(a)  {fila:SP}
Vanda vai para casa           {fila:SP}
Silas está sendo atendido(a)  {fila:P}
Pedro está na fila do caixa   {fila:PP}
...
```

Nota: a ordem dos eventos acima é apenas um exemplo, devendo variar entre execuções (devido ao escalonamento das threads) e implementações. Você não deve esperar executar o programa e coincidentemente obter o mesmo exemplo acima, mas deve obter o mesmo resultado, ou seja, respeitar as prioridades de uso do caixa (sincronização), evitar inanição e resolver deadlocks.

1.6 Sumarizando

- as regras de preferência definidas acima devem ser respeitadas;
- pessoas com uma mesma prioridade esperam entre si por ordem de chegada;
- deadlock deve ser resolvido pela atuação do Gerente;
- inanição deve ser evitada com o envelhecimento gradual (promoção de prioridade);

Para implementar a solução, analise as ações de cada pessoa em diferentes circunstâncias, como se outra pessoa de mesma prioridade já está esperando, se houver alguém com maior prioridade esperando, se há alguém com menor prioridade esperando mas que tenha sido promovido a nova prioridade. Determine o que fazer quando alguém já está na fila do caixa mas surge na fila um novo alguém de maior prioridade, o que fazer quando terminam de usar o caixa, etc.

Consulte as páginas de manual no Linux para entender as funções da biblioteca para **mutex** e variáveis de condição (**cond**)

```
- int pthread_mutex_init(pthread_mutex_t *mutex,
const pthread_mutexattr_t *attr);
- int pthread_mutex_lock(pthread_mutex_t *mutex);
- int pthread_mutex_unlock(pthread_mutex_t *mutex);
- int pthread_mutex_destroy(pthread_mutex_t *mutex);

-int pthread_cond_init(pthread_cond_t *cond,
const pthread_condattr_t *attr);
-int pthread_cond_signal(pthread_cond_t *cond)
-int pthread_cond_wait(pthread_cond_t *cond)
-int pthread_cond_destroy(pthread_cond_t *cond)
```

1.7 Dicas de Implementação

Como mencionado anteriormente, você deve usar uma mutex para implementar a funcionalidade equivalente a um monitor, isto é, as operações sobre o caixa serão parte de um monitor. Na prática, isso será implementado em C, mas deverá ter a funcionalidade equivalente a:

```
    //PSEUDOCÓDIGO
monitor caixa
{

...//variáveis compartilhadas, variáveis de condição

void esperar(int pessoa){
printf("%s está na fila do caixa\n", nome(pessoa));
...//verifica quem mais quer usar, contadores, variáveis de cond., etc.
}

void liberar(int pessoa){
printf("%s vai para casa\n", nome(pessoa));
...//verifica se tem que liberar alguém, atualiza contadores, etc.
void verificar(){
...//gerente verifica se há deadlock e corrige-o
}

};
```

Cada pessoa (exceto o gerente) executam as seguintes operações um certo número de vezes (definido pelo parâmetro de entrada na execução do programa):

```
caixa.esperar(p); //exige mutex
    atendido_pelo_caixa(p); //não exige exclusão mútua
caixa.liberar(p); //exige mutex
vai_embora_para_casa(p); //espera um certo tempo aleatório
```

Use **srand**() — confira a página do manual — para gerar números aleatórios entre 0 e 1, e faça uma multiplicação para gerar inteiros aleatórios.

Já o **gerente** executa as seguintes operações:

```
enquanto pessoas estão ativas faça:
    sleep(INTERVALO_VERIFICACAO);
    caixa.verificar();
```

2 Informações úteis

2.1 Forma de operação

O seu programa deve basicamente criar uma thread para cada pessoa e esperar que elas terminem. Cada pessoa executa um loop um certo número de vezes (parâmetro de entrada na linha de comando), exceto o gerente, que deve executar seu loop até que todos as outras pessoas tenham acabado.

2.2 Codificação das pessoas

Você deve buscar produzir um código elegante e claro. Em particular, note que o comportamento das equipes é basicamente o mesmo, você não precisa replicar o código para diferenciá-los. Além disso, o comportamento de todas as pessoas (exceto o gerente) é tão similar que você deve ser capaz de usar apenas uma função para todos eles, parametrizada por um número, que identifique cada pessoa. As prioridades podem ser descritas como uma lista circular.

2.3 Manipulação de argumentos de linha de comando

Os argumentos que são passados para um processo na linha de comando são visíveis para o processo através dos parâmetros da função main():

```
int main(int argc, char *argv[]);
```

o parâmetro argc contém um a mais que o número de argumentos passados (no caso deste enunciado o argc deverá ter valor igual a 2) e argv é um vetor de strings, ou de apontadores para caracteres (no caso deste enunciado o número de iterações deverá ser obtido em argv[1]) mais informações sobre os parâmetros das funções podem ser vistas aqui.

2.4 Processamento de Desenvolvimento

Lembre-se de conseguir fazer funcionar a funcionalidade básica antes de se preocupar com todas as condições de erro e os casos extremos. Por exemplo, primeiro foque no comportamento de uma pessoa e certifique-se de que ela funciona. Depois dispare duas pessoas apenas, para evitar que deadlocks

aconteçam. Verifique se pessoas de uma mesma prioridade funcionam, inclua o gerente e verifique se deadlocks são detectados (use um pouco de criatividade no controle dos tempos das outras atividades para forçar um deadlock, para facilitar a depuração. DICA: economize/comente prints e sleeps). Finalmente, certifique-se que o mecanismo de prevenção da inanição funciona (p.ex., use apenas dois casais e altere os tempos das outras atividades para fazer com que um deles (o de maior prioridade) esteja sempre querendo usar o caixa.

Exercite bem o seu próprio código! Você é o melhor testador dele (* mas não se esqueça de pedir para outras pessoas testarem para tentar identificar problemas que você tenha deixado passar desapercebidos). Mantenha versões do seu código. Ao menos, quando você conseguir fazer funcionar uma parte da funcionalidade do trabalho, faça uma cópia de seu arquivo C ou mantenha diretórios com números de versão. Ao manter versões mais antigas, que você sabe que funcionam até um certo ponto, você pode trabalhar confortavelmente na adição de novas funcionalidades, seguro no conhecimento de que você sempre pode voltar para uma versão mais antiga que funcionava, se necessário. Ou mais recomendável ainda utilize algum repositório git como o github.

3 Considerações finais

Este trabalho não é tão complexo quanto pode parecer à primeira vista. Talvez o código que você escreva seja mais curto que este enunciado. Escrever o seu monitor será uma questão de entender o funcionamento das funções de pthreads envolvidas e utilizá-las da forma correta. **O programa final deve ter apenas poucas centenas de linhas de código** . Se você observar que esteja escrevendo código mais longo que isso, provavelmente é uma boa hora para parar um pouco e pensar mais sobre o que você está fazendo (Ex.: será que realmente precisa copiar e colar tantos IFs/ELSEs ou consegue usar arranjos e laços de repetição para verificar IF/ELSE mais genéricos?). Entretanto, dominar os princípios de funcionamento e utilização das chamadas para manipulação de variáveis de condição e mutexes e conseguir a sincronização exata desejada pode exigir algum tempo e esforço.

1. Dúvidas: envie e-mail para eduardomiranda@cefetmg.br ou procure o professor fora do horário de aula (vide horário de atendimento).

2. Comece a fazer o trabalho logo, pois apesar do programa final ser relativamente pequeno, o tempo não é muito e o prazo de entrega não vai ficar maior do que ele é hoje (independente de que dia é hoje).

3. Será valorizado também a clareza, qualidade do código e da documentação e, obviamente, a execução correta com programas de teste.

4 Créditos

O enunciado deste trabalho foi baseado no material do Prof.º Everthon Valadão (IFMG).