P.PORTO ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO	Tipo de Prova Trabalho Prático	Ano letivo 2025/2026	Data 03-12-2025
	Curso LEI e LSIRC		Hora
	Unidade Curricular Sistemas Operativos		Duração

Observações

Trabalho Prático

Sistemas Operativos

Marco Gomes
mfg@estg.ipp.pt

Ronaldo Salles
rmo@estg.ipp.pt

Maria Tavares
mct@estg.ipp.pt

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Dezembro de 2025

1. Considerações Gerais

O trabalho prático consiste na elaboração de um programa em Linguagem Java que faça uso das técnicas de multiprocessamento, comunicação e sincronização aprendidas nas aulas. O trabalho deverá ser desenvolvido em grupo. Serão aceites trabalhos individuais, desde que o aluno manifeste atempadamente a intenção de o fazer.

A deteção de trabalhos fraudulentos invalida a nota de todos os grupos de todos os trabalhos envolvidos. Serão considerados trabalhos fraudulentos, aqueles onde se verifique trabalho desenvolvidos por pessoas que não façam parte do grupo, na totalidade do trabalho ou apenas em parte deste.

1.1. Defesa

Todos os trabalhos práticos estão sujeitos a defesa por parte do grupo que o elaborou. A defesa decorrerá numa sessão remota seguinte à data de entrega (excecionalmente prevista para o dia 22 de dezembro). A não comparência de um aluno à defesa implica a não consideração do trabalho para a nota do aluno em questão. Uma defesa considerada como não satisfatória por parte do docente da disciplina implica a não consideração do trabalho para a nota do aluno em questão.

1.2. Outras considerações

Quando não seja respeitado o formato de entrega (tipos de ficheiros e nomes), os alunos que compõem o grupo sofrerão uma penalização de 10% na nota final do trabalho.

As entregas de trabalhos não passíveis de compilação e execução na respetiva defesa implicam a não consideração do trabalho para a nota dos alunos do grupo em questão.

2. Datas

A indicação da composição do grupo deverá ser efetuada pelo moodle (**até um máximo de três elementos**).

A data-limite para a entrega é **19 de dezembro de 2025, pelas 23h59.** Os trabalhos entregues fora de prazo não serão considerados. A entrega deverá ser efetuada pelo moodle. Deverá ser entregue o código fonte e o relatório num ficheiro ZIP com o nome: **so_grupoX.zip** (onde X deverá ser substituído pelo número do grupo).

3. Enunciado: «Simulador de concorrência com monitorização»

O trabalho visa a simulação de um simulador de concorrência com a funcionalidade de monitorizar a execução de *thread* sob a perspectiva de cibersegurança. A linguagem de programação a utilizar é o Java e deverão recorrer à matéria abordada na unidade curricular, em particular, aos mecanismos de sincronização e comunicação lecionados nas aulas práticas. Neste trabalho, será solicitado o desenvolvimento de cenários de execução que explore os principais desafios de sistemas multiprogramados, a saber: *race conditions*, *deadlocks*, *starvation* e a execução coordenada de *thread* em ambientes de produção.

Alguns conceitos fundamentais sobre a matéria lecionada que deverão ter presentes para melhor entendimento do trabalho pedido:

- Recurso partilhado: um recurso computacional que é partilhado por múltiplas *thread*.
- Seção crítica: seção do código que envolve o acesso ou a alteração do estado de um recurso partilhado por múltiplas *thread*.
- Situação de competição *(race condition*): uma situação em que não é possível determinar o valor final de uma variável que ocorre numa seção crítica.
- Situação de bloqueio (*deadlock*): estado em que dois ou mais processos/*thread* ficam bloqueados permanentemente porque cada um espera por um recurso que o outro mantém, tornando impossível que qualquer um deles continue a execução.
- Situação de míngua (*starvation*): ocorre quando um processo/*thread* continua ativo no sistema não temdo acesso aos recursos computacionais de que necessita, devido a políticas de escalonamento ou ao comportamento de outros processos/*thread*.
- Sincronização e/ou sinalização: uma solução para tratar uma situação de competição que permite que uma *thread* execute sobre uma seção crítica.

Como inspiração para o trabalho poderão recorrer à descrição e funcionalidade de eBPF (*extended Berkeley Packet Filter*) que pode ser visto como um exemplo moderno de extensão controlada das funcionalidades do *kernel*, permitindo implementar perspectivas fundamentais como cibersegurança, isolamento, monitorização de chamadas ao sistema, gestão de eventos e monitorização de recursos sem alterar diretamente o código do núcleo do sistema operativo.

O objetivo principal é permitir aos estudantes explorarem os fenómenos de *race conditions, deadlocks, starvation* e a execução coordenada de *thread* em sistemas operativos e analisar as suas implicações para a cibersegurança, introduzindo, ao mesmo tempo, um mecanismo de monitorização no estilo eBPF para detetar esses problemas num ambiente simulado em Java.

Pretende-se, assim, o desenvolvimento de um programa que simule concorrência entre *thread* no acesso a recursos partilhados com um mecanismo de monotorização para identificar e rastrear eventos inseguros.

Os seguintes requisitos são de implementação recomendada no trabalho com a finalidade de testar a fiabilidade do sistema a desenvolver.

3.1. Simulador de concorrência [5.0 valores]

- Race condition:
  a) Criar recursos partilhados;
  b) Criar múltiplas *thread* que acedem e modificam os recursos sem sincronização adequada.
  c) Demonstrar resultados inconsistentes causados por *race conditions*.
  d) Corrigir o problema utilizando as estratégias lecionadas.
- Deadlock:
  a) Criar várias *thread* que competem pela exclusividade dos recursos.
  b) Implementar uma sequência de eventos que provoque um *deadlock*.
  c) Registar os *deadlocks* detetados no ficheiro de registo de atividade específico (e.g. um log);
  d) Corrigir o problema utilizando as estratégias lecionadas.
- Starvation:
  a) Criar várias *thread* com diferentes prioridades ou padrões de acesso.
  b) Demonstrar uma situação de *starvation*, em que uma ou várias *thread* são continuamente preteridas e não conseguem aceder ao recurso.
  c) Registar as ocorrências de *starvation* detetados no ficheiro de registo de atividade específico (e.g. um log);
  d) Corrigir o problema utilizando as estratégias lecionadas.

3.2. Mecanismos de Monitorização ao estilo eBPF [3.0 valores]

Implementar uma classe que monitorize os eventos das *thread* e regista comportamentos anómalos. Deve, portanto, registar todos os acessos aos recursos partilhados e ter as seguintes funcionalidades:
1) Detetação de *race conditions, deadlocks* (potenciais e confirmados), padrões de *starvation* (longos tempos de espera). Deve gerar alertas e registá-los num ficheiro de registo de atividade específico.
2) Manter estatísticas como número de acessos por *thread*, a ordem de obtenção de exclusividade; o tempo de espera para entrada em secções críticas.

3.3. Pespetiva de cibersegurança [3.0 valores]

O programa deve simular os seguintes cenários de aplicação:

1) **Race conditions** que podem levar a aumento de privilégios, corrupção de dados ou *bypass* de verificações.

2) *Deadlocks* explorados como ataques de DoS (*Denial of Service*).

3) *Starvation* que pode atrasar serviços críticos, causando falhas de segurança.

3.4. Relatório [9.0 valores]

Em conjunto com o código fonte do trabalho prático, deverá ser entregue um relatório contendo a seguinte lista de tópicos:

1. Introdução
    a. Importância da concorrência em sistemas operativos.
    b. Introdução ao eBPF: para que serve, como monitoriza eventos do sistema e a sua relevância para a cibersegurança.
2. Problemas de concorrência
    a. Explicar cada um dos problemas: *race conditions, deadlocks, starvation*, ordem conflituante da execução;
    b. Utilizar os exemplos retirados da implementação prática para os ilustrar.
3. Problemas de concorrência
    a. Descrever como cada problema pode ser explorado em sistemas operativos reais: corrupção de memória, acesso indevido, ataques DoS e bloqueio de serviços críticos.
4. Monitorização ao Estilo eBPF
    a. Explicar a arquitetura da classe de monitorização.
    b. Descrever as métricas recolhidas e como são detetados os comportamentos anómalos.
    c. Relacionar a monitorização com proteção e mitigação de ataques.
5. Implementação
    a. Explicar a estrutura do código desenvolvido;
    b. Descrever a simulação;
    c. Apresentação de *logs* reais da simulação.
6. Conclusão
    a. Principais aprendizagens sobre concorrência, segurança e monitorização.
    b. Equacionar como seria possível implementar algo semelhante com eBPF.

O único formato aceite para o relatório é o formato PDF.