Trabalho de Projeto

2ª Fase

1 Objetivo

Continuar a realização do trabalho de programação em Python, nesta fase com a comunicação e a sincronização entre processos, a manipulação de ficheiros e o tratamento de sinais e alarmes.

2 Introdução

Este trabalho pretende estender o comando `pwordcount` da 1ª Fase com algumas funcionalidades adicionais. Pretende-se que os alunos concretizem o comando `pwordcount` descrito de seguida:

NOME

`pwordcount` – calcula palavras em um ou mais ficheiros passados como argumento ao programa, a escrever na saída a respetiva contagem pedida pelo utilizador com o nível de paralelização também escolhido pelo mesmo.

SINOPSE

```
./pwordcount [-m t|u|o] [-p n] [-i s] [-l f] ficheiros…
```

DESCRIÇÃO

A descrição dos argumentos `-m`, `-p` e `ficheiros` são idênticas ao texto apresentado no enunciado da 1ª Fase. Quanto às novas opções:

`-i s`: opção que define o intervalo de tempo (a cada `s` segundos) em que serão obtidos os resultados parciais da execução. Caso não seja passado o argumento `-i s`, o programa deverá emitir os resultados parciais, por defeito, a cada `3` segundos.

`-l f`: opção que define o destino da emissão dos resultados parciais, onde `f` é o nome de um ficheiro de *log* (*e.g.*, `count.log`). Se o argumento `-l f` não for passado ao programa, este deverá imprimir os resultados parciais na saída padrão (*stdout*).

2.1 Comunicação Entre os Processos

Na 1ª Fase do projeto, os processos filhos não comunicavam entre si nem com o processo pai. Desta forma, os resultados intermédios impressos por cada processo filho não levavam em conta o estado atual dos outros processos e o processo pai não conseguia calcular os resultados agregados dos processos filhos.

Nesta 2ª Fase do projeto, os processos comunicarão entre si através dos modelos de comunicação estudados em SO (*i.e.*, comunicação por memória partilhada e por troca de mensagens) consoante o modo de execução do `pwordcount` definido pelo utilizador:

- Os processos, na opção `-m t` (*i.e.*, contagem total de palavras):
  - Vão partilhar um contador inteiro em memória partilhada do tipo `Value` do módulo `multiprocessing`.
    - Este contador servirá para os processos filhos acumularem o número total de palavras contadas no(s) ficheiro(s) de *input* (incluindo as palavras repetidas).
- Os processos, na opção `-m u` (*i.e.*, contagem de palavras únicas/diferentes):
  - Vão comunicar entre si por troca de mensagens através de uma fila do tipo `Queue` do módulo `multiprocessing`.
    - Cada processo filho colocará nesta fila o conjunto (*set*) de palavras únicas encontradas num (ou mais) ficheiro(s) ou bloco(s) de dados de um ficheiro que lhe tenha(m) sido atribuído(s).
    - O processo pai recolherá desta fila todos os resultados intermédios dos processos filhos e calculará o resultado agregado (*i.e.*, o tamanho do conjunto unificado de palavras únicas---sem duplicados---encontradas por todos os processos filhos).
  - Também vão partilhar um *array* de contadores inteiros do tipo `Array` do módulo `multiprocessing` com o tamanho `n`, o qual equivale ao número de processos filhos que estarão a trabalhar no problema em questão.
    - Cada um destes `n` contadores será utilizado por um processo filho diferente para acumular o número de palavras únicas encontradas por si até ao momento.

        - O processo pai utilizará estes contadores para emitir os resultados parciais ao longo da execução (ver a Secção 2.4). No final, espera-se que a soma destes contadores seja igual à soma dos tamanhos dos conjuntos enviados por mensagem através da `Queue` para o processo pai poder calcular o resultado agregado.
- Os processos, na opção `-m o` (*i.e.*, contagem de ocorrências de palavras):
    - Vão se comunicar por troca de mensagens através de uma fila do tipo `Queue` do módulo `multiprocessing`.
        - Cada processo filho colocará nesta fila a contagem de ocorrências de cada palavra num determinado ficheiro ou bloco de dados de um ficheiro que lhe tenha sido atribuído.
        - O processo pai recolherá todos os resultados intermédios e calculará o resultado agregado (*i.e.*, a soma das ocorrências de cada palavra nos vários ficheiros ou blocos de um mesmo ficheiro).
    - Também vão partilhar um *array* de contadores inteiros do tipo `Array` do módulo `multiprocessing` com o tamanho `n`, o qual equivale ao número de processos filhos que estarão a trabalhar no problema em questão.
        - Cada um destes `n` contadores será utilizado por um processo filho diferente para acumular o número de palavras únicas encontradas por si até ao momento.
        - O processo pai utilizará estes contadores para emitir os resultados parciais ao longo da execução (ver a Secção 2.4). No final, espera-se que a soma destes contadores seja igual à soma dos tamanhos dos conjuntos enviados por mensagem através da `Queue` para o processo pai poder calcular o resultado agregado.

2.2 Sincronização entre os Processos

Sempre que vários processos estiverem a aceder a zonas de memória partilhada (*e.g.*, `Value` e `Array` do módulo `multiprocessing`), estes devem sincronizar antes de fazer qualquer escrita ou leitura nas mesmas. Esta sincronização evitará que as operações concorrentes entrem em conflito e gerem resultados inesperados na execução do `pwordcount`. Desta forma, os grupos devem utilizar os mecanismos de sincronização estudados em SO (e.g., *mutex* e semáforos dos vários tipos) para garantir que os acessos (*i.e.*, as escritas e leituras) nas zonas de memória partilhada sejam corretamente sincronizadas.

Note que a fila de mensagens `Queue` do módulo `multiprocessing` permite a comunicação entre processos através da troca de mensagens e já possui mecanismos transparentes de sincronização, de modo que os processos não precisam sincronizar explicitamente entre si para escrever e ler mensagens neste tipo de serviço do SO.

2.3 Tratamento do Sinal Ctrl+C (SIGINT)

Caso o **processo pai** receba o sinal SIGINT (*i.e.*, CTRL+C), o processamento deve terminar corretamente. Isto é, os processos filhos devem concluir o processamento nos ficheiros ou blocos de dados correntes e terminar de seguida. O processo pai não deverá atribuir novos ficheiros ou blocos de dados aos processos filhos após receber este sinal. Após os processos filhos concluírem o processamento já iniciado, o processo pai escreve para a saída padrão (*stdout*) a contagem de palavras do modo solicitado pelo utilizador, considerando apenas os ficheiros ou blocos de dados que foram processados pelos processos filhos até ao momento.

Note que os **processos filhos** devem ignorar o sinal SIGINT pois eles não possuem o contexto completo para terminar corretamente a execução do programa. Além disso, consoante a implementação do vosso grupo pode ser necessário criar mecanismos adicionais de comunicação e sincronização entre os processos para o processo pai poder coordenar com os processos filhos a finalização do processamento.

2.4 Resultados Parciais Temporizados

A cada `s` segundos (`3` por defeito, ou o valor `s` passado no argumento `-i s`), o processo pai deve escrever para *stdout* ou para o ficheiro de *log* (consoante o argumento `-l f` ter sido passado ao programa) o estado da contagem de palavras até ao momento, com a seguinte informação separada por um espaço:

(1) *Timestamp* do momento atual a que se refere a entrada no log, num formato sem espaços à vossa escolha;
(2) Tempo decorrido (em microssegundos) desde o início da execução do programa.
(3) Número acumulado de palavras contadas até ao momento por todos os processos filhos;
(4) Número de ficheiros (ou blocos de dados) completamente processados;
(5) Número de ficheiros (ou blocos de dados) ainda em falta por processar;

Note que a função de tratamento de alarmes temporais pode ter de adquirir os *locks* apropriados para ter acesso às variáveis necessárias para imprimir as informações acima mencionadas sem que haja conflitos do tipo leitura-escrita (*read-write conflicts*). Além disso, é preciso refletir sobre como fazer a emissão dos resultados parciais exatamente a cada `s` segundos de forma consistente (ou seja, não devem utilizar funções como `sleep(s)`).
Segue um exemplo simples com algumas entradas de resultados parciais emitidos a cada `3` segundos num ficheiro de *log*:

```
22/11/2023-16:29:38 3000137 3728 2 4
22/11/2023-16:29:41 6000245 9238 4 2
22/11/2023-16:29:44 9000526 21752 6 0
```

2.5 Resultados Agregados

Nesta 2ª Fase, os processos filhos não devem imprimir os seus resultados parciais na saída padrão (*stdout*) ao terminarem os trabalhos que lhes forem atribuídos. Apenas o processo pai fará as impressões dos resultados agregados no *stdout* para finalizar a execução do programa.

3 Ficheiros Iniciais de Teste

Juntamente com o enunciado da 1ª Fase do projeto, foram disponibilizados um ficheiro ZIP (`SO-TI-XX.zip`) com a estrutura inicial do projeto e alguns ficheiros de texto que servirão para os alunos testarem a solução do comando `pwordcount`. Os alunos terão de descarregar estes ficheiros de teste para a sua máquina. Não os deverão abrir no Moodle, pois alguns dos ficheiros podem ser bastante grandes (*e.g.*, centenas de megabytes).

4 Entrega

A entrega do trabalho é realizada de forma semelhante à 1ª Fase, da seguinte forma:

- O grupo coloca os ficheiros `pwordcount` e `pwordcount.py` do projeto numa diretoria cujo nome deve seguir exatamente o padrão `SO-TI-XX` (e.g., `SO-TI-01` ou `SO-TI-23`). Juntamente com os dois ficheiros, devem incluir um ficheiro de texto `README.txt` (não é `pdf`, `rtf`, `odt` nem `docx`) que deve conter:
    1. A identificação dos elementos do grupo;
    2. Exemplos de chamadas do comando `pwordcount`;
    3. As limitações da implementação;
    4. A abordagem usada para a divisão dos ficheiros pelos processos;
    5. Outras informações que acharem pertinente sobre a implementação do projeto (*e.g.*, métodos de comunicação e sincronização utilizados, formatos dos *timestamps*).
- A diretoria será incluída num ficheiro ZIP cujo nome deve seguir exatamente o padrão grupo `SO-TI-XX.zip`, novamente onde `XX` corresponde ao número do vosso grupo. Este ficheiro deverá ser submetido no Moodle (apenas uma submissão por grupo).

De notar que o ficheiro ZIP entregue deve conter apenas a diretoria com os ficheiros (`pwordcount`, `pwordcount.py` e `README.txt`), pois qualquer outro ficheiro será ignorado.

Se não se verificar algum destes requisitos, o trabalho é considerado não entregue.
Não serão aceites trabalhos entregues por email nem por qualquer outro meio não definido nesta secção.

5 Prazo de Entrega

O trabalho deve ser entregue até o dia **10 de dezembro de 2023 (domingo) às 23:59h**.

6 Avaliação dos Trabalhos

A avaliação do trabalho será realizada:

1. Pelos alunos, através do preenchimento do formulário de contribuição de cada aluno no desenvolvimento do projeto. O formulário será disponibilizado no Moodle e preenchido após a entrega do projeto.
2. Pelo corpo docente, sobre dois conjuntos de ficheiros de texto: os ficheiros de teste disponibilizados aos alunos e outros somente usados pelos docentes.

Para além dos testes a efetuar, os seguintes parâmetros serão avaliados: funcionalidade, estrutura, desempenho, algoritmia, comentários, clareza do código, validação dos parâmetros de entrada e tratamento de erros.

7 Plágios

Não é permitido aos alunos partilharem códigos com soluções, ainda que parciais, de nenhuma parte do projeto com outros alunos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso será reportado aos órgãos responsáveis em Ciências@ULisboa.

Chamamos a atenção para o facto das plataformas generativas baseadas em Inteligência Artificial (*e.g.*, o *ChatGPT* e o *GitHub Co-Pilot*) gerarem um número limitado de soluções diferentes para o mesmo problema, as quais podem ainda incluir padrões característicos deste tipo de ferramenta e que podem ser detetáveis pelos verificadores de plágio. Desta forma, recomendamos fortemente que os alunos não submetam trechos de código gerados por este tipo de ferramenta a fim de evitar riscos desnecessários.

Por fim, é responsabilidade de cada aluno garantir que a sua *home*, as suas diretorias e os seus ficheiros de código estão protegidos contra a leitura de outras pessoas (que não o utilizador dono dos mesmos). Por exemplo, se os ficheiros estiverem gravados na sua área de aluno nos servidores de Ciências@ULisboa, então todos os itens mencionados anteriormente devem ter as permissões de acesso `700`. Se os ficheiros estiverem no *GitHub*, garantam que o conteúdo do vosso repositório não esteja visível publicamente.