UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

João Vitor Belmonte Rates

TRABALHO FINAL DA DISIPLINA DE SISTEMAS OPERACIONAIS DE TEMPO-REAL:

Santa Maria, RS

LISTA DE FIGURAS

Figura 1 – Funcionamento das principais threads. 8
Figura 2 – Detecção do veículo em relação à rota. 9
Figura 3 – Comandos do cliente. 12
Figura 4 – Função implementada no software. 16
Figura 5 – Funções auxiliares, com destaque na função `wait_enable_dec`. 17
Figura 6 – Estrutura de arquivos do software. 19

LISTA DE SIGLAS

GPS	global positioning system

SUMÁRIO

1 INTRODUÇÃO **7**
2 SERVIDOR **8**
2.1 THREADS 8
2.1.1 `set_gps()` **8**
2.1.2 `record_data()` **9**
2.1.3 `blocker_tracker()` **9**
2.1.4 `blocker()` **10**
2.1.5 `reduce_speed()` **10**
2.2 SENSORES E DISPOSITIVOS EXTERNOS 10
2.2.1 Sensor GPS **10**
2.2.2 Velocímetro **10**
2.2.3 Limitador de Velocidade **11**
3 CLIENTE **12**
3.1 COMANDOS 12
3.1.1 `GPS read_interval <int:seconds>` **13**
3.1.2 `load_route <string:file_name>` **13**
3.1.3 `locker` **13**
3.1.3.1 `locker conf <int:km_reduc> <int:t_interval> <int:t_tolerance>` 13
3.1.3.2 `locker set enable|disable` 13
3.1.4 `record` **14**
3.1.4.1 `record get <int:start_line> <int:end_line>` 14
3.1.4.2 `record set enable|disable` 14
3.1.4.3 `record snapshot` 14
4 DEMAIS DETALHES **15**
5 IMPLEMENTAÇÃO **16**
5.1 THREADS 16
5.1.1 Organização **16**
5.1.2 Funções e Estruturas Auxiliares **17**
5.2 COMUNICAÇÃO COM CLIENTE 17
5.2.1 Comandos Simples **18**
5.2.2 `load_route <string:file_name>` **18**
5.2.3 `record get <int:start_line> <int:end_line>` **19**
5.3 ESTRUTURA DE ARQUIVOS 19
6 CONCLUSÃO **20**

1 INTRODUÇÃO

Este trabalho visa satisfazer os objetivos de: desenvolver um software embarcado(servidor), implementando no mínimo 2 threads periódicas(1 thread para lidar com a comunicação em rede e 1 thread para lidar com a variável de condição), uso de mutexes e variáveis de condição; desenvolver um software cliente que implemente o mínimo de 5 comandos para comunicação com o servidor.

Para isto propõe-se o desenvolvimento de um software de rastreamento veicular que realize o registro de informações do mesmo(através do velocímetro e GPS) e, além disso, possa verificar quando o veículo saí de sua rota e então reduzir a velocidade do mesmo até que este pare.

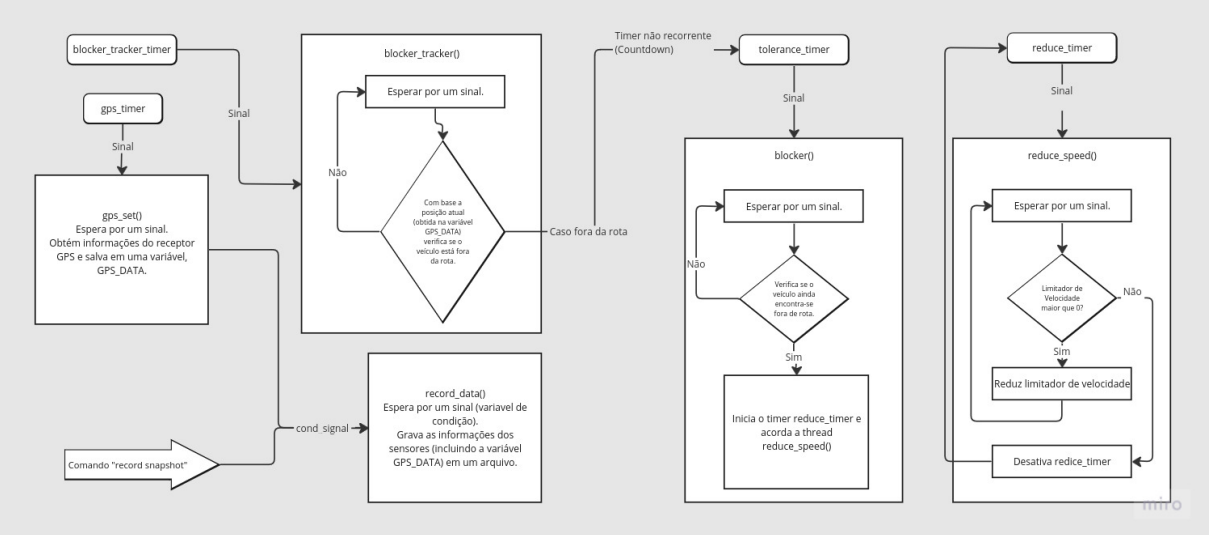
2 SERVIDOR

O software servidor é a principal parte a ser desenvolvida, visto que neste serão aplicados os principais conceitos aprendidos na disciplina de Sistemas Operacionais de Tempo-Real. Será função do software servidor: Registrar periodicamente a posição do veículo, a velocidade instantânea e o limite de velocidade; quando requisitado, enviar as informações registradas ao cliente; verificar, a partir de um arquivo pré-carregado, se o veículo encontra-se na rota determinada; caso o veículo estiver fora da rota definida, reduzir a velocidade do veículo até que este pare completamente.

2.1 THREADS

O servidor contará com as seguintes threads (além das threads necessárias à comunicação com o cliente).

Figura 1 – Funcionamento das principais threads.



Fonte: Elaboração Própria.

2.1.1 `set_gps()`

Está thread executa em um loop e espera até que um sinal seja enviado, quando recebe o sinal a thread grava a posição atual do veículo na variável global GPS_DATA,

está variável será acessada por outras funções, portanto estará associada a um mutex para garantir o acesso a atômico durante a escrita.

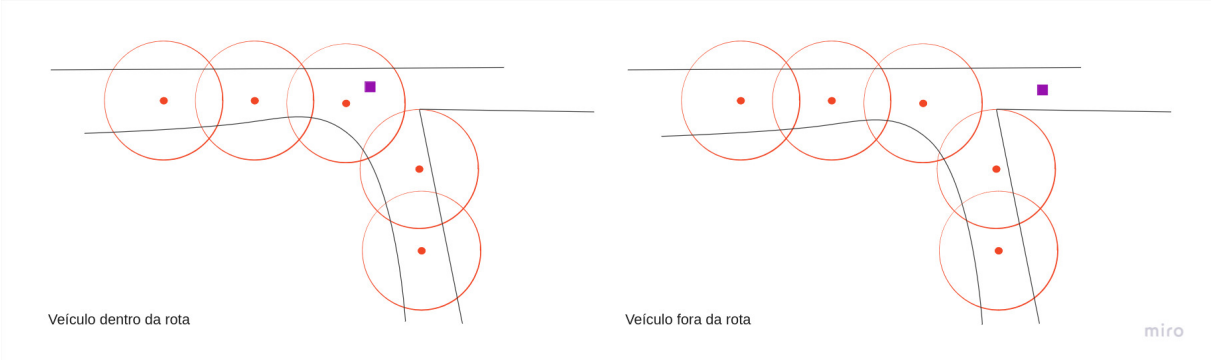
2.1.2 `record_data()`

Está thread executa em loop e espera até que um sinal seja enviado (espera condicional), quando receber o sinal ela "acorda"e grava os valores dos sensores no arquivo de registros (incluindo a valor da variável GPS_DATA).

2.1.3 `blocker_tracker()`

Está thread espera até que um sinal seja enviado, quando receber o sinal ela verifica se o veículo está dentro rota nada, caso esteja fora da rota, ativará um timer (tolerance_timer) que após um dado período enviará um sinal para a thread `blocker()`. A verificação da rota se dá pela verificação da distância do veículo a uma série de coordenadas, caso o veículo não se encontre a uma distância máxima de pelo menos um dos pontos, então conclui-se que o veículo está fora de rota. O tempo para a ativação do timer tolerance_timer tem função de permitir que o motorista do veículo tenha tempo para voltar a rota permitida.

Figura 2 – Detecção do veículo em relação à rota.



Fonte: Elaboração Própria.

As linhas pretas representam a estrada, os círculos laranja menores representam os pontos coordenados que definem a rota, o quadrado roxo representa o veículo. Quando o veículo abaixo da distância máxima(representada pela circunferência externa) de pelo menos um dos pontos considera-se dentro da rota.

2.1.4 `blocker()`

Está thread espera pelo sinal enviado pelo timer tolerance_timer, quando recebe o sinal ela verifica novamente se o veículo está fora de rota. Caso esteja fora de rota, inicia-se o procedimento de redução do limitador de velocidade do veículo (cria a thread reduce_speed() e o timer reduce_timer).

2.1.5 `reduce_speed()`

Está thread executa em um loop, enquanto o limitador de velocidade do veículo for maior que 0, espera por um sinal e quando recebe o sinal reduz a velocidade do veículo em uma constante. O sinal que está thread espera é recorrente e enviado pelo timer reduce_speed(), a função desta thread é reduzir a velocidade do veículo suavemente. Quando o limitador de velocidade estiver não permitir mais que o veículo ande, a thread desativa o timer reduce_timer e finaliza-se.

2.2 SENSORES E DISPOSITIVOS EXTERNOS

Não está no escopo deste trabalho compreender profundamente o funcionamento dos dispositivos externos e sensores usados, entretanto é necessário o entendimento da função de cada um destes dispositivos.

2.2.1 Sensor GPS

Para obtenção da posição real do veículo o embarcado usa um sensor GPS conectado em uma porta serial e grava, quando necessário, a posição em uma variável global nomeada GPS_DATA.

2.2.2 Velocímetro

Para obtenção da velocidade instantânea usa-se um velocímetro, para a implementação deste trabalho o velocímetro é simulado. A velocidade instantânea será gravada na variável global nomeada INST_SPEED.

2.2.3 Limitador de Velocidade

Um limitador de velocidade é um dispositivo que ao identificar que o veículo ultrapassa uma determinada velocidade instantânea, corta a capacidade de aceleração deste, fazendo com que o mesmo mantenha-se abaixo da velocidade definida. O valor definido como limite pelo velocímetro será gravado na variável global SPEED_LIMIT.

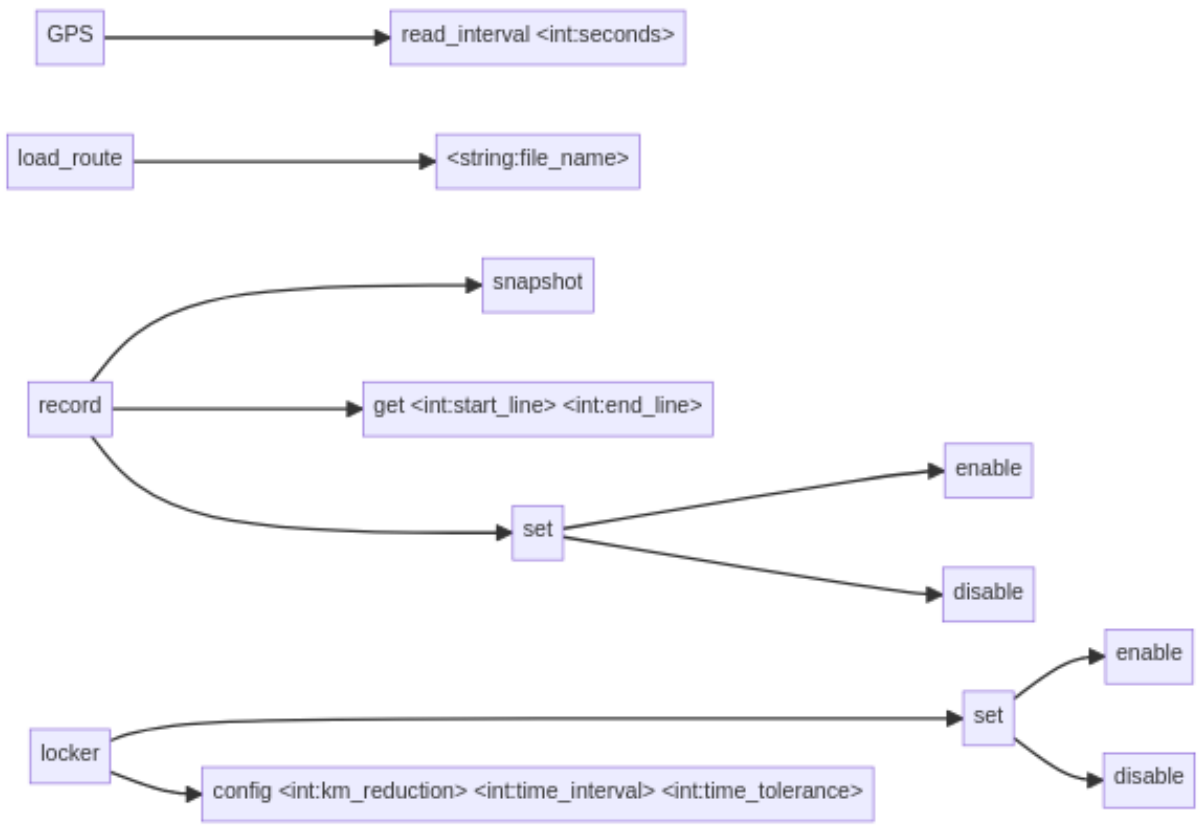
3 CLIENTE

O software cliente realiza a configuração das funções do software servidor, assim como pode carregar o arquivo com a rota a ser seguida pelo veículo e ainda requisitar os dados do arquivo de registro das informações do veículo.

3.1 COMANDOS

O funcionamento do cliente e servidor ficam mais evidentes com a descrição dos comandos do que podem ser passados através do cliente.

Figura 3 – Comandos do cliente.



Fonte: Elaboração Própria.

3.1.1 GPS read_interval <int:seconds>

Este comando define a intervalo no qual a variável GPS_DATA é atualizada pela função `gps_set()`, ou seja, altera o intervalo de execução do timer `gps_timer()`.

3.1.2 load_route <string:file_name>

Este comando permite carregar o arquivo com as coordenadas usadas para verificar se o veículo encontra-se na rota definida.

3.1.3 locker

O grupo de comandos `locker` está relacionado com a função do servidor de bloquear o veículo caso este desvie-se de sua rota.

3.1.3.1 locker conf <int:km_reduc> <int:t_interval> <int:t_tolerance>

Este comando configura o procedimento, os seguintes argumentos são necessários:

`km_reduc` indica o decremento de velocidade a cada execução da função `reduce_speed()`, em quilômetros por hora;

`t_interval` indica o tempo, em minutos, entre cada execução da função supracitada, tecnicamente altera o tempo de execução do timer `reduce_timer`;

`t_tolerance` indica o tempo que o motorista tem para retornar à rota antes que o sistema verifique novamente se veículo está na rota e caso necessário bloqueie o mesmo.

3.1.3.2 locker set enable|disable

Este comando ativa e desativa a funcionalidade, ao nível técnico altera a variável condicional verificada pela função `blocker_tracker()`.

3.1.4 `record`

O grupo de comandos `record` está relacionado com a função do servidor registrar informações do veículo em um arquivo de forma periódica.

3.1.4.1 `record get <int:start_line> <int:end_line>`

Este comando requisita dados do arquivo de registro, os seguintes argumentos são necessários à requisição:

`start_line` primeira linha a ser lida e enviada pelo servidor.

`end_line` última linha a ser lida e enviada pelo servidor.

O servidor envia todas as linhas no intervalo entre `start_line` e `end_line`. Quando não são passados argumentos o servidor envia o número total de linhas do arquivo.

3.1.4.2 `record set enable|disable`

Este comando ativa e desativa a funcionalidade, ao nível técnico altera a variável condicional verificada pela thread `record_data()`.

3.1.4.3 `record snapshot`

Este comando realiza o registro das informações atuais do veículo, ao nível técnico, ele envia um sinal à thread `record_data()`;

4 DEMAIS DETALHES

Alguns procedimentos não estão citados aqui, mas devem ser considerados visto que compõem o desenvolvimento do software e aumentam o trabalho necessário. Nestes procedimentos não citados encontram-se as threads necessárias a conexão cliente-servidor, o `parser` dos comandos enviados pelo cliente, e demais variáveis e funções necessárias à sincronização e controle das múltiplas threads.

5 IMPLEMENTAÇÃO

5.1 THREADS

5.1.1 Organização

A implementação servidor é composta por seis threads, cinco destas referentes as threads apresentadas na figura 1, e uma que implementa a função, responsável por inicializar as demais threads, e após isto torna-se responsável pela conexão com o software cliente.

Ambas threads recebem seus argumentos por meio de `structs` para facilitar a manipulação das variáveis, também apresentam um padrão na estrutura, separando a parte funcional da parte relacionada a sincronização de dados (com exceção da função `connection`). Por exemplo, a função `data_record_thread` apresentada na Figura 4, a maior parte da função é para obter cópias das variáveis a serem gravados, o procedimento de gravar os dados é delegado a outra função (`data_record`), mantendo assim o código mais modularizado e legível.

Figura 4 – Função implementada no software.

```
void *data_record_thread(void *structure){
  data_record_thread_arg  *arg = (data_record_thread_arg * )structure;

  while(arg->enable){
    printf("Data_record: Esperando enable \n\n");
    wait_enable_dec(arg->record_cond);
    printf("Data_record: Passou enable \n\n");

    // Cria a linha que será escrita
    data_line dl;

    pthread_mutex_lock(arg->gps.mutex);
    dl.position = *(arg->gps.data);
    pthread_mutex_unlock(arg->gps.mutex);

    pthread_mutex_lock(arg->speed_limit.mutex);
    dl.max_speed = *(arg->speed_limit.data);
    pthread_mutex_unlock(arg->speed_limit.mutex);

    pthread_mutex_lock(arg->instant_speed.mutex);
      get_speed((arg->instant_speed.data));
      //Tudo isso para manter instant_speed abaixo de max_speed
     dl.instant_speed = (*(arg->instant_speed.data) -
                         (int)(*(arg->instant_speed.data))) +
                        ((int)(*(arg->instant_speed.data)) % (int)dl.max_speed);
    pthread_mutex_unlock(arg->instant_speed.mutex);

    data_record(arg->fs, dl);

    pthread_mutex_lock(arg->fs->mutex);
      arg->fs->line_count++;
    pthread_mutex_unlock(arg->fs->mutex);

  }

  pthread_exit(NULL);

}
```

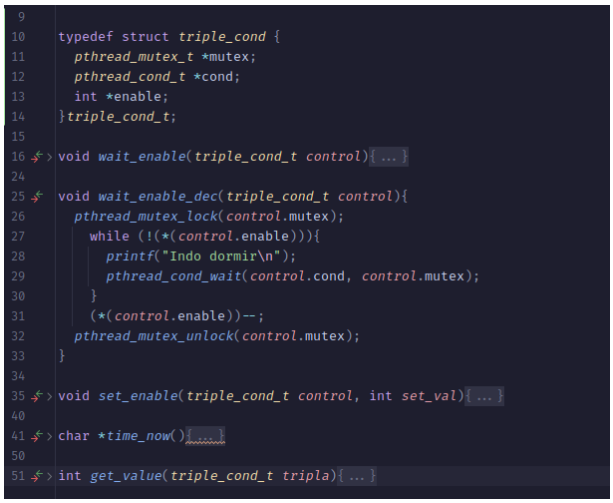
Fonte: Elaboração Própria.

A passagem de argumentos por meio de `structs` também facilitou a passagem destes para a função `command_control` contida em `connection` cuja responsabilidade é interpretar os comandos do software cliente e realizar as operações necessárias, que podem ser alterar as configurações ou ativar um timer, ou alterar variáveis que regem o funcionamento das threads.

5.1.2 Funções e Estruturas Auxiliares

Percebeu-se que o conjunto de variáveis mutex, variável de condição e valor inteiro é recorrentemente usado no software servidor, assim como os procedimentos de manipulação como `lock-unlock`, `wait_cond` e `wait_cond` com decremento do valor inteiro. Tendo em vista isto, foi desenvolvido um conjunto de funções, demonstradas na figura 5, com objetivo de reduzir a redundância de código e consequente torná-lo mais legível.

Figura 5 – Funções auxiliares, com destaque na função `wait_enable_dec`.



```
9
10  typedef struct triple_cond {
11    pthread_mutex_t *mutex;
12    pthread_cond_t *cond;
13    int *enable;
14  }triple_cond_t;
15
16  void wait_enable(triple_cond_t control){ ... }
24
25  void wait_enable_dec(triple_cond_t control){
26    pthread_mutex_lock(control.mutex);
27      while (!(*(control.enable))){
28        printf("Indo dormir\n");
29        pthread_cond_wait(control.cond, control.mutex);
30      }
31      (*(control.enable))--;
32    pthread_mutex_unlock(control.mutex);
33  }
34
35  void set_enable(triple_cond_t control, int set_val){ ... }
40
41  char *time_now(){ ... }
50
51  int get_value(triple_cond_t tripla){ ... }
```

Fonte: Elaboração Própria.

Obviamente, pode haver situações em que estas funções genéricas não satisfazem o requerido, entretanto na maioria das situações são suficientes.

5.2 COMUNICAÇÃO COM CLIENTE

O software cliente foi desenvolvido de maneira simples, com uma única thread e uma única conexão de socket. A maioria dos comandos segue o fluxo de enviar o comando para o servidor e este responder com uma mensagem, estes comandos serão chamados de comandos simples a seguir. Os comandos `record get;` e `load_route` pos-

suem um fluxo de envio e recepção diferente das demais e serão especificados em suas correspondentes subsecções.

5.2.1 Comandos Simples

Correspondem os comandos que seguem o fluxo de envio para o servidor, o servidor realiza a operação correspondente e então envia uma mensagem de resposta ao cliente. Assim como o fluxo de comunicação, as operações descadeadas por estes comandos são simples e compreendem alteração de variáveis de condição, ativar e/ou configurar timers e alterar outras variáveis que definem o comportamento das threads.

5.2.2 `load_route <string:file_name>`

Como já descrito neste texto, este comando permite carregar o arquivo com as coordenadas usadas para verificar se o veículo encontra-se na rota definida. Tendo em vista que o tamanho do arquivo não está limitado o tamanho do buffer usado para comunicação, é fácil compreender o porquê este comando não pode seguir o fluxo de comunicação apresentado acima.

Para a implementação deste comando, o mesmo tornou-se um pseudo-comando que não é enviado diretamente ao servidor. Ao invés disso ele gera uma sequência de comandos de fluxo semelhante ao explicado acima.

O processo inicia-se com o envio do comando `load_route open;` que abre um arquivo temporário no software servidor, seguido da leitura linha a linha do local a ser enviado e enviado para o servidor escrever no arquivo temporário através do comando `load_route write <double:latitude> <double:longitude> <double:raio>;`, após a leitura e envio de todas as linhas do arquivo local é enviado o comando `load_route close;` que fecha o arquivo no lado do servidor.

Até então o arquivo de rotas que deveria ser enviado ao servidor foi enviado, mas como um arquivo temporário, isto é, com um nome diferente do que é buscado pelo software servidor. Isto é feito para evitar que uma falha durante a comunicação corrompa o arquivo em uso, e também para diminuir o período em que as threads dependentes do arquivo ficarão em espera. Logo, ao fim do processo, caso não haja erros, é enviado o comando `load_route commit;` que faz o servidor renomear o arquivo temporário para o nome do arquivo em uso, sobrescrevendo o mesmo.

5.2.3 `record get <int:start_line> <int:end_line>`

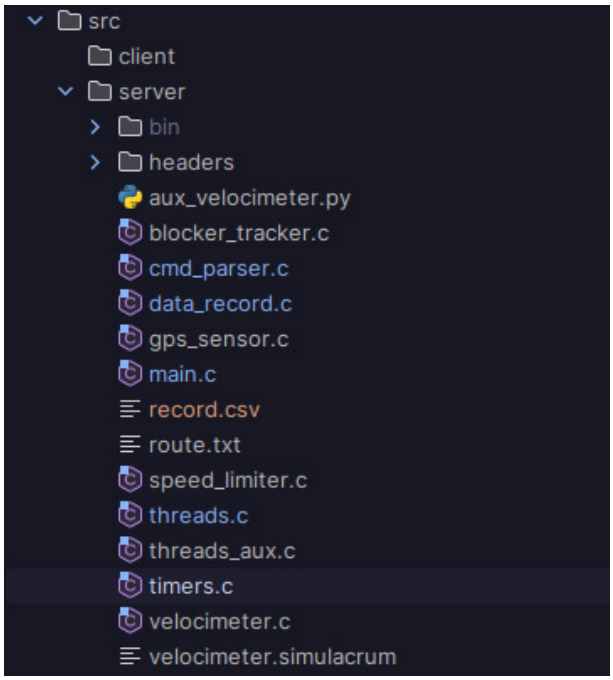
Assim como o comando acima, este não se encaixa na definição de comando simples, o procedimento, entretanto é mais simples.

Após o envio do comando ao servidor, o cliente entrará em um laço de repetição recebendo os dados linha a linha, e escrevendo em arquivo local. O procedimento encerra quando o servidor indica, por meio de um valor específico enviado.

5.3 ESTRUTURA DE ARQUIVOS

O agrupamento das responsabilidades também é implementado na organização dos arquivos (Figura 6), cada arquivo contém um grupo de funções relacionadas entre si, o arquivo `threads.c` contém a implementação das funções contidas nos demais arquivos na forma de threads, o arquivo `timers.c` contém funções que abstraem as bibliotecas `time.h` e `signal.h` a fim de facilitar o desenvolvimento do código.

Figura 6 – Estrutura de arquivos do software.



Fonte: Elaboração Própria.

6 CONCLUSÃO

Quanto aos desafios no desenvolvimento deste sistema, estão relacionados ao grande número de variáveis e aos cuidados necessários para impedir que ocorram problemas de sincronização.

Para solucionar estes desafios foi importante o planejamento prévio do sistema antes de iniciar o desenvolvimento, também o uso de `structs` que agrupassem variáveis usadas juntas e a criação de funções para a realização de procedimentos recorrentes.

Os problemas de sincronização foram evitados ao fazer as threads mais independentes o possível e ao diminuir o tempo de acesso exclusivo as variáveis, isto é, diminuir a quantidade de código dentro de trechos `lock-unlock`.