SIMULADOR DE MONITORAMENTO E CONTROLE DE UM INVERSOR FOTOVOLTAICO

25/08/21

João Felipe Amaral Santiago
Matrícula: 201720884

ELC1095
SISTEMAS OPERACIONAIS DE TEMPO REAL

monitor.c : **IMPLEMENTADO**
inversor.c : **IMPLEMENTADO**
painel.c **: IMPLEMENTADO**

painel.c

Resumo Geral

O código é capaz de simular, registrar e compartilhar todas as variáveis físicas (tensão, corrente, temperatura e potência), com o inversor. Para compartilhar dados, o software configura o socket necessário para realizar a comunicação com o código `inversor.c`.

A simulação das variáveis físicas se dá por meio de incrementos e decrementos periódicos. De tal forma que há intervalos de tempo específicos para amplificação e decaimento. Para as variáveis elétricas há três fatores que influenciam a modificação de seus valores: `var_gauss`, `grow_flag` e `eletric_increment`. Inicialmente, `var_gauss` vale -1, o que significa que essa variável deve incrementar seus valor, portanto `grow_flag=1`. A variável gaussiana cresce por um fator = `eletric_increment` (explicado em detalhes mais adiante) até que atinja o valor de 1, quando tal patamar é alcançado a curva deve iniciar seu decaimento `grow_flag=0`, até atingir o valor de -1 e voltar a crescer novamente.

A partir de var_gauss tem-se que:

Na amplificação:

```
tensão     = var_gauss*5
corrente   = var_gauss*10
```

No decaimento:

```
tesão     = var_gauss*(-0,005)
corrente = var_gauss*(-0,01)
```

A alteração dos valores da variável `temperatura` é diferente das variáveis elétricas. Além dos valores de crescimento e redução, um timer é utilizado para criar a periodicidade, de tal modo que nos momentos de crescimento a seguinte fórmula é utilizada:

```
temperatura+=( (potência + hora_atual) - refrigeração )/1000
```

Como os valores de potência ficam na casa das centenas à medida que o horário se aproxima das 13hrs, optou-se por uma divisão por 1000 para simular pequenas alterações de temperatura entre um minuto e outro.

A lógica para o decremento é a seguinte:

```
if(hra >=14){
  if(temp > -5){
    inc  = (( pot + (hra-reducer_factor) + ref_lic_local ))/1000000;
    temp -= inc;
    reducer_factor+=2;
  }
}else
  if(temp > -5)
    inc   = (( pot + hra + ref_lic_local ))/1000000;
    temp -= inc;
```

Nota-se que a única mudança se dá no fato da existência da variável `reducer_factor`, a qual tem como objetivo aproximar o valor de temperatura entre os horários 14 e 12, 15 e 11, 16 e 10, e assim sucessivamente. Ou seja, se por exemplo, às 10hrs ocorreu um incremento X, então às 16hrs deve ocorrer um decremento X.

Outra grande responsabilidade do painel é a de se comunicar com o inversor. Portanto, instanciou-se um socket o qual funciona como servidor, pois ele escuta na porta 8081 aguardando requisições do inversor. Após configurar o socket, já na thread que irá receber os dados de requisição, inicia-se o processo de registrar as referências dos valores das variáveis que o painel irá eventualmente transmitir no seu canal de comunicação. Após essa etapa, o código armazena no array `code_string[]` as strings que irão determinar qual variável deve ser transmitida.

Com o canal de comunicação estabelecido, a thread irá aguardar por requisições. Uma requisição de leitura é organizada da seguinte forma:

```
                    |MODO| |NOME|
|MODO|  : Leitura.
|NOME|  : Pode ser tensão, corrente, potência ou temperatura.
```

A resposta para esse tipo de requisição é a que se segue:

```
                    |NOME| |VALOR|

|NOME|  : Pode ser tensão, corrente, potência ou temperatura.
|VALOR| : Valor da variável requisitada.
```

Uma requisição de leitura tem a seguinte estrutura:

```
                    |MODO| |VALOR|
```

```
|MODO|  : Escrita.
|VALOR| : Valor da variável.
```

Sendo assim, quando a requisição é de escrita, basta atribuir o valor à variável de refrigeração. Quando é de leitura, o código deve descobrir qual é o nome da variável de interesse e seu respectivo valor. Após esta etapa uma mensagem é enviada contendo o nome da variável que foi requisitada e o seu valor mais recentemente calculado pelo painel.

Período dos timers, portas de comunicação e IP do inversor e painel podem ser alterados nas diretivas.

Organização

main()

Inicializa socket, threads, arquivos, sinais, temporizador e algumas variáveis. Ela também simula o comportamento das variáveis elétricas e escreve seus valores em arquivos.

Fluxo de funcionamento

1. Configura o socket que irá ouvir na porta 8081, esperando requisições de leitura das variáveis físicas, ou escrita na variável de refrigeração.

2. Define arquivos para cada variável física, com o intuito de viabilizar a verificação do comportamento das mesmas. O objetivo é que haja uma variável análoga à uma curva gaussiana para que, a partir dela, definir os valores de tensão, corrente e potência.

3. Faz uma configuração prévia dos sinais que serão utilizados no temporizador das medidas das variáveis elétricas e da temperatura.

4. Cria thread para simular temperatura e thread para receber dados via socket.

5. Cria temporizador para simular comportamento das variáveis elétricas.

6. Define parâmetros de incremento da curva gaussiana.

7. Aguarda sinal do timer.

8. Abre os arquivos das variáveis elétricas.

9. Incrementa ou decrementa curva “gausseana”.

10. Calcula corrente, tensão e potência.

11. Escreve os valores elétricos e o valor da curva em seus respectivos arquivos.

12. Fecha os arquivos.

13. Volta ao passo 7.

receive_info()

Thread que escuta na porta 8081. Ela deve aguardar por requisições do inversor, para então modificar o valor da refrigeração ou enviar dados das variáveis físicas.

Fluxo de Funcionamento

1. Armazena referência das variáveis físicas.

2. Define string associada a cada variável física.

3. Aguarda requisição de conexão do inversor.

4. Aguarda requisição de dados.

5. Descobre se a requisição é de leitura ou de escrita.

6. Se for escrita, modifica o valor da refrigeração e volta ao passo 4; do contrário, algum valor das variáveis físicas terá de ser entregue ao inversor.

7. Descobre qual variável física o inversor quer ler.

8. Cria mensagem do tipo “`nome_da_variavel: valor_da_variavel`”.

9. Envia mensagem ao inversor.

10. Volta ao passo 4.

temperatura()

Simula o comportamento da temperatura. De tal modo que o objetivo é que ela aumente das 7hrs até as 14hrs, fora desses horários a temperatura deve naturalmente reduzir. A temperatura calculada em cada simulação é escrita em um arquivo adequado.

Fluxo de Funcionamento

1. Configura timer.
2. Abre ou cria arquivo de temperatura.
3. Aguarda sinal do timer.
4. Lê valor de potência e refrigeração.
5. Caso 7 < hora < 14, aumenta a temperatura com o desconto da refrigeração.
   Do contrário, reduz a temperatura com o incremento da refrigeração.
6. Aumenta minuto em 1.
7. Verifica se min e hora devem ser setados.
8. Escreve valor da temperatura calculada no arquivo temp.txt
9. Atualiza o valor da temperatura global.
10. Volta ao passo 3.

Erros, Problemas & Limitações

- As curvas das variáveis físicas não ficaram satisfatórias, pois o objetivo era que as tais se aproximassem de um comportamento gaussiano.

Aspectos de Implementação Relevantes

° A curva “gaussiana”, que se parece mais com uma dente de serra com ângulo < 90° na subida precisa ser tal forma que seu valor de pico (1) seja às 13 hrs. Portanto, o valor de incremento/decremento precisa viabilizar esse objetivo.

Sabendo que a curva começa em -1 às 00:00 hrs, quando o sistema começa, conclui-se que serão necessários 780 min para se chegar às 13hrs. Com isso é preciso saber em quanto tempo o timer da temperatura chegará nesse horário. Sabendo disso, é possível descobrir quantos ciclos o timer das variáveis elétricas precisarão para chegar no horário objetivo. No final, o incremento será o intervalo de valores dividido pela quantidade de ciclos que este intervalo deve ser alcançado.

Ou seja:

```
temp_time           = 780*temp_periodo;
eletric_ciclos      = temp_time/eletric_periodo;

incremento_da_curva = (1-(-1))/eletric_ciclos;
```

° A curva “gaussiana” aumenta até alcançar o valor 1, após isso ela inicia seu decremento até o valor -1, para então repetir o processo.

° Os valores presentes nos arquivos criados pelo software podem ser utilizados para verificar o comportamento das variáveis simuladas.

Componentes

Structs:

° `struct periodic_info:` Struct que viabiliza a configuração do timer das threads que usam cronômetro, de tal forma que cada uma tenha sinais distintos para seus respectivos timers.

° `struct inv_socket_infos:` Struct que tem seus valores atribuídos na main, os quais futuramente serão passados como parâmetro para a thread receive_info, ou seja a main faz a configuração do socket que irá aguardar por requisições do inversor, ao passo que a thread `receive_info` apenas usa esses dados para fazer a posterior conexão.

Funçẽos:

° `read_buffer():` atribui à string `aux1,` X caracteres da string `r_buffer.` A quantidade de chars é exatamente o tamanho da string `search_string.` Caso `aux1 == search_string, answear = search_string;` do contrário, `answear = "NOP".`

Essa função é utilizada toda vez que se quer descobrir o que a requisição do inversor está exigindo do servidor painel.

° `make_periodic():` Tem como entradas o período e a struct que conterá uma variável do tipo sinal e o conjunto de sinais, o qual ele pertence. A partir disso, a função irá definir o valor do sinal e adicionará no conjunto. Posteriormente haverá um timer que irá setar o sinal da struct de entrada. O período desse cronômetro é o valor da entrada `period` .

Essa função é usada por todas as threads periódicas, não só as de painel.c, mas de todas as demais contidas no software inversor.c

° `wait_period():` Função que operacionaliza a periodicidade, pois internamente ela chama a função `sigwait()`, a qual bloqueia a thread até que o sinal configurado em `make_periodic()` chegue para desbloquear.

Variáveis Globais:

° `potencia:` recurso acessado pelas 2 threads e a main. A main a utiliza para determinar seu valor e escrever nos arquivos de debug, `receive_info` lhe acessa para encaminhar quando o inversor requisita, e a thread `temperatura` a usa para calcular o valor da temperatura instantânea.

°`voltage e current`: recursos utilizados pela main e pela thread `receive_info` pelos mesmos motivos que a variável `potencia`.

°`temperature e ref_lic`: variáveis acessadas nas threads `receive_info` e `temperatura`.

Justamente por tais variáveis globais serem acessadas por diferentes threads, elas exigem que haja mutexes para garantir a exclusão mútua.

inversor.c

Resumo Geral

Possui 4 responsabilidades, recebe comandos do monitor, acessa e escreve valores no painel e gerencia bateria. A thread `inv_commands` se encarrega de, a partir de um canal de comunicação, aguardar requisições do monitor. `read_vars` irá periodicamente realizar leituras no painel, cujo o período simula a duração de uma conversão A/D. `write_refr_value` enviará, periodicamente, o valor da refrigeração para o painel, de acordo com a temperatura limite (definida em `inv_commands)` e a temperatura lida em `read_vars`. Por último, em `battery_consumer` haverá uma avaliação periódica da potência armazenada na bateria e do seu valor após um decremento de consumo.

O funcionamento de `inv_commands` é bastante similar com a thread `receive_info`, de painel.c, porém no canal de comunicação é possível enviar uma quantidade maior de requisições de escrita, o que exige uma pequena alteração na forma como as requisições são recebidas e respondidas. De tal modo que a estrutura de uma requisição de leitura é similar com a de escrita. O que significa que a estrutura de uma requisição é organizada da seguinte forma:

```
|MODO| |NOME| |VALOR|

|MODO|:  Pode ser de escrita ou leitura.
|NOME|: Indica qual recurso será acessado. Pode ser consumo,
        potência armazenada, variável elétrica, ou
        temperatura.
|VALOR|: Usada, quando o modo é de escrita.
```

Com essa organização o monitor será capaz de:

1. Definir temperatura limite, a qual abrirá a válvula,
2. Definir se inversor irá ler tensão ou corrente,
3. Definir se haverá armazenamento ou não da potência lida do painel e
4. Definir se haverá consumo ou não da potência da bateria.

Para as requisições de leitura, o monitor terá acesso a:

1. Temperatura instantânea do painel,
2. Potência instantânea da bateria e
3. Valor instantâneo da variável elétrica que foi escolhida na escrita.

Para interagir com essa estrutura a thread, antes de entrar no leu loop infinito, armazena as referências das variáveis que eventualmente receberão atribuições ou serão lidas, além de armazenar em duas strings, as palavras dos nomes correspondentes aos acessos para escrita e leitura. Por fim, a thread irá aguardar a requisição de conexão de monitor. Dentro do loop descobrirá se a requisição é de escrita ou leitura, para posteriormente saber qual o nome do recurso de interesse para que finalmente seja enviado ou atribuído algum valor para a variável requisitada.

A thread `battery_consumer` nada mais faz do que acessar o valor total da potência armazenada e a flag (setada em `inv_commands`) que indica se o consumo deve ocorrer ou não. Com os devidos acessos em mãos, a thread decrementa a potência da bateria por um fator de 2,5W, caso a potência resultante seja menor ou igual a zero, o decremento não ocorre.

Na thread `write_refr_value` há o acesso da temperatura mais recentemente lida por `read_vars` e da temperatura limite definida em `inv_commands`. Caso a temperatura esteja menor ou igual ao limite, a refrigeração vale 0 (válvula fechada), do contrário, a válvula se abrirá e terá um valor específico de refrigeração o qual respeita a seguinte fórmula:

```
refrigeração = 4*(temperatura_lida -temperatura_limite)
```

Depois de ter seu valor calculado, a refrigeração é encaminhada via socket para o painel.

A última thread presente no inversor é `read_vars`, cuja a principal tarefa é simular uma conversão A/D entre as medições do painel e o sensoriamento do inversor. Sua organização é bastante similar com as demais threads que trocam informações via canais de comunicação, visto que ela é cliente da thread

`receive_info` do código painel.c. O que significa que ela encaminha requisições, respeitando a estrutura que foi demonstrada e explicada na página 3 desta documentação. Apesar da comunicação seguir a mesma lógica que as demais threads que recebem dados via socket, `read_vars` abre arquivos para cada variável que requisita leitura e escreve seu valor em formato txt. Com esses arquivos é facilitada a verificação do comportamento das variáveis físicas e o impacto que a refrigeração tem na temperatura. Outro aspecto peculiar desta thread é que toda a vez que uma leitura de potência é feita, deve-se chamar a função `battery_charge`, a qual avalia se a bateria deve ser carregada ou não com o valor lido. Caso a leitura seja de corrente ou tensão é preciso avaliar qual foi o comando definido pelo monitor, para que então se decida quais das duas será registrada.

Organização

main()

Inicializa socket que será cliente de painel e outro que será servidor de monitor. Cria e configura threads, arquivos e conjunto de sinais.

Fluxo de funcionamento

1. Configura socket que irá encaminhar requisições na porta 8081. Ou seja, o socket que será cliente de painel.

2. Configura socket que irá receber requisições na porta 8080. Ou seja, o socket que será servidor de monitor.

3. Faz uma configuração prévia dos sinais que serão utilizados no temporizador das threads.

4. Cria ou abre arquivos de debug.

5. Fecha arquivos.

6. Cria todas as threads.

7. Faz `join()` de todas as threads.

8. Fecha socket cliente.

inv_commands()

Thread periódica que possui um socket, o qual escuta na porta 8080, aguardando requisições de monitor.

Fluxo de Funcionamento

1. Armazena as referências das variáveis que indicam as condições operacionais dos recursos sujeitos a comandos vindos do monitor.

2. Armazena as referências das variáveis que eventualmente o monitor irá querer ler.

3. Define strings, as quais representarão os nomes dos recursos sujeitos a serem transmitidos no canal de comunicação.

4. Aguarda requisição de conexão.

5. Aguarda requisição de dados.

6. Descobre se requisição é de escrita ou leitura

7. Se for de leitura pula para passo 10, do contrário descobre qual recurso o monitor deseja escrever.

8. Após descobrir o nome do recurso, desreferência a variável adequada, a qual foi armazenada no passo 1 e atribui a ela o valor contido no buffer de comunicação.

9. Volta ao passo 5.

10. Descobre o nome do recurso que o monitor deseja conhecer.

11. Após descobrir o nome do recurso, atribui ao buffer de comunicação o valor da variável desreferenciada, a qual foi armazenada no passo 2.

12. Em uma string, concatena nome do recurso requisitado e seu respectivo valor.

13. Encaminha mensagem para monitor.

14. Volta ao passo 5.

battery_charge()

Gerência consumo da potência existente na bateria.

Fluxo de Funcionamento

1. Configura timer.
2. Inicializa valor de consumo.
3. Acessa flag de consumo.
4. Acessa potência total da bateria.
5. Caso o consumo resulte em uma carga total zero ou negativa, pula para o passo 7.
6. Decrementa potência total.
7. Aguarda sinal de cronômetro.
8. Libera recursos críticos.
9. Volta ao passo 3.

write_refr_value()

Atribui valor de refrigeração. Ou seja, a thread tem um socket que é cliente de painel.

Fluxo de Funcionamento

1. Configura timer.
2. Acessa temperatura recentemente lida.
3. Acessa temperatura limite.

4. Caso a temperatura lida > tempertatura limite, calcula valor da refrigeração, do contrário atribui zero à variável de refrigeração

5. Estrutura mensagem para escrever valor calculado.

6. Faz requisição de escrita no painel.

7. Volta ao passo 2.

read_vars()

Realiza requisições de leitura ao painel, escreve valores lidos em arquivos e em variáveis locais.

Fluxo de Funcionamento

1. Configura timer.

2. Armazena as referências das variáveis que eventualmente serão lidas do painel.

3. Define strings, as quais representarão os nomes das requisições sujeitos a serem transmitidos no canal de comunicação.

4. Registra o nome dos arquivos que serão acessados

5. Encaminha requisição de leitura.

6. Aguarda resposta.

7. Descobre qual requisição foi respondida.

8. Abre arquivo associado a variável respondida.

9. Acessa valor.

10. Caso o nome seja de potência, encaminha o valor para verificar se deve ser armazenado na bateria.

11. Verifica se nome é tensão, em caso negativo pula para passo 15.

12. Caso leitura de tensão não esteja habilitada, pula para o passo 20.

13. Armazena valor de tensão na variável correspondente e salva em arquivo.

14. Pula para passo 20.

15. Caso a leitura não seja de corrente, pula para passo 19.

16. Caso a leitura de corrente não esteja habilitada, pula para o passo 20.

17. Armazena valor de corrente na variável e arquivo correspondente.

18. Pula para passo 20.

19. Armazena valor de temperatura ou potência na variável correspondente.

20. Fecha arquivo.

21. Volta ao passo 5.

Erros, Problemas & Limitações

- O controle da temperatura não foi satisfatório, uma vez que a temperatura do painel oscila entre 2 a 10 graus acima do limite estabelecido no comando enviado pelo monitor.

Aspectos de Implementação Relevantes

° Buscou-se implementar as threads periódicas (`battery_consumer`, `read_vars e write_refr_value`) do inversor de tal forma que houvesse uma execução sequencial e colaborativa entre elas. Portanto, no início do loop infinito de

cada uma há uma variável de condição que impede a sua execução enquanto até que outra thread lhe dê algum sinal de liberação. Para atingir este objetivo a variável `threads_cond` inicia em 0, a qual permite apenas que `read_vars` execute. Essa thread então irá incrementar o valor de `threads_cond` em uma unidade fazer sua computação enviar um sinal de liberação broadcast e liberar o recurso condicional.

Com `threads_cond = 1,` a thread `write_refr_value` estará liberada para executar. O mesmo processo acontece (incremento, computação e liberação), porém agora com `threads_cond = 2`. Agora `battery_consumer` executará, com a diferença que o novo valor para `threads_cond` será 0, liberando a execução de `read_vars` e fechando o ciclo.

Componentes

Structs:

° `struct periodic_info:` Explicada na página 8.

° `struct inv_socket_infos:` Explicada na página 8.

Funçẽos:

° `read_buffer():` Explicada na página 8.

° `make_periodic():` Explicada na página 8.

° `wait_period():` Explicada na página 8.

°`battery_charge():` recebe como parâmetro de entrada o valor a ser incrementado na bateria. Posteriormente, acessa flag e potência total. Verifiqua se a soma irá ultrapassar o valor máximo estipulado no código. Se sim, não executa a soma, do contrário incrementa carga da bateria.

Variáveis Globais:

° `voltage e current:` recursos acessados em `read_vars` e em `inv_commands`. Na primeira thread, as variáveis são usadas para receber o

valor de resposta, vindo do painel. Já na segunda, seus valores são encaminhados para o inversor.

° `temperature`: recurso acessado em `read_vars`, `inv_commands` e `write_refr_value`. Seu uso tanto na segundo, quanto na primeira segue a mesma lógica que os recursos anteriormente explicados. Para a terceira thread, seu valor precisa ser acessado para poder saber qual deve ser o valor da refrigeração.

º `total_pot`: recurso acessado em `read_vars`, `inv_commands` e `battery_consumer`. Na primeira tarefa, a variável poderá ou não receber incremento de acordo com o valor lido de `potencia`. Na segunda, tal recurso é útil quando o usuário requisita acesso ao valor total presente na bateria. Em relação à última thread sua utilidade advém da necessidade de saber se o consumo pode ocorrer ou não.

° `temp_lim`: recurso acessado em `write_refr_value` e `inv_commands`. A primeira tarefa a usa para definir qual deve ser o valor da refrigeração, e a segunda lhe acessa toda vez que o usuário quiser definir qual será a temperatura limite.

° `consume_flag`: recurso acessado em `battery_consumer` e `inv_commands`. No primeiro caso o recurso é usado para saber se o usuário habilitou consumo e para o segundo ele é usado para receber a orientação do usuário.

° `charge_flag`: recurso acessado em `inv_commands` e `battery_charg`. Na primeira situação, a variável irá receber a especificação vinda do usuário. Na segunda, ela será usada para sinalizar se deve ou não ocorrer o armazenamento da potência na bateria.

° `voltage_flag`: recurso acessado em `read_vars` e `inv_commands`. A primeira thread a usará para saber se deve registrar tensão ou corrente. A segundo irá definir o seu valor.

° `threads_cond`: recurso acessado em `read_vars`, `battery_consumer` e `write_refr_value`. As três irão usar esse recurso pelo mesmo motivo, saber se estão aptas ou não para serem executadas.

monitor.c

Resumo Geral

Este código estabelece uma conexão cliente com o inversor para posteriormente apresentar os comandos de escrita e leitura disponíveis ao usuário.

Num primeiro momento o usuário é questionado se deseja ler, configurar ou sair da aplicação. Caso deseje “ler”, o monitor perguntará se ele quer:

1. Conhecer o valor da potência acumulada na bateria.
2. Temperatura instantânea do painel
3. Acessar valor de alguma variável elétrica.

Após escolher as opções 1,2 ou 3 uma mensagem de leitura será encaminhada para o inversor o qual responderá, utilizando a thread `inv_commands`. Após o encaminhamento, o código irá aguardar pela resposta, para depois chamar a função `printBuffer`, a qual apresenta a variável em sua respectiva unidade de medida a partir da resposta transmitida pelo inversor.

Caso o usuário opte por “configurar”, o monitor perguntará se o usuário deseja:

1. Setar valor da temperatura limite.
2. Configurar leitura de tensão ou corrente.
3. Armazenar ou não energia na bateria.
4. Consumir ou não energia da bateria.

Se escolher a opção 1 ele deverá digitar algum valor inteiro qualquer. Se for 2, deverá digitar 1 para ler tensão ou 0 para ler corrente. Se digitar 3, terá que digitar 1 para armazenar ou 0 para não fazê-lo. Caso escolha 4, poderá escolher 1 para consumir ou 0 para não. Para cada decisão que o usuário fizer, uma mensagem requisitando escrita na variável de interesse será enviada para a thread `inv_commands` do código inversor.c.