Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas

# Sistemas Operativos 2021/2022

## Ficha Nº 6 Threads, Mutexes Vars Condicionais, ncurses

**Tópicos** ->
- Execução de várias tarefas em simultâneo com threads POSIX.
- Coordenação de gestão de exclusão mútua com Mutexes POSIX.
- Coordenação de tarefas com variáveis condicionais POSIX.
- Utilização da biblioteca ncurses para gestão do output na consola.

### Exercícios

Exercício 1: Utiliza *threads* e *mutexes* para executar e coordenar várias tarefas em simultâneo.

Exercício 2: Utiliza *threads* para conseguir ler de dois *named pipes* "em simultâneo".

Exercício 3: Utiliza a biblioteca *ncurses*.

Exercício 4: Junta o uso de pipes com threads com ncurses.

Exercício 5: Utiliza variáveis condicionais.

**>> Notas**

- Estes exercícios são maiores do que os das fichas anteriores. É **muito aconselhável** que os alunos vão preparando em casa, com antecedência, o código dos exercícios
- Alguns exercícios envolvem a biblioteca ***ncurses*** (manipulação de ecrã). Não tendo sido abordada este ano, esses a funcionalidade pode ser removida dos exercícios.

**1.** Escreva um programa que imprima sequências de caracteres no ecrã. O programa é organizado em tarefas. Existem pelo menos duas tarefas.

- A tarefa A imprime caracteres "A". Tem o seguinte comportamento: durante um intervalo de tempo aleatório entre 1 e 5 segundos imprime caracteres ".", um caracter por segundo. De seguida imprime 3 caracteres "A" aguardando também 1 segundo entre cada um. Este comportamento repete-se indefinidamente. A tarefa pode ser interrompida a pedido do "resto do programa" no início de cada iteração (antes do primeiro caracter ".").
- A tarefa B é exactamente como a tarefa A, mas imprime "B" em vez de "A".

O programa mantém-se atento ao utilizador. Se este escrever "sair", o programa desencadeia o encerramento das tarefas, aguarda que estas realmente terminem, imprime "adeus" e termina.

Importante:

- Neste programa não pode usar nem *select* nem sinais.
- É importante que os caracteres sejam impressos imediatamente. Use uma das técnicas que já viu nas aulas para garantir que assim é (caso contrário o exercício não vai funcionar).

a) Escreva o programa com as características pedidas. Teste-o e confirme que este funciona de acordo com o pretendido. Verifique que pode acontecer os caracteres A e B aparecerem misturados.

b) Modifique o programa de forma a cumprir o seguinte requisito adicional:

- Os caracteres A e B não podem aparecer misturados, ou seja, se uma das tarefas entrar na parte em que começa a imprimir os seus caracteres ("A" ou "B", conforma a tarefa), então a outra tem que aguardar que a primeira acabe de imprimir esses caracteres e entre na parte dos ".". Os caracteres "." podem ser impressos no meio dos "A" e "B".

**Objectivos do exercício - Threads e mutexes.**
- Explorar o conceito de threads com recurso ao API POSIX de threads em Linux.
- Explorar o conceito de exclusão mútua e resolver este tipo de situações com o API de Mutexes POSIX em Linux.

**2.** Escreva um programa chamado "stereo" que cria dois *named pipes* (p_esq e p_dir) e coloca no ecrã tudo o que receber nesses *pipes*. O programa não sabe em qual dos *pipes* vão chegar dados primeiro, mas consegue reagir imediatamente assim que chega alguma informação em qualquer deles. A informação que é enviada nos *pipes* é um nome (até 10 caracteres) e um valor inteiro. A informação é colocada no ecrã seguinte formado: "esq - " ou "dir - " (consoante o *pipe* de onde veio a informação seguido do nome e do inteiro recebidos. O programa termina ordeiramente quando recebe SIGINT.

- O mecanismo *select* poderia ajudar, mas não o pode usar neste exercício.
- Os sinais poderiam ajudar mas conduziriam a uma solução demasiado rebuscada. Também não os vai usar neste exercício (excepto para a terminação via SIGINT).
- Vai ser necessário construir também um programa "envia" que pede uma um nome ao utilizador, e depois, em ciclo, pede números. Para cada um, envia o nome e o número ao pipe p_esq ou p_dir. O *pipe* a usar é indicado por parâmetro na linha de comandos.

a) Identifique o mecanismo necessário para cumprir os requisitos e trace as linhas gerais da estrutura do programa. Esta alínea é para planear e debater com o docente e não envolve código.

b) Construa os programas "stereo" e "envia". Teste-os usando uma execução do "stereo" e duas ou mais do "envia" com um terminal para cada execução.

**Objectivos do exercício - Leitura de diversas fontes de dados em simultâneo através de *threads***
- Compreender o problema na leitura de diversas fontes de dados em simultâneo com recurso a *threads*
- Consolidar o uso de *named pipes*.

**3.** ***Este exercício envolve ncurses. Não tendo sido dado neste ano, fica exclusivamente como desafio/TPC.***

Volte ao exercício 2 e modifique o programa "stereo" de forma a cumprir o novo requisito:

- As mensagens oriundas do *pipe* p_esq aparecem apenas na metade superior da consola. As mensagens recebidas no *pipe* p_dir aparecem na metade inferior da consola.

a) Utilize as funções da biblioteca *ncurses* para posicionar o cursor na onde pretende imprimir.

Sugestões se implementação

- Pode ser útil manter variáveis para controlar em que parte da consola é que se vai imprimir a seguir para cada um dos *pipes*.
- Poderá ser necessário saber onde estava o cursor antes de o reposicionar para depois o voltar a colocar "onde estava".

b) Modifique a sua solução utilizando agora o mecanismo de janelas virtuais da biblioteca *ncurses* para simplificar separação das áreas dedicadas a cada *pipe*.

**Objectivos do exercício - Funcionalidade básica da biblioteca *ncurses***
- Explorar as funções de impressão em posições especificadas da consola.
- Explorar a funcionalidade de janelas virtuais da biblioteca *ncurses*.

**4.** Desenvolva um sistema que simule o funcionamento de um multibanco remoto. Este sistema é composto pelo programa ***servidor*** que armazena as contas dos clientes e atende os pedidos destes, e pelo programa ***cliente*** que serve de mera interface entre os utilizadores e o sistema. Devem ser cumpridos os seguintes requisitos:

a) Toda a lógica do sistema deve estar no lado do *servidor*, sendo o programa *cliente* um ponto de inserção de comandos e de apresentação de informação recebida do servidor;

b) Para cada cliente o *servidor* precisa "apenas" de armazenar o saldo da conta, que inicialmente é 0;

c) O *servidor* só suporta um máximo de 20 contas de clientes em simultâneo;

d) Ambos os programas seguem uma abordagem de linha de comandos, onde se podem inserir comandos que irão desencadear a execução de uma determinada ação;

e) Um cliente pode executar as seguintes ações:

   i. Obter o saldo, através do comando ***saldo***;

   ii. Levantar dinheiro, através do comando ***levantar <quantidade>*** (o saldo da sua conta tem de ser atualizado e pode ficar a negativo);

   iii. Depositar dinheiro, através do comando ***depositar <quantidade>*** (o saldo da sua conta tem de ser atualizado);

   iv. Sair do sistema, através do comando ***sair***, que termina o respetivo programa *cliente* e elimina a conta deste no servidor.

f) O servidor, para além de atender os pedidos dos clientes, permite a execução das seguintes ações:

   i. Listar todas as contas existentes no sistema, através do comando ***listar***;

   ii. Sair do sistema, através do comando ***sair***, que termina o programa *servidor* (não é necessário notificar os clientes de que o *servidor* terminou).

No que diz respeito a restrições técnicas da implementação, devem ser tidas em contas as seguintes:

a) Toda a comunicação entre os programas é feita através de ***named pipes***, o que levanta algumas questões que requerem uma resposta por parte dos alunos (por exemplo, como é que um programa conhece o(s) *named pipe(s)* do outro);

b) A execução em simultâneo de várias tarefas por parte dos programas tem de ser feita à custa de ***threads*** (o mecanismo *select* não pode ser utilizado);

c) Ao nível do *servidor*, a execução do comando ***listar*** e o atendimento dos pedidos dos clientes podem aceder de forma concorrente à mesma zona de memória, o que requer que se utilize um mecanismo de exclusão mútua para estes acessos;

d) Ambos os programas têm de terminar de forma ordeira quando recebem o sinal ***SIGINT***;

   i. Para resolver o "problema" do bloqueio das *threads* na leitura dos *named pipes* explore uma solução com sinais que tire partido da utilização das funções ***sigaction***, ***pthread_kill*** e ***pthread_exit***.

Comece por planear e discutir a arquitetura do sistema com o docente (e, eventualmente, com os seus colegas). De seguida faça a implementação dos programas *servidor* e *cliente*.

**Objetivos do exercício - Utilizar os conceitos de *named pipes*, *threads*, *mutexes* e sinais em simultâneo.**
- Definir uma arquitetura capaz de dar resposta aos requisitos de um sistema mais complexo.
- Consolidar o uso de *named pipes*, *threads*, *mutexes* e sinais em simultâneo.

**5.** Pretende-se um programa que implemente uma espécie de calculadora multi-threaded. Trata-se de um único processo no qual existirão várias threads. O objectivo é o de sortear uma expressão aritmética e depois apresentar o seu resultado. O funcionamento é o descrito a seguir.

- Algumas das *threads* constroem uma expressão aritmética envolvendo 2 operandos (números) e um operador (+ - * /). Os números e o operador são sorteados. Os números devem ser positivos maior que zero. Estas são as "*threads sorteadoras*". Existem 3 *threads* sorteadoras e executam todas a mesma função.

- Uma outra *thread* vai detectando a construção da expressão e quando estiver completa apresenta o seu resultado. Esta é a "*thread calculadora*". Existe uma única *thread* calculadora.

- Existe uma matriz de 3 inteiros, acessível a todas as *threads* (sem, no entanto, ser global) que corresponde à definição da expressão. O primeiro e o terceiro elementos têm os números (operandos) sorteados que integram a expressão. O segundo elemento tem o operador (sorteado) a usar, segundo o código: 1 -> soma, 2 -> subtracção, 3 -> multiplicação, 4 -> divisão.

- A matriz com a expressão é preenchida sequencialmente. Cada *thread* sorteadora analisa a matriz, vê qual o próximo elemento a definir e coloca esse elemento na matriz sorteando o seu valor (operando/operador). Ao fazer isso avisa a *thread* calculadora acerca da existência de um novo elemento da expressão.

- Cada *thread* sorteadora está em ciclo a sortear elementos da expressão. Antes de analisar em que ponto vai a matriz e sortear um elemento executa uma pausa de duração aleatória entre 1 e 3 segundos. A *thread* só preenche um elemento por expressão: após um elemento sorteado e colocado na matriz, a *thread* aguarda que a expressão seja completada e só depois repete.

- A *thread* calculadora aguarda por cada novo elemento na matriz, apresentando-o no ecrã assim que é definido (sorteado). Quando detecta que a expressão está completa, coloca o resultado no ecrã e avisa as restantes que podem sortear novos elementos para uma nova expressão. A *thread* calculadora está em ciclo, repetindo este comportamento.

Dependendo das instruções dadas pelo seu docente de laboratório, faça uma das seguintes alíneas, em alternativa uma à outra.

a) Escreva o código do programa, tendo atenção às questões de sincronização. Inclua uma forma das *threads* poderem ser paradas mediante a indicação de uma palavra introduzida pelo utilizador.

Em alternativa (se puder escolher, escolha a primeira)

a) Veja o código fornecido pelo docente/no-moodle. Tem tudo excepto mecanismos de sincronização. Acrescente-os. Corrija a forma de terminação das *threads*.

**Objetivos do exercício - Utilizar os conceitos de *threads*, *mutexes* e variáveis condicionais.**
- Rever e consolidar a gestão de *threads*.
- Rever e consolidar o uso de *mutexes*.
- Conhecer e praticar o uso de variáveis condicionais.