UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO

Relatório 3 da disciplina de
Sistemas Operacionais em Tempo Real

PROJETO DE UM SIMULADOR PARA CONTROLE DE ROBÔ MANIPULADOR UTILIZANDO COMUNICAÇÃO CLIENTE-SERVIDOR

Autor: NELSON ROBERTO WEIRICH JUNIOR
Professor: Osmar Marchi dos Santos

Santa Maria, RS
2 de julho de 2018

1 OBJETIVO

Desenvolver um simulador de software embarcado contendo duas partes:

- Interface de controle (monitor): que envia e recebe informações via rede para o software simulador;
- Simulador: software simulando um programa embarcado, onde deve se implementar no mínimo 3 tarefas periódicas e utilizar noções de concorrência (*mutex* e variáveis de condição).

1.1 REQUISITOS

- Implementação de (no mínimo) 3 tarefas periódicas;
- Comunicação em rede;
- Informação concorrente e controle de concorrência (no mínimo, uso de 1 *mutex* e 1 variável de condição);
- Envio e recepção de comandos da interface de controle para o simulador (no mínimo 3 comandos enviados entre interface de controle e simulador).

2 INTRODUÇÃO

Um robô pode executar muitas tarefas de acordo com o ambiente onde é inserido. Além disso, é preciso que haja controle de estabilidade quando esse está parado ou mesmo em movimento. Ele pode, também, ser comandado remotamente ou ser totalmente autônomo em suas tarefas. Robôs para manipulação de cirurgias ou para solda de peças industriais são alguns exemplos de robô controlado remotamente.

Para alcançar o objetivo deste trabalho, será desenvolvido um sistema de comunicação cliente-servidor para simulação de um robô manipulador de coordenadas cartesianas controlado remotamente. Esse será um sistema simulado, ou seja, as tarefas finais não serão executadas para acionamento de mecanismo algum, será somente uma representação do funcionamento de um sistema real.

O usuário poderá interagir com o a máquina utilizando-se de palavras-chave para definir a tarefa a executar. Enviando comandos através de uma interface de comunicação, o usuário controla as ações da máquina e recebe respostas dos sensores de estado do sistema. Dessa forma, serão necessárias tarefas para controle de comunicação e tarefas para controle do sensoriamento do sistema.

3 DESENVOLVIMENTO

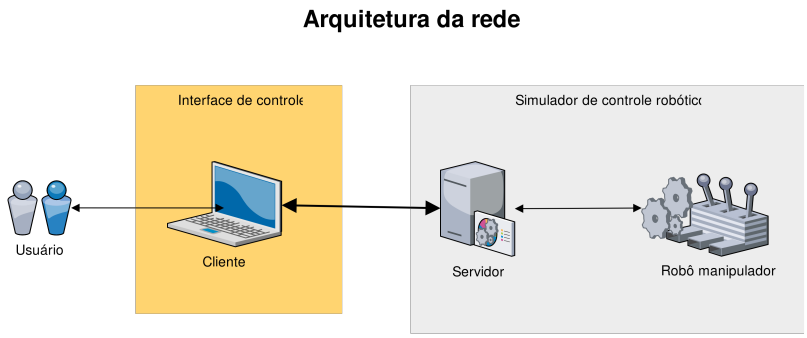
Serão criadas três tarefas periódicas as quais serão responsáveis por manter a estabilidade do robô nos três eixos e informar a posição da garra robótica.

Em nível de implementação, os comandos modificam variáveis globais de estado do sistema que serão acessadas pelas tarefas periódicas descritas. Para garantia de acesso a estas variáveis, será utilizado a noção de *mutex* e variáveis de condições. Estas tarefas serão escalonadas conforme a prioridade que exerce sobre o sistema, seguindo conforme um sistema de tempo real.

O usuário terá duas opções de interface de comunicação: CLI (*command line interface*) ou GUI (*graphical user interface*) – para facilitar a interação com o simulador/sistema. Para interface GUI será utilizada a biblioteca GTK para linguagem C.

A estrutura do sistema que será implementado é mostrada na Figura 3.1.

Figura 3.1 – Estrutura do sistema a ser implementado.



Fonte: Elaborada pelo autor.

Enquanto as tarefas estão sendo escalonadas e executadas no simulador, o robô mantém sua estabilidade garantida pelas mesma tarefas em tempo de deadline. Quando o controlador envia um comando via terminal (interface cliente), o servidor recebe e filtra-o para comandar a ação relacionada. Neste ponto, é interrompida a execução da tarefa corrente e, com garantia de acesso a uma seção crítica e condicionado a uma variável específica, uma mudança do estado do sistema é realizada de acordo com o comando enviado. O servidor envia uma resposta ao cliente sobre a operação realizada. O sistema retorna ao seu estado de execução normal.

Serão definidos os seguintes comandos iniciais para controle do sistema:

- **right** – movimenta-se para a direita
- **left** – movimenta-se para a esquerda
- **forward** – movimenta-se para a frente

- **back** – movimenta-se para trás
- **stop** – cessa o movimento
- **exit** – desconecta do simulador

Os comandos de movimento devem possuir um parâmetro que define a quantidade do movimento que deverá ser executado.

Mais alguns detalhes de implementação poderão ser acrescentados conforme o projeto é desenvolvido. Portanto, serão acrescentados nos relatórios seguintes.

4 IMPLEMENTAÇÃO

4.1 PARTE I

Partindo das ideias descritas, foi escrito o código que segue junto a este documento. Esse é um sistema primitivo que funciona conforme as especificações. Porém, há melhorias a serem feitas até a entrega final, como, por exemplo, melhorar a organização de algumas partes e tratar informação dentro das tarefas criadas.

Foram criadas as três tarefas periódicas: uma para controle de posição linear, uma para controle de posição angular e outra para controle de velocidade. Para cada tarefa, até o momento, dentro de um laço infinito, é impresso o valor da variável a partir de uma condição definida como sendo diferente de zero (esta condição será melhorada até o final do projeto). Essa parte de código está protegida utilizando *mutex*, pois a variável só pode ser acessada com garantias de que ela não está em uso por outra tarefa do sistema.

Dentro do código do servidor, foi realizado o tratamento dos parâmetros de entrada e, após a validação, são setadas as configurações da conexão e configuração do conjunto de sinais, e são criadas as três tarefas periódicas. A partir disso, a execução entra em um loop infinito onde, inicialmente, fica aguardando uma conexão com cliente através da chamada de função *accept()*. No código do cliente, são tratados os parâmetros de entrada e feita a configuração da conexão da mesma forma. Após isso, é feita a conexão através da função *connect()*.

Quando a conexão é estabelecida, o servidor cria uma *thread* para a mesma, a qual será responsável por manter a comunicação cliente-servidor. Ao iniciar esta comunicação, o servidor envia uma mensagem de entrada para o cliente que fica aguardando esse envio para criar a *thread* de comunicação com o servidor. Neste momento, o cliente fica esperando comandos do usuário, enquanto que o servidor aguarda envio de dados do cliente.

O usuário entra com um comando, o cliente envia-o para o servidor e fica aguardando resposta do mesmo. O servidor recebe a informação, processa e executa algo conforme o que foi enviado (se for algo fora do conjunto de comandos, é recomendado ao usuário digitar a opção help para ajudá-lo). Para os comandos *right* e *left*, a variável de posição angular é incrementada ou decrementada. Para os comandos *forward* e *back*, a variável de posição linear é incrementada ou decrementada. E para o comando *stop*, a variável de velocidade é zerada (será feito algo para alterar a velocidade para mais ou para menos do robô). Para todos esses tratamentos há garantias de acesso às variáveis através de *mutex*.

Assim, após o tratamento, o servidor responde ao cliente com uma mensagem que será mostrada ao usuário em seu terminal. Quando o usuário quiser desconectar do servidor, basta digitar ***exit***. O servidor irá interpretar este comando uma finalização de conexão e enviará para o cliente uma mensagem com a mesma palavra, **exit**. O cliente, ao receber este dado, irá

interpretá-lo como final de conexão. Portanto, ambos irão encerrar a execução da sua *thread* corrente. Desta forma, o cliente irá encerrar a execução do programa principal e finalizará. Por outro lado, o servidor, ao retornar ao programa principal, permanece dentro do laço infinito e aguardará uma nova conexão (função *accept()*). Agora, um novo usuário poderá tentar nova uma conexão com o servidor que iniciará o mesmo processo.

Enquanto esta comunicação é realizada, as tarefas periódicas ficam executando de tempos em tempos, seguindo a regulagem estabelecida para cada uma. Quando em execução, ela envia ao terminal o valor presente na variável respectiva ao seu monitoramento. No terminal do servidor, esta informação aparece entre as informações de conexão com o cliente. Isto será melhorado para o produto final.

Por motivos de tempo de desenvolvimento e prazos de entrega, não será implementada a interface gráfica. Dessa forma, somente será apresentada a interface por linha de comando, CLI, desenvolvida até então.

4.2 PARTE II

Nessa segunda etapa, foi melhorada a organização do código. Foi adicionada a biblioteca *lperiodic* para controle da periodicidade das tarefas definidas, melhorada a execução das tarefas periódicas e corrigido alguns erros de implementação.

O funcionamento geral do sistema permaneceu o mesmo. Dessa forma, o usuário insere um comando para controlar os movimentos do robô e, através da tarefa do cliente, é enviado o mesmo para o servidor. Esse é responsável por filtrar o comando e executar uma ação de acordo com o que for necessário. Então, ao final, ele retorna uma mensagem ao cliente que a repassa para o usuário.

Foi pensado em adicionar, no lugar das impressões de terminal do servidor, ações para envio de comandos a uma porta serial onde haveria um dispositivo que pudesse tratar isso, como um microcontrolador. Assim, seria uma simulação mais voltada à uma aplicação, onde o servidor teria o controle de uma terceira conexão (o microcontrolador) para envio de ações.

A organização dos diretórios é separada por aplicação. Um diretório responsável pelo armazenamento dos códigos do cliente, com separação de bibliotecas e códigos-fonte. E outro diretório responsável pelo armazenamento dos códigos do servidor, com separação de bibliotecas e códigos-fonte, da mesma forma.

Uma documentação mais completa do projeto pode ser encontrada em robot-controller.

5 CONCLUSÕES

Para construir um sistema para controle de robô manipulador utilizando conexão cliente-servidor, foram desenvolvidos dois principais programas em código de linguagem C. Esses são responsáveis pela comunicação cliente-servidor. O cliente fica encarregado de receber comandos do usuário e enviá-los para o servidor e de receber a resposta desse e enviá-la para o usuário de volta pelo terminal. O servidor é responsável por atender aos pedidos que o cliente envia em forma de comandos dentro do conjunto estabelecido.

Seguindo as especificações, foram criadas as três tarefas periódicas, que são responsáveis por controlar três variáveis de estado: posição linear, posição angular e velocidade. Para garantias de acesso a estas variáveis globais evitando resultados indesejados, foi utilizado o conceito de *mutex*.

Na segunda parte de desenvolvimento, o sistema já encontrava-se em fase de funcionamento, porém era um sistema primitivo. Para a finalização dele, foram modificadas algumas partes de código. Devido ao desenvolvimento de outros projetos em paralelo, a parte de interface gráfica ficou comprometida. Mas com uma boa interface de linha de comando no terminal na forma mais robusta e clássica, para um programador, pode ser melhor do que uma interface gráfica amigável. O contrário é melhor para apresentar um sistema mais didático a quem quer controlar um robô da forma mais simples e direta. No entanto, o funcionamento em si estará melhor finalizado.

O sistema final ficou conforme proposto e como planejado previamente. Portanto, é possível controlar remotamente um robô manipulador de coordenadas cartesianas, pelo menos em nível de simulação. O mesmo sistema pode ser aplicado em um controle real com garantias temporais, onde é necessário o uso dos conceitos de Sistemas Operacionais em Tempo Real.

REFERÊNCIAS BIBLIOGRÁFICAS

KERRISK, M. **Linux man pages online**. 2018. Disponível em: <http://man7.org/linux/man-pages/index.html>.

PROJECT, T. G. **Getting Started with GTK+**. Disponível em: <https://developer.gnome.org/gtk3/stable/gtk-getting-started.html>.

SIMPLíCIO BEATRIZ RêGO LIMA, J. A. J. P. V. G. **Manipuladores Robóticos Industriais**. 2016. Disponível em: <https://periodicos.set.edu.br/index.php/cadernoexatas/article/viewFile/3572/1950>.