TRABALHO PRÁTICO Nº 2

Acesso informático aos Quartos de Banho

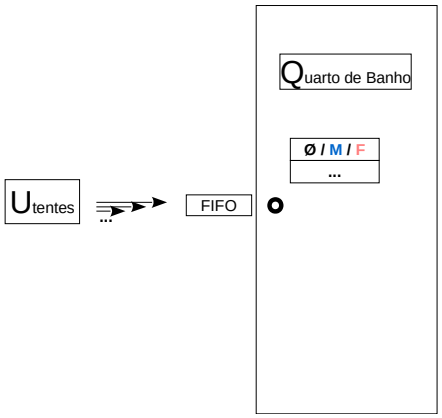
Objetivos

Completando com sucesso todas as fases deste trabalho, os alunos demonstram conhecer e saber utilizar a interface programática de UNIX em C para conseguir:

- criar programas multi*thread*;
- promover a intercomunicação entre processos através de canais com nome (*named pipes* ou FIFOs);
- evitar conflitos entre entidades concorrentes, por via de mecanismos de sincronização.

Descrição geral

Pretende-se obter uma aplicação do tipo cliente-servidor capaz de lidar com situações de conflito no acesso a zonas partilhadas.
A zona partilhada é um Quarto de Banho com vários lugares unisexo, controlado por um processo `Q` ao qual se dirigem pedidos de acesso de utentes. Os pedidos de acesso são enviados por intermédio de um processo *multithread* `U` (cliente), e neles se indica o tempo que o interessado deseja estar num lugar das instalações sanitárias. Os pedidos ficarão numa fila de atendimento até terem vez; nessa altura, o utente respetivo acede a um lugar nas instalações durante o tempo pedido, sob o controlo do servidor `Q`; depois o recurso é libertado para outro utente.
A aplicação deve ser desenvolvida em 2 etapas, de complexidade crescente, a segunda baseando-se no bom funcionamento da primeira. No final, e para avaliação, deverão ser entregues dois pacotes de código (com os programas `U`/`Q`), um para cada etapa.



Requisitos comuns a ambas as etapas

Para cada etapa:

- são necessários 2 programas *multithread*, `U`*n* e `Q`*n* (*n* será o número 1 na etapa 1 e 2 na seguinte)[1];
- ambos funcionam com o máximo de paralelismo possível e evitam situações de encravamento (*deadlock*), de "colisão" e "esperas ativas" (*busy-waiting*) com o auxílio de primitivas de sincronização do tipo das estudadas (e.g. mutexes);
- os pedidos de acesso ao Quarto de Banho são gerados por `U`*n* de forma (pseudo)aleatória (em termos de duração do acesso), mas com intervalos curtos, da ordem dos milissegundos, por forma a exacerbar condições de competição.

Especificação de `U`*n*

Um programa *multithread* `U`*n* :

- é invocado com o comando
  - `U`*n* `<-t nsecs> fifoname`
  - `nsecs` - nº (aproximado) de segundos que o programa deve funcionar
  - `fifoname` – nome do canal público (FIFO) para comunicação com o servidor
- inicialmente, recolhe os argumentos de linha de comando e executa as operações fundamentais para o funcionamento do programa;
- lança continuamente (i.e. com intervalos de alguns milissegundos) *threads*, cada um ficando associado a um pedido;
- cada *thread* gera aleatoriamente parâmetros do pedido pelo qual é responsável (especificamente, a duração do acesso), trata de toda a comunicação com o servidor (os pedidos são enviados ao servidor através do canal

1 Note-se que, em rigor, não são precisos dois programas `U`: não há razão para o cliente `U1` ter de ser diferente de `U2`!

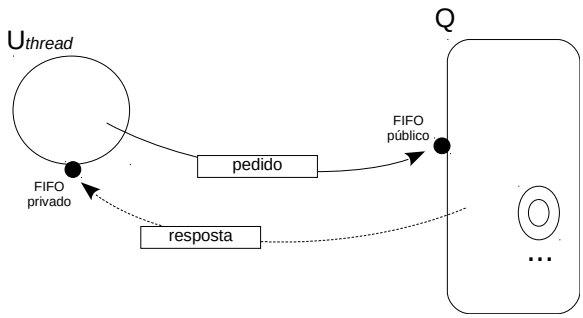
público (`fifoname`); as respostas do servidor são recebidas por um canal privado com nome, criado e posteriormente eliminado pelo cliente) e termina só após o atendimento ter sido completado;
- após o tempo estipulado para o funcionamento do programa (`nsecs`), ou após o encerramento do Quarto de Banho, o ciclo de geração de pedidos é terminado; contudo, os *threads* com pedidos já colocados deverão aguardar resposta: ou se foram atendidos (e o utente está a usar o Quarto de Banho) ou com a indicação de que o Quarto de Banho já está encerrado;
- no final, deve garantir-se que todos os recursos tomados ao sistema são libertados.

Especificação de `Qn`

Um programa *multithread* `Q`*n*:
- invocado com o comando
  - `Q`*n* `<-t nsecs> [-l nplaces] [-n nthreads] fifoname`
    - `nsecs` - nº (aproximado) de segundos que o programa deve funcionar
    - `nplaces` - lotação do Quarto de Banho
    - `nthreads` – nº (máximo) de *threads* a atender pedidos
    - `fifoname` – nome do canal público (FIFO) a criar pelo servidor para atendimento de pedidos
- inicialmente, recolhe os argumentos de linha de comando e executa as operações fundamentais para o funcionamento do programa;
- cada pedido é atendido por um *thread*, que comunica com o *thread* cliente e controla o tempo de utilização de um lugar do Quarto de Banho; se não houver lugares disponíveis, espera que haja e prossegue;
- após o tempo estipulado para o funcionamento do programa (`nsecs`), que corresponde ao encerramento do Quarto de Banho, o canal de comunicação `fifoname` é desativado, mas de forma a que os pedidos pendentes no "*buffer*" do canal sejam notificados do facto e os pedidos em utilização do Quarto de Banho sejam completados;
- no final, todos os recursos tomados ao sistema devem ser libertados.

Comunicação entre `Un` e `Qn`

Para ambos os programas `Q`*n*, `U`*n*:
- as mensagens trocadas são sempre aos pares:
  - cada pedido terá sempre uma resposta
- os canais de comunicação são:
  - um canal público do tipo FIFO, cujo nome é passado como argumento das linhas de comando de `U`*n* e de `Q`*n*, e que recebe os pedidos;
  - canais privados do tipo FIFO, cada um criado pelo *thread* responsável por um pedido e que receberá a correspondente resposta do servidor; os nomes têm a estrutura:
    - "`/tmp/`*`pid.tid`*"
      - *`pid`* - identificador de sistema do processo cliente
      - *`tid`* - identificador no sistema do *thread* cliente



- as mensagens trocadas são estruturalmente idênticas, diferindo no conteúdo de alguns campos; a sua estrutura é:
  - [*`i`*, *`pid`*, *`tid`*, *`dur`*, *`pl`*]
    - *`i`* - o número sequencial do pedido (gerado por `U`*n*)
    - *`pid`* - identificador de sistema do processo (cliente, no caso do pedido; servidor, no caso da resposta)
    - *`tid`* - identificador no sistema do *thread* cliente (cliente, no caso do pedido; servidor, no caso da resposta)
    - *`dur`* - duração, em milissegundos, de utilização (de um lugar) do Quarto de Banho (valor atribuído no pedido e repetido na resposta, se se der a ocupação; se não se der, por motivo de o Quarto de Banho estar em vias de encerrar, o servidor responde aqui com o valor `-1`
    - *`pl`* – nº de lugar que eventualmente lhe será atribuído no Quarto de Banho (no pedido, este campo é preenchido com o valor `-1` e na resposta terá o valor do lugar efetivamente ocupado ou também `-1`, na sequência de insucesso de ocupação, por motivo de encerramento)

Registo das operações

Diversas fases da operação dos programas, intimamente associadas à expedição e receção de mensagens, devem ser registadas na saída padrão (*stdout*), através de linhas de texto, emitidas pelo processo apropriado. Cada linha tem a seguinte estrutura:

- "*inst ; i ; pid ; tid ; dur ; pl ; oper*"
  - *i* , *pid* , *tid* , *dur* , *pl* – têm o mesmo significado que os campos apresentados acima
  - *inst* - valor retornado pela chamada ao sistema `time()`, na altura da produção da linha
  - *oper* – siglas de 5 letras ajustadas às fases da operação que cada processo/*thread* acabou de executar e que variam conforme se trate do cliente ou do servidor:
    - `IWANT` - cliente faz pedido inicial
    - `RECVD` - servidor acusa receção de pedido
    - `ENTER` - servidor diz que aceitou pedido
    - `IAMIN` - cliente acusa a utilização do Quarto de Banho
    - `TIMUP` - servidor diz que terminou o tempo de utilização
    - `2LATE` - servidor rejeita pedido por Quarto de Banho já ter encerrado
    - `CLOSD` - cliente acusa informação de que o Quarto de Banho está fechado
    - `FAILD` - cliente já não consegue receber resposta do servidor
    - `GAVUP` - servidor já não consegue responder a pedido porque FIFO privado do cliente fechou

Ligação das comunicações aos registos

A tabela seguinte sumariza a ligação entre as mensagens trocadas entre os processos/*threads* e os registos efetuados na saída padrão de cada um. O tempo cresce de linha para linha.

CLIENT `U`			**SERVER `Q`**	
Operation log	**Msg type / channel**	**Msg Direction**	**Msg type / channel**	**Operation log**
`IWANT`	request	→	single public FIFO	`RECVD`
	... multiple private FIFOs	←	... reply	
`IAMIN` or `CLOSD` or `FAILD`				`ENTER` ... `TIMUP` or `2LATE` or `GAVUP`

Etapa 1 – Requisitos específicos

Nesta etapa restringe-se a operação do programa `Q`1 à situação em que:

- a lotação e o nº de *threads* do servidor é, à partida, ilimitado (excetuando limitações impostas pelo sistema, como o nº máximo de *threads* por processo); nessa situação, os argumentos opcionais "`-l nplaces`" e "`-n nthreads`" não devem ser indicados na linha de comando de `Q`1.

Etapa 2 - Requisitos específicos

Nesta etapa, o programa `Q`2:

- pode ser invocado com qualquer combinação dos argumentos de linha de comando.

Produto final

A entrega do trabalho para avaliação tem duas datas limite[1]:

- 2.Maio.2020, inclusive, para a Etapa 1;
- 16.Maio.2020, inclusive, para a Etapa 2.

Em cada etapa, a entrega deve ser feita mediante um ficheiro compacto com:

- código-fonte

1 As datas poderão ser alteradas, por motivo de diretiva superior devido à actual situação de pandemia.

- *makefile* preparado para facilitar a geração dos executáveis
- sucinto ficheiro `README` com pormenores de implementação relevantes.

O compacto, identificado com um nome do tipo `TxGy.zip`, sendo `x` o número da turma e `y` o número do grupo, deverá ser submetido via Moodle, numa área a abrir e anunciar oportunamente.

Avaliação

Serão conduzidos testes simples de avaliação do trabalho aos dados produzidos pelo programa numa dada etapa. Tipicamente envolvem:

- a leitura do `README`
- para cada etapa
  - a análise superficial do código
  - a execução dos programas com argumentos de linha de comando razoáveis. Por exemplo:
    - `Q2 -t 30 -l 6 -n 15 door > q2.log 2> q2.err`
    - `U2 -t 20 door > u2.log 2> u2.err`
  - a análise dos registos produzidos, com o auxílio de utilitários, o que permitirá a verificação aproximada da correcção dos programas. Por exemplo:
    - `n2LATE=`grep 2LATE q1.log | wc -l` ; echo $n2LATE`
    - `nCLOSD=`grep CLOSD u1.log | wc -l` ; echo $nCLOSD`
      - Neste caso `n2LATE` deveria ser igual a `nCLOSD`.

A classificação do trabalho é distribuída em 60% para a Etapa 1 e 40% para a Etapa 2. A avaliação da Etapa 2 só será efetuada caso os programas da Etapa 1 funcionem globalmente bem e não exibam falhas graves.

SOPE, 2019/20