Sistemas Operativos

Trabalho Prático

Rastreamento e Monitorização da Execução de Programas

Grupo de Sistemas Distribuídos
Universidade do Minho

17 de Março de 2023

Informações gerais

- Cada grupo deve ser constituído por até 3 elementos;
- O trabalho deve ser entregue até às 23:59 de 13 de Maio;
- Deve ser entregue o código fonte e um relatório de até 10 páginas (A4, 11pt) no formato PDF (excluindo capas e anexos), justificando a solução, nomeadamente no que diz respeito à arquitectura 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 15 e 19 de Maio;
- O trabalho representa 50% da classificação final.

Resumo

Pretende-se implementar um serviço de monitorização dos programas executados numa máquina.

Os utilizadores devem conseguir executar programas, através do cliente, e obter o seu tempo de execução (*i.e.*, o tempo total desde que o utilizador pede ao cliente para correr o programa até ao término do mesmo).

Um administrador de sistemas deve conseguir consultar, através do servidor, todos os programas que se encontram atualmente em execução, incluíndo o tempo dispendido pelos mesmos. Finalmente, o servidor deve também permitir a consulta de estatísticas sobre programas já terminados (*p.ex.*, o tempo de execução agregado de um certo conjunto de programas).

Programas cliente e servidor

Deverá ser desenvolvido um cliente (programa *tracer*) que ofereça uma interface com o utilizador via linha de comandos. Deverá ser também desenvolvido um servidor (programa *monitor*), 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 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 projecto 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 comandos directa ou indirectamente através do interpretador de comandos (p. ex.: *bash* ou `system()`).

Funcionalidades Básicas (14 valores)

O serviço deverá suportar as seguintes funcionalidades básicas:

Execução de programas do utilizador

- O cliente é responsável por executar programas dos utilizadores (*p.ex.*, *cat*, *grep*, *wc*) através da opção `execute` e comunicar ao servidor o estado dessa execução. O servidor é responsável por armazenar essa informação de execução e disponibilizá-la para consulta.

  Assim, para executar um programa, o utilizador passa ao cliente os seguintes argumentos de linha de comando:

  1. A opção `execute -u`;
  2. O nome do programa a executar;
  3. Os argumentos do programa, caso existam.

  Nota 1: Assuma que o número de argumentos é variável.
  Nota 2: Para o utilizador, a execução do seu programa deve ser idêntica a correr o mesmo via linha de comando (*p.ex.*, informação produzida pelo programa para o *standard output* e para o *standard error*).

- Aquando a execução de um novo programa do utilizador, o cliente deve primeiro informar o servidor do mesmo, enviando a seguinte informação:

  1. O *PID* do processo a executar o programa;
  2. O nome do programa a executar;
  3. O *timestamp* atual, que representa o tempo imediatamente antes do inicio de execução do programa.

  Ainda o cliente deve notificar o utilizador, via *standard output*, do PID do programa que será executado. Após notificar o utilizador e servidor, o cliente deve executar o programa.

  Nota 1: O *PID* serve como identificador único para uma dada execução de um programa, já que os utilizadores podem executar várias vezes programas com o mesmo nome.
  Nota 2: A função *gettimeofday*[1] pode ser utilizada para obter o *timestamp* referido anteriormente.

- Após a terminação de um programa, o cliente deve informar o servidor do mesmo, enviando a seguinte informação:

  1. O *PID* do processo que terminou;
  2. O *timestamp* atual, que representa o tempo imediatamente após o término da execução do programa.

  O cliente deve também notificar o utilizador, via *standard output*, do tempo de execução (em milisegundos) utilizado pelo programa.

Consulta de programas em execução

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

  1. Através da opção `status` devem ser listados, um por linha, os programas em execução no momento. A informação de cada programa deve conter o seu *PID*, nome, e tempo de execução até ao momento (em milisegundos).

  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.

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

Servidor

- O servidor deve suportar, sempre que possível, o processamento concorrente de pedidos, evitando que clientes a realizar pedidos que obriguem a um maior tempo de processamento possam bloquear a interação de outros clientes com o servidor.

Funcionalidades Avançadas (6 valores)

A avaliação do trabalho prático será valorizada pelas seguintes funcionalidades avançadas:

Execução encadeada de programas

- Através da opção `execute -p`, 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
```

Nota 1: O utilizador e servidor devem ser notificados, tal como na execução de um programa único (funcionalidade básica), do inicio e fim da execução da *pipeline* de programas. Esta *pipeline* deve ser identificada por um *PID* (*p.ex.*, do processo que cria a *pipeline*, ou do primeiro processo que faz parte da mesma). A escolha deste indentificador fica ao critério de cada grupo.
Nota 2: Ao ser chamada a opção `status` (funcionalidade básica), o tracer deve imprimir a informação da *pipeline* numa única linha, nomeadamente: *PID*, nome dos programas, e tempo de execução até ao momento (em milisegundos).

Armazenamento de informação sobre programas terminados

- O servidor deve armazenar em ficheiros o estado de programas terminados. Deve existir um ficheiro diferente por programa executado. O nome deste ficheiro deve corresponder ao PID do programa em questão. Ainda, o ficheiro deve conter o nome, e tempo de execução total (em milisegundos) do programa. A estrutura do ficheiro fica ao critério de cada grupo.

  Nota 1: Os ficheiros devem ser guardados dentro de uma pasta, cujo caminho é passado ao servidor como argumento durante o seu arranque.
  Nota 2: Caso o projeto suporte *pipelines* (funcionalidade avançada), a informação relativa a cada *pipeline* executada deve ser guardada num ficheiro independente. O nome do ficheiro deve corresponder ao *PID* atribuído à *pipeline*. O ficheiro deve conter o nome dos programas e o tempo de execução total (em milisegundos).

Consulta de programas terminados

- O cliente deve poder ser usado pelos utilizadores, via linha de comando, para realizar as seguintes interrogações sobre programas já terminados:

  1. Através do comando `stats-time`, deve ser impresso no *standard output* o tempo total (em milisegundos) utilizado por um dado conjunto de programas identificados por uma lista de *PIDs* passada como argumento.
  2. Através do comando `stats-command`, deve ser impresso no *standard output* o número de vezes que foi executado um certo programa, cujo nome é passado como argumento, para um dado conjunto de *PIDs*, também passado como argumento.
  3. Através do comando `stats-uniq`, deve ser impressa no *standard output* a lista de nomes de programas diferentes (únicos), um por linha, contidos na lista de *PIDs* passada como argumento.

  O processamento para dar resposta às interrogações anteriores deve ser realizado pelo servidor. O cliente apenas envia o pedido e recebe a resposta, apresentado-a ao utilizador.

  Nota: O servidor deve otimizar o tempo de pesquisa quando múltiplos *PIDs* são passados como argumento e é necessário realizar operações sobre vários ficheiros.

Interface e Modo de Utilização

O serviço deverá ser usado do seguinte modo:

- submeter pedidos de execução de um programa (funcionalidade básica), conforme o exemplo abaixo:

```
$ ./tracer execute -u "prog-a arg-1 (...) arg-n"
Running PID 3090
(output do programa)
Ended in 400 ms
```

  Nota: A flag *-u* indica a execução de um programa individual.

- submeter pedidos de execução de uma *pipeline* de programas (funcionalidade avançada), conforme o exemplo abaixo:

```
$ ./tracer execute -p "prog-a arg-1 (...) arg-n | prog-b arg-1 (...) arg-n | prog-c arg-1 (...) arg-n"
Running PID 3083
(output do programa)
Ended in 730 ms
```

  Nota: A flag *-p* indica a execução de uma *pipeline* de programas.

- interrogar programas em execução (funcionalidade básica):

```
$ ./tracer status
3033 prog-c 10 ms
3320 prog-h 20 ms
3590 prog-d | prog-e | prog z 100ms
```

  Nota: A listagem de *pipelines* de programas (última linha do exemplo) faz parte da funcionalidade avançada.

- interrogar tempo total de execução de um conjunto de programas terminados (funcionalidade avançada):

```
$ ./tracer stats-time PID-1 PID-2 (...) PID-N
Total execution time is 1000 ms
```

- interrogar número de vezes que um certo programa (*prog-a*) foi executado (funcionalidade avançada):

```
$ ./tracer stats-command prog-a PID-1 PID-2 (...) PID-N
Prog-a was executed 4 times
```

- interrogar lista de programas únicos executados (funcionalidade avançada):

```
$ ./tracer stats-uniq PID-1 PID-2 (...) PID-N
prog-a
prog-c
prog-h
```

- executar o servidor:

```
$ ./monitor PIDS-folder
```

  Nota: A pasta onde são guardados os ficheiros de programas terminados (*PIDS-folder*) faz parte da funcionalidade avançada.

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 objectivos (*targets*) especificados: `all`, `server`, `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).

```
all: folders server client

server: bin/monitor

client: bin/tracer

folders:
@mkdir -p src obj bin tmp

bin/monitor: obj/monitor.o
gcc -g obj/monitor.o -o bin/monitor

obj/monitor.o: src/monitor.c
gcc -Wall -g -c src/monitor.c -o obj/monitor.o

bin/tracer: obj/tracer.o
gcc -g obj/tracer.o -o bin/tracer

-o obj/tracer.o: src/tracer.c
gcc -Wall -g -c src/tracer.c -o obj/tracer.o

clean:
rm -f obj/* tmp/* bin/{tracer,monitor}
```