MIEIC, 2ºA - Sistemas Operativos – 2020/21 - FEUP

MINI PROJETO Nº 2 (v.2)

Application Server

Objectivos

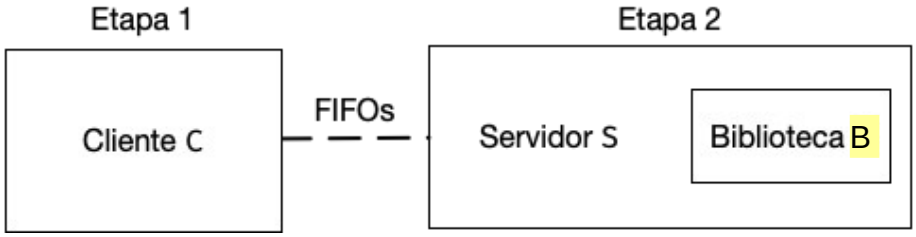
Completando com sucesso todas as fases deste trabalho, os alunos demonstram conhecer e saber utilizar a interface programática de UNIX em C para conseguir:

- criar programas *multithread*;
- 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 pedidos de execução de tarefas de diferente carga. As tarefas são executadas por código numa Biblioteca **B** externa, e um Servidor **S** *multithread* gere os pedidos de execução e devolve os resultados. Os pedidos de tarefas são enviados ao Servidor **S** por um processo Cliente **C** *multithread*; ficarão numa fila de atendimento até terem vez, altura em que a tarefa requisitada é passada por um *thread* Produtor à Biblioteca para ser executada, sendo o resultado colocado num armazém (*buffer*). Este armazém é monitorizado por um *thread* Consumidor que, havendo resultados, os recolhe um a um e os envia de volta ao Cliente.

A aplicação deve ser desenvolvida em 2 etapas, de complexidade crescente, a primeira focando-se no desenvolvimento do Cliente **C**, assumindo a existência de uma implementação do Servidor **S** (que será disponibilizada), e a segunda focando-se no desenvolvimento do Servidor **S**, sendo então disponibilizado um exemplo de um Cliente **C** (os alunos podem usar o cliente que desenvolveram anteriormente, mas devem sempre validar o funcionamento com a versão disponibilizada). Em ambas as etapas a Biblioteca **B** externa atua como caixa-negra e é disponibilizada. No final de cada etapa, o código desenvolvido será submetido para avaliação: código Cliente **C** no final da 1ª etapa e código Servidor **S** no final da 2ª etapa.

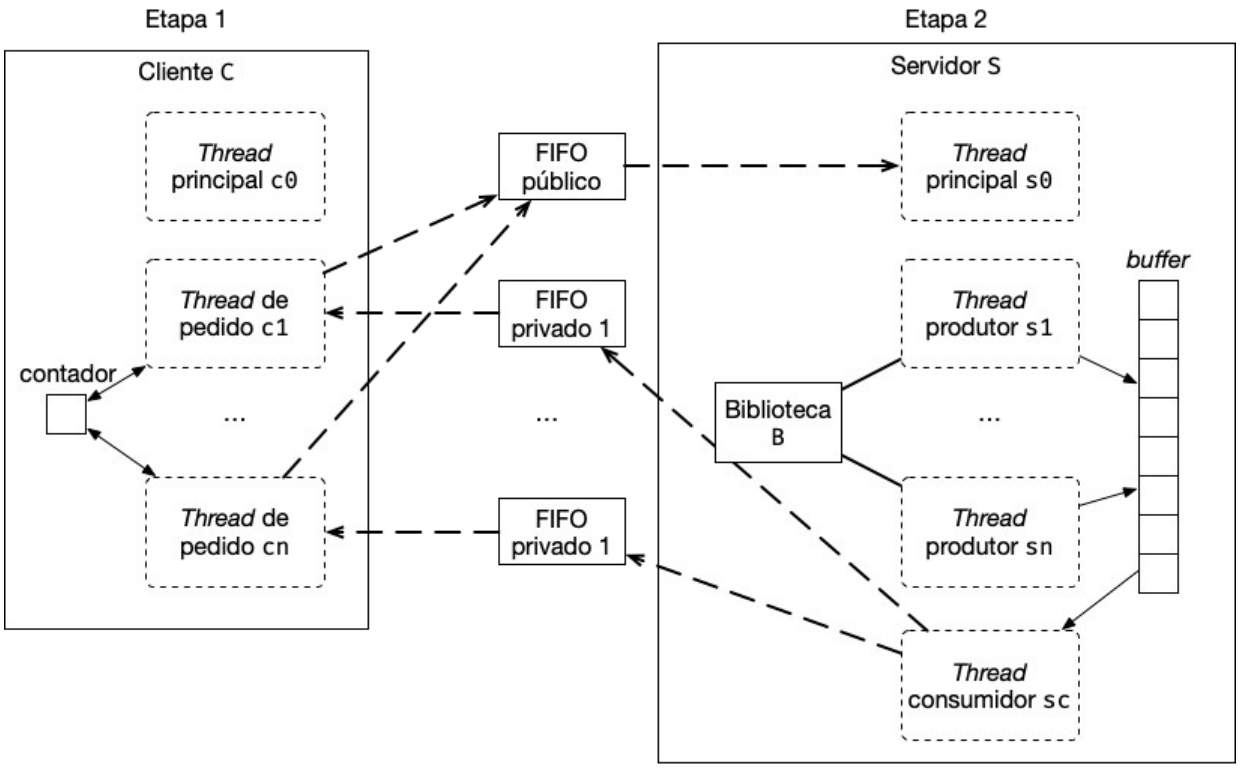


Requisitos

Requisitos comuns ao Cliente e ao Servidor

- tanto o Cliente como o Servidor deverão ser programas *multithread*;
- ambos funcionam com o máximo de paralelismo possível e evitam situações de encravamento (*deadlock*), de "colisão" e "esperas activas" (*busy-waiting*) com o auxílio de primitivas de sincronização do tipo das estudadas (e.g. *mutexes*);
- as tarefas a executar no Servidor (pelo código na Biblioteca) são identificadas por um inteiro único universal e exigem uma carga de processamento que é classificada por um número entre 1 e 9; o resultado da execução de cada tarefa é um valor inteiro único.

[**NOTA**: o comportamento da Biblioteca **B** pode ser parametrizado durante a compilação com um *delay* em milissegundos para facilitar a execução de testes. Este valor pode ser alterado na *Makefile* partilhada.]



Requisitos para o Cliente C

O Cliente **C** é invocado com o comando:

c <-t nsecs> <fifoname>

- nsecs - nº (aproximado) de segundos que o programa deve funcionar
- fifoname - nome (absoluto ou relativo) do canal público de comunicação com nome (FIFO) por onde o Cliente envia pedidos ao Servidor

O seu funcionamento, de forma sumarizada, é o seguinte:

- inicialmente, recolhe os argumentos de linha de comando e executa as operações fundamentais para o funcionamento do programa;
- o *thread* principal `c0` lança continuamente *threads* de pedido `c1`, …, `cn` (com intervalos (pseudo-)aleatórios de alguns milisegundos, por forma de exacerbar condições de competição), cada um ficando associado a um pedido ao Servidor;
- cada *thread* de pedido `c1`, …, `cn` gera os parâmetros do pedido pelo qual é responsável (especificamente, o seu número de identificação universal único e a carga da tarefa pedida, valor aleatório inteiro entre 1 e 9), trata de toda a comunicação com o servidor, incluindo a criação e posterior eliminação de um FIFO privado por onde receberá a resposta do Servidor, pela qual terá de esperar em bloqueamento; chegada a resposta, termina (não sem antes efectuar eventuais operações de registo ou disponibilização do resultado ao utilizador);
- caso o Servidor feche, o FIFO, o Cliente deverá detectar tal situação e terminar a geração de novos pedidos; os *threads* com pedidos já colocados deverão aguardar resposta, que será: o resultado normal do pedido de tarefa (que indica que o pedido ainda foi atendido a tempo pelo Servidor) ou uma indicação de encerramento (no caso de o pedido já não ser atendido);
- após o prazo de funcionamento especificado pelo utilizador, o Cliente deve terminar a execução fazendo com que os threads em espera de resposta desistam mas não sem antes garantir que todos os recursos tomados ao sistema são libertados.

[**NOTA**: o executável **S**, integrando a biblioteca **B**, é disponibilizado na 1ª Etapa.]

Requisitos para o Servidor S

O Servidor **S** é invocado com o comando:

s <-t nsecs> [-l bufsz] <fifoname>

- `nsecs` - nº (aproximado) de segundos que o programa deve funcionar
- `bufsz` - tamanho do armazém (*buffer*) a ser utilizado para guardar os resultados dos pedidos
- `fifoname` - nome (absoluto ou relativo) do canal público de comunicação com nome (FIFO) por onde serão recebidos pedidos de execução de tarefas

O seu funcionamento, de forma sumarizada, é o seguinte:

- inicialmente, recolhe os argumentos de linha de comando e executa as operações fundamentais para o funcionamento do programa;
- os pedido do Cliente são recolhidos pelo *thread* principal `s0` que cria *threads* Produtores `s1`, …, `sn` e lhes passa os pedidos; cada Produtor invoca a correspondente tarefa em `B` e coloca o resultado obtido no armazém; depois, termina;
- o acesso ao armazém deve ser gerido usando as primitivas de sincronização estudadas (e.g., semáforos), devendo os *threads* Consumidor e Produtores ficar bloqueados quando o armazém estiver, respectivamente, vazio ou cheio;
- o (único) *thread* Consumidor deve continuamente tentar retirar valores do armazém pela ordem de inserção e enviar cada um ao correspondente *thread* Cliente que fez o pedido, através do respectivo FIFO privado;

- mesmo não havendo pedidos do Cliente (estando o canal público vazio), o Servidor deve continuar a executar, ficando à espera da chegada de algum pedido, até ao final do prazo de execução especificado (em argumento da linha de comando);
- no final, deve garantir-se que todos os recursos tomados ao sistema são libertados.

[**NOTA**: o código binário da biblioteca **B**, que deve ser associado ao Servidor aquando da compilação de **S**, é disponibilizado.]

Comunicação entre C e S

Para ambos os programas **C** e **S**:

- as mensagens trocadas são sempre aos pares:
  - cada pedido terá sempre uma resposta;
- os pontos de comunicação são:
  - um canal público do tipo FIFO, cujo nome é passado como argumento das linhas de comando de **C** e de **S**, por onde são colocados 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 **C**;
    - **tid** - identificador de sistema Posix do respectivo *thread* de pedido;
- as mensagens trocadas em ambas as direções são estruturalmente idênticas, diferindo possivelmente no conteúdo de alguns campos. A sua estrutura é, em C:

```
struct msg { int i; int t; pid_t pid; pthread_t tid; int res; };
```

  - **i** - número universal único do pedido (gerado pelo Cliente);
  - **t** - inteiro entre 1 e 9 que representa a carga associada à execução tarefa do pedido;
  - **pid** - identificador de sistema do processo (Cliente **C**, no caso do pedido; Servidor **S**, no caso da resposta);
  - **tid** - identificador de sistema Posix do *thread* (do Cliente **C**, no caso do pedido; do Servidor **S**, no caso da resposta)
  - **res** - do Servidor, pode ser:
    - o resultado devolvido pela execução da tarefa;
    - o valor -1 como indicação ao Cliente de que o serviço já está encerrado (pelo que o pedido não foi atendido);
  - **res** - do Cliente, é sempre o valor -1.

Registo das operações

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

inst ; i ; t ; pid ; tid ; res ; oper

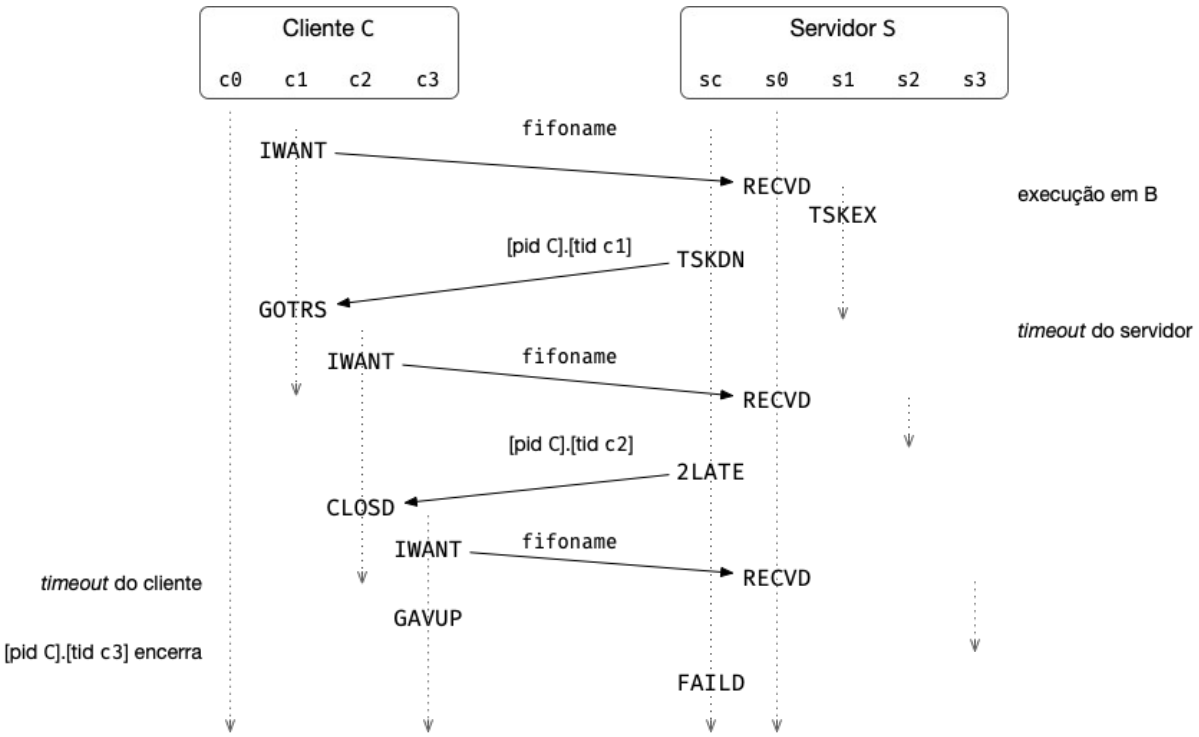
- **inst** - valor retornado pela chamada ao sistema `time()`, na altura da produção da linha;

- **i**, **t**, **pid, tid**, **res** - têm o mesmo significado que os campos correspondentes na estrutura das mensagens;
- **oper** - são palavras 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** - um *thread* de pedido do Cliente `c1`, …, `cn` faz pedido inicial
    - **RECVD** - o *thread* principal `s0` do Servidor acusa a recepção de pedido
    - **TSKEX** - um *thread* produtor do Servidor `s1`, …, `sn` informa que já tem o resultado de tarefa
    - **TSKDN** - o *thread* consumidor do Servidor `sc` informa que enviou ao Cliente o resultado de tarefa
    - **GOTRS** - um *thread* de pedido do Cliente `c1`, …, `cn` acusa a recepção do resultado da execução da tarefa
    - **2LATE** - o *thread* consumidor do Servidor `sc` informa que o pedido foi rejeitado por o serviço já ter encerrado
    - **CLOSD** - um *thread* de pedido do Cliente `c1`, …, `cn` acusa informação de que o Servidor rejeitou pedido por estar encerrado
    - **GAVUP** - um *thread* de pedido do Cliente `c1`, …, `cn` informa que não pode esperar mais pela resposta do Servidor, porque o prazo de funcionamento do Cliente venceu, e desiste
    - **FAILD** - o *thread* consumidor do Servidor `sc` informa que não consegue responder a pedido, porque a FIFO privada do *thread* do cliente fechou, e termina

Ligação das comunicações aos registos

O diagrama seguinte exemplifica a ligação entre as mensagens trocadas entre os processos/*threads* e os registos efectuados na saída padrão de cada um. Cada seta representa uma mensagem emitida por um *thread* e recebida por outro; o tempo cresce de cima para baixo. Neste exemplo, o Servidor termina a execução antes do Cliente terminar.

Neste exemplo, o *thread* principal do Cliente `c0` gerou três *threads* de pedidos (`c1`, `c2` e `c3`). O primeiro deles envia um pedido ao Servidor pelo FIFO público `fifoname` e regista esse pedido (**IWANT**). O *thread* principal do Servidor `s0` recebe esse pedido e acusa esse evento (**RECVD**), criando um *thread* Produtor `s1` para tratar da execução da tarefa pedida, chamando **B**; este Produtor, assim que recebe o resultado de **B**, acusa a receção do resultado (**TSKEX**) e coloca-o no armazém. O *thread* Consumidor `sc` vai lendo os resultados das execuções e responde ao Cliente através do respectivo *thread* privado, informando do envio (**TSKDN**). Quando o *thread* `c1` do Cliente recebe o resultado da execução, acusa essa receção (**GOTRS**). Quando o pedido de `c2` chega ao Servidor, já se deu o *timeout* e o pedido já não será atendido, pelo que o *thread* Produtor `s2` não executa a tarefa em **B** e coloca um item disso indicativo no armazém (-1). Nesse caso o *thread* Consumidor `sc` vai responder ao Cliente com o resultado -1 e informa desse envio (**2LATE**); quando `c2` recebe essa mensagem acusa a receção do encerramento do Servidor (**CLOSD**). Enquanto o thread de pedido `c3` espera pela resposta, dá-se o *timeout* do Cliente. Nessa altura o *thread* deixa de esperar pela resposta e acusa esse evento (**GAVUP**). Vai também fechar o seu FIFO privado, pelo que quando o Servidor tentar eventualmente responder vai falhar e acusar essa ocorrência (**FAILD**).



Produtos finais

Na 1ª etapa deve ser desenvolvido o código de um Cliente e na 2ª etapa o código de um Servidor.
Tais códigos (e ficheiros anexos) devem ser submetidos para avaliação da forma já publicada e nos prazos também já publicados.

Avaliação

O processo de avaliação será semelhante ao do MP1, passando pelas seguintes fases.

1. nota base do grupo, pesando cada uma das etapas 50%:
    - São disponibilizados testes que permitirão a avaliação da qualidade dos programas desenvolvidos. A especificação apresentada permitirá aos grupos despistar erros e corrigi-los antes da avaliação dos programas, a efectuar pelos docentes. O objectivo é a automação dos testes e uma avaliação razoavelmente objectiva e homogénea, minimizando a dependência do avaliador e do ambiente de avaliação. A automatização ainda não está garantida, mas uma razoável objectividade, sim. A máquina onde deverão ser feitos os testes e a avaliação é a GNOMO.FE.UP.PT, onde deverão estar instaladas todas as ferramentas auxiliares necessárias aos testes.
    - Estes testes estão divididos nos seguintes eixos:
        i. execução do código (70%): ver em baixo;
        ii. correção do código (20%): ver Adenda do MP1;
        iii. aspecto do código (10%): ver Adenda do MP1;
2. nota ajustada do grupo:
    - o docente pode ajustar a nota base do grupo em 10% (± 5%) com base na sua impressão global do processo de avaliação;
    - isso inclui a interacção com os elementos do grupo na aula de

avaliação/apresentação, a apreciação do pacote submetido com o código (incluindo o mini-relatório) e a consulta do seu instinto;

3. nota individual de cada elemento:
    - considera a auto-avaliação (interpares) fornecida pelos membros do grupo no mini-relatório;
    - a nota individual de cada aluno poderá variar em torno da nota de grupo de até ± 2 valores, sendo o docente o moderador desse ajuste.

Os testes à execução do código focar-se-ão principalmente na análise dos registos produzidos pelas aplicações **C** e **S**:

- algumas regras que podem ser consideradas são:
    - o número de mensagens IWANT registadas pelo Cliente deve ser igual ao somatório das mensagens GOTRS, CLOSD e GAVUP;
    - o número de mensagens TSKEX e TSKDN registadas pelo Servidor deve ser igual;
    - o número de mensagens GOTRS do Cliente deve ser igual ao de mensagens TSKDN do Servidor;
    - os identificadores das tarefas i, indicados pelo Cliente em IWANT devem ser únicos;
    - os resultados das tarefas res, indicados pelo Servidor em TSKDN devem ser únicos;
- como exemplo, a terceira destas regras pode ser verificada, por exemplo, com:
    - `nTSKEX=`grep TSKEX s.log | wc -l`; echo $nTSKEX`
    - `nTSKDN=`grep TSKDN s.log | wc -l`; echo $nTSKDN`
    - o valor de nTSKEX deve ser igual ao de nTSKDN
- está disponibilizado no Moodle um *shell script* que automatiza alguns destes testes (os alunos são encorajados a estender o *script* com testes adicionais para maior garantia de correção);
- dada esta automação dos testes, os alunos devem ter particular cuidado em garantir que as linhas dos registos seguem o formato definido em cima, sob pena dos resultados não poderem ser avaliados.