MilkV – Real-Time Edge Data Sensing and Fusion

Relatório Final do Projeto Integrador

Angy Pita - up202007253
Diogo Leandro – up202005304
Luís Ganço – up202004196
Martim Ferreira - up20228429

U.PORTO

Licenciatura em Engenharia Informática e Computação

Orientador na U.Porto: Pedro Ferreira Souto

Data: 27 de junho de 2025

Conteúdo

1 Introdução

1.1 Enquadramento

O presente relatório tem como objetivo documentar o estudo desenvolvido pelo grupo sobre o *Single Board Computer* Milk-V Duo S, com o intuito de avaliar as suas capacidades e a relevância da sua utilização tanto na Licenciatura em Engenharia Informática e Computação (LEIC) como no Mestrado em Engenharia Informática e Computação (MEIC).

1.2 Objetivos e resultados esperados

Inicialmente é esperado conseguir inicializar (*boot*) o sistema operativo Linux no Milk-V e avaliar a sua facilidade da instalação. Posteriormente, serão testadas as suas capacidades através do desenvolvimento de programas com foco na utilização de *system calls* do Linux semelhantes aos desenvolvidos nas unidades curriculares de Sistemas Operativos e Redes de Computadores.

O principal objetivo do projeto consiste em instalar um módulo de *kernel* que permita emular *system calls* do sistema Minix em ambiente Linux. A funcionalidade deste módulo será avaliada através do desenvolvimento de aplicações que interagem com dispositivos de entrada/saída (I/O), validando assim a viabilidade do Milk-V Duo S para atividades de ensino e experimentação.

No final do projeto, é esperada uma avaliação objetiva das funcionalidades de *hardware* disponíveis na plataforma Milk-V, bem como uma avaliação subjetiva da dificuldade do processo de desenvolvimento dessas aplicações.

1.3 Estrutura do relatório

O presente relatório inicia-se com uma introdução ao tema abordado neste projeto, incluindo os seus objetivos, a metodologia adotada, os intervenientes envolvidos e os requisitos necessários à sua realização.

De seguida, é apresentado em detalhe o trabalho desenvolvido ao longo de quatro etapas, nomeadamente: **Inicialização do Sistema Linux**, **Desenvolvimento de Aplicações que utilizam chamadas ao sistema Linux *System Calls***, **Ligação à *Internet* e Teste de Comunicação com *Sockets***, e **Compilação e Instalação do Módulo Minix**. Para cada etapa, descrevem-se os respetivos objetivos, o trabalho realizado, os problemas encontrados, os resultados obtidos e as conclusões retiradas.

Por fim, é feita uma reflexão sobre o projeto desenvolvido, destacando os resultados alcançados, as aprendizagens adquiridas e o trabalho futuro a considerar com base no que foi realizado.

2 Metodologia utilizada

2.1 Metodologia utilizada

No desenvolvimento deste projeto, seguimos uma abordagem estruturada baseada na divisão de tarefas e na comunicação contínua entre os membros do grupo.

As reuniões presenciais ocorreram semanalmente na sala B332, às sextas-feiras, das 9h30 às 12h30, onde discutimos o progresso do projeto e trabalhamos ativamente no seu desenvolvimento. Além disso, utilizámos o Discord como plataforma principal para comunicação assíncrona e reuniões adicionais, permitindo a partilha de informações, documentação e ficheiros relevantes para o desenvolvimento. Complementarmente, mantivemos contacto com o professor através do *e-mail* e também realizámos reuniões remotas, onde esclarecemos dúvidas e recebemos *feedback* sobre o nosso trabalho.

O projeto foi dividido em quatro fases principais, conforme planeado:

- **1. Configuração do ambiente (3 semanas) -** Instalação e configuração do Linux no Milk-V Duo S, criando o ambiente necessário para o desenvolvimento das aplicações;
- **2. Desenvolvimento e execução de aplicações com chamadas de sistema Linux (2 semanas) -** Implementação de aplicações baseadas em chamadas de sistema, conforme abordado na unidade curricular de Sistema Operativos;
- **3. Desenvolvimento de aplicações de rede (2 semanas) -** Implementação e teste de aplicações relacionadas com a unidade curricular de Redes de Computadores;
- **4. Desenvolvimento de um módulo do *kernel* e emulação de chamadas do sistema Minix (3 semanas) -** Instalação e teste de un módulo do *kernel* e de uma biblioteca que emula chamadas do sistema Minix no Linux.

As últimas semanas foram dedicadas à finalização do relatório, revisão e ajustes finais.

Esta metodologia permitiu-nos manter uma organização clara das tarefas, garantindo um fluxo de trabalho eficiente e uma comunicação eficaz entre todos os envolvidos.

2.2 Intervenientes, papéis e responsabilidades

A equipa do projeto trabalhou de forma colaborativa, com todos os membros a contribuírem em todas as fases do desenvolvimento. As responsabilidades foram partilhadas, garantindo um esforço integrado e coordenado ao longo do projeto.

O único *stakeholder* externo envolvido foi o orientador Pedro Ferreira Souto, que forneceu supervisão, orientação técnica e *feedback*, auxiliando na condução e validação dos resultados obtidos.

3 Requisitos e Restrições

3.1 Requisitos de *Hardware*

Para a realização deste projeto, é necessário contar com certos componentes físicos:

- **Milk-V Duo S:** O Milk-V Duo S é uma plataforma de desenvolvimento embebida ultracompacta baseada no processador SG2000, projetada para executar Linux e RTOS. Um dos seus diferenciais é a capacidade de alternar entre as arquiteturas RISC-V e ARM, permitindo maior adaptabilidade a diferentes cenários de desenvolvimento.
- **Computador:** Utiliza-se para descarregar os ficheiros necessários, escrever a imagem do sistema operativo no cartão *microSD* e comunicar com o Milk-V Duo S durante a sua configuração e utilização.
- **Cartão *microSD* com uma capacidade superior a 1GB:** O cartão *microSD* atua como o principal meio de armazenamento do Milk-V Duo S, sendo nele que a imagem do Sistema Operativo Linux é gravada. Esse processo é essencial para permitir o arranque e o correto funcionamento do dispositivo, viabilizando a execução do sistema e das aplicações necessárias.
- **Adaptador de cartões SD ou leitor de cartões:** Para transferir a imagem do sistema operativo para o cartão *microSD* a partir de um computador, é necessário um leitor de cartões SD ou um adaptador USB. Isto é essencial quando o computador não dispõe de uma ranhura para cartões *microSD*.
- **Cabo USB-C para USB-A:** Este cabo é necessário tanto para alimentar o Milk-V Duo S como para estabelecer a comunicação entre o computador e o Milk-V Duo S durante a configuração e o desenvolvimento do projeto.

3.2 Requisitos de *Software*

A execução deste projeto requer certos componentes de *software*, que são necessários para a configuração e funcionamento do sistema:

- **Imagem Linux do Milk-V Duo S (compatível com a arquitetura escolhida ARM ou RISC-V):** Este é o sistema operativo que será executado no Milk-V Duo S. Pode ser descarregado do site oficial ou do repositório do Milk-V e instalado no cartão *microSD*, permitindo o arranque e funcionamento do dispositivo.

- **Programa de *flashing* (para escrever a imagem no cartão *microSD*):** Para instalar o sistema operativo no Milk-V Duo S, é necessário um *software* de gravação, como BalenaEtcher ou Rufus. Estes programas permitem escrever corretamente a imagem do sistema no cartão *microSD*, garantindo que o dispositivo possa arrancar sem problemas.
- **Git(Acesso ao repositório oficial da Milk-V):** O repositório oficial do Milk-V no GitHub disponibiliza ficheiros essenciais, incluindo o SDK e ferramentas necessárias para o desenvolvimento na plataforma. Além disso, fornece a imagem do Linux utilizada no dispositivo.

3.3 Restrições

A versão SDK V1 não oferece suporte para arquiteturas ARM, o que impossibilita o seu uso em dispositivos baseados nesta arquitetura, como o Milk-V Duo S. Para garantir compatibilidade e funcionamento adequado, é necessário utilizar a versão SDK V2, que inclui suporte para ARM e RISC-V, permitindo o desenvolvimento correto no ambiente escolhido.

4 Atividades Desenvolvidas

4.1 Inicialização Sistema Linux

4.1.1 Descrição

Nesta primeira fase do projeto, configuramos o Milk-V Duo S para o desenvolvimento de aplicações, incluindo a instalação do sistema Linux, a configuração da sua memória e o ajuste do ambiente de trabalho dos nossos computadores. Para isso, gravamos a imagem do sistema no cartão *microSD* e garantimos que o *switch* de arquitetura estivesse corretamente ajustado para RISC-V.

Durante a inicialização, enfrentamos desafios como falhas no cartão *microSD*, que impediram o arranque do sistema. Após testar outro cartão e ajustar a partição corretamente, conseguimos estabelecer comunicação via SSH, garantindo que o dispositivo estivesse pronto para receber código e executar aplicações.

4.1.2 Instalação Linux

Para preparar o ambiente de desenvolvimento e testar a execução de programas no Milk-V Duo S, foi seguido um conjunto de passos descritos na documentação oficial disponível no site, criado pelos desenvolvedores da Milk-V, Milk-V Duo S - Getting Started.

Primeiramente, realizou-se o *download* da imagem Linux compatível com a arquitetura RISC-V (SDK V2) a partir do repositório git oficial da Milk-V Duo S GitHub - duo-buildroot-sdk. Antes de gravá-la no cartão *microSD*, verificamos a configuração do *switch* de arquitetura, um pequeno interruptor físico localizado

na parte superior do Milk-V Duo S, próximo ao conector USB. Esse *switch* permite alternar entre os modos RISC-V e ARM, definindo qual processador será utilizado para a execução do sistema. Para garantir a compatibilidade, ajustamos o *switch* para a posição correspondente ao RISC-V, conforme indicado na documentação.

Em seguida, a gravação (*burning*) da imagem foi feita no cartão *microSD* utilizando o *software* BalenaEtcher. Para isso, abrimos o BalenaEtcher, selecionamos a imagem descarregada, escolhemos o cartão *microSD* como destino e iniciamos a gravação. No final do processo, removemos o cartão do computador e inserimo-lo no Milk-v Duo S, utilizando o *slot* localizado na parte inferior do Milk-V Duo S, próximo ao conector USB-C.

Por fim, conectamos o dispositivo ao computador por meio do cabo USB-C, que fornece alimentação e comunicação. Para aceder ao sistema, foi estabelecida uma ligação via SSH ao Milk-V pelo IP predefinido `192.168.42.1`, possibilitando a interação com a linha de comandos do Milk de forma remota com o comando `ssh root@192.168.42.1`.

AVISO: Se mais do que um dispositivo Milk-V Duo S estiver a ser usado, pode ocorrer um conflito com a ligação SSH, porque ambos têm o mesmo IP por predefinição.

4.1.3 Configuração da memória do Milk-V Duo S

Após a instalação do sistema operativo, foi necessário configurar o ambiente de desenvolvimento na Milk-V Duo S e no computador utilizado para a compilação e execução de aplicações.

Para preparar a memória do cartão *microSD* para o desenvolvimento de aplicações, configuramos uma partição de memória dedicada ao Milk-V Duo S. Este processo foi automatizado através do *script* ('resize.sh') disponibilizado na documentação oficial do Milk-V Duo S. Este *script* redimensionou automaticamente a partição *root* para utilizar toda a capacidade disponível no cartão *microSD*. Para isso, executamos o seguinte comando diretamente no terminal do Milk-V Duo S:

```
./resize.sh
```

Após a execução, reiniciamos o dispositivo para que as alterações tivessem efeito. Em seguida, verificamos se a expansão havia sido aplicada corretamente utilizando o comando:

```
df -h
```

Este comando exibiu o uso do armazenamento no sistema, permitindo confirmar que a partição *root* havia sido expandida para a capacidade total do cartão *microSD*. Caso a expansão não tivesse ocorrido corretamente, poderíamos ter realizado o processo manualmente utilizando ferramentas como `fdisk` ou `parted`, mas o *script* `resize.sh` automatizou esta tarefa de forma eficiente.

Com esta configuração concluída, o sistema passou a ter espaço suficiente para armazenar bibliotecas, arquivos de desenvolvimento e aplicações compiladas diretamente no Milk-V Duo S.

4.1.4 Configuração do Ambiente de Desenvolvimento

Além disso, configurámos o ambiente de desenvolvimento no computador, garantindo a compatibilidade com a Milk-V Duo S para a criação e transferência de aplicações. Para isso, seguimos as recomendações do repositório oficial do projeto na documentação oficial do Buildroot SDK e preparámos o sistema para compilar código compatível com o dispositivo.

Com o ambiente devidamente configurado, realizámos o *download* do SDK oficial da Milk-V Duo S, disponível no GitHub. Para isso, clonámos o repositório e acedemos ao diretório correspondente GitHub oficial do Milk-V.

A partir desse repositório, foi possível aceder aos recursos necessários para a construção e desenvolvimento de aplicações para o Milk-V Duo S, incluindo ferramentas para compilar código, gerar binários e realizar a configuração avançada do ambiente. Para garantir a correta configuração do ambiente de desenvolvimento, foi necessário preparar tanto o Milk-V Duo S quanto o computador utilizado para a compilação e transferência de aplicações, seguindo as recomendações do repositório oficial do projeto. Com o ambiente devidamente configurado, compilámos um programa simples (*Hello World*) em C e executámos no Milk-V Duo S para validar o processo de construção e execução de binários.

Para finalizar a configuração do ambiente, executámos o *script* responsável pela configuração das variáveis de ambiente:

```
source envsetup.sh
```

Sempre que abrirmos um novo terminal, para compilar para o Milk-V Duo S, precisamos de executar este *script* novamente.

Além disso, adicionámos o *Makefile* fornecido pelo repositório oficial da Milk-V, que foi modificado por nós para garantir o correto carregamento das variáveis de ambiente durante a compilação.

No caso de se usar WSL (Windows Subsystem for Linux), os passos da configuração são semelhantes.

AVISO: Ao usar WSL para a configuração é necessário resolver o problema das variáveis de ambiente do Windows através da alteração do ficheiro /etc/wsl.conf e adicionar as seguintes linhas:

```
[interop]
appendWindowsPath = false
```

4.1.5 Teste de Execução de um Programa no Milk-V Duo S

O ficheiro executável gerado foi copiado para o Milk-V utilizando o protocolo SCP. Por fim, o programa foi executado diretamente no dispositivo, sendo o resultado observado no terminal, validando assim a funcionalidade básica da plataforma e a integração entre o ambiente de desenvolvimento e o Milk-V Duo S.

4.1.6 Problemas Encontrados

Durante a fase inicial da configuração do Milk-V Duo S, identificámos um problema relacionado ao cartão *microSD*, que impediu a inicialização correta do sistema. Após instalar a imagem do sistema operativo Linux e inserir o cartão na ranhura correspondente, conectámos o Milk-V ao computador. No entanto, embora a luz LED azul tenha acendido, esta permaneceu fixa, sem piscar, e não foi possível inicializar o sistema.

Perante esta situação, realizámos um diagnóstico para identificar a possível causa do problema. Primeiramente, testámos diferentes cabos USB, verificando se poderia tratar-se de uma falha na alimentação ou na comunicação entre o Milk-V e o computador. Como essa alternativa não apresentou resultados, considerámos a possibilidade de um defeito no cartão *microSD*. Para validar essa hipótese, testámos o mesmo cartão em diferentes dispositivos e utilizámos diferentes programas de *flashing* para gravar a imagem do sistema, garantindo que o problema não estivesse relacionado com o processo de gravação. Ainda assim, o sistema não inicializou corretamente.

Por conseguinte, adquirimos um novo cartão *microSD* com a mesma capacidade, repetimos o procedimento de instalação do sistema operativo e, desta vez, o Milk-V Duo S iniciou corretamente. Esse teste confirmou que o problema estava no cartão *microSD* original, que possivelmente apresentava falhas de leitura ou escrita.

4.1.7 Resultados

A instalação do sistema Linux no Milk-V e a preparação do ambiente de desenvolvimento foram bem-sucedidas.

4.2 Desenvolvimento de Aplicações que fazem uso de Linux *System Calls*

4.2.1 Descrição

Nesta etapa do projeto, estudámos a possibilidade de desenvolver programas que fazem uso de Linux *System Calls* no Milk-V Duo S e procurámos detetar possíveis erros aquando do desenvolvimento de aplicações para o Milk-V.

Para o desenvolvimento dos programas de teste, começámos por programá-los e testá-los no computador local. Após confirmar o correto funcionamento dos

programas, seguimos o processo descrito nas secções 4.1.4 e 4.1.5 do presente relatório para realizar a compilação cruzada e transferir os ficheiros executáveis.

Durante a fase de desenvolvimento, encontrámos um erro na compilação cruzada devido à má execução do *script* de configuração de ambiente, um erro na transferência de ficheiros para o Milk-V via `scp`, dado um problema de compatibilidade de protocolos de transferência de ficheiros e um erro na conexão ao Milk-V causado pela utilização do mesmo endereço `IP` em dois Milk-V Duo S diferentes.

Os programas desenvolvidos nesta componente fazem uso das seguintes *system calls*:

- *System calls* relacionadas a processos:
  - `getpid()`, `fork()`, `exec()`, `pipe()`;
- *System calls* para a manipulação de ficheiros:
  - `open()`, `close()`, `read()`, `write()`;
- *System calls* para a criação e gestão de *threads*:
  - `pthread_create()`, `pthread_join()`, `pthread_mutex_init()`, `pthread_mutex_destroy()`, `pthread_mutex_lock()` e `pthread_mutex_unlock()`.

4.2.2 Desenvolvimento de Aplicações

Para dar início ao desenvolvimento de aplicações, começámos por criar um repositório no GitHub para organizar e documentar os programas implementados. De seguida, procedemos à programação de diversas aplicações de teste, garantindo o seu correto funcionamento no computador local.

Após a fase de programação, para realizar a compilação cruzada dos programas para o ambiente do Milk-V Duo S, seguimos os passos descritos na secção 4.1.4 do presente relatório. Assim, executámos o *script* de configuração do ambiente e utilizámos a *Makefile* fornecida no repositório oficial do Milk-V Duo S.

4.2.3 Teste de Execução

O teste de execução do programa seguiu os mesmos passos descritos na secção 4.1.5 do presente relatório. Primeiramente, o ficheiro executável foi copiado para o dispositivo Milk-V Duo S utilizando o protocolo SCP. Após a a transferência do ficheiro, estabelecemos a ligação com o dispositivo Milk-V Duo S via SSH, utilizando as credenciais do utilizador *root*. De seguida, o programa foi executado diretamente no dispositivo, permitindo a observação do resultado no terminal.

4.2.4 Problemas Encontrados

Durante esta fase, analisámos a forma correta de carregar as variáveis de ambiente no Milk-V Duo S. Constatámos que o uso do comando *source* era essencial para garantir que as variáveis necessárias fossem corretamente definidas e mantidas no ambiente de trabalho. Especificamente, verificámos que o *script envsetup.sh* não era executado corretamente quando chamado com *sh envsetup.sh* ou *./envsetup.sh*, pois essas abordagens criavam um novo processo, impedindo que as configurações fossem preservadas. A execução destes comandos resultava na mensagem de erro "*TOOLCHAIN not set*", indicando que as variáveis necessárias para a compilação cruzada não estavam carregadas. Somente ao utilizar *source envsetup.sh* foi possível aplicar corretamente as variáveis de ambiente, garantindo que o sistema funcionasse adequadamente durante a inicialização.

Aquando da fase de teste de execução dos programas no Milk-V Duo S, deparámo-nos com um problema na transferência do ficheiro executável para o dispositivo através do *scp*. Por exemplo, ao executar o comando:

```
scp tmutex_duos root@192.168.42.1:/root/
```

Recebemos a mensagem de erro "sh: /usr/libexec/sftp-server: *not found*" seguida de "*scp: Connection closed*". Esse erro ocorre porque, por padrão, o 'scp' tenta utilizar o protocolo 'sftp', mas o Milk-V Duo S não possui o servidor 'sftp' configurado corretamente. Para resolver essa questão, foi necessário utilizar a *flag -O*, garantindo que o *scp* operasse no modo clássico, evitando o uso do 'sftp' e permitindo a transferência bem-sucedida do executável para o Milk-V Duo S. O comando corrigido ficou da seguinte forma:

```
scp -O tmutex_duos root@192.168.42.1:/root/
```

Com essa abordagem, a transferência foi realizada com sucesso, possibilitando a execução do programa diretamente no dispositivo. Esse problema foi documentado para futuras referências, assegurando que os mesmos erros possam ser evitados em desenvolvimentos posteriores.

No decorrer desta etapa, decidimos adquirir um Milk-V Duo S adicional. Ao tentarmos conectar ao dispositivo via **SSH**, verificámos um erro relacionado às *host keys* utilizadas pelo computador local para identificar e estabelecer ligações seguras entre dispositivos. Este erro aconteceu devido à utilização do mesmo endereço *IP* em ambos os dispositivos Milk-V, pelo que foi necessário alterar o endereço *IP* de um dos dispositivos Milk-V para solucionar este problema.

4.2.5 Resultados

Após a execução dos programas de teste no Milk-V Duo S, verificámos que o uso das *system calls* selecionadas foi bem-sucedido, comprovando a viabilidade do desenvolvimento de aplicações baseadas em chamadas ao sistema Linux neste dispositivo.

Os testes de criação e gestão de processos com `fork()`, `exec()` e `pipe()` funcionaram corretamente, permitindo a comunicação entre processos de forma eficiente. Da mesma forma, as operações de manipulação de ficheiros com `open()`, `close()`, `read()` e `write()` foram executadas sem erros, garantindo a integridade dos dados lidos e escritos no sistema de ficheiros do Milk-V Duo S.

No que diz respeito à criação e sincronização de *threads*, os testes com `pthread_create()` e `pthread_join()` demonstraram um funcionamento adequado, permitindo a execução concorrente de tarefas. Além disso, o uso de *mutexes* através das chamadas `pthread_mutex_init()`, `pthread_mutex_lock()` e `pthread_mutex_unlock()` preveniu corretamente condições de corrida entre *threads*, garantindo a consistência dos dados partilhados.

Apesar dos desafios encontrados, as soluções adotadas permitiram a execução bem-sucedida das aplicações desenvolvidas. Assim, os testes realizados validam a possibilidade de desenvolver e executar programas que fazem uso de Linux *System Calls* no Milk-V Duo S, confirmando a viabilidade da plataforma para aplicações que requerem interação direta com o sistema operativo.

4.3 Ligação *Internet* e Teste de Comunicação Com *Sockets*

4.3.1 Descrição

Nesta estapa do projeto, enfrentámos inicialmente o problema apresentado na secção 4.2.4, relacionado com a impossibilidade de estabelecer uma ligação SSH ao segundo dispositivo Milk-V Duo S. Para resolver esta limitação, realizámos uma mudança de *IP*, configurando manualmente um novo endereço para o segundo dispositivo.

Após esta configuração, avançámos para a ligação do dispositivo à rede *internet*, garantindo assim o seu acesso à rede. Com a conectividade estabelecida, testámos a comunicação entre dispositivos através de uma transferência de ficheiros, de acordo com o protocolo FTP. Para tal, desenvolvemos e executámos um programa que faz uso de *sockets* para comunicação com um servidor FTP e a receção dos ficheiros.

4.3.2 Ligação à *Internet*

Para testar a capacidade do Milk-V Duo S no desenvolvimento de programas relacionados com redes informáticas, começámos por estabelecer uma ligação à *internet*.

Devido à ausência de computadores pessoais com entrada para cabo Ethernet, bem como a indisponibilidade de um cabo adequado, optámos por estabelecer uma ligação *wireless* à *internet* através de uma rede *hotspot* de um telemóvel.

Para tal, começámos por ativar a *interface internet*. Depois, inicializámos a funcionalidade *wpa_suplicant* e configurámos o ficheiro de acordo com as con-

figurações e credenciais correspondentes à rede a estabelecer ligação. Após a configuração do *wpa_supplicant*, inicializámos uma tentativa de ligação ao *hotspot*, e verificámos a correta conexão do dispositivo à rede *wireless*.

4.3.3 Teste de Comunicação Com *Sockets*

Após a ligação à *internet*, testámos a execução de um programa responsável pela transferência do ficheiro *readme.txt* disponibilizado no servidor FTP *test.rebex.net*, um servidor utilizado para teste de conexões FTP, com o objetivo de analisar o correto funcionamento do Milk-V e a sua acessibilidade no desenvolvimento de programas no âmbito de redes informáticas.

Para tal, desenvolvemos uma aplicação para o *download* de ficheiros via FTP e seguimos o procedimento descrito nas secções 4.2.2 e 4.2.3 para a compilação cruzada e execução da aplicação no Milk-V.

Após a execução do programa no dispositivo, verificámos a transferência bem-sucedida do ficheiro.

4.3.4 Problemas Encontrados

Durante esta fase, constatámos que o Milk-V obteve dificuldades em conectar-se a um rede local *wireless* (*eduroam*) devido à proteção da rede, e, consequentemente, à falta de autenticação com a mesma. Assim, para contornar esse problema, utilizámos um *hotspot* de um dispositivo móvel, que por não ter qualquer tipo de proteção facilita a conexão.

4.3.5 Resultados

Depois de estabelecida a ligação à *internet* e concluídos os testes com *sockets*, confirmámos que o Milk-V Duo S é perfeitamente capaz de lidar com comunicação em rede. A ligação *wireless*, feita através de um *hotspot* de telemóvel, funcionou bem e manteve-se estável durante toda a realização dos testes.

O programa que desenvolvemos conseguiu ligar-se ao servidor *test.rebex.net* sem problemas, fazer *login* e transferir corretamente o ficheiro *readme.txt*. A comunicação com o servidor foi feita usando *sockets* e o protocolo FTP, e não houve qualquer falha durante o processo.

O uso de *sockets* mostrou-se eficaz e simples de implementar. Todo o processo — desde o desenvolvimento da aplicação, passando pela compilação cruzada até à execução no Milk-V — decorreu sem dificuldades de maior.

No geral, os testes mostraram que o dispositivo está bem preparado para aplicações que envolvam ligação à *internet* e troca de dados com outros sistemas. Isto confirma que o Milk-V Duo S pode ser usado com confiança em projetos que exijam comunicação de rede.

4.4 Compilação e Instalação do Módulo Minix

4.4.1 Descrição

Nesta fase, pretende-se compilar e instalar o módulo Minix do *kernel* no Milk-V Duo S, em RISC-V. Como primeiro passo, procedeu-se à compilação e instalação de um módulo de teste "*Hello World*", com o objetivo de validar o processo. Confirmado o seu correto funcionamento, avançou-se para a integração do módulo do Minix.

4.4.2 Preparação do Ambiente para Compilação de Módulos do *Kernel*

Este teste de compilação segue a metodologia apresentada num tutorial da Unidade Curricular de Sistemas Operativos do MIEEC (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores), baseado no livro "Linux Device Drivers, 3rd Edition". Este tutorial tem como objetivo orientar a compilação e execução de um módulo simples e personalizado do *kernel*.

O tutorial disponibiliza o código de teste `hello.c`, bem como o respetivo `Makefile`. Tendo em conta que o desenvolvimento de programas e módulos para o Milk-V é realizado externamente, num computador anfitrião, através de compilação cruzada, é necessário adaptar alguns aspetos do processo às restrições impostas por este ambiente.

Inicialmente, é necessário dispor de uma *toolchain* compatível para compilar para RISC-V, uma vez que o uso do *script* `envsetup.sh` e do compilador por ele fornecido gera erros de incompatibilidade entre versões. Para contornar este problema, optou-se por utilizar uma versão mais antiga da *toolchain* (10.5.0), que permite acomodar comandos incompatíveis com a versão problemática, garantindo assim a correta compilação.

Para instalar a *toolchain* na versão 10.5.0, utilizou-se o seguinte comando: `sudo apt install gcc-10-riscv64-linux-gnu`.

De seguida, é necessário obter o código-fonte do *kernel* correspondente ao Milk-V, bem como o seu ficheiro de configuração `.config`.

Para guardar uma cópia do ficheiro de configuração `.config` do Milk-V Duo S, executámos o seguinte comando no dispositivo:

```
zcat /proc/config.gz > milk_v_config
```

Posteriormente, transferimos o ficheiro para o computador anfitrião (*host PC*) utilizando o protocolo SCP.

Importante: Não nomear o ficheiro como `.config`, pois durante a transferência via SCP este pode sobrescrever o ficheiro `.config` já existente no computador anfitrião, podendo causar problemas indesejados.

O código-fonte do *kernel* do Milk-V Duo S encontra-se disponível no

repositório oficial do Milk-V, na pasta `linux_5.10`. É fundamental utilizar o código-fonte correspondente à versão do *kernel* em que o módulo será compilado, garantindo assim a compatibilidade entre o módulo e o *kernel* do Milk-V.

O passo seguinte consiste em mover o ficheiro de configuração para a pasta `linux_5.10` e, posteriormente, renomeá-lo para `.config`.

Para garantir a utilização correta da *toolchain* RISC-V durante a compilação do módulo, foram realizados os seguintes passos de configuração:

- Criação de *links* simbólicos para ferramentas RISC-V específicas numa pasta, garantindo o uso da versão correta:

  ln -sf /usr/bin/riscv64-linux-gnu-gcc-10
  "TOOLCHAIN_DIR/riscv64-linux-gnu-gcc"

  ln -sf /usr/bin/riscv64-linux-gnu-ld
  "TOOLCHAIN_DIR/riscv64-linux-gnu-ld"

  ln -sf /usr/bin/riscv64-linux-gnu-as
  "TOOLCHAIN_DIR/riscv64-linux-gnu-as"

  ln -sf /usr/bin/riscv64-linux-gnu-strip
  "TOOLCHAIN_DIR/riscv64-linux-gnu-strip"2¿/dev/null

- Exportação de variáveis de ambiente para definir o compilador a ser usado e a arquitetura para a qual vamos compilar o módulo:

  export PATH="$TOOLCHAIN\_DIR$:PATH"CROSS_COMPILE="riscv64-linux-gnu"ARCH="riscv"

Dada a possível necessidade de repetir estes comandos, foi criado um *script* em

bash para automatizar a configuração da *toolchain*.

```
#!/bin/bash

TOOLCHAIN\_DIR="$HOME/toolchains/riscv-gcc10"
mkdir -p "$TOOLCHAIN\_DIR"

ln -sf /usr/bin/riscv64-linux-gnu-gcc-10 \
    "$TOOLCHAIN_DIR/riscv64-linux-gnu-gcc"

ln -sf /usr/bin/riscv64-linux-gnu-ld \
    "$TOOLCHAIN_DIR/riscv64-linux-gnu-ld"

ln -sf /usr/bin/riscv64-linux-gnu-as \
    "$TOOLCHAIN_DIR/riscv64-linux-gnu-as"

ln -sf /usr/bin/riscv64-linux-gnu-strip \
    "$TOOLCHAIN_DIR/riscv64-linux-gnu-strip" 2>/dev/null

export PATH="$TOOLCHAIN_DIR:$PATH"
export CROSS\_COMPILE=riscv64-linux-gnu-
export ARCH=riscv
```

De seguida, preparámos o código do *kernel* para a compilação cruzada de módulos, executando os seguintes comandos na pasta `linux_5.10`:

- ***make* olddefconfig:** Atualiza a configuração existente, preenchendo opções novas com os valores por defeito.
- ***make* prepare:** Gera ficheiros necessários para compilar o *kernel* ou módulos internos.
- ***make* modules_prepare:** Prepara os *headers* e estruturas para compilar módulos externos ao *kernel*.

Após a execução destes comandos, o sistema encontra-se preparado para compilar módulos do *kernel* para o Milk-V Duo S.

4.4.3 Teste Hello World

Após configurado o ambiente de compilação cruzada para o Milk-V Duo S, procedeu-se ao teste de compilação e execução de um módulo simples — neste caso, o módulo *Hello World*, disponibilizado no livro "Linux Device Drivers, 3rd Edition".

O código de teste e o respetivo ficheiro *Makefile* são fornecidos pelo livro, sendo que foi apenas necessário executar o *script* de configuração do ambiente de compilação e, em seguida, compilar o módulo com o comando *make*.

Concluída a compilação, o ficheiro binário gerado (*hello.ko*) foi transferido para o Milk-V via SCP.

A inserção do módulo no *kernel* é realizada com o comando `insmod hello.ko`, enquanto a sua remoção é feita com `rmmod hello.ko`.

Por fim, para validar a correta execução do módulo, foi utilizado o comando `dmesg | tail -5`, devendo ser visíveis as mensagens "*Hello, world*"aquando da inserção e "*Goodbye, cruel world*"aquando da remoção.

4.4.4 Compilação e Instalação Módulo Minix

Depois de verificar a correta compilação e instalação do módulo *Hello World*, avançámos para o módulo Minix.

Antes de compilar o módulo, foi necessário executar o *script* de configuração da *toolchain* para compilação de módulos, descrito na secção "Preparação do Ambiente para Compilação de Módulos do *Kernel*".

Durante a primeira tentativa de compilação do módulo do Minix, deparámo-nos com erros relacionados com a ausência de *headers*, incompatibilidades de sintaxe e múltiplos *warnings*. Foram efetuadas alterações estritamente estruturais, assegurando que a funcionalidade do módulo se manteve inalterada.

A versão corrigida do código encontra-se disponível na tag *"milkV_riscV"* do repositório no GitHub, criado especificamente para este fim.

Após aplicar as correções mencionadas, foi possível compilar com sucesso o módulo do Minix, gerando o ficheiro `umdp.ko`.

O módulo foi então transferido para a plataforma Milk-V através do protocolo SCP.

Em seguida, procedeu-se à inserção do módulo com o comando:

```
insmod umdp.ko
```

Através da análise da saída do comando:

```
dmesg | tail -5
```

Foi possível verificar que a *netlink family* foi registada com sucesso, o que confirma que o módulo foi corretamente carregado pelo *kernel*.

Para remover o módulo, utilizou-se o comando:

```
rmmod umdp
```

Mais uma vez, a verificação com `dmesg | tail -5` confirmou a remoção, indicando que a *netlink family* foi *unregistered*, demonstrando que o ciclo de inserção e remoção do módulo está a funcionar como esperado.

4.4.5 Problemas Encontrados

Durante esta fase, foram identificados diversos problemas no processo de teste de compilação através do *kernel*:

- **Incompatibilidade de versões do *kernel*:** Após a compilação do cabeçalho ELF, verificou-se que a versão do *kernel* estava incorreta. O código-fonte do *kernel* utilizado correspondia a uma versão mais recente do que a exigida pela plataforma Milk-V, o que resultou em erros durante a execução. **Solução:** Obter e utilizar o código-fonte do *kernel* correspondente exatamente à versão suportada pelo Milk-V.
- **Incompatibilidade de versões do compilador:** A versão do compilador utilizada para a compilação cruzada era demasiado recente em relação àquela usada para compilar os módulos do *kernel* do Milk-V, o que levou a incompatibilidades. **Solução:** Instalar uma versão do compilador compatível com a utilizada no ambiente de compilação do *kernel* do Milk-V.
- **Ficheiro de *headers* em falta:** Durante a compilação do módulo "*hello world*", foi detetada a ausência de alguns ficheiros *header*, essenciais para o processo. **Solução:** Garantir que todos os ficheiros *header* necessários estão presentes na pasta esperada do código fonte do *kernel*.
- **Problemas com dependências:** Ao utilizar o comando `insmod` para inserir módulos no kernel, foram observados erros relacionados a símbolos indefinidos. Isto ocorre porque o `insmod` não resolve automaticamente as dependências entre módulos, diferentemente do `modprobe`, que não está disponível no Milk-V. **Solução:** Para evitar esses problemas, foi necessário compilar a biblioteca de forma estática e transferi-la para o Milk-V Duo S via SCP, garantindo que todas as funções necessárias estivessem incluídas no binário final.

4.4.6 Resultados

A compilação do módulo para o Minix foi realizada com sucesso, resultando na geração do ficheiro executável umdp.ko. Este resultado confirma que o processo decorreu sem erros, ficando o módulo pronto para ser transferido e testado no Milk-V Duo S.

5 Guião de Configuração e Desenvolvimento com o Milk-V Duo S

5.1 Enquadramento

No decurso do projeto, foi elaborado um guião técnico com o objetivo de documentar, de forma sistemática, clara e acessível, o processo de configuração e desenvolvimento de aplicações no dispositivo Milk-V Duo S. Este guião teve como propósito apoiar a execução das atividades desenvolvidas no âmbito do

projeto, bem como servir de recurso reutilizável por futuros estudantes ou docentes em edições subsequentes de projetos semelhantes. Para além disso, poderá também ser utilizado como material complementar em unidades curriculares da Licenciatura em Engenharia Informática e Computação (LEIC), como é o caso de Laboratório de Computadores (LCOM).

5.2 Conteúdo do Guião

O guião encontra-se estruturado em diversas secções, alinhadas com as necessidades práticas do projeto:

- **Requisitos**: componentes de *hardware* e *software* essenciais para o funcionamento do dispositivo;
- **Descrição dos componentes do Milk-V Duo S**: explicações visuais e funcionais sobre os elementos físicos do dispositivo;
- **Inicialização e instalação do sistema Linux**: procedimentos para transferência e gravação da imagem do sistema operativo no cartão *microSD*;
- **Configuração do ambiente de desenvolvimento**: passos para ligação via SSH, expansão de memória e preparação da compilação cruzada;
- **Compilação cruzada e testes**: criação e execução de programas no Milk-V, com resolução de problemas típicos (ex. permissões e conflitos de IP);
- **Configuração Wi-Fi e de rede**: soluções para ligação a redes protegidas ou alternativas como *hotspot* móvel;
- **Ambiente para desenvolvimento de módulos do kernel**: configuração da *toolchain*, obtenção do código-fonte e testes com módulos personalizados.

5.3 Importância para o Projeto

A criação deste guião revelou-se essencial para:

- Documentar e sistematizar os procedimentos técnicos adotados;
- Facilitar a reprodutibilidade das ações por terceiros;
- Promover a autonomia de aprendizagem e consolidação do conhecimento técnico da equipa;
- Servir de recurso pedagógico para unidades curriculares da Licenciatura em Engenharia Informática e Computação.

5.4 Disponibilização

O guião foi produzido em formato PDF e encontra-se no repositório do projeto.

6 Acesso aos Nossos Repositórios

- **Programas de Teste e Guião:** github.com/DiogoLeandro06/PI
- **Módulo Minix:** github.com/up202004196/minix_milk_v_duo_s/tag/milkV_riscV

7 Conclusões

7.1 Resultados alcançados

Com base nos resultados obtidos ao longo do projeto, conclui-se que o uso do Milk-V Duo S é viável no contexto do curso de Licenciatura em Engenharia Informática e Computação (LEIC) da Universidade do Porto. O Milk-V Duo S demonstrou ser uma escolha adequada para a consolidação de conhecimentos práticos em áreas como sistemas embutidos, programação em C, sistemas operativos, controlo de periféricos e interação com o sistema operativo em *kernel-space* e *user-space*.

Entre as vantagens identificadas, destacam-se o suporte à compilação cruzada, a existência de um ambiente de execução autónomo, a compatibilidade com aplicações de sistemas operativos, e o suporte a funcionalidades de rede, como Wi-Fi, Ethernet e Bluetooth. Estas características tornam o Milk-V Duo S uma ferramenta adequada para o ensino de desenvolvimento de *software* em contextos de baixo nível e de interação direta com o sistema.

Por outro lado, destaca-se como desvantagem o número ainda reduzido de recursos técnicos e de membros ativos na comunidade do Milk-V Duo S, o que dificultou algumas fases do desenvolvimento. No entanto, esses obstáculos foram superados com sucesso através de investigação autónoma, da consulta dos recursos disponíveis nos repositórios oficiais e do aconselhamento prestado pelo professor orientador, cujo apoio foi fundamental na clarificação de dúvidas e na definição de caminhos técnicos mais adequados.

Um dos marcos mais significativos do projeto foi a compilação bem-sucedida do modulo Minix no *kernel* Linux do Milk-V Duo S, demonstrando a sua aplicabilidade em contextos de ensino avançado, nomeadamente no desenvolvimento e teste de módulos a nível de *kernel*, além de possibilitar o desenvolvimento de programas compatíveis com Minix.

No âmbito do trabalho em grupo, todos os elementos participaram ativamente tanto na execução técnica como na redação do relatório. As tarefas foram distribuídas de forma equilibrada entre os membros, estimando-se uma contribuição equivalente de aproximadamente 25% por cada elemento, o que refletiu um esforço colaborativo consistente ao longo de todo o projeto.

7.2 Aprendizagens

A experiência com o Milk-V revelou-se extremamente enriquecedora para a consolidação de conceitos de sistemas embutidos e operações de baixo nível, como a comunicação direta com o *hardware* e interação com o sistema operativo a nível *kernel-space* e *user-space*. Foi também evidente a importância de ferramentas como os *scripts* de automatização.

Destaca-se que a plataforma se adequa especialmente a disciplinas que exijam contacto com conceitos como programação em C, sistemas operativos, arquitetura de computadores, ou eletrónica digital. O nível de abstração reduzido do Milk-V favorece uma aprendizagem mais direta e prática.

7.3 Trabalho futuro

Como trabalho futuro, seria interessante desenvolver aplicações que explorem as funcionalidades do Milk-V, nomeadamente a sua capacidade de comunicação em rede e de processamento. Consideramos também que seria benéfico seguir o guião para a compilação de um módulo do *kernel* ou para o desenvolvimento e teste de aplicações em ambientes distintos do nosso computador, através de compilação cruzada. Acreditamos que esta abordagem proporcionaria uma experiência benéfica para a aprendizagem e consolidação de diversos conceitos na área da Engenharia Informática.

A documentação geral do dispositivo está disponível em *Documentação Geral - Milk-V Duo* [2]. Para interações com a comunidade e suporte informal, pode-se recorrer ao *Comunidade/Forum - Milk-V* [1]. As instruções de início rápido encontram-se em *Milk-V Duo S (Getting Started)* [5]. O processo de ligação e configuração do Milk-V Duo S está documentado em *Milk-V Duo S (Connection and Setup)* [3]. Exemplos práticos para desenvolvimento estão disponíveis no GitHub em *Milk-V Duo S (Connection and Setup - GitHub Examples)* [4]. A inicialização via cartão MicroSD está descrita em *Milk-V Duo S (MicroSD Initialization)* [6]. A ligação à rede WiFi é explicada em *Milk-V Duo S (WiFi Connection)* [8]. Por fim, o processo de compilação de programas é abordado em *Milk-V Duo S (Program Compilation)* [7].

Referências

[1] Milk-V. Comunidade/forum - milk-v. `https://community.milkv.io/`.

[2] Milk-V. Documentação geral - milk-v duo. `https://milkv.io/docs/duo/overview`.

[3] Milk-V. Milk-v duo s (connection and setup). `https://milkv.io/docs/duo/getting-s/setup/setup`.

[4] Milk-V. Milk-v duo s (connection and setup - github examples). https://github.com/milkv-duo/duo-examples/blob/main.

[5] Milk-V. Milk-v duo s (getting started). https://milkv.io/docs/duo/getting-started/duos.

[6] Milk-V. Milk-v duo s (microsd initialization). https://milkv.io/docs/duo/getting-started/boot.

[7] Milk-V. Milk-v duo s (program compilation). https://milkv.io/docs/duo/getting-started/buildroot-sdk.

[8] Milk-V. Milk-v duo s (wifi connection). https://community.milkv.io/t/connect-to-wi-fi-on-duo-s/1540.

8 Anexos

8.1 *texec.c*

```c
#include <stdio.h>
#include <unistd.h>
int main(){
    int ret;
    printf("Testing command :\nls -l\n");
    ret = execl ("/bin/ls", "ls", "-1", (char *)0);
    if(ret < 0){
        printf("Error execl.\n");
    }
    return 0;
}
```

Listing 1: Programa texec.c

8.2 *tread.c*

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
int main(int argc, char ** argv){
    if (argc != 2){
        printf("Utilizar de acordo com o exemplo:\n%s <
            ficheiro_teste>\n",argv[0]);
        return 0;
    }
    int file_id = open(argv[1],O_RDONLY);
    if (file_id == -1){
        printf("Error opening file %s\n",argv[1]);
```

```
14          printf("%s\n",strerror(errno));
15          return 0;
16      }
17      char buffer[258];
18      buffer[257] = '\0';
19      ssize_t bytes_r = read(file_id,buffer, 256);
20      if(bytes_r < 0){
21          printf("Erro reading file %s\n",argv[1]);
22          printf("%s\n",strerror(errno));
23      }
24      else {
25          printf("%zd bytes read:\n%s",bytes_r,buffer);
26      }
27      int close_s = close(file_id);
28      if (close_s != 0){
29          printf("Error closing file %s\n",argv[1]);
30          return 0;
31      }
32      return 0;
33  }
```

Listing 2: Programa tread.c

8.3 *twrite.c*

```
1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <unistd.h>
4  #include <errno.h>
5  #include <string.h>
6  int main(int argc, char ** argv){
7      if (argc != 2){
8          printf("Utilizar de acordo com o exemplo:\n%s <
               ficheiro_teste>\n",argv[0]);
9          return 0;
10     }
11     int file_id = open(argv[1],O_WRONLY,O_CREAT);
12     if (file_id == -1){
13         printf("Error opening file %s\n",argv[1]);
14         printf("%s\n",strerror(errno));
15         return 0;
16     }
17     char * buffer = "Teste 1\n  Texto \n\nMais  Texto\n";
18     int size_w = (int)strlen(buffer);
19     ssize_t bytes_w = write(file_id,buffer, size_w);
20     if(bytes_w < 0){
21         printf("Error writing to file%s\n",argv[1]);
22         printf("%s\n",strerror(errno));
23     }
```

```
24      else {
25          printf("%zd bytes written\n",bytes_w);
26      }
27      int success = close(file_id);
28      if (success != 0){
29          printf("Error closing file %s\n",argv[1]);
30          return 0;
31      }
32      return 0;
33  }
```

Listing 3: Programa twrite.c

8.4 *tgetpid.c*

```
1  #include <stdio.h>
2  #include <unistd.h>
3  int main (){
4      long id = (long) getpid();
5      printf("Process id = %ld\n",id);
6      return 0;
7  }
```

Listing 4: Programa tgetpid.c

8.5 *tfork.c*

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  int main() {
5      int id;
6      id =  getpid();
7      printf("%d\n", id);
8      fork();
9      id = getpid();
10     printf("%d\n", id);
11     fork();
12     id = getpid();
13     printf("%d\n", id);
14     return 0;
15 }
```

Listing 5: Programa tfork.c

8.6 *tpipe.c*

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main(){
    int pipefd[2];
    char buf;
    pid_t cpid;
    if (pipe(pipefd) == -1){
        printf("Error pipe creation.\n");
        return 1;
    }
    cpid = fork();
    if (cpid == -1){
        printf("Error forking.\n");
        return 1;
    }
    if (cpid == 0) {    //child process
        if (close(pipefd[1]) == -1){
            printf("Error closing pipe.\n");
            return 1;
        }
        while (read(pipefd[0], &buf, 1) > 0) {
            if (write(STDOUT_FILENO, &buf, 1) != 1){
                printf("Error writing to stdout.\n");
                return 1;
            }
        }
        if (write(STDOUT_FILENO, "\n", 1) != 1){
            printf("Error writing to stdout.\n");
            return 1;
        }
        if (close(pipefd[0]) == -1){
            printf("Error closing pipe.\n");
            return 1;
        }
        return 0;
    }
    else {    //parent process
        if (close(pipefd[0]) == -1) {
            printf("Error closing pipe.\n");
            return 1;
        }
        char * msg = "Testing..";
        if (write(pipefd[1], msg, strlen(msg)) !=
            strlen(msg)){
            printf("Error writing to pipe.\n");
```

```
49              return 1;
50          }
51          if (close(pipefd[1]) == -1){
52              printf("Error closing pipe.\n");
53              return 1;
54          }
55          if (wait(NULL) == -1){
56              printf("Error waiting for child process.\n");
57              return 1;
58          }
59          return 0;
60      }
61  }
```

Listing 6: Programa tpipe.c

8.7 *tthread.c*

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  int x = 0;
5  void * sum2(){
6      sleep(1);
7      x += 2;
8  }
9  void * printx(){
10     printf("x = %d\n",x);
11     sleep(2);
12     printf("x = %d\n",x);
13 }
14 int main(){
15     pthread_t t1, t2;
16     if (pthread_create(&t1,NULL,&printx,NULL)){
17         printf("Error starting thread 1\n");
18     }
19     if (pthread_create(&t2,NULL,&sum2,NULL)){
20         printf("Error starting thread 2\n");
21     }
22     if (pthread_join(t1, NULL)){
23         printf("Error waiting for thread 1\n");
24     }
25     if (pthread_join(t2, NULL)){
26         printf("Error waiting for thread 2\n");
27     }
28     return 0;
29 }
```

Listing 7: Programa tthread.c

8.8 *tmutex.c*

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
pthread_mutex_t mutex;
pthread_t thread[2];
int thread_i = 1;
void * tstatus(){
    pthread_mutex_lock(&mutex);
    printf("Thread %d: Start\n",thread_i);
    sleep(2);
    printf("Thread %d: End\n",thread_i);
    thread_i++;
    pthread_mutex_unlock(&mutex);
}
int main(){
    if (pthread_mutex_init(&mutex,NULL)){
        printf("Error init mutex\n");
        return 1;
    }
    for (int i = 0; i < 2; i++){
        if (pthread_create(&(thread[i]),NULL,&tstatus,NULL))
            {
            printf("Error create thread %d\n", i);
            return 1;
        }
    }
    if (pthread_join(thread[0], NULL)){
        printf("Error thread 1\n");
    }
    if (pthread_join(thread[1], NULL)){
        printf("Error thread 2\n");
    }
    if (pthread_mutex_destroy(&mutex)){
        printf("Error destroy mutex\n");
        return 1;
    }
    return 0;
}
```

Listing 8: Programa tmutex.c

8.9 Protocolo FTP

8.9.1 *commands.c* e *commands.h*

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

```c
#include "comands.h"
#include "parser.h"
#include <ctype.h>
#include <stdlib.h>
int sendCommand(int sockfd, const char *command) {
    int bytes = write(sockfd, command, strlen(command));
    if (bytes < 0) {
        perror("write()");
        close(sockfd);
        return 1;
    }
    return 0;
}
int sendUser(int sockfd, const char *username) {
    char buf[64];
    snprintf(buf, sizeof(buf), "USER %s\r\n", username);
    return sendCommand(sockfd, buf);
}
int sendPass(int sockfd, const char *password) {
    char buf[64];
    snprintf(buf, sizeof(buf), "PASS %s\r\n", password);
    return sendCommand(sockfd, buf);
}
int sendPasv(int sockfd) {
    return sendCommand(sockfd, "PASV\r\n");
}
int sendRetr(int sockfd, const char *urlPath) {
    char buf[64];
    snprintf(buf, sizeof(buf), "RETR %s\r\n", urlPath);
    return sendCommand(sockfd, buf);
}
int read_response(FILE *fl) {
    char *buf = NULL;
    size_t len = 0;
    ssize_t nread;
    int code = 0;
    int multiline = 0;
    while ((nread = getline(&buf, &len, fl)) != -1) {
        printf("%s", buf);
        if (isdigit(buf[0]) && isdigit(buf[1]) &&
            isdigit(buf[2])) {
            if (buf[3] == '-') {
                multiline = 1;
                code = atoi(buf);
            }
            else if (buf[3] == ' ') {
                break;
            }
        }
        else if (multiline) {
```

```
            if (atoi(buf) == code && buf[3] == ' ') {
                break;
            }
        }
    }
    free(buf);
    return 0;
}
int read_response_pasv(FILE * fl, char ip[32], int * port){
    char * buf = NULL;
    size_t bytes = 0;
    bytes = getline(&buf, &bytes, fl);
    buf[bytes] = '\0';
    printf(">%s",buf);
    if (bytes < 0){
        return 1;
    }
    return parse_pasv_response(buf, ip, port);
}
int read_response_retr(FILE * fl, const char * filename){
    FILE* download_file = fopen(filename, "w");
    if (download_file == NULL) {
        perror("Error opening file for writing");
        return 1;
    }
    char buf[1028];
    size_t bytes;
    while (!feof(fl)) {
        bytes = fread(buf, 1, sizeof(buf), fl);
        if (bytes < 0) {
            perror("Error reading from file");
            fclose(download_file);
            return 1;
        }
        size_t written = fwrite(buf, 1, bytes,
            download_file);
        if (written < 0) {
            perror("Error writing to file");
            fclose(download_file);
            return 1;
        }
    }
    fclose(download_file);
    return 0;
}
```

Listing 9: comands.c

```
#ifndef __COMANDS_H__
#define __COMANDS_H__
```

```c
#include <stdio.h>
int sendCommand(int sockfd, const char *command);
int sendUser(int sockfd, const char *username);
int sendPass(int sockfd, const char *password);
int sendPasv(int sockfd);
int sendRetr(int sockfd, const char *url_path);
int read_response(FILE * fl);
int read_response_pasv(FILE * fl, char ip[32], int * port);
int read_response_retr(FILE * fl, const char * filename);
#endif
```

Listing 10: comands.h

8.9.2 *main.c*

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <string.h>
#include "objects.h"
#include "parser.h"
#include "comands.h"
#define SERVER_PORT 21
int main(int argc, char **argv){
    if (argc != 2){
        fprintf(stderr,"Correct usage : ./projeto
            ftp://[<user>:<password>@]<host>/<url-path>\n");
        return EXIT_FAILURE;
    }
    Data_ info;
    init_data(&info);
    if ((parse_data(&info, argv[1])) != 0){
        fprintf(stderr,"The given url syntax is
            incorrect\n");
        return EXIT_FAILURE;
    }
    // PARSING DATA DONE
    if ((get_host_ip(&info)) != 0){
        return EXIT_FAILURE;
    }
    print_data(info);
    // GET HOST IP DONE
    int sockfd;
    struct sockaddr_in server_addr;
    //server address handling
    bzero((char *) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(info.host_ip);
```

```
33      server_addr.sin_port = htons(SERVER_PORT);
34      //open a TCP socket
35      if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
36          perror("socket()");
37          return EXIT_FAILURE;
38      }
39      //connect to the server
40      if (connect(sockfd,
41                  (struct sockaddr *) &server_addr,
42                  sizeof(server_addr)) < 0) {
43          perror("connect()");
44          return EXIT_FAILURE;
45      }
46      printf("\nConnection open on hostname = '%s' and ip =
            '%s'\n",info.host,info.host_ip);
47      FILE * fl = fdopen(sockfd,"r");
48      if ((read_response(fl)) != 0){
49          fprintf(stderr,"Error reading answer from
                server\n");
50          return EXIT_FAILURE;
51      }
52      if ((sendUser(sockfd, info.user)) != 0){
53          fprintf(stderr,"Error sending user\n");
54          return EXIT_FAILURE;
55      }
56      printf("\nSent command : 'USER %s'\n",info.user);
57      if ((read_response(fl)) != 0){
58          fprintf(stderr,"Error reading answer from
                server\n");
59          return EXIT_FAILURE;
60      }
61      if ((sendPass(sockfd, info.pw)) != 0){
62          fprintf(stderr,"Error sending password\n");
63          return EXIT_FAILURE;
64      }
65      printf("\nSent command : 'PASS %s'\n",info.pw);
66      if ((read_response(fl)) != 0){
67          fprintf(stderr,"Error reading answer from
                server\n");
68          return EXIT_FAILURE;
69      }
70      if ((sendPasv(sockfd)) != 0){
71          fprintf(stderr,"Error sending passive mode
                command\n");
72          return EXIT_FAILURE;
73      }
74      printf("\nSent command : 'pasv'\n");
75      char ip[32] = {0};
76      int port = 21;
77      if ((read_response_pasv(fl, ip, &port)) != 0){
```

```
78          fprintf(stderr,"Error read passive mode server
                answer\n");
79          return EXIT_FAILURE;
80      }
81      int sockfd2;
82      struct sockaddr_in data_server_addr;
83      //server address handling
84      bzero((char *) &data_server_addr,
            sizeof(data_server_addr));
85      data_server_addr.sin_family = AF_INET;
86      data_server_addr.sin_addr.s_addr = inet_addr(ip);
87      data_server_addr.sin_port = htons(port);
88      //open a TCP socket
89      if ((sockfd2 = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
90          perror("socket()");
91          return EXIT_FAILURE;
92      }
93      //connect to the server
94      if (connect(sockfd2, (struct sockaddr *)
            &data_server_addr, sizeof(data_server_addr)) < 0) {
95          perror("connect()");
96          return EXIT_FAILURE;
97      }
98      printf("\nConnection open on hostname = '%s' and ip =
            '%s'\n",info.host,info.host_ip);
99      FILE *fl2 = fdopen(sockfd2, "r");
100     if ((sendRetr(sockfd, info.url_path)) != 0) {
101         fprintf(stderr, "Error sending RETR command\n");
102         return EXIT_FAILURE;
103     }
104     printf("\nSent command : 'RETR %s'\n", info.url_path);
105     if ((read_response_retr(fl2, info.filename)) != 0) {
106         fprintf(stderr, "Error reading RETR command
                response\n");
107         return EXIT_FAILURE;
108     }
109     printf("\nDownloaded '%s' on host = '%s'\n",
            info.url_path, info.host);
110     if ((fclose(fl2)) < 0) {
111         perror("close()");
112         return EXIT_FAILURE;
113     }
114     if ((fclose(fl)) < 0) {
115         perror("close()");
116         return EXIT_FAILURE;
117     }
118     printf("\nClosed connection\n");
119     return EXIT_SUCCESS;
120 }
```

Listing 11: main.c

8.9.3 *objects.c* e *objects.h*

```c
#include <stdio.h>
#include "objects.h"
void init_data(Data_ * info){
    info->user[0] = '\0';
    info->pw[0] = '\0';
    info->host[0] = '\0';
    info->host_ip[0] = '\0';
    info->url_path[0] = '\0';
    info->filename[0] = '\0';
}
void print_data(const Data_ info){
    printf("\nuser = '%s'\npw = '%s'\nhost = '%s'\nhost_ip =
         '%s'\nurl_path = '%s'\nfilename = '%s'\n",info.user,
        info.pw,info.host,info.host_ip,info.url_path,info.
        filename);
}
```

Listing 12: objects.c

```c
#ifndef __OBJECTS_H__
#define __OBJECTS_H__

typedef struct Data_ {
    char user[64];
    char pw[64];
    char host[64];
    char url_path[64];
    char host_ip[64];
    char filename[64];
} Data_;
void init_data(Data_ * info);
void print_data(const Data_ info);
#endif
```

Listing 13: objects.h

```c
#ifndef __OBJECTS_H__
#define __OBJECTS_H__
typedef struct Data_ {
    char user[64];
    char pw[64];
    char host[64];
    char url_path[64];
    char host_ip[64];
```

```
    char filename[64];
} Data_;
void init_data(Data_ * info);
void print_data(const Data_ info);
#endif
```

Listing 14: objects.h

8.9.4 *parser.c* e *parser.h*

```
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include "parser.h"
int parse_data(Data_ * info, const char * arg){
    int control = 1;
    size_t len = strlen(arg);
    if (len > 6) {
        char tmp[64] = {0};
        strncpy(tmp, arg, 6);
        if ((strcmp(tmp, "ftp://")) != 0) {
            fprintf(stderr, "Received prefix %s", tmp);
            return 1;
        }
        strcpy(tmp, "\0");
        int j = 0;
        for (int i = 6; i < len; i++) {
            if (arg[i] == '/') {
                if (control != 4) {
                    tmp[j] = '\0';
                    strcpy(info->host, tmp);
                    j = 0;
                    strcpy(tmp, "\0");
                    control++;
                }
                else {
                    tmp[j] = arg[i];
                    j++;
                }
            }
            else if (arg[i] == ':') {
                tmp[j] = '\0';
                strcpy(info->user, tmp);
                j = 0;
                strcpy(tmp, "\0");
                control++;
```

```
            }
            else if (arg[i] == '@') {
                tmp[j] = '\0';
                strcpy(info->pw, tmp);
                j = 0;
                strcpy(tmp, "\0");
                control++;
            }
            else {
                tmp[j] = arg[i];
                j++;
            }
        }
        tmp[j] = '\0';
        strcpy(info->url_path, tmp);
        j = 0;
        strcpy(tmp, "\0");
        char *lastSlash = strrchr(info->url_path, '/');
        if (lastSlash != NULL) {
            strcpy(info->filename, lastSlash + 1);
        }
        else {
            strcpy(info->filename, info->url_path);
        }
        return 0;
    }
    return 1;
}
int get_host_ip(Data_ * info){
    struct hostent *h;
    if ((h = gethostbyname(info->host)) == NULL) {
        herror("gethostbyname()");
        return 1;
    }
    strcpy(info->host_ip ,inet_ntoa(*((struct in_addr *)
        h->h_addr)));
    return 0;
}
int parse_pasv_response(const char buf[64], char ip[32],
    int * port){
    int a, b, c, d, e, f;
    if (sscanf(buf, "227 Entering Passive Mode
        (%d,%d,%d,%d,%d,%d)",
                &a, &b, &c, &d, &e, &f) == 6) {
        snprintf(ip, 32, "%d.%d.%d.%d", a, b, c, d);
        *port = (e * 256) + f;
        return 0;
    }
    return 1;
}
```

Listing 15: parser.c

```
#ifndef __PARSER_H__
#define __PARSER_H__
#include "objects.h"
int parse_data(Data_ * info, const char * arg);
int get_host_ip(Data_ * info);
int parse_pasv_response(const char buf[64], char ip[32], int
    * port);
#endif
```

Listing 16: parser.h