Novos exemplos de Práticas Pedagógicas e Estratégias de Inovação Pedagógica no Iscte

CONSELHO
PEDAGÓGICO

iscte
INSTITUTO
UNIVERSITÁRIO
DE LISBOA

FICHA TÉCNICA

Título
Novos Exemplos de práticas pedagógicas
e estratégias de inovação pedagógica no Iscte

Suporte
Eletrónico

Formato
PDF

Organizadores
Sónia Pintassilgo, Alexandre Almeida,
Ana Catarina Nunes, Helena Soares, Isabel
Correia, Patrícia Dinis Costa, Vania Baldi,
David Isaac, Henrique Lage, Beatriz Saavedra,
Gonçalo Tomé Ribeiro, Helena Alvito

Autores
António Luís Lopes & ChatGPT; Joana Martinho
Costa; Sara Soares & Rita Jerónimo; Marília
Prada & Margarida Vaz Garrido; Sibila Marques,
Mariana Montalvão e Silva & André
Samora-Arvela; Rodrigo Vieira de Assis & Filipa
Pinho; Elsa Justino & Inês Casquilho-Martins;
Malwina Wojciechowska & Sofia Gomes;
Dulce Morgado Neves & Adriana Albuquerque;
Mara Clemente; Carlos Rocha, Inês Gama
& Sara Mourato; Mel Campos, Leo Oliveira &
Paulo Raposo; Cristiane Souza, Sofia Frade
& Marília Prada; Conceição Pereira; Ricardo
Mendes Correia; André Almeida Pinho;
Ana Simaens; Ana Lúcia Martins; Adriana Rosa
& Arlindo Ribeiro; Carlos Coutinho

Edição
Iscte – Instituto Universitário de Lisboa

Projeto gráfico e paginação
Gabinete de Comunicação Iscte

Local e data
Lisboa, junho, 2025

Responsabilidade
Cada capítulo é da exclusiva
responsabilidade dos seus autores

ISBN
978-989-584-117-2

Princípios pedagógicos sobre a Unidade Curricular de Sistemas Operativos no Iscte

Carlos Coutinho
carlos.eduardo.coutinho@iscte-iul.pt

Iscte – Instituto Universitário de Lisboa

SUMÁRIO EXECUTIVO

Os sistemas operativos são uma componente fundamental da computação moderna, desempenhando um papel crucial na gestão dos recursos informáticos, facilitando a comunicação entre *hardware* e *software* e proporcionando uma interface de fácil utilização para os utilizadores interagirem com o computador.

A história dos sistemas operativos remonta aos primórdios da computação, na década de 1950, quando os primeiros computadores, usados principalmente para fins científicos e militares, foram desenvolvidos. Nessa altura, o propósito primordial dos sistemas operativos era controlar as entradas e saídas do computador. Com o passar do tempo, os sistemas operativos tornaram-se mais sofisticados, passando a proporcionar multiprocessamento, permitindo o acesso a múltiplos utilizadores segundo o modelo *time-sharing*. Para facilitar a interação entre os utilizadores e a máquina, desenvolveram-se interfaces de linha de comandos, e, mais tarde, interfaces gráficas (GUI) que se destinaram a simplificar e apresentar um ambiente *user-friendly* para utilizadores comuns, democratizando o acesso e utilização dos computadores, e possibilitando-lhes o acesso rápido e eficaz às funcionalidades mais populares e comuns dos sistemas operativos.

Na atualidade, como resultado da heterogeneidade de acessos a recursos computacionais em todas as áreas da sociedade, que vão desde a computação móvel, aos sistemas de computador tradicionais, a dispositivos *wearables*, ou sensores interligados em rede para a *Internet* das Coisas, modelos de *digital twin* de elementos físicos reais, ou de realidade aumentada, até à computação em nuvem, estes recursos apresentam propósitos muito diferentes. Naturalmente, também as suas formas de gestão, armazenamento, interação e outros são eles próprios também heterogéneos. Nesse sentido, os sistemas operativos tornaram-se um importante aliado para consolidar e homogeneizar conceitos, soluções e interfaces, simplificando e democratizando uma vez mais o acesso a todas essas funcionalidades, que agora incluem como a gestão de recursos, a interface com o *hardware* e dispositivos associados, o multiprocessamento de tarefas, e a segurança no acesso, entre outras.

Os conhecimentos e ideias aqui abrangidos não só têm utilidade em si mesmos, como também contribuem para o desenvolvimento de metodologias e

de abstração de conceitos que serão fundamentais para obterem uma visão mais consolidada do mundo na área informática dos alunos.

Neste capítulo é descrita a Unidade Curricular (UC) de Sistemas Operativos (SO), transversal às licenciaturas em Informática e Gestão de Empresas (LIGE e LIGE-PL), Engenharia Informática (LEI e LEI-PL), e Engenharia de Telecomunicações e Informática (LETI) do Iscte – Instituto Universitário de Lisboa (Iscte), aproveitando a experiência adquirida no seu leccionamento nos anos letivos de 2016/17 até 2024/25. Esta UC tem sido lecionada no primeiro semestre do segundo ano e no segundo semestre do primeiro ano destas licenciaturas. A UC não tem pré-requisitos formais, embora se assuma que os alunos tenham bons conhecimentos de algoritmia e estrutura de dados e conhecimentos em pelo menos uma linguagem de programação de alto nível (e.g., Java, C++, JavaScript).

Esta UC enquadra-se na área científica de Arquitetura de Computadores e Sistemas Operativos do Departamento de Ciências e Tecnologias da Informação (DCTI) do Iscte, do qual também fazem parte a UC de Fundamentos de Arquitetura de Computadores e a UC de Microprocessadores, todas do primeiro ano, embora a UC de Microprocessadores apenas seja dada na Licenciatura em Engenharia Informática.

1. OBJETIVOS GERAIS

A UC de Sistemas Operativos tem o objetivo principal de apresentar os fundamentos dos Sistemas Operativos e relacioná-los, por um lado com a experiência do aluno, e por outro, com as matérias que são dadas noutras UCs, nomeadamente nas Arquiteturas de Computadores, da qual esta UC é uma progressão lógica, as Redes de Computadores, ou Segurança. O propósito principal desta UC é enumerar as (principais) múltiplas diferentes funcionalidades dos sistemas operativos, avaliando os algoritmos e soluções alternativas, e comparando as abordagens tomadas pelos principais fornecedores em cada uma dessas funcionalidades. De seguida, ainda no âmbito de cada funcionalidade, descrevemos em maior pormenor a abordagem tomada pelo sistema operativo Linux.

Como em qualquer UC, uma preocupação central no planeamento desta foi a motivação dos alunos. Nesse sentido, a UC tem por um lado uma componente teórico-prática, onde se exploram os conceitos e filosofias que levaram às soluções técnicas usadas nos SO mais populares, realçando as soluções existentes nos sistemas Linux e Windows. É nesta componente também que se explicam as diferenças de utilização entre a linha de comandos e os ambientes gráficos, e se ensinam as bases de interação com o sistema operativo tanto usando

comandos Linux, Shell *scripting* ou programação usando a linguagem C. Por outro lado, tem uma componente prática laboratorial de importância premente, onde os alunos podem consolidar os conhecimentos adquiridos na parte teórico-prática (e.g., funcionalidades do SO, gestão de recursos, multiprocessamento, segurança, persistência) em sessões *hands-on* usando uma interface linha de comandos para um sistema Linux.

Esta componente prática laboratorial habilitará o aluno a utilizar a linha de comandos do sistema operativo Linux e a desenvolver programas, ao nível do sistema, usando os mecanismos do sistema operativo, tendo em conta os modelos de programação sequencial e concorrente. A escolha do Linux como sistema operativo teve em conta a tendência atual para a utilização massiva deste SO ou semelhantes na esmagadora maioria de equipamentos do mercado, tanto em dispositivos móveis (na sua vertente Android ou iOS), como nos sistemas de computação tradicionais (correndo Unix, Linux, OSX e até Windows), aos paradigmas mais disruptivos como sejam a configuração de dispositivos *things* usando Arduíno (Xinu) ou Raspberry Pi OS, até à configuração e gestão de *containers* e sistemas de computação na nuvem.

Em termos de planeamento, a componente laboratorial ocorre em todas as semanas de aulas, tendo o dobro da duração das aulas teórico-práticas. Cada aula teórico-prática tem um documento (conjunto de slides) disponibilizado no início do semestre letivo, com a matéria específica dada nessa aula. Cada aula de prática laboratorial tem um guião, que permite aos alunos terem linhas de orientação para as atividades a serem realizadas nas aulas de laboratório, o que promove terem um acompanhamento que cobre praticamente todos os conceitos do programa da UC.

Os alunos, para além das horas semanais teórico-práticas e de prática laboratorial, têm também de preencher um mini-teste semanal no Moodle que se destina a fazer com que consolidem os conhecimentos transmitidos nessa semana, num total de nove mini-testes. Para além disso, têm de espelhar os seus conhecimentos num projeto de desenvolvimento em Linux, que já foi realizado em grupo e é agora individual, dividido em três partes, com entregas a cada quatro semanas de aulas.

Todos os programas, conteúdos, exercícios, um conjunto extenso de testes anteriores, e material auxiliar da cadeira estão disponíveis na página da UC no Moodle Iscte, página essa que serve também como principal ponto de contacto entre os alunos e a cadeira fora da sala de aula.

2. PROGRAMA

O programa da disciplina está previsto para 12 semanas de aulas, que incluem, em cada semana, 1h30 de aula teórico-prática e 3h de aula prático-laboratorial. O programa seguido introduz os conceitos referidos nos objetivos da disciplina mediante uma sequência lógica de apresentação de cada funcionalidade dos sistemas operativos alinhada sempre que possível com a sua experimentação em aulas práticas laboratoriais, e enquadrando as mesmas com desafios reais do mundo à nossa volta, por forma a alinhar os conceitos e trabalhos com potenciais soluções que possam ser reutilizáveis pelos alunos no futuro. O programa espelha os seguintes conteúdos programáticos (CP) enumerados:

CP1: Introdução aos Sistemas Operativos: Evolução histórica, tipos, funções e características dos vários sistemas operativos.

Processos:

CP2: Concorrência e gestão de processos: Concorrência de processos, pseudo-paralelismo e multiprogramação, criação e execução de processos.

CP3: Escalonamento de processos: Estados de um processo, gestão de processos e *context-switching,* algoritmos de escalonamento mais utilizados.

CP4: Sincronização entre processos: Processos e *threads*, zonas críticas e exclusão mútua, mecanismos de sincronização entre processos
IPC: semáforos.

CP5: Comunicação entre processos: Pipes, FIFOs, mecanismos de comunicação IPC: memórias partilhadas, filas de mensagem.

Gestão de Memória:

CP6: Modelos e algoritmos de gestão de memória: Organização hierárquica da memória, segmentação e paginação, endereçamento real e virtual.

CP7: Memória Virtual: Algoritmos de substituição de páginas, conceito de *Thrashing* e de *Working Set*.

CP8: Entradas e Saídas: Periféricos, interrupções, *device drivers*, chamadas ao sistema e *spooling*.

CP9: Sistema de Ficheiros: ficheiros, diretórios, redireccionamento, expansão, *standard streams*, *hard links*, *soft links*.

CP10: Administração e Segurança: Conhecer os diversos tipos de autenticação de utilizadores, conhecer vários tipos de ataques, de dentro e de fora do sistema, acesso SSH, SFTP.

Utilização do Linux (aulas práticas laboratoriais)

CP11: Comandos Shell e programação em Shell: Comandos comuns de manipulação do sistema de ficheiros, manipulação de texto, estruturas de controlo, criação de *scripts*.

CP12: Mecanismos de comunicação e sincronização: Sinais, PIPEs, IPC.

3. PROCESSO DE ENSINO – APRENDIZAGEM

O processo de ensino/aprendizagem nesta UC é orientado pelos seguintes princípios:

- Enfase na compreensão dos conceitos base sobre os sistemas operativos e no relacionamento destes conceitos com a experiência e matérias de outras UC.
- Forte relacionamento entre a componente teórica e a expressão prática dos mesmos conceitos.
- Trabalho prático de desenvolvimento como meio de consolidação dos conhecimentos e competências.

A metodologia adotada não só dá um conhecimento amplo sobre os diversos aspetos de funcionamento dos diversos sistemas operativos, como permite fortalecer o conhecimento do aluno em aspetos mais relevantes, devido à forte componente laboratorial.

A forte componente laboratorial baseia-se na utilização do sistema operativo Linux, um sistema operativo *unix-like*, de código aberto e amplamente divulgado, o que permite dar uma ênfase ao trabalho remoto com base na linha de comandos e na utilização direta das chamadas do sistema usando a linguagem C.

Os conteúdos necessários para a compreensão da matéria da UC encontram-se centralizados, em primeiro lugar, na sebenta da UC, que acompanha de forma extensa a matéria que é dada nas aulas. Esta sebenta está organizada em capítulos, sendo que vários destes capítulos são diretamente mapeados em correspondentes semanas de aulas, contemplando tanto os conceitos teóricos associados como contém dicas práticas e sugestões de exercícios práticos que são próximos dos dados nas aulas práticas de laboratório. Para além da sebenta, os conteúdos tanto das aulas teórico-práticas como as das aulas de prática laboratorial estão divididos em apresentações separadas para cada aula, todas disponíveis de forma clara no Moodle Iscte.

O planeamento do semestre é apresentado claramente na primeira aula teórico-prática, mostrando um diagrama com o calendário de aulas e associação

entre as aulas teórico-práticas e as aulas de prática laboratorial, como mostrado na Figura 1.

Week	Monday	Sunday	Week	Theoretical	Practical	Output
SO-1	6-fev	12-fev	1	Concepts, OS structures	Shell (Command Line)	HW1
SO-2	13-fev	19-fev	2	Concepts, OS structures	Text Manipulation Commands	HW2
	20-fev	26-fev				
SO-3	27-fev	5-mar	3	Inputs/Outputs (I/O)	Shell Script Programming	HW3
SO-4	6-mar	12-mar	4	File Systems	**Apoio ao Trabalho Prático**	**PA1**
SO-5	13-mar	19-mar	5	C Language (revision)	Files and I/O	HW4
SO-6	20-mar	26-mar	6	Processes: creation, management	C Language (exercises)	HW5
SO-7	27-mar	2-abr	7	Processes: threads and synchronisation	Processes and Signals	HW6
	3-abr	9-abr				
	10-abr	16-abr				
SO-8	17-abr	23-abr	8	Processes: Synchronisation	**Apoio ao Trabalho Prático**	**PA2**
SO-9	24-abr	30-abr	9	Processes: Communication	Inter Process Communication (IPC)	HW7
SO-10	1-mai	7-mai	10	Memory Management,	Inter Process Communication (IPC)	HW8
SO-11	8-mai	14-mai	11	Virtual Memory	Inter Process Communication (IPC)	HW9
SO-12	15-mai	21-mai	12	Management / Security	**Apoio ao Trabalho Prático**	**PA3**
	22-mai	28-mai		Oral evaluations		

Figura 1 – Planeamento da UC de SO apresentado aos alunos na primeira aula

Como se pode ver, o planeamento está normalmente dividido em três blocos principais de quatro semanas cada. No primeiro bloco, são dados nas aulas teórico-práticas os conceitos básicos dos sistemas operativos, introduzindo os alunos à necessidade de gestão de periféricos e de sistemas de gestão de ficheiros, complementadas por demonstrações em sala de aula, por acesso remoto via SSH ao servidor Linux "Tigre" (alojado no *Data Centre* do Iscte no endereço tigre.iul.lab), para apresentar de forma direta as diferenças e semelhanças entre, por exemplo, o sistema de ficheiros Linux e o Windows. Este bloco de quatro semanas de aulas é complementado por correspondentes aulas semanais de prática laboratorial, realizadas nos laboratórios de informática do Iscte (Figura 2), em que são ensinados aos alunos os fundamentos de ligação via SSH ao servidor "Tigre", e onde são dados os primeiros passos na desenvoltura de utilização de um terminal que permite o acesso via linha de comandos ao Linux, explicando o que é um terminal, o que é uma Shell e assim por diante.

No primeiro bloco de quatro semanas de aulas são também ensinados os fundamentos de utilização de um editor de texto remoto (vi), assim como apresentados os fundamentos ao sistema de ficheiros do Linux, incluindo a descrição de atributos dos ficheiros, definição de i-nodes, definição de *standard-streams* e a sua utilidade (STDIN, STDOUT, STDERR), redireccionamento dessas *streams* para ficheiros, noção de conjunção de comandos em Linux e interligação do resultado de vários comandos usando *pipes*. Noção de *soft-links* e *hard-links*, semelhanças e diferenças, e são apresentados vários pequenos programas utilitários disponíveis em Linux, assim como várias opções que podem ser utilizadas nesses programas. É também explicado de que forma se podem conjugar esses programas para obter vários resultados de valor acrescentado, e muitas outras dicas na utilização do sistema operativo Linux.

Figura 2 – Um dos laboratórios de Informática no Iscte

No segundo bloco de quatro semanas de aulas, os alunos de SO aprendem de forma breve a especificação da linguagem C. Diz-se de forma breve, já que se assume que já frequentaram a UC de Introdução à Programação no semestre anterior, onde se dá a linguagem Java, derivada do C, e em que grande parte das estruturas de controlo e a própria sintaxe das duas linguagens tem vários pontos em comum. São também ensinados os fundamentos da criação de novos processos, em Windows e Linux, a hierarquia entre processos, e os princípios do multiprocessamento e multiprogramação. São também ensinadas técnicas de interação entre processos, nomeadamente o estabelecimento de canais de comunicação entre processos inter-relacionados via programação em C, e interrupções usando sinais, quer usando a linguagem C ou a própria linha de comandos.

Finalmente, no terceiro bloco de quatro semanas de aulas, os alunos aprendem como utilizar os mecanismos de comunicação e sincronização de processos IPC do Linux, enquanto compreendem a necessidade da utilização dos mesmos, assim como os problemas associados ao multiprocessamento em cenários reais. Aprendem também as técnicas associadas à gestão de memória, as noções de memória virtual e mapeamento de memória virtual em memória real. São apresentados os conceitos de partições *swap*, de fragmentação interna e externa, da necessidade de *overlays* e mapeamento de endereços de memória em sistemas com multiprocessamento e concorrência.

Como já indicado anteriormente, as aulas teórico-práticas têm cada semana uma apresentação com os conteúdos a dar nessa semana, que é explicada em período de aula, em conjunção com demonstrações e exemplos de como os conceitos teóricos se transportam para problemas reais dos sistemas operativos, e como lidar com eles no SO Linux. Da mesma forma, a aula de prática laboratorial tem também um guião para direcionar mais facilmente os alunos na conclusão dessa aula prática *hands-on*.

Da experiência de vários anos a lecionar esta UC, foram recolhidos e analisados com toda a atenção os comentários, críticas e sugestões de melhoria apresentados pelos alunos, e uma delas que era recorrente era o fato de que, apesar de reconhecerem o esforço de ligação entre a componente teórico-prática e a prática laboratorial, por vezes sentiam a falta de algum suporte teórico para apoiar as aulas práticas, cobrindo conceitos associados com estas aulas mas que, por pura falta de tempo, não teria sido possível apresentar, como por exemplo as alternativas na escolha da melhor aplicação para terminal, ou o melhor editor de texto, ou as melhores técnicas *best-practices* para trabalhar com os mecanismos de comunicação em Linux. Ou, também, técnicas inovadoras como a utilização de Git e os problemas de segurança associados com a utilização de determinadas funcionalidades do C. Por forma a mitigar este problema, têm progressivamente sido construídos novos conteúdos, denominados "suporte teórico das aulas práticas" sob a forma de apresentações semanais sobre tópicos relacionados com a utilização dos mecanismos que estão propostos para a aula de prática laboratorial em questão.

Ou seja, nas aulas de prática laboratorial, para além do guião, são disponibilizados vários conteúdos adicionais, sob a forma de apresentações e vídeos, tutoriais, *cheat-sheets* e outros, todos disponíveis na plataforma Moodle, de forma tabular com separação semana a semana, para que os alunos percebam claramente quais são os conteúdos que deverão usar para se preparar para as respetivas aulas de SO, como se pode ver na Figura 3, um exemplo da tabela que mostra os conteúdos das primeiras 4 semanas. A tabela disponível no Moodle tem as 12 semanas completas.

Todos estes conteúdos e metodologias destinam-se a dotar os alunos de uma maior capacidade de análise e resiliência de conhecimentos, estruturando a matéria de uma forma sequencial e lógica, e promovendo os conceitos de comunidade e partilha de conhecimentos.

Como se pode ver na Figura 3, os conteúdos disponibilizados no Moodle quer para as aulas teórico-práticas (à esquerda) como para as aulas de prática laboratorial são apresentados de forma clara e separados por semanas, de acordo com o que é esperado dos alunos nessa semana.

Para além da frequência das aulas teórico-práticas, e da frequência das aulas de prática laboratorial, os alunos também realizam de forma autónoma um mini-teste semanal que se destina a avaliar os conhecimentos do aluno durante essa semana, num total de 9 mini-testes (carinhosamente apelidados de TPC ou *Homework* (HW) pelos alunos) para as 12 semanas de aulas, sendo os mesmos distribuídos, como mostrado na Figura 1, da seguinte forma: a cada bloco de 4 semanas de aulas, as três primeiras semanas culminarão cada uma com a entrega do tal mini-teste. Logo no início da segunda semana de aulas, é apresentado aos alunos o enunciado da primeira parte (de três) do trabalho prático de SO, trabalho esse que tem de ser submetido no Moodle até ao final

Theoretical Class	Slides
01 - Concepts	Aula T01
02 - Concepts	Aula T02
03 - Input/Output	Aula T03
04 - File Systems	Aula T04

Aula Prática (Laboratório)	Documentação
01 - Introdução ao trabalho remoto, comandos e vi (cap. 1 e 10 da sebenta)	Aula P01 - Suporte Teórico Aula P01 - Guião Cheat Sheet: Shell Cheat Sheet: vi Text Editor Video: VPN Windows Video: Install PuTTY Video: Install MobaXterm Video: Install VSCode+SSHFS Aula P01
02 - Comandos de manipulação de texto (cap. 2 da sebenta)	Aula P02 - Suporte Teórico Aula P02 - Guião Cheat Sheet: Text Commands [PT / EN] Aula P02
03 - Shell Programming (cap. 3 da sebenta)	Aula P03 - Suporte Teórico Aula P03 - Guião Cheat Sheet: BASH Scripts Aula P03
04 - Linux Administration (cap. 4 da sebenta)	Aula P04 - Suporte Teórico Video: Criar Ficheiro ZIP para submeter trabalho no Moodle

Figura 3 – Moodle: Tabela de conteúdos semanais para as primeiras 4 semanas

da quarta semana, sendo, portanto, que nessa quarta semana, os alunos não farão mini-teste de SO, devendo nessa semana submeter a primeira parte do trabalho. De forma análoga, nas três semanas seguintes à entrega do trabalho, novamente os alunos terão de completar mini-testes (Figura 4) todas as semanas, submetendo na oitava semana a segunda parte do trabalho de SO, e acontecendo o mesmo para a terceira parte do trabalho. Apesar destes mini-testes contarem para a avaliação da UC (ver capítulo 6), o propósito principal é garantir que os alunos aprendam e retenham o conhecimento.

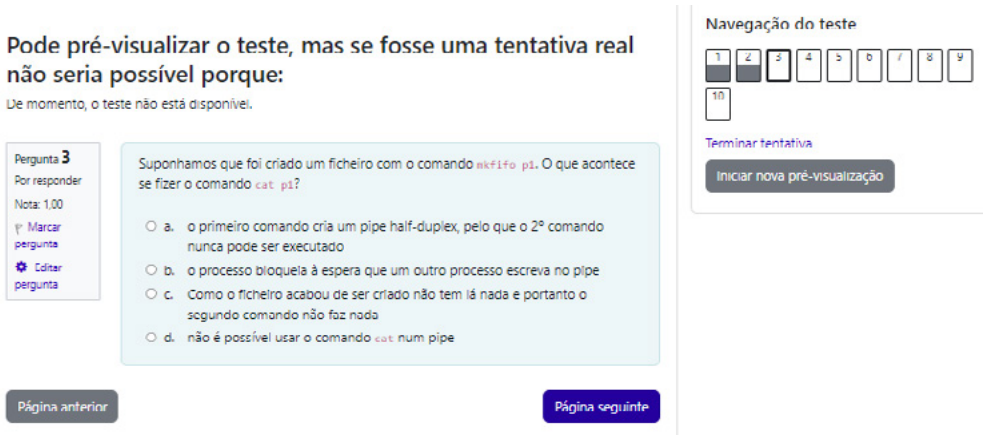


Figura 4 – Mini-teste semanal de SO

Com vista a melhorar ainda mais a experiência pedagógica dos alunos, o docente Carlos Coutinho desenvolveu um conjunto de *Knowledge Bases*

(bases de dados de conhecimento), criadas com base nas perguntas que os alunos vão fazendo ao docente, sendo depois as respostas colocadas de modo a serem mais explicativas. Foram desenvolvidas 6 *Knowledge Bases* (KB), como descrito na Figura 5:

KB Basics: Fórum dedicado a perguntas iniciais, de acesso ao servidor, perda de password, escolha de uma aplicação para terminal, acesso à VPN.

KB Shell Commands & Scripting: Fórum dedicado à matéria de Linux commands & Shell, dada nas primeiras 4 semanas de aulas.

KB C Language & Compiling: Fórum dedicado à parte básica de criação e compilação de programas em C, não diretamente relacionado com a parte de SO.

KB Linux Processes: Fórum dedicado à matéria de criação e gestão de processos, sinais e comunicação usando *pipes*, dada nas segundas 4 semanas de aulas.

KB Linux Process Communication: Fórum dedicado à matéria de comunicação usando mecanismos IPC do Linux, dada nas últimas 4 semanas de aulas.

KB Practical Assignment: Fórum dedicado especialmente às questões e dúvidas relacionadas com o projeto prático do ano em questão.

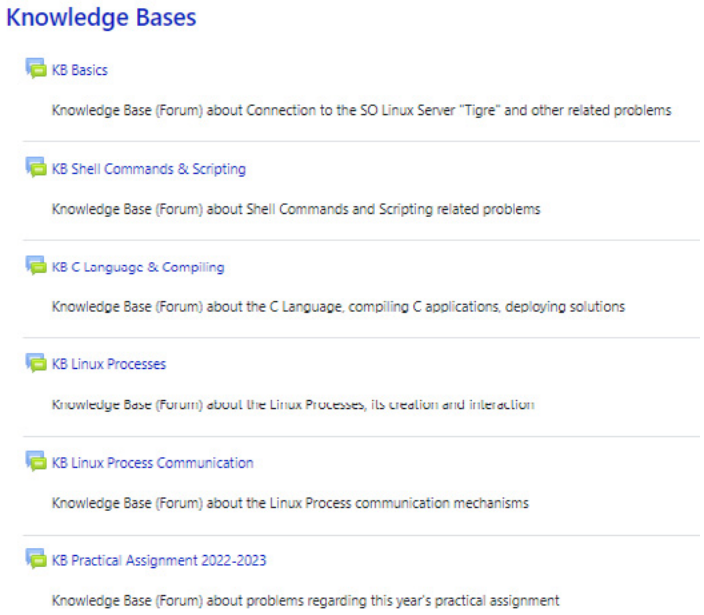


Figura 5 – Knowledge Bases criadas para os alunos de SO

Cada uma dessas KB é depois formatada por forma a apresentar uma interface muito informal para que o aluno tenha maior empatia para procurar a solução em vez de ir pesquisar na *internet* pelas soluções, como apresentado na Figura 6.

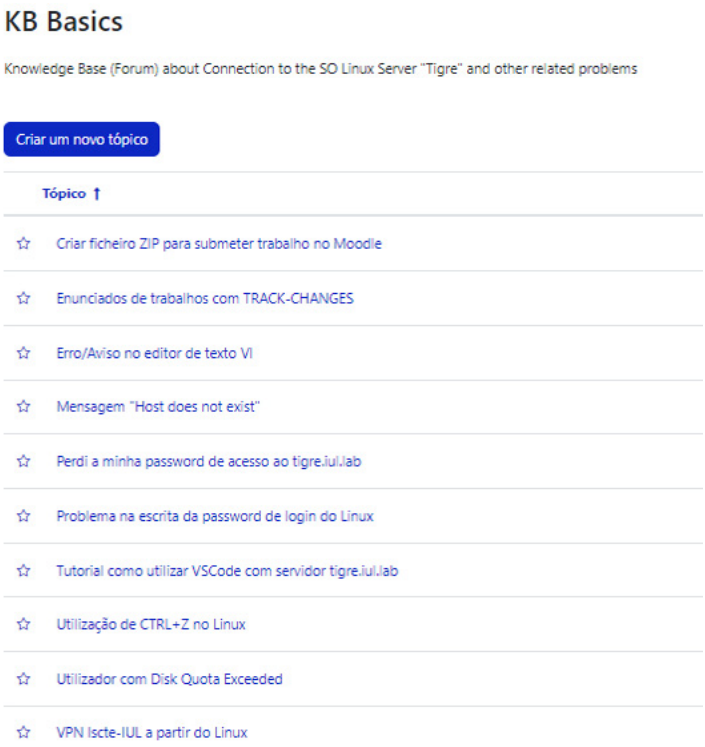


Figura 6 – Exemplo de KB na UC de SO

Na Figura 7 podemos ver um exemplo da forma de apresentação de uma pergunta na KB. Dado o carácter extremamente visual e *web-based* das perguntas, as mesmas não foram colocadas na lista de conteúdos, mas ainda assim estão todas disponíveis no Moodle Iscte.

O trabalho prático de SO é normalmente um projeto de sistemas de informação que é montado com base num cenário fictício, mas verosímil, de um negócio empresarial. O trabalho em si é dividido em três partes:

› Na primeira parte, alinhada com o que os alunos estão a aprender nas aulas, o propósito é focado na manipulação de ficheiros de texto e extração de dados utilizando as funcionalidades poderosas do SO Linux para esse efeito. São, portanto, pedidas funcionalidades de administração de um sistema de informação sob a forma de *scripts* Bash que cuidam detarefas como o registo de utilizadores, a criação de estatísticas de acesso, ou um menu interativo apresentado na Shell.

Figura 7 – Exemplo de apresentação de pergunta numa KB

› Na segunda parte, o projeto já é realizado através do recurso a programação em linguagem C, e onde o foco é a criação de múltiplos processos, tipicamente num cenário de utilização usando um padrão de aplicação cliente-servidor, onde o servidor tem um comportamento típico àsaplicações realizadas na indústria, recebendo pedidos e delegando os mesmos a novos processos dedicados a cada um dos clientes. O projeto implica também uma boa dose de comunicação e interação entre clientes e servidores dedicados, pelo que se utilizam *pipes*, acesso a ficheiros sequenciais e de acesso direto, e sinais para tratamento de situações excecionais (como erros, ou processamento de cancelamentos por parte doutilizador).
› Finalmente, a terceira parte do projeto de SO é sobretudo dedicado à comunicação usando os mecanismos de comunicação inter-processual do Linux (IPC), onde realça-se a utilização de filas de mensagem e

memórias partilhadas. Da mesma forma, evidencia-se o grave problema de concorrência que os próprios alunos puderam verificar nos seus trabalhos, na segunda parte do projeto, que diz respeito a termos múltiplos processos (servidores dedicados) todos a aceder a um sistema de informação de forma descontrolada e desordenada, com enormes riscos de incoerência devidas ao acesso concorrente a recursos partilhados. Desse modo, também esta parte do projeto se dedica a colmatar esse problema através do uso de mecanismos de sincronização (semáforos IPC).

O projeto de Sistemas Operativos foi realizado em grupo durante os primeiros anos do período reportado, sendo que atualmente a equipa docente considera ser pedagogicamente mais proveitoso para os alunos que este projeto seja feito individualmente pelos alunos. Esta UC não se destina a ensinar ou avaliar os alunos em programação, mas sim em compreender os sistemas operativos e as técnicas de utilização das funcionalidades disponibilizadas pelos mesmos para resolver os problemas aplicacionais, e, portanto, considera-se ser este o meio mais eficaz para os alunos aprenderem e ganharem uma maior proficiência na matéria em questão.

Curiosamente, apesar destas alterações terem sido no sentido de passar o trabalho de grupo para individual, os proponentes são fervorosos e aguerridos defensores do trabalho em grupo e do espírito de comunidade. A questão aqui é que o modelo tradicional de trabalho de grupo universitário, nos moldes que estava a ser realizado durante vários anos, foi o de um elemento do grupo que trabalhava e os restantes que se "encostavam", e, por mais que as avaliações de grupo incluíssem avaliações orais, o espírito de entreajuda dos colegas tentava sempre evitar que os alunos que não trabalhavam (nesta UC, provavelmente o fariam noutras) ficassem com nota mais baixa. A partir da altura em que os trabalhos passaram a ser individuais, notou-se claramente uma enorme diferença na qualidade dos alunos, facilmente demonstrada na qualidade das dúvidas apresentadas aos docentes, à proficiência com que os mesmos demonstram no manuseamento das ferramentas e ambientes propostos, etc. Ainda assim, não é proibido o ajuntamento, é perfeitamente aceite que os alunos mostrem abertamente que se juntam ainda em grupos informais para discussão de ideias e soluções de forma colaborativa, e depois extraiam daí os resultados para os seus trabalhos individuais.

4. AVALIAÇÃO

Mediante aprovação em Comissão Pedagógica da Escola, foi estabelecido que esta UC é feita apenas por Avaliação ao longo do semestre, não contemplando Exame Final. As diferentes componentes da avaliação são explicadas aos alunos na primeira aula teórico-prática.

4.1. Nota final de avaliação

A avaliação à UC de Sistemas operativos é resultante da soma de duas componentes principais, uma teórica e uma prática, sendo que ambas têm um peso equivalente de 50% para a nota final:

Nota Final de SO = Nota Teórica (50%) + Nota Prática (50%)

Apesar de cada uma das componentes teórica e prática terem os seus requisitos para aproveitamento, para aproveitamento na UC de SO é preciso que a soma das duas componentes, pesada da percentagem correspondente, resulte numa nota igual ou superior a 9,5 valores.

4.2. Componente teórica de avaliação

A componente teórica para avaliação da UC de SO resulta da nota obtida num teste escrito, onde é avaliada:

- A capacidade do aluno para compreensão da matéria teórica (peso 8/20), assim como
- A capacidade de utilização de comandos Linux e desenvolvimento de scripts Bash (4/20),
- A capacidade de desenvolvimento de programas em C onde haja necessidade de multiprocessamento e interação usando sinais (4/20), e finalmente,
- A capacidade de utilizar mecanismos de comunicação e sincronização IPC (4/20).

As quotas mencionadas anteriormente são puramente indicativas, podendo pontualmente ser alteradas em algum teste em particular.

As perguntas feitas na prova escrita podem envolver aspetos relativos aos trabalhos feitos na componente laboratorial.

Para poder ter aproveitamento em Avaliação ao longo do semestre (a única possível), os alunos terão de ter uma nota de teste teórico de pelo menos 7 valores. A assiduidade nas aulas não é um requisito para o aproveitamento nesta componente de avaliação.

4.3. Componente prática de avaliação

A componente prática de avaliação da UC de SO é composta pelos resultados obtidos em duas provas:

- Um projeto (trabalho prático), realizado em grupo ou individualmente, dividido em três etapas, cada um deles com o peso de 25% da nota da componente prática;
- 9 (nove) mini-testes "TPC" para submissão no Moodle (contam apenas os 8 melhores), contando a nota médias desses 8 melhores testes com o peso de 25% da nota prática.

Os mini-testes são disponibilizados todas as semanas durante as semanas 1, 2, 3, 5, 6, 7, 9, 10 e 11, publicados à segunda-feira à noite, e o período de submissão de cada mini-teste encerra ao domingo às 23h59.

As três partes do projeto têm de ser submetidas até às 23h59 de domingo das semanas 4, 8 e 12.

A nota da componente prática do trabalho pode ser sujeita a cada aluno através de uma prova oral. A nota dependerá dos relatórios, do desempenho do aluno na oral e poderá ter em conta a assiduidade.

Para poder ter aproveitamento em Avaliação ao longo do semestre (a única possível), os alunos terão de ter uma nota na componente prática de pelo menos 9,5 valores, sendo que a nota máxima da componente prática é limitada automaticamente para não poder ser superior a 6 valores da nota obtida no teste teórico.

Até há poucos anos, a realização do projeto de SO era efetuada em grupo, com todas as vantagens de promoção do trabalho em equipa que é totalmente valorizada. Com a experiência do contacto com os alunos nos projetos, o corpo docente começou a aperceber-se de que, por norma, em cada grupo de 3 alunos, apenas um deles fazia o trabalho e os restantes não faziam esforço para se colocarem a par do que tinha sido feito (dado que a complexidade do trabalho assim não o permitia). Devido a esse fato, e após acordo entre todos os docentes da UC, foi decidido que o trabalho iria reduzir um pouco em complexidade, mas que iria passar a ser individual. Este simples fator, segundo o feedback que o corpo docente passou a ter dos alunos, fez com que a proficiência dos alunos globalmente melhorasse muito. Continua a haver alunos que não adquirem conhecimentos necessários para passar à UC, mas globalmente o nível das turmas melhorou substancialmente.

Claro que isso trouxe um acréscimo substancial de peso para os docentes, que em vez de corrigirem 100 trabalhos complexos, passariam a ter de corrigir 400 ou 500 trabalhos mais simples. A solução passou por desenvolver-se uma forma automatizada de correção dos trabalhos, através de um inteligente conjunto de scripts realizados em Python que fazem *overload* dos métodos principais avaliados para os conteúdos de Sistemas Operativos, e que com isso determinam de forma inteligente qual é o padrão esperado para a solução. No primeiro ano, a experiência revelou-se pouco produtiva, já que os alunos

não faziam o seu desenvolvimento de acordo com os padrões esperados pelos scripts de validação e avaliação, por isso, foi necessário realizar a avaliação manual dos trabalhos. Esse esforço adicional trouxe a motivação necessária para fazer as alterações que resultariam no modelo atual.

Mais uma vez, a UC de SO não faz parte da área de disciplinas de ensino à programação e algoritmia, logo não é esse tipo de avaliação que é pretendida ser feita aos alunos. O projeto de SO atualmente é composto de um esqueleto completo do conteúdo do projeto (ver Figura 8).

```c
/*****************************************************************************
** ISCTE-IUL: Trabalho prático 3 de Sistemas Operativos 2023/2024, Enunciado Versão 1+
**
** Este Módulo não deverá ser alterado, e não precisa ser entregue
** Nome do Módulo: common.h
** Descrição/Explicação do Módulo:
**     Definição das estruturas de dados comuns aos módulos servidor e cliente
**
*****************************************************************************/
#ifndef __COMMON_H__
    #define __COMMON_H__

    #include "/home/so/utils/include/so_utils.h"
    #include <signal.h>     // Header para as constantes SIG_* e as funções signal() e kill()
    #include <unistd.h>     // Header para as funções alarm(), pause(), sleep(), fork(), exec*() e get*pid()
    #include <sys/ipc.h>    // Header para as funções de IPC
    #include <sys/sem.h>    // Header para as funções de IPC

    #define FILE_DATABASE_PASSAGEIROS   "bd_passageiros.dat"    // Ficheiro de acesso direto que armazena a lista de passageiros
    #define FILE_DATABASE_VOOS          "bd_voos.dat"           // Ficheiro de acesso direto que armazena a lista de voos

    #define RETURN_SUCCESS       0      // Defines utilitários para valores de retorno
    #define RETURN_ERROR        -1      // Defines utilitários para valores de retorno
    #define CICLO1_CONTINUE      2      // Valor de retorno que indica que o main() deve recomeçar o CICLO1
    #define CYCLE1 CONTINUE      CICLO1 CONTINUE


    typedef struct {
        long msgType;                   // Tipo da Mensagem
        struct {
            CheckIn infoCheckIn;        // Informação sobre o CheckIn
            Voo infoVoo;                // Informação sobre um Voo
        } msgData;                      // Dados da Mensagem
    } MsgContent;

    typedef struct {                    // Base de dados de Negócio, em Memória Partilhada
        CheckIn listClients[MAX_PASSENGERS]; // Lista de passageiros
        Voo listFlights[MAX_FLIGHTS];        // Lista de voos dos passageiros
    } DadosServidor;

    /* Protótipos de funções */
    int  initShm_S1 ();                     // S1:   Função a ser implementada pelos alunos
    int  initMsg_S2 ();                     // S2:   Função a ser implementada pelos alunos
    int  initSem_S3 ();                     // S3:   Função a ser implementada pelos alunos
    int  triggerSignals_S4 ();              // S4:   Função a ser implementada pelos alunos
    int  readRequest_S5 ();                 // S5:   Função a ser implementada pelos alunos
    int  createServidorDedicado_S6 ();      // S6:   Função a ser implementada pelos alunos
    void terminateServidor_S7 ();           // S7:   Função a ser implementada pelos alunos
    void trataSinalSIGINT_S8 (int);         // S8:   Função a ser implementada pelos alunos
    void trataSinalSIGCHLD_S9 (int);        // S9:   Função a ser implementada pelos alunos
    int  triggerSignals_SD10 ();            // SD10: Função a ser implementada pelos alunos
    int  searchClientDB_SD11 ();            // SD11: Função a ser implementada pelos alunos
    int  searchFlightDB_SD12 ();            // SD12: Função a ser implementada pelos alunos
```

Figura 8 – Exemplo de cabeçalho de projeto de SO

Este esqueleto inclui uma função main() fechada que os alunos não devem alterar, que chama de forma metódica funções que, essas sim, deverão ser desenvolvidas pelos alunos.

```c
/**
 * @brief Processamento do processo Cliente
 *        "os alunos não deverão alterar a função main(), apenas compreender o que faz.
 *         Deverão, sim, completar as funções seguintes à main(), nos locais onde está claramente assinalado
 *         '// Substituir este comentário pelo código da função a ser implementado pelo aluno' "
 */
int main () {
    // C1
    msgId = initMsg_C1();
    so_exit_on_error(msgId, "initMsg_C1");
    // C2
    so_exit_on_error(triggerSignals_C2(), "triggerSignals_C2");
    // C3
    so_exit_on_error(getDadosPedidoUtilizador_C3(), "getDadosPedidoUtilizador_C3");
    // C4
    so_exit_on_error(sendRequest_C4(), "sendRequest_C4");
    // C5: CICLO6
    while (TRUE) {
        // C5
        configureTimer_C5(MAX_ESPERA);
        // C6
        so_exit_on_error(readResponseSD_C6(), "readResponseSD_C6");
        // C7
        int lugarEscolhido = trataResponseSD_C7();
        if (RETURN_ERROR == lugarEscolhido)
            terminateCliente_C9();
        // C8
        if (RETURN_ERROR == sendSeatChoice_C8(lugarEscolhido))
            terminateCliente_C9();
    }
}
```

Figura 9 – Exemplo de função *main()* do projeto de SO

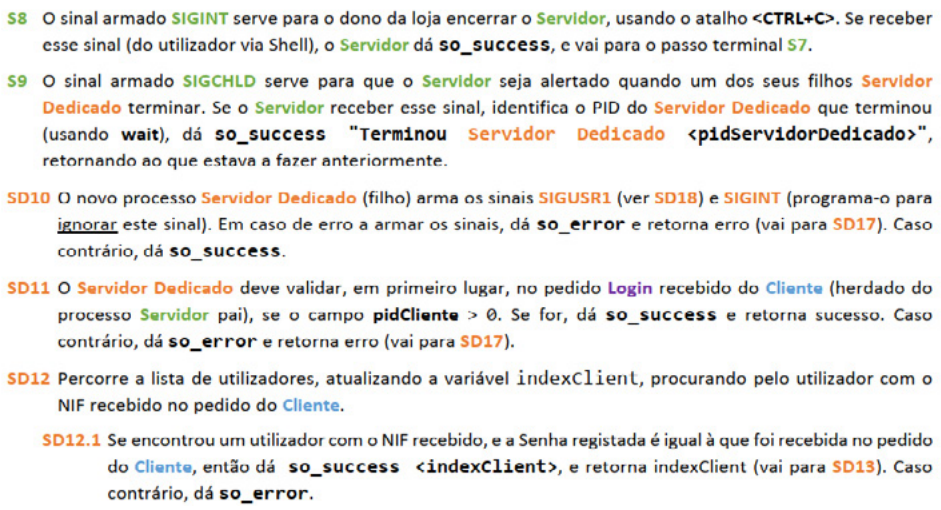
Nestas funções invocadas pelo main(), é onde os alunos se podem limitar a mostrar a sua capacidade de trabalhar com os elementos e conceitos de SO, e não (tanto) de algoritmia e programação (Figura 10). O processo não é fácil, já que o docente Carlos Coutinho tem o trabalho de especificar com todo o detalhe possível o que é suposto que os alunos façam, e depois passa todas as linhas desse enunciado de projeto gigantesco (Figura 11) para código e para comentários no trabalho. Como resultado, passou a ser possível avaliar de forma automática o trabalho dos alunos. Podemos, com orgulho, dizer que o processo foi tão popular que os alunos têm passado a palavra, e já começa a haver outras UC que iniciaram o mesmo processo na UC respetiva.

Ainda assim, as notas nos projetos de SO não eram muito boas, dado que os alunos se limitavam a realizar o que pensavam ser o pedido nos enunciados, esquecendo-se de realizar muitas das validações e verificações essenciais para um bom funcionamento do projeto. Esse fato associado agora a avaliação ser automatizada, logo, sem a tolerância e "vista grossa" do docente humano, levou a que muitos alunos pensassem que teriam notas elevadas e não era o caso. Depois de alguns casos em que foi necessário realizar provas orais para garantir o conhecimento, e dado que os docentes querem mesmo que os alunos saibam utilizar corretamente os elementos de SO, em vez de facilitarem de futuro o validador para ser menos exigente, tomaram outra abordagem, que foi a de entregarem o script validador aos alunos. Desta forma, na atualidade, os alunos após fazerem os seus trabalhos correm um script validador que lhes diz que têm vários dos seus testes errados, explicando-lhes qual é o

teste que está a ser realizado, e os resultados esperados. Esse fator levou a que, desde então, as notas do projeto de SO melhoraram substancialmente, e levou em consequência a que até nos testes escritos, os alunos já têm em consideração esses testes e verificações, o que faz com que haja uma maior satisfação tanto por parte dos docentes com a qualidade dos conhecimentos dos alunos, como dos próprios alunos que agora conseguem fazer os trabalhos de forma mais alinhada com as expectativas dos docentes.

```
/**
 * @brief S2    Cria a Message Queue (MSG) do projeto, que tem a KEY IPC_KEY, realizando as
 *              seguintes operações:
 *        S2.1  Se já existir, deve apagar a fila de mensagens. Em caso de qualquer erro, dá
 *              so_error e retorna erro (vai para S7). Caso contrário, dá so_success.
 *        S2.2  Cria a Message Queue com a KEY IPC_KEY. Em caso de erro, dá so_error e retorna
 *              erro (vai para S7). Caso contrário, dá so_success <msgId> e retorna o ID da MSG.
 *
 * @return int RET_ERROR em caso de erro, ou a msgId em caso de sucesso
 */
int initMsg_S2() {
    int result = RET_ERROR;    // Por omissão retorna erro
    so_debug("<");

    // Substituir este comentário pelo código da função a ser implementado pelo aluno

    so_debug("> [@return:%d]", result);
    return result;
}

/**
 * @brief S3    Cria um grupo de semáforos (SEM) que tem a KEY IPC_KEY, realizando as seguintes
 *              operações:
 *        S3.1  Se já existir, deve apagar o grupo de semáforos. Em caso de qualquer erro, dá
 *              so_error e retorna erro (vai para S7). Caso contrário, dá so_success.
 *        S3.2  Cria um grupo de três semáforos com a KEY IPC_KEY. Em caso de qualquer erro, dá
 *              so_error e retorna erro (vai para S7). Caso contrário, dá so_success <semId>.
 *        S3.3  Inicia o valor dos semáforos SEM_USERS e SEM_PRODUCTS para que possam trabalhar em
 *              modo "exclusão mútua", e inicia o valor do semáforo SEM_NR_SRV_DEDICADOS com o
 *              valor 0. Em caso de erro, dá so_error e retorna erro (vai para S7).
 *              Caso contrário, dá so_success e retorna o ID do SEM.
 *
 * @return int RET_ERROR em caso de erro, ou a semId em caso de sucesso
 */
int initSem_S3() {
    int result = RET_ERROR;    // Por omissão retorna erro
    so_debug("<");

    // Substituir este comentário pelo código da função a ser implementado pelo aluno

    so_debug("> [@return:%d]", result);
    return result;
}
```

Figura 10: Exemplo de funções totalmente documentadas, para serem realizadas pelos alunos



S8 O sinal armado SIGINT serve para o dono da loja encerrar o Servidor, usando o atalho **<CTRL+C>**. Se receber esse sinal (do utilizador via Shell), o Servidor dá **so_success**, e vai para o passo terminal S7.

S9 O sinal armado SIGCHLD serve para que o Servidor seja alertado quando um dos seus filhos Servidor Dedicado terminar. Se o Servidor receber esse sinal, identifica o PID do Servidor Dedicado que terminou (usando **wait**), dá **so_success "Terminou Servidor Dedicado <pidServidorDedicado>"**, retornando ao que estava a fazer anteriormente.

SD10 O novo processo Servidor Dedicado (filho) arma os sinais SIGUSR1 (ver SD18) e SIGINT (programa-o para ignorar este sinal). Em caso de erro a armar os sinais, dá **so_error** e retorna erro (vai para SD17). Caso contrário, dá **so_success**.

SD11 O Servidor Dedicado deve validar, em primeiro lugar, no pedido Login recebido do Cliente (herdado do processo Servidor pai), se o campo **pidCliente** > 0. Se for, dá **so_success** e retorna sucesso. Caso contrário, dá **so_error** e retorna erro (vai para SD17).

SD12 Percorre a lista de utilizadores, atualizando a variável `indexClient`, procurando pelo utilizador com o NIF recebido no pedido do Cliente.

SD12.1 Se encontrou um utilizador com o NIF recebido, e a Senha registada é igual à que foi recebida no pedido do Cliente, então dá **so_success <indexClient>**, e retorna indexClient (vai para SD13). Caso contrário, dá **so_error**.

Figura 11 – Exemplo de trecho do enunciado de um projeto de SO

5. CONCLUSÕES

O modelo de UC aqui proposto tem sido experimentado e tem evoluído na UC de Sistemas Operativos do Iscte, uma UC que engloba anualmente cerca de 600 alunos. Da experiência desta evolução podem-se tirar várias conclusões. A primeira é de que os alunos aqui chegados têm ainda hábitos e proficiência muito baixos, e, segundo os relatos de vários deles passados alguns anos, é de que adquirem nesta UC competências que se revelaram essenciais para o seu desenvolvimento pessoal. Desde o início, a UC promove e propõe o trabalho num ambiente remoto (um servidor Linux), que é algo de muito estranho aos alunos, habituados a trabalhar nos ambientes locais das suas máquinas. Esta competência é cada vez mais importante num mundo cada vez mais global, e onde o trabalho remoto na indústria é cada vez mais uma realidade diária, com trabalhadores a realizarem o seu trabalho remotamente a partir da sua casa, ou a partir da sua cidade, ou até a partir do seu país, elementos esses que cada vez mais são diferentes do local da empresa onde trabalham. O trabalho em ambiente remoto *Cloud* é também algo que os alunos terão de enfrentar vulgarmente, portanto é essencial que desde o primeiro ano estejam familiarizados com estes ambientes. Também a transição de trabalho em ambientes gráficos, tipicamente destinados a utilizadores de massa, para ambientes em linha de comandos, muito mais poderosos e versáteis, é algo que é altamente valorizado para os profissionais dos cursos da Escola de Tecnologias. Os resultados também demonstraram que a proficiência de um aluno desta UC há alguns anos é claramente muito inferior ao dos alunos atuais da UC, que demonstram de uma forma clara uma destreza e capacidade notáveis. Outros aspetos foram altamente promovidos, o desenvolvimento de um ambiente

de trabalho customizado, com primitivas e funções próprias da própria UC, o que faz com que seja mais difícil obter respostas na *internet* ou em motores de Inteligência Artificial Generativa. Isto também promoveu que os alunos não façam tanto o trabalho isoladamente com recurso às tecnologias anteriores, mas promovam sim o trabalho colaborativo, onde os alunos aprenderam a consultar fóruns internos da UC (onde estão várias respostas específicas que resolvem os seus problemas) antes de irem pesquisar em recursos externos, e também a colocar as suas dúvidas nos fóruns da UC, onde outros colegas mais experientes são incentivados a exprimir as suas ideias e propostas para resolver os problemas dos menos habilitados.

BIBLIOGRAFIA E MATERIAIS PEDAGÓGICOS RELEVANTES

A maior parte das apresentações, quer teóricas ou práticas, dadas no âmbito desta UC tem no seu último slide uma lista de referências, quer porque tenham sido utilizadas na elaboração da apresentação em si, ou porque o autor considerou que seriam interessantes para que os alunos possam detalhar e aprofundar mais os conhecimentos da apresentação em questão:

Brian Kernighan, Dennis Ritchie (1988), "The C Programming Language", second Edition, Prentice Hall, ISBN: 9780131101630
https://www.infopedia.pt/$sistema-operativo
https://kids.pplware.sapo.pt/o-meu-computador/o-que-e-um-sistema-operativo/
https://kids.pplware.sapo.pt/o-meu-computador/que-outros-sistemas-operativos-existem/
https://www.e-konomista.pt/o-que-e-um-sistema-operativo/
https://conceito.de/sistema-operativo
https://www.tecmundo.com.br/sistema-operacional/2031-a-historia-dos-sistemas-operacionais-ilustracao-.htm
https://pt.wikibooks.org/wiki/Sistemas_operacionais/História
https://ricardoribeiro21.wordpress.com/2013/11/22/historia-e-evolucao-dos-sistemas-operativos/
https://en.wikipedia.org/wiki/History_of_Linux
https://www.digitalocean.com/community/tutorials/brief-history-of-linux
https://gs.statcounter.com/os-market-share
https://blog.codinghorror.com/understanding-user-and-kernel-mode
https://www.sciencedirect.com/topics/engineering/kernel-mode
https://en.wikipedia.org/wiki/Protected_mode
https://www.geeksforgeeks.org/functions-of-operating-system
https://www.daydreameducation.co.uk/poster-operating-systems
https://www.tutorialspoint.com/operating_system/os_multi_threading.htm
https://www.quora.com/What-is-Memory-hierarchy/answer/Atul-Kumar-1028
https://www.slideshare.net/sgpraju/os-swapping-paging-segmentation-and-virtual-memory