Sistemas Operacionais

TP2: Sincronização de processos

Trabalho individual ou em dupla.

Data de entrega: verifique o calendário do curso

(Política de penalização por atraso na página do curso)

Enunciado do trabalho em PDF

Este trabalho tem por objetivo fazer com que os alunos experimentem na prática a programação de problemas de sincronização entre processos e os efeitos da programação concorrente. Isso deve ser feito utilizando-se os recursos de threads POSIX (pthreads) -- em particular, mutexes (`pthread_mutex_t`) e variáveis de condição (`pthread_cond_t`).

O problema

Sheldon, Leonard e Howard compraram um micro-ondas juntos (tempos difíceis). Para decidir quem podia usar o forno, definiram o seguinte conjunto de regras:

- se o micro-ondas estiver liberado, quem chegar primeiro pode usar;
- caso contrário, obviamente, quem chegar depois tem que esperar;
- se mais de uma pessoa estiver esperando para usar, valem as precedências:
  - Sheldon pode usar antes do Howard;
  - Howard pode usar antes do Leonard;
  - Leonard pode usar antes do Sheldon.

Como agora eles têm namoradas (Leonard tem a Penny, Howard tem a Bernardette e Sheldon tem a Amy, caso você não saiba), elas entram no esquema do forno da seguinte forma: cada namorada tem a mesma prioridade do namorado. Só que se os dois quiserem usar o micro-ondas, um tem que esperar depois do outro. (Quer dizer, se Penny, Leonard e Howard estiverem esperando para esquentar algo e ninguém mais chegar, Howard usa primeiro, depois Penny ou Leonard, dependendo de quem chegou primeiro; mas se a Bernardette chegasse antes do Howard acabar, ela teria preferência de usar em seguida.

Isso pode levar à inanição em casos de uso intenso do forno, daí é preciso criar uma regra para resolver o problema: se o casal usar o forno em seguida, o segundo é obrigado a ceder a vez para um membro do casal que normalmente teria que esperar por eles.

Cada personagem, como os filósofos daquele problema, dividem seu tempo entre períodos em que fazem outras coisas e períodos em que resolvem esquentar algo para comer. O tempo que cada um gasta com outras coisas varia entre 3 e 6 segundos e usar o forno gasta um segundo. (Os tempos das outras coisas são apenas uma referência, você pode experimentar tempos um pouco diferentes, se for mais adequado. 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.

Se você reparar as precedências definidas, vai notar que é possível ocorrer um deadlock -- é, eles sabem, mas são muito teimosos para mudar. Para evitar isso, o Raj periodicamente (a cada 5 segundos) confere a situação e, se encontrar o pessoal travado, pode escolher um deles aleatoriamente e liberá-lo para usar o forno.

Implementação

Sua tarefa neste trabalho é implementar os personagens do Big Bang Theory como threads e implementar o forno como um monitor usando pthreads, mutexes e variáveis de condição.

Parte da solução requer o uso de uma mutex para o forno, que servirá como a trava do monitor para as operações que os personagens podem fazer sobre ele. Além da mutex, você precisará de um conjunto de variáveis de condição para controlar a sincronização do acesso dos casais ao forno: uma variável para enfileirar o segundo membro de um casal se o outro já estiver esperando, e outra variável de condição para controlar as regras de precedência no acesso direto ao forno.

O programa deve criar os sete personagens e receber como parâmetro o número de vezes que eles vão tentar usar o forno -- 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:

```
$ ./bbtmwov 2
Sheldon quer usar o forno
Sheldon começa a esquentar algo
Leonard quer usar o forno
Howard quer usar o forno
Amy quer usar o forno
Sheldon vai comer
Penny quer usar o forno
Raj detectou um deadlock, liberando Howard
Howard começa a esquentar algo
Sheldon quer usar o forno
Howard vai comer
Leonard começa a esquentar algo
Leonard vai comer
Penny começa a usar o forno
Leonard quer usar o forno
Amy começa a usar o forno
...
```

Nota: a ordem dos eventos acima deve variar entre execuções e implementações. Você não deve esperar ter exatamente o mesmo resultado.

Sumarizando:

- membros do mesmo casal esperam um atrás do outro para usar o forno;
- as regras de preferência definidas acima valem a não ser que um casal tenha que ceder a vez;
- deadlock deve ser resolvido pela atuação do Raj;
- inanição deve ser evitada pela regra do casal ceder a vez.

Para implementar a solução, analise as ações de cada personagem em diferentes circunstâncias, como o outro membro do casal já está esperando, há alguém com mais prioridade esperando, há alguém com menos prioridade esperando. Determine o que fazer quando alguém decide que quer usar o forno e tem alguém de maior prioridade, o que fazer quando terminam de usar o forno, etc.

Consulte as páginas de manual no Linux para entender as funções da biblioteca para mutexes e variáveis de condição:

- `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, pthread_mutex_t *mutex)`
- `int pthread_cond_destroy(pthread_cond_t *cond);`

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

```
monitor forno
{
  ... // variáveis compartilhadas, variáveis de condição

  esperar(personagem p) {
    printf("%s quer usar o forno\n", nome(b.num));
    ... // verifica quem mais quer usar, contadores, variáveis de cond., etc.
  }

  usar(personagem p) {
    printf("%s começa a esquentar algo\n", nome(b.num));
    ... // verifica preferência, usa contadores, variáveis de cond.
    sleep(1); // tempo de usar o forno
  }

  liberar(personagem p) {
```

```
    printf("%s vai comer\n", nome(b.num));
    ... // verifica se tem que liberar alguém, atualiza contadores, etc.
  }

  verificar() {
    ... // Raj verifica se há deadlock e corrige-o
  }
};
```

Os personagens com namoradas(os) executam as seguintes operações um certo número de vezes (definido pelo parâmeto de execução do programa):

```
forno.esperar(p);
forno.usar(p);
forno.liberar(p);
comer(p);          // espera um certo tempo aleatório
```

Use `drand48()` (confira a página do manual) para gerar números aleatórios entre 0 e 1, use uma multiplicação para gerar inteiros aleatórios.

O Raj executa as seguintes operações:

```
enquanto personagens estão ativos faça
  sleep(5);
  forno.verificar();
```

Informações úteis

Forma de operação

O seu programa deve basicamente criar uma threa para cada personagem e esperar que elas terminem. Cada personagem executa um *loop* um certo número de vezes (parâmetro da linha de comando), exceto Raj, que deve executar seu loop até que todos os outros personagens tenham acabado.

Codificação dos personagens

Você deve buscar ter um código elegante e claro. Em particular, note que o comportamento dos casais é basicamente o mesmo, você não precisa replicar o código para diferenciá-los. Além disso, o comportamento de todos os personagens (exceto o Raj) é tão similar que você deve ser capaz de usar apenas uma função para todos eles, parametrizada por um número, que identifique cada personagem. As prioridades podem ser descritas como uma lista circular.

Acesso às páginas de manual

Para encontrar informações sobre as rotinas da biblioteca padrão e as chamadas do sistema operacional, consulte as páginas de manual *online* do sistema (usando o comando Unix `man`). Você também vai verificar que as páginas do manual são úteis para ver que arquivos de cabeçalho que você deve incluir em seu programa. Em alguns casos, pode haver um comando com o mesmo nome de uma chamada de sistema; quando isso acontece, você deve usar o número da seção do manual que você deseja: por exemplo, `man read` mostra a página de um comando da shell do Linux, enquanto `man 2 read` mostra a página da chamada do sistema.

Processamento da entrada

Não há entrada neste programa além do parâmetro de execução do programa.

Certifique-se de verificar o código de retorno de todas as rotinas de bibliotecas e chamadas do sistema para verificar se não ocorreram erros! (Se você ver um erro, a rotina `perror()` é útil para mostrar o problema.) Você pode achar o `strtok()` útil para analisar a linha de comando (ou seja, para extrair os argumentos dentro de um comando separado por espaços em branco).

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 e `argv` é um vetor de *strings*, ou de apontadores para caracteres.

Processo 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 um personagem e certifique-se de que ele funciona. Depois dispare dois personagens apenas, para evitar que deadlocks aconteçam. Verifique se os casais funcionam, inclua o Raj e verifique se deadlocks são detectados (use um pouco de criatividade no controle dos tempos das outras atividades para forçar um deadlock). Finalmente, certifique-se que o mecanismo de prevenção da inanição funciona (p.ex., use apenas dois casais de altere os tempos das outras atividades para fazer com que um deles (o de maior prioridade) esteja sempre querendo usar o forno. Exercite bem o seu próprio código! Você é o melhor (e neste caso, o único) testador dele.

Mantenha versões do seu código. Programadores mais avançados utilizam um sistema de controle de versões, tal como RCS, CVS ou SVN. 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.

O que deve ser entregue

Você deve entregar no moodle um arquivo .zip ou .tgz contendo o(s) arquivo(s) contendo o código fonte do programa (.c e .h), um `Makefile` e um relatório sobre o seu trabalho, que deve conter:

- Um resumo do projeto: alguns parágrafos que descrevam a estrutura geral do seu código e todas as estruturas importantes.
- Decisões de projeto: descreva como você lidou com quaisquer ambiguidades na especificação.
- Bugs conhecidos ou problemas: uma lista de todos os recursos que você não implementou ou que você sabe que não estão funcionando corretamente

Não inclua a listagem do seu código no relatório; afinal, você já vai entregar o código fonte!

Finalmente, embora você possa desenvolver o seu código em qualquer sistema que quiser, certifique-se que ele execute corretamente na máquina virtual com o sistema operacional Linux que foi distribuída no início do curso. A avaliação do seu funcionamento será feita naquele ambiente.

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 algumas dezenas de linhas de código, talvez umas poucas centenas. Se você se ver escrevendo código mais longo que isso, provavelmente é uma boa hora para parar um pouco e pensar mais sobre o que você está fazendo. Entretanto, dominar os princípios de funcionamento e

utilização das chamadas para manipulacão de variáveis de condição e mutexes e conseguir a sincronização exata desejada pode exigir algum tempo e esforço.

1. **Dúvidas:** usem o moodle (minha.ufmg).
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. Vão valer pontos clareza, qualidade do código e da documentação e, obviamente, a execução correta da chamada do sistema com programas de teste. A participação nos fóruns de forma positiva também será considerada.

Última alteração: May 2, 2013