Sistemas Operativos

Trabalho Prático

Orquestrador de Tarefas

Grupo de Sistemas Distribuídos
Universidade do Minho

4 de Março de 2024

Informações gerais

- Cada grupo deve ser constituído por até 3 elementos;
- O trabalho deve ser entregue até às 23:59 de 7 de Maio;
- Deve ser entregue o código fonte e um relatório de até 10 páginas de conteúdo (A4, 11pt) no formato PDF (não são contabilizadas capas e anexos para o limite de 10 páginas), justificando a solução, nomeadamente no que diz respeito à arquitetura de processos e da escolha e uso concreto dos mecanismos de comunicação;
- O trabalho deve ser realizado tendo por base o sistema operativo Linux como ambiente de desenvolvimento e de execução;
- O trabalho deve ser submetido num arquivo Zip com nome `grupo-xx.zip`, em que `xx` deve ser substituído pelo número do grupo de trabalho (*p.ex.*, `grupo-01.zip`);
- A apresentação do trabalho ocorrerá em data a anunciar, previsivelmente entre os dias 27 e 29 de Maio;
- O trabalho representa 50% da classificação final.

Resumo

Pretende-se implementar um serviço de orquestração (*i.e.*, execução e escalonamento) de tarefas num computador.

Os utilizadores devem usar um programa cliente para submeter ao servidor a intenção de executar uma tarefa, dando uma indicação da duração em milissegundos que necessitam para a mesma, e qual a tarefa (*i.e.*, programa ou conjunto/pipeline de programas) a executar.

Cada tarefa tem associado um identificador único que deve ser transmitido ao cliente mal o servidor recebe a mesma. O servidor é responsável por escalonar e executar as tarefas dos utilizadores. Informação produzida pelas tarefas para o *standard output* ou *standard error* devem ser redirecionadas pelo servidor para um ficheiro cujo nome corresponde ao identificador da tarefa.

Através do programa cliente, os utilizadores podem ainda consultar o servidor para saberem quais as tarefas em execução, em espera para execução, e terminadas.

Servidor e Cliente (12 valores)

Deverá ser desenvolvido um cliente (programa *client*) que ofereça uma interface ao utilizador via linha de comandos. Deverá ser também desenvolvido um servidor (programa *orchestrator*), com o qual o cliente irá interagir. O servidor deve manter em memória e em ficheiros a informação relevante para suportar as funcionalidades descritas neste enunciado.

O *standard output* deverá ser usado pelo cliente para apresentar as respostas necessárias ao utilizador, e pelo servidor apenas para apresentar informação de *debug* que julgue necessária.

Tanto o cliente como o servidor deverão ser escritos em C e comunicar via *pipes com nome*. Na realização deste projeto não deverão ser usadas funções da biblioteca de C para operações sobre ficheiros, salvo para impressão no *standard output*. Da mesma forma não se poderá recorrer à execução de programas direta ou indiretamente através do interpretador de comandos (*p.ex.*, *bash* ou `system()`).

O serviço deverá suportar as seguintes funcionalidades básicas:

Execução de tarefas do utilizador

- O cliente é responsável por receber, através da opção `execute`, a indicação do tempo que o utilizador acha que é necessário para executar a sua tarefa, e qual a tarefa a executar.

  Assim, para executar uma tarefa, usando a opção `execute time -u "prog-a [args]"`, o utilizador passa ao programa cliente os seguintes argumentos de linha de comando:

  1. `time`: indicação do tempo estimado, em milissegundos, para a execução da tarefa;
  2. `prog-a`: o nome/caminho do programa a executar (*p.ex.*, *cat*, *grep*, *wc*);
  3. `[args]`: os argumentos do programa, caso existam (*p.ex.*, `arg-1 (...) arg-n`).

  Nota 1: Assuma que o número de argumentos é variável.
  Nota 2: Assuma que o tamanho total dos argumentos passados à opção `execute` não excede os 300 bytes.

- O cliente deve comunicar ao servidor o pedido de execução, recebendo do mesmo um identificador único da tarefa, o qual deve ser comunicado ao utilizador. Após o mesmo, o programa cliente termina e o utilizador deve poder efetuar outras operações, mesmo que as tarefas que submeteu ainda não tenham sido terminadas. A escolha do identificador de tarefa fica ao critério de cada grupo.
- O servidor deve escalonar as tarefas dos múltiplos clientes de acordo com políticas que permitam reduzir o seu tempo médio de espera. A política a ser utilizada (*p.ex.*, FCFS, SJF) fica ao critério de cada grupo, e deve tirar proveito da indicação da duração de cada tarefa fornecida pelos utilizadores.
- O servidor deve executar os programas das tarefas, tal como executariam num ambiente Linux normal. O *standard output* e *standard error* destes programas devem ser redirecionados para um ficheiro com o identificador da tarefa.
- Após a terminação de uma tarefa, o servidor deve registar em ficheiro, de forma persistente, o identificador da tarefa e o seu tempo de execução (*i.e.*, o tempo total desde que o servidor recebe o pedido do cliente até o término do programa). Todas as tarefas terminadas devem ser registadas no mesmo ficheiro.

  Nota 1: O tempo fornecido pelo utilizador para a duração de uma tarefa é apenas indicativo. O servidor deve registar o tempo real de execução, que pode ser inferior ou superior ao indicado.

  Nota 2: Assim que uma tarefa termine, o servidor deve executar a próxima, não precisando de ficar à espera que o tempo indicado pelo utilizador seja cumprido.

  Nota 3: A função `gettimeofday`[1] pode ser utilizada para calcular o tempo de execução referido anteriormente.

Consulta de tarefas em execução

- O cliente deve também permitir aos utilizadores, via linha de comando, realizar a seguinte interrogação:

  1. Através da opção `status` devem ser listadas as tarefas em execução, as tarefas em espera para executar, bem como as tarefas terminadas.

  Nota 1: O processamento para dar resposta à interrogação anterior deve ser realizado pelo servidor. O cliente apenas envia o pedido e recebe a resposta, apresentado-a ao utilizador.

  Nota 2: Para as tarefas em execução ou à espera para executar devem ser listados os respetivos identificadores e programas. Para as tarefas terminadas devem ser listados os seus identificadores, programas, e tempo de execução.

- O servidor não deve bloquear clientes de realizarem pedidos de `execute` devido a um `status` a ser atendido.

[1] `https://man7.org/linux/man-pages/man2/gettimeofday.2.html`

Encadeamento de programas, multi-processamento e avaliação (8 valores)

A avaliação do trabalho prático será valorizada pelas seguintes funcionalidades:

Execução encadeada de programas

- Através da opção `execute time -p "prog-a [args] | prog-b [args] | prog-c [args]"`, o cliente deve suportar a execução encadeada de programas do utilizador (*pipelines*) tal como acontece na linha de comandos quando usado o operador | . Por exemplo:

```
cat fich1 | grep "palavra" | wc -l
```

- O utilizador deve ser notificado, tal como na execução de um programa único por tarefa, do identificador da tarefa que contém a *pipeline* de programas. A escolha do identificador fica ao critério de cada grupo.

Processamento de várias tarefas em paralelo

- Considere que o servidor consegue processar `N` tarefas em simultâneo (parâmetro definido no arranque do servidor). Utilize esta funcionalidade para otimizar o algoritmo de escalonamento e a velocidade de processamento de tarefas. O limite de paralelização (`N`) deve ser considerado ao nível da tarefa, independentemente desta ser constituída por um ou mais programas. Por exemplo, se `N` tiver o valor 4, então o servidor consegue processar, em paralelo, 4 tarefas de cada vez, independentemente do número de processos que estas tarefas necessitam para executar.

Avaliação de políticas de escalonamento

- Desenvolva testes que permitam perceber a eficiência da sua política de escalonamento. Considere comparar com outras políticas implementadas por si. Os testes devem considerar diferentes execuções de utilizadores (combinações de tarefas com programas e durações distintas) de forma a perceber que políticas de escalonamento permitem melhorar a experiência dos utilizadores, por exemplo, ao reduzir o tempo médio de execução das suas tarefas;
- Teste diferentes configurações de paralelização do servidor. Compreenda de que forma estas configurações alteram a experiência (*p.ex.*, tempo médio de execução das suas tarefas) para os utilizadores.

  Nota 1: O relatório deve incluir uma reflexão sobre os resultados dos testes efetuados.

  Nota 2: Tire partido de / crie *scripts* (*p.ex.*, em *bash*) para ajudar na automatização dos testes.

Interface e Modo de Utilização

O serviço deverá ser usado do seguinte modo:

- Executar o servidor:

  ```
  $ ./orchestrator output_folder parallel-tasks sched-policy
  ```

  Argumentos:

  1. `output-folder`: pasta onde são guardados os ficheiros com o output de tarefas executadas.
  2. `parallel-tasks`: número de tarefas que podem ser executadas em paralelo.
  3. `sched-policy`: identificador da política de escalonamento, caso o servidor suporte várias políticas.

- Submeter pedidos de execução de uma tarefa, conforme o exemplo abaixo:

  ```
  $ ./client execute 100 -u "prog-a arg-1 (...) arg-n"
  TASK 1 Received
  ```

  Nota: A flag `-u` indica a execução de um programa individual. O valor 100 indica que a duração da tarefa prevista é 100 milissegundos.

- Submeter pedidos de execução de uma *pipeline* de programas, conforme o exemplo abaixo:

  ```
  $ ./client execute 3000 -p "prog-a arg-1 (...) arg-n | prog-b arg-1 (...) arg-n | prog-c arg-1 (...) arg-n"
  TASK 30 Received
  ```

  Nota: A flag `-p` indica a execução de uma *pipeline* de programas. O valor 3000 indica que a duração da tarefa prevista é 3000 milissegundos.

- Interrogar programas em execução:

  ```
  $ ./client status
  Executing
  12 prog-c

  Scheduled
  8 prog-h
  13 prog-d | prog-e | prog z

  Completed
  1 prog-a 100 ms
  2 prog-c | prog-b 500 ms
  (...)
  ```

  Nota: As tarefas terminadas devem incluir o seu tempo real de execução (*p.ex.*, no exemplo anterior, a tarefa 1 demorou 100 ms e a tarefa 2 demorou 500 ms).

Makefile

Tenha em conta que a Makefile que se apresenta deverá ser usada como ponto de partida mas poderá ter necessidade de a adaptar de modo a satisfazer, por exemplo, outras dependências do seu código-fonte. Em todo o caso, deverá manter sempre os objetivos (*targets*) especificados: `all`, `orchestrator`, `client`, e `clean`. Não esqueça que por convenção a indentação de uma Makefile é especificada com uma tabulação (*tab*) no início da linha (nunca com espaços em branco).

```
CC = gcc
CFLAGS = -Wall -g -Iinclude
LDFLAGS =

all: folders server client

server: bin/orchestrator

client: bin/client

folders:
	@mkdir -p src include obj bin tmp

bin/orchestrator: obj/orchestrator.o
	$(CC) $(LDFLAGS) $^ -o $@

bin/client: obj/client.o
	$(CC) $(LDFLAGS) $^ -o $@

obj/%.o: src/%.c
	$(CC) $(CFLAGS) -c $< -o $@

clean:
	rm -f obj/* tmp/* bin/*
```