Processos, Tarefas e Eventos

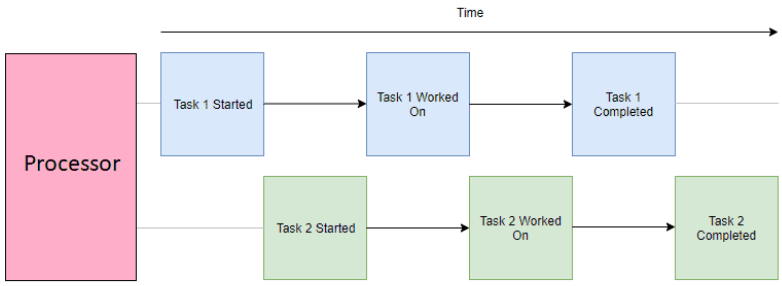
1. Processos
    1.1. Multiprogramação
    1.2. Processo
    1.3. Máquina Virtual
    1.4. Gestor de Processos
2. Objeto Processo
    2.1. **Processo Pai e Processo Filho**
    2.2. **Segurança**
    2.3. **Recursos do Sistema**
    2.4. **Operações sobre Processos**
3. Modelo Multitarefa
    3.1. Threads
    3.1.1. Operações Sobre Threads
    3.2. Programação num Ambiente Multitarefa
    3.3. Pseudo-Threads ou User-Threads
    3.4. Pseudo-Threads vs. Real Threads
4. Modelo de Eventos
    4.1. Eventos

1. Processos

1.1. Multiprogramação

Aqui considera-se a situação onde apenas se tem 1 CPU e 1 core.

- Execução, em paralelo, de múltiplos programas, na mesma máquina;
- Cada instância de um programa em execução é um **Processo;**
- O paralelismo não é real:
    - **pseudo paralelismo/concorrência** é o termo utilizado para designar sistemas multiprogramados que se executam num computador com 1 processador.



Pseudo Concorrência

1.2. Processo

> ❗ Um **processo** é uma instância de um programa que é executado por um (ou mais) *threads* que se executa continuamente desde a sua criação exceto quando explicitamente bloqueado/findado. É um objeto do SO que suporta a execução dos programas, contendo informações sobre, por exemplo, o código programa que está a ser executado e o seu estado;

- Pode ser considerado uma máquina virtual que executa um programa.

1.3. Máquina Virtual

- Um processo tem:
  - Espaço de endereçamento;
  - Reportório de instruções;
  - Contexto de execução.
- A máquima virtual suporta a execução de um programa:
  - Estende a máquina física:
    - disponibiliza funções do sistema.
  - restringe a execução do mesmo:
    - confina-a ao espaço de endereçamento associado;
    - impede a execução de certas operações.

1.4. Gestor de Processos

- Está na base da multiprogramação;
- Responsável por:
  - Criar e eliminar processos;
  - Comutar o processador entre processos;
  - Escalonar a execução dos processos;
  - Garantir o isolamento do processo.

---

2. Objeto Processo

- Atributos:
  - ID
    - Identificador único (número inteiro) atribuído pelo SO;
    - Designado por **PID**;
      - Permite referenciar o processo nas operações que sobre ele atuam.
  - Programa
    - Cada processo executa um programa carregado em memória a partir de um **ficheiro executável**;
  - Espaço de Endereçamento
    - Privado a um processo;
    - **Isola-o em relação a outros processos**;
    - Contem:
      - Código do programa;
      - Zona para dados;
      - Zona para *stack.*
    - Mantém uma zona de memória para guardar **informação de gestão** que não pode ser acedida em *modo user.*
  - Prioridade
    - Tida em conta no **escalonamento**;
    - Pode ser **fixa** ou **dinâmica.**

    - Processo Pai
    - Canais de I/O, Ficheiro
    - Quotas de utilização de recursos
    - Contexto de Segurança
- Operações:
    - Criar
    - Eliminar
    - Esperar pela terminação do subprocesso

2.1. Processo Pai e Processo Filho

- Um processo é criado por outro processo;
- O processo criador é o **processo pai**;
- O processo criado é o **processo filho**/**subprocesso**;
- A estrutura hierárquica facilita a criação dos processos:
    - Ambiente pode ser herdado;
    - Gestão de quotas de utilização dos recursos pode ser global.
- Tratamento da relação hierárquica:
    - Fim do processo pai elimina todos os processo filhos;
    - Fim do processo pai não altera estado dos processo filhos.

2.2. Segurança

- Cada processo está associado a um *user*;
- Os *users* são representados por um **UID** — User Identifier;
- O *user* pode pertencer a um grupo/s;
- O grupo é também representado por um **UID**;

2.3. Recursos do Sistema

- Um processo tem necessidade de executar certos recursos durante a sua execução:
    - Canais de I/O;

    - Ficheiros;
    - Mecanismos de sincronização;
    - Mecanismos de comunicação;
- O SO é responsável por:
    - Mecanismos de segurança de gestão dos recursos;
    - Manter coerentemente o estado dos recursos;
    - Garantir a contabilização da sua utilização.

2.4. Operações sobre Processos

- Criação:
    - Processo pai cria processo filho.
- Eliminação de Processo:
    - O processo termina por si mesmo;
    - O processo pai termina o processo filho;
    - Um processo com privilégios especiais termina a execução de um outro processo.

---

3. Modelo Multitarefa

- Um processo é um **ambiente de execução** e um **fluxo de execução**:
    - Memória;
    - Segurança;
    - Recursos;
- Necessidade de os separar:
    - Surgem as **threads**;

3.1. Threads

- Múltiplos fluxos de execução no mesmo processo;
- Fluxos de execução concorrentes que executam o mesmo programa no espaço de endereçamento do processo;
- Vantagens:

    - As *Threads* partilham o mesmo espaço de endereçamento;
    - Código pertence ao processo englobante;
    - Comutação mais simples;
    - Validações de segurança mais simples porque as tarefas se executam dentro do mesmo processo.
- **Contexto**:
    - *Program counter*;
    - *Stack*;
    - Registos do CPU;
- As *threads* pertencentes a um procesos constam de uma **tabela de tarefas**;
- As tarefas são comutadas por uma **função de despacho** de forma semelhante aos processos;

3.1.1. Operações Sobre Threads

- `thread_id = create_thread(procedure)`
- `delete_thread(thread_id)`
- `wait_thread(thread_id)`

Exemplo: Interface POSIX:

- `err = pthread_create(&thread_id, attr, function, arg)`
    - `&tid`: apontador para o ID da *thread;*
    - `attr`: atributos da tarefa;
    - `function`: função a executar;
    - `arg`: parâmetros passados à função.
- `pthread_exit`
- `pthread_join(pthread_t thread)`

3.2. Programação num Ambiente Multitarefa

- Tarefas partilham o mesmo espaço de enderaçamento, tendo acesso às mesmas variáveis globais;

- As modificações das variáveis globais têm de ser efetuadas com precauções para evitar erros de sincronização;

3.3. Pseudo-Threads ou User-Threads

- Os fluxos de atividade independentes já existiam nas linguagens de programação;
- As *threads* eram implementadas numa biblioteca de funções no *user space* (***pseudo-threads***);
    - No *user mode*, o kernel não pode fazer comutação;
    - Comutação feita pelo programador - `thread_yield`
        - Nos SO mais recentes, as tarefas são objetos do sistemas e comutadas pelo kernel (**real threads** ou **kernel threads**);

3.4. Pseudo-Threads vs. Real Threads

- Pseudo-Threads:
    - Vantagens:
        - Podem ser implementadas em todos os SO porque são uma biblioteca;
        - Não envolvem *syscalls*:
            - Muito mais rápidas.
    - Desvantagens:
        - Obrigam a comutação explícita;
        - O programador tem de detetar e programa a comutação, ao passo que nas Real Threads o *kernel* faz isso automaticamente quando deteta uma situação de bloqueio.

---

4. Modelo de Eventos

- Situações esporádicas que têm de ser tratadas:
    - Exceções;
    - Provocadas pelo utilizador (e.g. `CTRL+C`);
    - I/O;
    - Sinais enviados de outros processos.

- Numa programação sequencial, é difícil tratar dessas situações porque obrigariam a um teste sistemático:
  - Penalizante para o desempenho.

4.1. Eventos

> ❗ **Rotinas assíncronas para o tratamento de acontecimentos assíncronos e exceções;**

- Alguns SO oferecem maneiras de associar um evento a um procedimento;
- Quando o evento é detetado, o programa é redirecionado para sub-rotina que efetua o tratamento;
  - A execução continua na linha posterior após o término.