# zenith.h - v1.0

João Matheus Siqueira Souza

## Introdução:

A biblioteca zenith.h foi criada durante o projeto **Z-Sat**, o primeiro CubeSat do grupo Zenith Aerospace, da Escola de Engenharia de São Carlos, o qual foi projetado para a competição CubeSat Design, organizada pelo INPE. A biblioteca contém funções úteis para o desenvolvimento de sistemas aeroespaciais, como na comunicação entre subsistemas internos de nano-satélites, sondas espaciais e drones; na comunicação do sistema com uma base de operação; na gestão do sistema de alimentação do sistema; além de outras funções.

### #include "zenith.h"

**(DESCONSIDERAR ESSA PARTE ATÉ DIA 30/06)**

Antes de começar a utilizá-la, certifique-se de que a biblioteca está corretamente instalada. Para isso, baixe os arquivos do repositório: https://github.com/jmssouza/cubesat , em seguida, execute os seguintes comandos num terminal Linux:

```
cd ~/<PATH_TO_DIRETORY_THAT_YOU_PUT_THE_FILES>/instalation

gcc install_zenith_h.c -o zenith_install

./zenith_install

./testZenith
```

Após esse comando, se a biblioteca estiver corretamente instalada, deverá aparecer no terminal a seguinte tela:

## • Constantes:

Aqui descreveremos cada uma das constantes definidas no arquivo "constants.h" e como elas podem ser alteradas para suas aplicações. Dividimos em dois tipos: constantes numéricas e nomes de arquivos utilizados.

### - Constantes numéricas:

`#define PACK_SIZE 255` : Essa constante representa o tamanho em bytes que cada pacote armazena. Esse valor pré-definido de 255 refere-se ao tamanho máximo permitido pelo transceiver utilizado em nosso primeiro Cubesat. Caso utilizemos outro dispositivo, podemos mudar esse número para atender as capacidades deste novo dispositivo.

`#define BLOCK_SIZE 248` : Essa constante representa o tamanho em bytes de informação que cada pacote carrega. A diferença entre esses valor e o PACK_SIZE é dada pelo "cabeçalho" que cada pacote possui. Esse "cabeçalho", no nosso sistema continha em 3 bytes o valor do modo de operação do sistema, em outros 3 bytes o valor do pacote e em um outro byte o ciclo do pacote.

### - Nomes de arquivos:

`#define TM_FILE "tm_file.dat"`

Contém todos os pacotes enviados pelo CubeSat, em ordem.

`#define TM_NUMBER "tm_num.dat"`

Contém o número de pacotes enviados.

`#define TM_CYCLE "tm_cycle.dat"`

Como é dado um byte para armazenar o valor do pacote, quando mais de 256 pacotes são enviados, haveria reinicialização do número de pacotes, pois um byte consegue

representar apenas 256 valores diferentes. Assim, quando essa reinicialização ocorre, é acrescida uma unidade no valor armazenado pelo pacote TM_CYCLE, o qual possibilita obter o valor de pacotes enviados.

```
#define TC_FILE         "tc_file.dat"
#define TC_NUMBER       "tc_num.dat"
#define TC_CYCLE        "tc_cycle.dat"
```

Esses arquivos são análogos aos TM's, no entanto, usados para os pacotes recebidos pelo CubeSat, não para os enviados.

```
#define NEW_TC          "new_tc.dat"
```

Quando um novo pacote é recebido pelo sistema, antes de ser escrito no arquivo TC_FILE, ele é mantido no arquivo NEW_TM até que o pacote seja analisado e passado para o outro arquivo.

```
#define MISSED_PACKAGES "missed_packets.dat"
```

Esse arquivo contém o número dos pacotes perdidos a cada recebimento de um novo pacote. Permitindo, pois, que estes pacotes perdidos sejam requisitados da base.

```
#define MODE_FILE       "op_mode.dat"
```

Esse arquivo armazena o valor do modo de operação atual do CubeSat, o qual é recebido em um telecomando.

- ## Funções:

Aqui será descrito como funciona cada uma das funções existentes na biblioteca zenith.h e como elas devem ser utilizadas.

- `int valueGetter(char *file_name, int *value);`

Essa função armazena valores contidos em arquivos. Ou seja, ela abra um arquivo ( cujo nome está armazenado no ponteiro file_name ) e armazena em uma variável o valor contido naquele arquivo. Por exemplo, temos um arquivo chamado "packages_received_number.txt" , queremos saber o valor armazenado nesse arquivo:

```
int number = 0;
char file [20] = "packages_received_number.txt";
valueGetter(file, &number);
```

nesse exemplo, a variável number agora armazena o valor contido pelo arquivo "packages_received_number.txt". Essa funçaõ retorna 1, caso todos os processos tenha ocorrido de maneira esperada, e 0, caso tenha ocorrido algum problema na abertura do arquivo.

- `int valueSetter(char *file_name, int value);`

Essa função é análoga à função valueSetter, no entanto, ela muda o valor contido dentro do arquivo o qual é passado como argumento. Assim, considerando um exemplo análogo ao da função valueGetter, se number fosse igual a 3, a função valueSetter armazenaria no arquivo "packages_received_number.txt" o valor 3. Assim como a função valueGetter, é retornado 1 caso a função não tenha apresentado nenhum problema durante sua execução e 0, para quando o arquivo tenha algum problema e não abra como esperado.

- `int writeMessage(char *file_name, char *message, int position, int size, int check);`

Essa função é uma das mais importantes da biblioteca, ela permite a escrita dos dados de operação em seus respectivos arquivos. O primeiro argumento da função, file_name, refere-se ao nome do arquivo no qual deseja-se escrever. O segundo

argumento, message, é o que deseja-se escrever no arquivo file_name. O terceiro, position, refere-se a posição na qual deseja-se escrever no arquivo file_name, mas cuidado, cada arquivo é padronizado, tendo uma sequência periódica de escrita. Assim, essa posição referente ao terceiro argumento significa a posição da mensagem que deseja-se escrever no arquivo. Por exemplo, se um pacote tem 255 bytes, a posição não faz referência nenhuma à posição dentro do pacote (algum valor menor que 255 bytes) mas sim à posição de um determinado pacote no arquivo. Se recebemos 320 pacotes e desejamos escrever no arquivo o pacote 321, então 321 é o valor ao qual esse parâmetro faz referência. O quarto parâmetro, size, é referente ao tamanho do pacote em bytes, no exemplo anterior, 255 bytes. O último parâmetro faz referência a uma posição de checagem opcional para cada pacote. Vamos supor que todos os pacotes recebidos da base sejam escritos num arquivo "tc_file.dat", cada pacote tem tamanho de 255 bytes, então a função writeMessage irá escrever um pacote em cada intervalo de 255 bytes. No entanto, queremos ler esse arquivo "tc_file.dat" de forma não sequencial, talvez de modo aleatório. Para determinarmos se já havíamos lido algum pacote, pode-se fazer o uso do parâmetro check, o qual irá pular a quantidade de bytes desejada entre um pacote e outro. Assim, pode-se usar esses bytes pulados para armazenar informações, como se esse pacote já foi lido ou não. Essa quantidade de bytes pulados entre um pacote e outro pode assumir qualquer valor, porém deve-se tomar bastante cuidado para que as funções writeMessage e readMessage associadas a esse arquivo tenham o mesmo valor de check. Se não é desejado que haja "pulos" entre os pacotes, basta colocar o valor "0" para esse parâmetro. Essa função retorna 1, se todo seu processamento ocorreu da maneira esperada e 0, se algum problema ocorreu durante sua execução.

- **`int readMessage(char *file_name, char *message, int position, int size, int check);`**

Essa função é análoga à função writeMessage, no entanto, deve-se passar um vetor de char, message, para que se armazene nele o conteúdo lido no arquivo. Os mesmos retornos da função writeMessage também são apresentados aqui.

- `int blockBuilder(char *block, int operating_mode, int aux);`

**(FUNÇÃO AINDA EM CONSTRUÇÃO)**

Definimos como "block", o bloco de informações que não contém o cabeçalho de um pacote. A função blockBuilder constrói esse bloco de informação com base no modo de operação do CubeSat. Assim, o primeiro parâmetro é um vetor de char, com tamanho do bloco, no qual o bloco de informações, para aquele modo de operação, será armazenado. O segundo parâmetro é o modo de operação atual do sistema e o terceiro é uma variável auxiliar, a qual muda seu papel em função do modo de operação. Como cada um dos desenvolvedores do CubeSat é responsável por implementar a parte da função blockBuilder referente ao seu modo de operação, a utilização da variável auxiliar fica ao critério de cada um.

- `int packageCreator(char *pack_num_file, char *pack_cycle_file, char *block, char *message);`

Essa função é responsável por unir o "block" ao "cabeçalho", finalizando, pois, o pacote para o envio. Os parâmetros dessa função são, respectivamente, referentes ao nome do arquivo que guarda o número do último pacote enviado pelo CubeSat (TM_NUMBER), o número do ciclo do pacote (TM_CYCLE), o bloco de informações obtido pela função blockBuilder e o vetor de char que deve ser passado para se obter o pacote final, message.

- `int packageAnalyzerCube();`

Essa função é responsável por analisar os pacotes recebidos pelo CubeSat da base. Mudando os valores de número do pacote recebido (TC_NUMBER), o modo de operação do sistema (MODE_FILE) e, caso ocorra perda de pacote, alterar o número dos pacotes perdidos (MISSED_PACKAGES) em seus respectivos arquivos. Essa função retorna 1 caso o pacote tenha recebido

com precisão o modo de operação e 0, caso não se possa concluir sobre esse modo de operação.

- **Desenvolvedores:**

Francesco Rossi Lena
João Matheus Siqueira Souza
Orlando Wozniak de Lima Nogueira
Tiago Bachiega de Almeida
Vinicius Eiji Sasaki