Trabalho 1 - INE5645

Bernardo De Marco Gonçalves - 22102557

Design da aplicação

A aplicação é composta por N sensores (valor parametrizável via *input*). Cada sensor é uma *thread* que executa um *loop* inifinito, produzindo valores aleatórios e os inserindo no *buffer* do produtor/consumidor.

O *core* da aplicação concentra-se na central de controle (*orchestrator*). Esse componente é uma única *thread* que gerencia os atuadores do veículo, através de um *thread pool* de `4` *threads* e uma tabela de controle. Além disso, o *orchestrator* é o o responsável por consumir os dados produzidos pelos sensores do veículo.

Sempre que o *orchestrator* consumir um valor do *buffer*, ele envia o valor juntamente com uma *task* a ser executada para uma *thread* do *thread pool*. Cada *task* é responsável por determinar o atuador e o nível de atividade, atualizar o registro do atuador e gerar um *log* no terminal.

A atualização e a geração dos *logs* são executados em paralelo (*fork-join*). Caso ocorra algum erro em alguma dessas subtarefas, a *task* é responsável por gerar uma saída informando o erro ocorrido.

Para atualização do atuador, o seu nível de atividade na tabela de atuadores é atualizado, e permanece inalterado por dois ou três segundos. Para não definir a tabela inteira como uma única zona de exclusão mútua, ela foi dividida em seções críticas com uma granularidade pré-determinada de `20`. Ou seja, quando um atuador tiver seu nível de atividade atualizado apenas a sua seção será travada, e não a tabela inteira.

Concomitantemente à atualização da tabela, o *log* é gerado. Sempre que um *log* é impresso, ele será mantido no console por um segundo. Por fim, a *task* do *thread pool*, ao sincronizar o *fork-join*, checa se houve algum erro em alguma tarefa (`20%` de chance de erro em cada uma). Caso algum erro tenha ocorrido, um *log* de erro é gerado e impresso no terminal.

Tecnologias adotadas

- Linguagem de programação C
- POSIX *library*
- C-Thread-Pool

A biblioteca C-Thread-Pool foi utilizada para o uso de *thread pools*. Ela possui uma API clara e de fácil utilização. Além disso, seu código-fonte foi verificado e foi certificado que ela utiliza uma *queue* como *buffer* de *tasks* do *pool*. Com isso, foi possível utilizá-la para cumprir requisitos da aplicação.

Instruções de *setup*

Foi utilizado o `Makefile` para facilitar a compilação da aplicação. O arquivo compilado é o arquivo `app` localizado na pasta `build`.

- Compilar:

```
make compile
```

- Rodar:

```
make run
```

- Limpar:

```
make clean
```

Exemplos de saídas de execução

Quantidade de atuadores igual ao dobro da quantidade de sensores

Parâmetros de entrada

Número de sensores	Número de atuadores
100	200

Saída

```
 Changing actuator [90] with value [89]
Changing actuator [22] with value [69]
Changing actuator [73] with value [7]
An error occurred in the actuator [73]
Changing actuator [63] with value [17]
Changing actuator [105] with value [38]
An error occurred in the actuator [105]
Changing actuator [180] with value [61]
Changing actuator [103] with value [27]
Changing actuator [136] with value [95]
An error occurred in the actuator [136]
Changing actuator [72] with value [36]
Changing actuator [115] with value [55]
An error occurred in the actuator [115]
Changing actuator [197] with value [8]
Changing actuator [19] with value [39]
An error occurred in the actuator [19]
Changing actuator [172] with value [32]
Changing actuator [59] with value [3]
Changing actuator [59] with value [7]
Changing actuator [165] with value [26]
Changing actuator [37] with value [93]
Changing actuator [104] with value [93]
Changing actuator [72] with value [37]
Changing actuator [124] with value [92]
Changing actuator [39] with value [32]
An error occurred in the actuator [104]
An error occurred in the actuator [39]
Changing actuator [61] with value [88]
Changing actuator [112] with value [38]
```

Total de *logs*	Número de alterações nos atuadores	Número de erros
30	23	7

Durante essa execução, observou-se que o fluxo de geração de *logs* foi bem fluído, respeitando-se o requisito de cada *log* de alteração segurar o console por um segundo.

Quantidade de sensores igual ao dobro da quantidade de atuadores

Parâmetros de entrada

Número de sensores	Número de atuadores
200	100

Saída

```
 Changing actuator [73] with value [7]
Changing actuator [80] with value [84]
Changing actuator [3] with value [14]
An error occurred in the actuator [80]
Changing actuator [36] with value [45]
Changing actuator [72] with value [42]
An error occurred in the actuator [72]
Changing actuator [15] with value [38]
An error occurred in the actuator [15]
Changing actuator [97] with value [35]
An error occurred in the actuator [97]
Changing actuator [94] with value [77]
Changing actuator [72] with value [68]
Changing actuator [59] with value [45]
Changing actuator [20] with value [50]
Changing actuator [65] with value [59]
An error occurred in the actuator [65]
Changing actuator [37] with value [77]
Changing actuator [4] with value [4] # Change of 4th actuator's activity level
Changing actuator [72] with value [36]
An error occurred in the actuator [72]
Changing actuator [24] with value [38]
Changing actuator [39] with value [1]
An error occurred in the actuator [39]
Changing actuator [61] with value [12]
An error occurred in the actuator [4] # error in the 4th actuator
Changing actuator [49] with value [78]
Changing actuator [92] with value [22]
Changing actuator [97] with value [65]
An error occurred in the actuator [97]
```

Total de *logs*	Número alterações nos atuadores	Número de erros
30	24	6

Em alguns momentos, observou-se que o erro foi impresso depois de uma quantidade significativa de *logs* de alteração, como observa-se com o atuator `4` no *output*. Isso deve-se ao fato de que a quantidade de sensores é igual ao dobro da quantidade de atuadores.

Com isso, a quantidade de valores que são inseridos no *buffer* do produtor/consumidor tende a dobrar. Concomitantemente, a quantidade de atuadores diminui em `100`. Isso leva a dividir a tabela de atuadores em `5` regiões críticas (`100 / 20`). No exemplo anterior eram `10` regiões críticas.

Por conseguinte, uma quantidade maior de dados sensoriais disputam menos regiões críticas. Isso ocasiona um enfileiramento de tarefas em estado suspenso maior e mais frequente, retardando a modificação do campo da tabela de um dado atuador.