Sistemas Operativos 2022/2023

Trabalho Prático

Simulador para Internet das Coisas em contexto habitacional

NOTA: Antes de começar o trabalho leia o enunciado até ao fim!

1 Introdução

Com o aparecimento da Internet das Coisas (*Internet of Things* em inglês), tornou-se atrativo instalar sensores nas várias divisões de uma habitação de forma a coletar, medir e atuar sobre diferentes aspetos. Por exemplo, é possível ligar e desligar luzes em divisões não ocupadas, variar o aquecimento, detetar intrusões, etc. Para suportar estes casos de uso é necessário um sistema capaz de receber informação de vários sensores em simultâneo, para depois a armazenar e processar.

O trabalho a desenvolver deve simular um ambiente simplificado de Internet das Coisas onde vários sensores enviam informação (leituras) para um ponto centralizado, que por sua vez armazena estes dados, gera estatísticas e despoleta alertas quando certas condições são atingidas. De seguida são descritos os detalhes do sistema simplificado a simular neste trabalho.

2 Descrição do sistema de simulação

Nesta secção serão descritos os componentes do sistema, as suas funcionalidades e a forma como comunicam entre si.

2.1 Estrutura do sistema

A estrutura do sistema é apresentada na Figura 1.

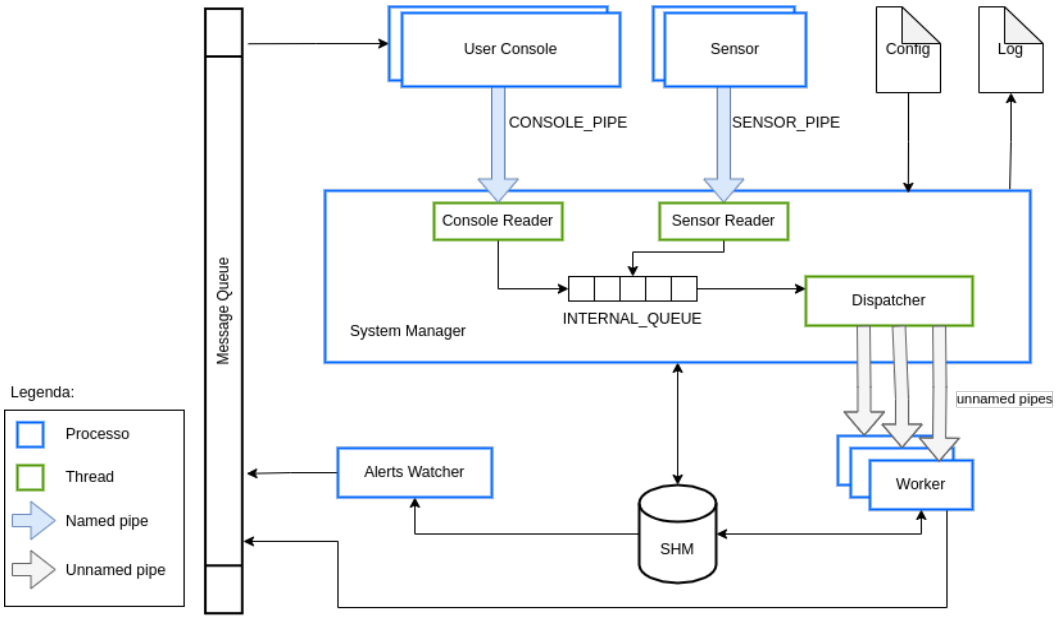

Figura 1. Visão geral do simulador a desenvolver.

Tal como é representado na Figura 1, o sistema é baseado em vários processos e *threads* que se descrevem a seguir:

- ***Sensor*** - Processo que periodicamente gera dados e os envia por um *named pipe* ao ***System Manager***. Podem existir vários destes processos.
- ***User Console*** - Processo que interage com o utilizador e que comunica por *named pipe* com o ***System Manager*** para enviar comandos. Efetua operações de gestão do sistema, consulta estatísticas e outras informações, e recebe alertas em tempo real. A informação é recebida através da *Message Queue*. Podem existir vários destes processos.
- ***System Manager*** - Processo responsável por iniciar o sistema, ler o ficheiro de configuração e criar os processos ***Worker*** e o processo ***Alerts Watcher***.
- ***Worker*** - Processo responsável por satisfazer os pedidos recebidos dos processos ***Sensor*** e ***User Console***. Podem existir vários destes processos.
- ***Alerts Watcher*** - Processo responsável por verificar se os valores dos sensores estão fora dos limites e por enviar um alerta aos processos ***User Console*** respetivos através da *Message Queue*.
- ***Console Reader*** - *Thread* que lê os comandos das ***User Consoles*** enviados no *named pipe* **CONSOLE_PIPE**.
- ***Sensor Reader*** - *Thread* que lê os dados dos processos ***Sensor*** enviados no *named pipe* **SENSOR_PIPE**.
- ***Dispatcher*** - *Thread* que pega nos pedidos que estão armazenados na fila **INTERNAL_QUEUE** e envia-os através de *unnamed pipes* para um ***Worker*** que esteja disponível.

E também por vários IPCs:

- ***Named Pipe*** **SENSOR_PIPE** - Permite que os processos ***Sensor*** enviem dados ao processo ***System Manager*****.**
- ***Named Pipe*** **CONSOLE_PIPE** - Usado para enviar comandos efetuados pelo ***User Console*****.**
- **SHM** - Zona de memória partilhada acedida pelos processos ***Worker*** e ***Alerts Watcher***.
- ***Unnamed pipes*** - Permitem a comunicação entre o processo ***System Manager*** e cada um dos processos ***Worker***.
- ***Message Queue*** (fila de mensagens) - Permite o envio de alertas a partir do ***Alerts Watcher*** para os ***User Consoles*** e o envio das respostas aos pedidos dos ***User Consoles***.

Dentro do processo ***System Manager*** existe também uma estrutura de dados de tamanho fixo que armazena os dados ou comandos que têm de ser processados pelos ***Workers*** - **INTERNAL_QUEUE**.

Existirá também um ficheiro de ***log*** onde serão escritas todas as informações para análise posterior. Todas as informações escritas para o ***log*** também devem aparecer no ecrã.

2.2 Descrição dos componentes e das funcionalidades

De seguida são apresentadas com detalhe as características e funcionalidades dos diversos componentes.

Sensor

Processo que envia dados para o sistema em intervalos regulares. Podemos ter um ou mais processos destes a correr em simultâneo, cada um com os seus parâmetros. Cada processo escreverá os dados no *named pipe* **SENSOR_PIPE**.

O identificador único do sensor, a chave referente aos dados que vão ser enviados, o valor mínimo e máximo a ser enviado e o intervalo entre cada envio, são fornecidos por parâmetro de linha no arranque do programa.

Sintaxe do comando:

```
$ sensor {identificador do sensor} {intervalo entre envios em segundos (>=0)} {chave} {valor inteiro mínimo a ser enviado} {valor inteiro máximo a ser enviado}
```

Exemplo de uso:

```
$ sensor SENS1 3 HOUSETEMP 10 100
```

Informação a enviar para o *named pipe*:

```
ID sensor#Chave#Valor
```

Exemplo:

```
SENS1#HOUSETEM#20
```

O identificador de um sensor é um código alfanumérico com um tamanho mínimo de 3 caracteres e tamanho máximo de 32 caracteres. A chave é uma *string* com tamanho entre 3 e 32 caracteres (inclusive) formada por uma combinação de dígitos, caracteres alfabéticos e *underscore* (_), que serve para identificar qual o tipo de informação que está a ser enviada. Esta chave vai ser usada para agrupar os dados na memória partilhada.

Se receber o sinal SIGTSTP, o processo deve apresentar no terminal o número de mensagens que enviou desde o seu início.

O processo termina ao receber um sinal SIGINT, ou em caso de erro. Um erro pode acontecer se algum parâmetro estiver errado ou ao tentar escrever para o *named pipe* e a escrita falhar, casos em que deverá escrever a mensagem de erro no ecrã. Sempre que termina, o processo deve limpar todos os recursos.

User Console

Processo que disponibiliza um menu interativo ao utilizador e que permite enviar comandos de gestão e consulta de dados para o sistema, apresentando o resultado dos mesmos. Para além disso, é capaz de receber e apresentar no ecrã qualquer alerta que receba em tempo real.

Sintaxe do comando a executar na linha de comando:

```
$ user_console {identificador da consola}
```

Este processo deve disponibilizar um menu onde o utilizador pode escrever os seguintes comandos:

- `exit` - Sai do ***User Console***
- `stats` - Apresenta estatísticas referentes aos dados enviados pelos sensores
- `reset` - Limpa todas as estatísticas calculadas até ao momento pelo sistema (relativa a todos os sensores, criados por qualquer ***User Console***)
- `sensors` - Lista todos os **Sensors** que enviaram dados ao sistema
- `add_alert [id] [chave] [min] [max]` - Adiciona uma nova regra de alerta ao sistema
- `remove_alert [id]` - Remove uma regra de alerta do sistema
- `list_alerts` - Lista todas as regras de alerta que existem no sistema

O *id* de um alerta é um código alfanumérico com um tamanho mínimo de 3 caracteres e máximo de 32 caracteres. Os campos *min* e *max* são números inteiros e correspondem ao intervalo de valores aceitável para a chave. Todos os valores menores que o mínimo ou maiores que o máximo geram alertas. Existe um limite máximo do número de alertas que podem estar ativos no sistema (este limite é definido no ficheiro de configuração). Sempre que houver uma tentativa de adicionar um novo alerta quando o limite já foi atingido, o comando deve falhar com erro. O comando de adicionar um alerta deve também falhar se o *id* do novo alerta for igual ao *id* de um alerta que já existe.

Exemplo de uma sequência de uso dos comandos:

```
> stats
Key   Last   Min   Max   Avg   Count
ROOM1_TEMP 15 10 30 14 123
ROOM1_CO2 120 100 300 150 10
> sensors
ID
ROOM1S1
ROOM1S2
> add_alert AL1 ROOM1_TEMP 10 25
OK
> reset
OK
> stats
Key   Last   Min   Max   Avg   Count
> sensors
ID
> list_alerts
ID    Key MIN MAX
AL1   ROOM1_TEMP 10 25
> remove_alert AL1
OK
> list_alerts
ID    Key MIN MAX
```

Todos estes comandos (exceto o exit) devem ser enviados para o ***System Manager*** através do **CONSOLE_PIPE** .

A resposta aos comandos (exceto o exit) devem ser devolvidos à respetiva ***User Console*** pelo ***Worker*** que processou o pedido e através da fila de mensagens. As notificações dos alertas que ocorreram também devem ser transmitidas pela fila de mensagens, para todos os processos ***User Console*** ativos, e a mensagem que dá conta do alerta deve ser impressa no terminal imediatamente após ser recebida (mesmo que o utilizador esteja a executar um comando).

Ao receber o sinal SIGINT ou o comando `exit`, o processo deve limpar todos os recursos e terminar. O processo deve terminar (limpando os recursos) se ao tentar escrever para o *named pipe* a escrita falhar porque este já não existe.

System Manager
Este processo lê as configurações iniciais e arranca todo o sistema. Em concreto tem as seguintes funcionalidades:

- Lê e valida as informações no ficheiro de configurações - ***Config File*** (neste enunciado é fornecido um exemplo deste ficheiro)
- Cria os *named pipes* **SENSOR_PIPE** e **CONSOLE_PIPE**
- Cria os processos ***Worker***
- Cria os *unnamed pipes* para cada ***Worker***
- Cria o processo ***Alerts Watcher***
- Cria as *threads* ***Sensor Reader*** e ***Console Reader***
- Cria a *thread* ***Dispatcher***
- Cria a fila de mensagens
- Cria a estrutura de dados **INTERNAL_QUEUE**
- Cria a memória partilhada; a utilização da memória partilhada deve ser otimizada e não gastar espaço desnecessário. A memória partilhada deve, pelo menos, conter informação sobre os sensores, os dados que são recebidos dos sensores e informação sobre as regras para geração de alertas.
- Captura o sinal SIGINT para terminar o programa.

Sintaxe do comando a executar na linha de comando:
```
$ home_iot {ficheiro de configuração}
```

Sensor Reader
Thread responsável por gerir a receção de dados dos sensores e de os colocar numa estrutura de dados (fila **INTERNAL_QUEUE**) para serem processados pela *thread* ***Dispatcher***.

Quando uma nova mensagem chega através do *named pipe* **SENSOR_PIPE**, a informação (ou seja, o ID do sensor, chave e valor) é colocado na fila **INTERNAL_QUEUE**. Esta fila tem um tamanho máximo de QUEUE_SZ (este parâmetro é fornecido pelo ficheiro de configurações). Se a fila já estiver cheia, o pedido é eliminado e essa informação é escrita no ecrã e no ficheiro de ***log***.

Console Reader
Thread responsável por gerir a receção de comandos das ***User Consoles*** e de os colocar numa estrutura de dados (fila **INTERNAL_QUEUE**) para serem processados pela *thread* ***Dispatcher***.

Quando uma nova mensagem chega através do *named pipe* **CONSOLE_PIPE**, a informação (ou seja, o ID da consola, o comando e os seus argumentos) é colocada na fila **INTERNAL_QUEUE**. Esta é a mesma fila utilizada para armazenar as mensagens dos sensores. Se a fila estiver cheia então a *thread* deve bloquear até haver espaço livre na fila para acomodar a nova entrada.

Dispatcher
Thread responsável por pegar nas entradas que estão na fila e de as enviar para um ***Worker*** que esteja livre (não ocupado). Caso não existam ***Workers*** disponíveis, a *thread* ***Dispatcher*** aguarda (sem *busy wait*) até um ***Worker*** ficar disponível.

As mensagens enviadas pelos processos **Sensor** têm menor prioridade do que as mensagens dos ***User Console***. Isto quer dizer que o ***Dispatcher*** deve dar prioridade às mensagens dos ***User Console*** e enviá-las para serem processadas pelos processos ***Worker***.

A comunicação entre o ***Dispatcher*** e o ***Worker*** é feita através do *unnamed pipe* que é criado para cada ***Worker***. Quando uma entrada é enviada para um ***Worker***, esta deve ser removida da fila e o ***Worker*** deve passar a estar num estado ocupado (para que o ***Dispatcher*** não lhe envie outra mensagem enquanto esta não tiver sido processada).

Worker
Processo que recebe mensagens através do seu *unnamed pipe* e as processa. Quando o ***Worker*** terminar de processar o pedido, deve colocar o seu estado como livre.

As mensagens recebidas pelo *unnamed pipe* podem ser de dois tipos: 1) dados que foram enviados pelos sensores; ou 2) comandos enviados pelas consolas. O ***Worker*** deve ser capaz de processar e satisfazer ambos os tipos de mensagens.

Se receber dados dos sensores, deve aceder à memória partilhada, obter a informação existente para aquela chave (se existir) e atualizar a informação. Os dados provenientes dos sensores a serem armazenados para cada chave são: último valor recebido (inteiro), valor mínimo recebido (inteiro), valor máximo recebido (inteiro), média dos valores recebidos (vírgula flutuante de precisão dupla, ou seja, *double*), e total de vezes que a chave foi atualizada. Existe um limite para o nº de chaves que podem ser armazenadas no sistema (definido no ficheiro de configuração). Se este limite tiver sido atingido, então a operação deve ser descartada e uma mensagem a indicar que tal aconteceu deve ser impressa no *log*.

Para além disso, após receber dados de um sensor, é necessário incluir esse sensor na lista de sensores que já comunicaram com o sistema. Esta lista tem um tamanho limitado (definido no ficheiro de configuração), pelo que se a lista já tiver atingido o seu limite, a operação de adicionar um novo sensor à lista deve ser descartada e uma mensagem a indicar que tal aconteceu deve ser impressa no *log*.

Se receber um comando da consola que implique apenas a leitura de dados (ou seja, um dos `stats`, `sensors`, `list_alerts`), deve aceder à memória partilhada para ler os dados pedidos e devolvê-los de volta à respetiva consola através da fila de mensagens. Se for um comando que implique alterações (ou seja, `reset`, `add_alert`, `remove_alert`), deve fazer

essas alterações e depois devolver uma mensagem de sucesso (OK) ou erro (ERROR) à consola, através da fila de mensagens.

Alerts Watcher
Deve ser possível enviar notificações de alertas para os ***User Consoles*** em tempo real. Para isso é necessário que na memória partilhada estejam os valores a serem verificados. Quando os valores saírem do intervalo especificado, então o ***Alerts Watcher*** deve enviar um alerta para as ***User Consoles*** através da fila de mensagem. Sempre que isto ocorra durante a execução da simulação, deve ser gerada uma mensagem indicando que determinado alerta foi despoletado.

Para todos os processos do simulador
Todos os processos mantêm a memória partilhada atualizada, usando os mecanismos necessários para que não seja possível a existência de corrupção de dados.

Fim controlado do simulador de Internet das Coisas
O sistema que controla a recolha de dados dos sensores tem de terminar de forma controlada. Ao receber um SIGINT através do ***System Manager*** o sistema segue as seguintes etapas:

- Escreve no log que o programa vai acabar.
- As *thread**s Dispatcher***, ***Sensor Reader*** e ***Console Reader*** param de funcionar, deixando assim o sistema de receber dados dos processos ***Sensor*** ou ***User Console***.
- Aguarda que todas as tarefas (processamento de dados ou comandos) que estejam a executar nos ***Workers*** terminem.
- Escreve no log as tarefas que estão na fila **INTERNAL_QUEUE** e que não chegaram a ser executadas.
- Após as tarefas terminarem, remove todos os recursos utilizados pelo sistema (inclui todos os recursos utilizados, com exceção dos processos ***Sensors*** e ***User Console***) .

Ficheiro de Configurações
O ficheiro de configurações deverá seguir a seguinte estrutura:

```
QUEUE_SZ - número de slots na fila INTERNAL_QUEUE, que é usada pelo Dispatcher, Sensor Reader e Console Reader (>=1)
N_WORKERS - número de processos Worker a serem lançados (>=1)
MAX_KEYS - número máximo de chaves que podem ser armazenadas na memória partilhada (>=1)
MAX_SENSORS - número máximo de sensores diferentes que podem ser usados (>=1)
MAX_ALERTS - número máximo de alertas que podem estar registados (>=0)
```

Exemplo do ficheiro de configurações:

```
10
5
300
10
50
```

Log da aplicação
Todo o *output* da aplicação deve ser escrito de forma legível num ficheiro de texto "log.txt". Cada escrita neste ficheiro deve ser sempre precedida pela escrita da mesma informação na consola, de modo a poder ser facilmente visualizada enquanto decorre a simulação.
Deverá pôr no ***log*** todos os eventos relevantes acompanhados da sua data e hora, incluindo:

- Início e fim do programa;
- Criação de cada um dos processos

- Erros ocorridos
- Mudança de estado de cada *Worker*
- Alertas enviados pelo *Alert Watcher*
- Sinais recebidos

Exemplo do ficheiro de *log*:

```
18:00:05 HOME_IOT SIMULATOR STARTING
18:00:06 THREAD SENSOR_READER CREATED
18:00:06 THREAD CONSOLE_READER CREATED
18:00:06 THREAD DISPATCHER CREATED
18:00:06 PROCESS ALERTS_WATCHER CREATED
(...)
18:00:23 WORKER 1 READY
18:00:23 WORKER 2 READY
(…)
18:00:10 WRONG COMMAND => SENS1#CHAVE 1#
(...)
18:02:00 SIGNAL SIGTSTP RECEIVED
(…)
18:04:00 DISPATCHER: ROOM1_TEMP DATA (FROM ROOM1T1 SENSOR) SENT FOR PROCESSING ON WORKER 1
18:05:00 DISPATCHER: ADD ALERT AL1 (ROOM1_TEMP 10 TO 20) SENT FOR PROCESSING ON WORKER 2

(...)
18:04:30 WORKER1: ROOM1_TEMP DATA PROCESSING COMPLETED
18:05:30 WORKER2: ADD ALERT AL1 (ROOM1_TEMP 10 TO 20) PROCESSING COMPLETED

(...)
18:06:00 ALERT AL1 (ROOM1_TEMP 10 TO 20) TRIGGERED

(...)
18:05:50 SIGNAL SIGINT RECEIVED
18:06:00 HOME_IOT SIMULATOR WAITING FOR LAST TASKS TO FINISH
18:06:10 HOME_IOT SIMULATOR CLOSING
```

3 Checklist

Esta lista serve apenas como indicadora das tarefas a realizar e assinala as componentes que serão objeto de avaliação na defesa intermédia. Tarefas com "(preliminar)" não precisam de estar completas na defesa intermédia.

Item	Tarefa	Avaliado na defesa intermédia?
User Console	Criação do processo User Console	S
	Leitura correta dos parâmetros da linha de comando e de comandos do utilizador	S
	Geração e escrita das tarefas no *named pipe* CONSOLE_PIPE	
Sensor	Criação do processo Sensor	S
	Leitura correta dos parâmetros da linha de comando	S
	Geração e escrita das tarefas no *named pipe* CONSOLE_PIPE	
System Manager	Arranque do sistema, leitura do ficheiro de configurações, validação dos dados do ficheiro e aplicação das configurações lidas.	S
	Criação da memória partilhada	S
	Criação dos *named pipes* SENSOR_PIPE e CONSOLE_PIPE	
	Criação dos processos *Worker*	S
	Criação do processo *Alerts Watcher*	S
	Criação das *threads*	S
	Criação da fila de mensagens	
	Capturar o sinal SIGINT, terminar a corrida e liberta os recursos	
Alerts Watcher	Gerir alertas	
	Envio de alerta para os *User Console* através da fila de mensagens	
Worker	Ler trabalhos do unnamed pipe	
	Executar as tarefas	
Ficheiro *log*	Envio sincronizado do *output* para ficheiro de *log* e ecrã.	S
Geral	Criar um *makefile*	S
	Diagrama com a arquitetura e mecanismos de sincronização	S (preliminar)
	Suporte de concorrência no tratamento de pedidos	
	Deteção e tratamento de erros.	
	Atualização da shm por todos os processos e *threads* que necessitem	
	Sincronização com mecanismos adequados (semáforos, *mutexes* ou variáveis de condição)	S (preliminar)
	Prevenção de interrupções indesejadas por sinais não especificados no enunciado; fornecer a resposta adequada aos vários sinais especificados no enunciado	
	Após receção de SIGINT, terminação controlada de todos os processos e *threads,* e libertação de todos os recursos.	

4 Notas importantes

- **Não será tolerado plágio, cópia de partes de código entre grupos ou qualquer outro tipo de fraude**. Tentativas neste sentido resultarão na **classificação de ZERO valores** e na consequente **reprovação na cadeira**. Dependendo da gravidade poderão ainda levar a processos disciplinares.
- Todos os trabalhos serão escrutinados para deteção de cópias de código.
- Para evitar cópias, os alunos não podem colocar código em repositórios de acesso público.
- Leia atentamente este enunciado e esclareça dúvidas com os docentes.
- Em vez de começar a programar de imediato pense com tempo no problema e estruture adequadamente a sua solução. Soluções mais eficientes e que usem menos recursos serão valorizadas.
- Inclua na sua solução o código necessário à deteção e correção de erros.
- Evite esperas ativas no código, sincronize o acesso aos dados sempre que necessário e assegure a terminação limpa do servidor, ou seja, com todos os recursos utilizados a serem removidos.
- **Penalizações:**
  - A não compilação do código enviado implica uma classificação de **ZERO valores** na meta correspondente.
  - Esperas ativas serão fortemente penalizadas! Use o comando `top` num terminal, de modo a assegurar que o programa não gasta mais CPU que o necessário!
  - Acessos concorrentes que, por não serem sincronizados, puderem levar à corrupção de dados, serão fortemente penalizados!
  - Uso da função `sleep` ou estratagemas similares para evitar problemas de sincronização, serão fortemente penalizados!
- Inclua informação de *debug* que facilite o acompanhamento da execução do programa, utilizando por exemplo a seguinte abordagem:

```
#define DEBUG //remove this line to remove debug messages
(...)
#ifdef DEBUG
printf("Creating shared memory\n");
#endif
```

- Todos os trabalhos deverão funcionar na VM fornecida ou, em alternativa, na máquina student2.dei.uc.pt.
  - Compilação: o programa deverá compilar com recurso a uma *makefile*; não deve ter erros em qualquer uma das metas; evite também os *warnings*, exceto quando tiver uma boa justificação para a sua ocorrência (é raro acontecer!).
- A defesa final do trabalho é obrigatória e todos os elementos do grupo devem participar. A não comparência na defesa final implica a classificação de **ZERO valores** no trabalho.
- O trabalho pode ser realizado em grupos de até 2 alunos (grupos com apenas 1 aluno devem ser evitados e grupos com mais de 2 alunos não são permitidos).
- A nota da defesa é individual pelo que cada um dos elementos do grupo poderá ter uma nota diferente;
- Os alunos do grupo devem pertencer a turmas PL do mesmo docente. Grupos com alunos de turmas de docentes diferentes são exceções que carecem de aprovação prévia.
- Ambas as defesas, intermédia e final, devem ser realizadas na mesma turma e com o mesmo docente.

5 Metas, entregas e datas

Data	**Meta**	
Data de entrega no Inforestudante 11/04/2023-09h00	Entrega intermédia	Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho e submeta-o no Inforestudante: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Inclua todos os ficheiros fonte e de configuração necessários e também um Makefile para compilação do programa. ● Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões) ● Com o código deve ser entregue **1 página A4 com a arquitetura e todos os mecanismos de sincronização a implementar descritos**. ● **Não serão admitidas entregas por e-mail**.
Semana de 11/04/2022	Demonstração /defesa intermédia	● A demonstração deverá contemplar todos os pontos referidos na *checklist* que consta deste enunciado. ● A demonstração/defesa será realizada nas aulas PL. ● A defesa intermédia vale **20%** da cotação do projeto.
Data de entrega final no Inforestudante 13/05/2023-22h00	Entrega final	Crie um arquivo no formato **ZIP** (**NÃO SERÃO ACEITES OUTROS FORMATOS**) com todos os ficheiros do trabalho e submeta-o no Inforestudante: ● Os **nomes** e **números** dos alunos do grupo devem ser colocados no início dos ficheiros com o código fonte. ● Inclua todos os ficheiros fonte e de configuração necessários e também um Makefile para compilação do programa. ● Não inclua quaisquer ficheiros não necessários para a compilação ou execução do programa (ex. diretórios ou ficheiros de sistemas de controlo de versões) ● Com o código deve ser entregue um **relatório** sucinto (no máximo 2 páginas A4), no formato **pdf (NÃO SERÃO ACEITES OUTROS FORMATOS)**, que explique as opções tomadas na construção da solução. Inclua um esquema da arquitetura do seu programa. Inclua também informação sobre o tempo total despendido (por cada um dos dois elementos do grupo) no projeto. ● **Não serão admitidas entregas por e-mail**.
15/05/2023 a 02/06/2022	Defesa final	● A defesa final vale **80%** da cotação do projeto e consistirá numa análise detalhada do trabalho apresentado. ● Defesas em grupo nas aulas PL. ● É necessária inscrição para a defesa.

Depois da submissão é aconselhado fazer o download dos ficheiros para verificar se tudo o que é necessário foi submetido. Caso submetam a pasta errada, apenas parte dos ficheiros, etc., isso não poderá contar para a meta.