LICENCIATURA EM ENGENHARIA INFORMÁTICA
LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

# Sistemas Operativos

*Recurso*

14 de junho de 2024 — Duração: 2h

Por favor responda ao grupo I e a **cada** exercício do grupo II em folhas de teste **separadas**. Obrigado.

## I

**1** Considere um serviço que permite a execução de programas (*p.ex.*, editores de texto, *bash*, *chatbots*) que requerem interação frequente com os utilizadores. Como este serviço nem sempre tem capacidade para executar todos os programas simultaneamente, é necessário escalonar a execução dos mesmos.

Assumindo que este serviço suporta mecanismos de desafetação forçada (*i.e.*, consegue interromper um programa durante a sua execução e resumir a mesma mais tarde) que algoritmo de escalonamento escolheria para o mesmo? Justifique a sua resposta indicando **uma vantagem** e **uma possível preocupação ou desvantagem** da sua escolha.

**2** Ao analisar o comportamento de acesso ao disco rígido (HDD) de um servidor, observou que algumas aplicações exibem mau desempenho (*i.e.*, os pedidos ao disco destas aplicações demoram muito tempo a serem servidos). Ao analisar o sistema operativo repara que este está configurado com um algoritmo de escalonamento Shortest Seek Time First (SSTF). Relembre que este algoritmo escolhe os próximos pedidos a servir de acordo com a proximidade dos cilindros do disco a que estes acedem (*i.e.*, minimiza o movimento da cabeça do disco).

Indique **uma possível razão** para o mau desempenho das aplicações. Ainda, indique **um algoritmo de escalonamento alternativo** que podia ser mais justo e melhorar o desempenho destas aplicações. Justifique a sua resposta.

## II

Considere um sistema de processamento de tarefas composto por dois programas: `scheduler` e `executor`. O programa `scheduler` lê tarefas de um ficheiro em formato binário e envia-as, via *pipe com nome*, para o programa `executor` que executará as mesmas. Cada tarefa armazenada é representada pela estrutura de dados `Job`. O campo `id` identifica a tarefa, o campo `request` contém dados para a execução da tarefa e, por fim, o campo `sent` indica se a tarefa já foi previamente lida pelo programa `scheduler` e enviada para o programa `executor`.

```
1  typedef struct job {
2      int id;
3      char request[128];
4      int sent; // 0 (não enviada) ou 1 (enviada)
5  } Job;
```

**1** Implemente a função `void schedule_jobs(char* path, int n_jobs)` do programa `scheduler`. A função recebe como argumentos `path`, que indica o caminho para o ficheiro que contém as tarefas, e `n_jobs`, o número total de tarefas nesse mesmo ficheiro. A função deve enviar as tarefas que não tenham sido previamente lidas e enviadas para o programa `executor`. O ficheiro de tarefas deve ser atualizado, modificando as entradas existentes de modo a registar o envio das tarefas. Assuma que o *pipe com nome* usado é `"executor_fifo"` e já se encontra criado.
**Valorização**: Permita que `N` processos leiam e enviem tarefas concorrentemente para o programa `executor`.

**2** Implemente a função `void handle_job(Job job)` do programa `executor`. Esta função, para além de executar uma tarefa (`Job`), deve também gerir os *outputs* produzidos pela mesma. Para o ajudar na implementação desta função, assuma a existência de um programa `run_job <request>` que executa uma tarefa, recebendo como argumento os dados armazenados no campo `request` da estrutura `Job`. Assuma também a existência de um programa `proc_output` que processa linhas de texto recebidas via `stdin`.
A função `handle_job` deve executar a tarefa, redirecionando o conteúdo escrito no `stdout` para o programa `proc_output`, e o conteúdo escrito no `stderr` para um ficheiro com o nome `"error_<id>"`, sendo `id` o identificador da tarefa.