Software de Controle para Robótica Cirúrgica

Autoria do projeto


```
Nome : Dayanne Fernandes da Cunha
Matrícula : 13/0107191
```


Introdução

Existem problemas computacionais os quais é necessário que se utilize estratégias mais elaboradas para solucionar estes. Por muitas vezes, uma execução serial de um determinado problema pode não somente aumentar o tempo de execução, mas também pode desconfigurar a lógica de sua solução.

A programação concorrente é uma destas estratégias utilizadas para resolver alguns casos críticos, como por exemplo, problemas de condição de corrida. Nestes casos é necessário que exista algum escalonamento quando dois ou mais processos tentam acessar a mesma região crítica simultaneamente.

Uma região critica é caracterizada como parte do código onde é realizado o acesso de um recurso compartilhado. Para evitar o acesso destas regiões críticas por mais de um processo simultaneamente é utilizado soluções como exclusão mútua. A exclusão mútua proibe que mais de um processo acesse estes recursos compartilhados ao mesmo tempo.

Para criar e/ou manipular rotinas que possam ser executadas intercaladas ou de forma concorrente na linguagem C/C++ é utilizado a biblioteca POSIX Threads.

Neste projeto iremos tratar de um problema concorrente na área de controle de robótica cirúrgica. Será utilizada a biblioteca `POSIX Threads` e a linguagem `C++11` para solucionar o problema proposto.

Problema

Para movimentar um robô cirúrgico é necessário elaborar algum software que auxilie no controle das direções que serão emitidas para os servomotores. Este software se comportará como uma interface que conecta um dispositivo emissor de comandos (joystick, pedal, comando de voz, teclado, etc) com os dispositivos físicos (servomotores) que executarão os movimentos cirúrgicos na(o) paciente.

Responsabilidades da interface :

- Informar aos servomotores qual direção o dispositivo emissor de comandos enviou a eles
- Identificar o estado de conexão dos servomotoreses
- Evitar que dois ou mais motores se movimentem ao mesmo tempo
- Gerenciar um buffer de comandos e um protocolo de execução deste

Quando uma direção for acionada pelo dispositivo emissor de comandos, o servomotor só irá se movimentar se estiver conectado e se nenhum outro servomotor estiver em movimento no momento.

Caso algum servomotor esteja em execução, então o software de controle irá guardar em uma fila de M espaços as direções acionadas. Ele irá executar os movimentos da fila de acordo com que as execuções dos servomotores forem finalizando.

Dado um número limite M para o tamanho máximo do buffer de comandos, caso o buffer se encha em algum momento, então os novos comandos emitidos serão descartados.

Solução

Concorrência no problema/solução

Em um código serial não seria possível receber um comando do `Joystick`, processá-lo, executá-lo e ainda esperar outros comandos durante a etapa de processamento e execução de algum comando. Portanto são utilizadas `Threads` para o controle de etapas que podem funcionar de formar intercaladas/concorrentes.

Para o controle de memória compartilhada foram utilizados `Locks`. Abaixo temos exemplos de recursos controlados por esta ferramenta :

- Pilha de comandos emitidos pelo `Joystick`
- Estados de conexão dos servomotores

Já para o controle de escalonamento das etapas foram utilizados `Semáforos`. Os `Semáforos` foram utilizados para resolver as seguintes situações :

- Envio de comando do `Joystick` à `Interface`

- Controle da pilha de comandas, com cautela para que o limite M do tamanho máximo do buffer não fosse ultrapassado
- Controle de bloqueio dos motores enquanto um determinado motor executa algum comando

Foram criadas as seguintes `Threads` de controle no software :

- 1 do `Joystick`
- 3 dos 3 graus de liberdade dos `Servomotores`
- Uma para controle de `Conexão`
- M da `Interface` de controle entre o `Joystick` e os `Servomotores`, tal que M é igual ao tamanho máximo do buffer de comandos de entrada

Informações gerais do software

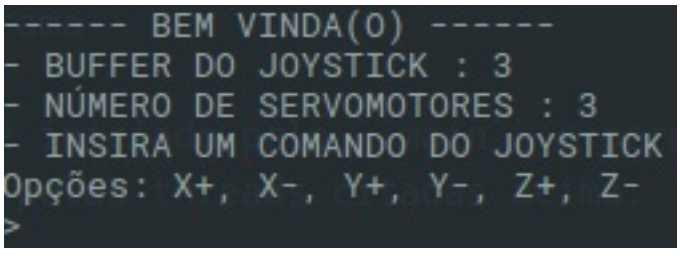
O teclado foi escolhido como dispositivo emissor de comandos, porém, no software iremos denominá-lo por `Joystick`. O `Joystick` poderá emitir 6 direções, sendo elas `X+, X-, Y+, Y-, Z+, Z-`. O software controla 3 graus de liberdade de servomotores, `X, Y, Z`. A pilha de controle dos comandos é empilhada/desempilhada através do protocolo FIFO (**First In First Out**) de pilhas.

Execução e Interface do Usuário

Para executar o código basta executar os seguintes comandos :

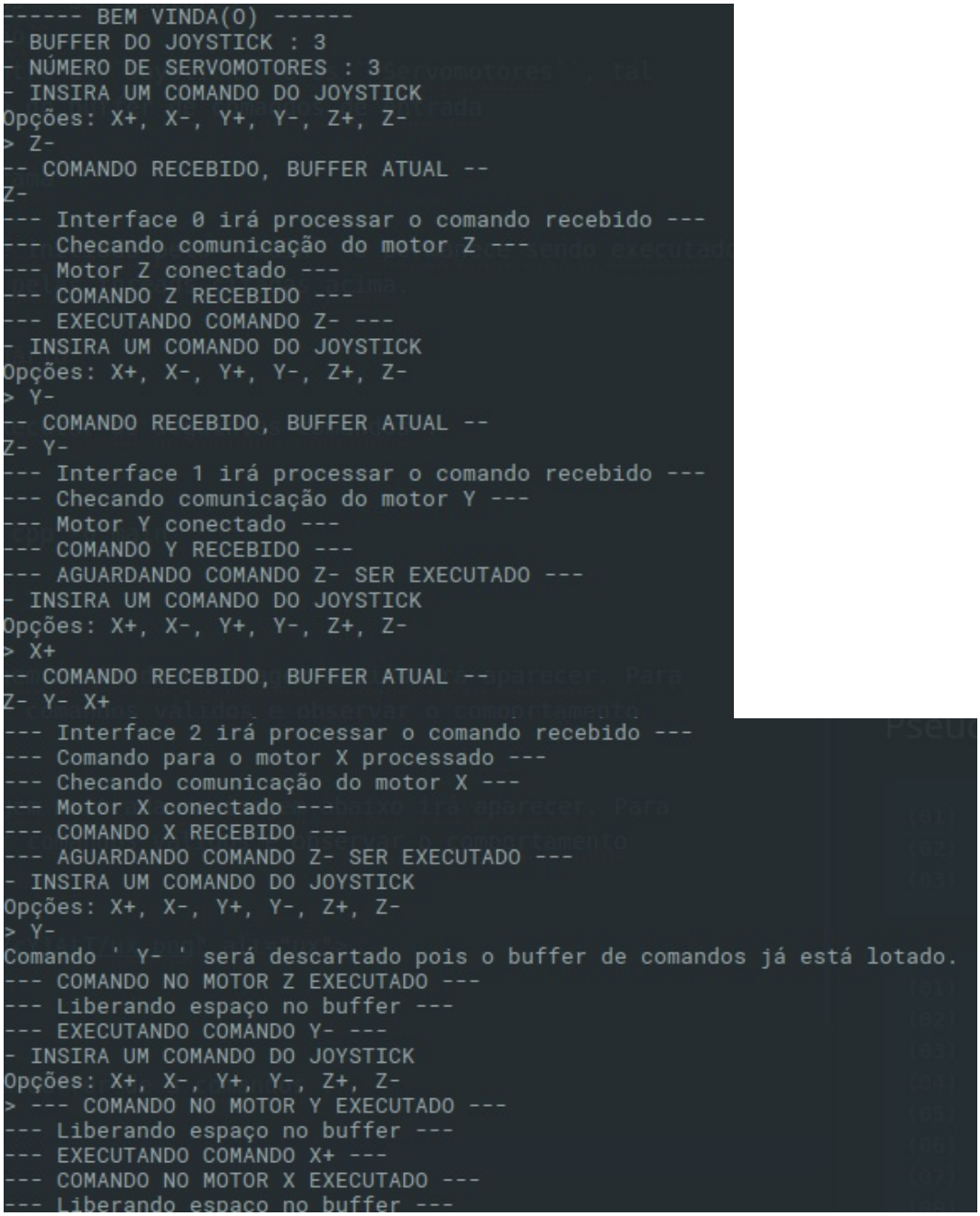
```
$ g++ -pthread -std=c++11 main.cpp -o main
$ ./main
```

Ao executar o programa a mensagem mostrada na imagem abaixo irá aparecer. Para seguir o programa basta inserir comandos válidos e observar o comportamento do software de controle :



Exemplo de execução :

- Inserção de 4 comandos (`Z-, Y-, X+, Y-`) com um buffer de 3 comandos :



Pseudocódigo

```
(01)  main:
(02)    inicia semaforos
(03)    cria threads
```

```
(01)  joystick:
(02)    while (True)
(03)      recebe comando
(04)      valida comando
(05)      se buffer nao estiver cheio
(06)        empilha comando no buffer
(07)        envia (comando) para interface
```

```
(08)      senao
(09)        descarta comando
```

```
(01)  interface:
(02)    while (True)
(03)      espera (comando) do joystick
(04)      lock (comando)
(05)      identifica comando
(06)      lock (conexão)
(07)      se motor cujo comando será executado estiver conectado
(08)        unlock (conexão)
(09)        envia (comando_x ou comando_y ou comando_z) ao motor correspondente
(10)      unlock (comando)
```

```
(01)  motor_x (ou motor_y ou motor_z):
(02)    while (True)
(03)      tenta esperar (comando_x ou comando_y ou comando_z)
(04)        lock (comando executando)
(05)        executa comando
(06)        desempilha comando do buffer
(07)        unlock (comando executando)
(08)      senao
(09)        lock (conexao)
(10)        conexao_x = true (ou motor_y = true ou motor_z = true)
(11)        unlock (conexao)
(12)        sleep(2)
```

```
(01)  controle_conexao:
(02)    while (True)
(03)      lock (conexao)
(04)      conexao_x, conexao_y, conexao_z = false
(05)      unlock (conexao)
(06)      sleep (30)
```

Conclusão

O problema de controle de robótica cirúrgica é muito mais sofisticado do que foi apresentado neste projeto, porém foi possível simular o caso mais simples de um exemplo real de aplicação de programação concorrente. A disputa de memória compartilhada não somente afeta a perfomance do código como também a segurança do usuário. Sistemas delicados como este exige-se que seja elaborado protocolos de escalonamento mais precisos e ótimos para minimizar ao máximo possíveis **deadlocks** e **starvation** que possam vir a ocorrer durante a execução de sistemas.

Referências

- Bibliografia base da matéria, Programação Concorrente, professor Alchieri