C:\Users\Pedro Tomás\Documents\IST\logo_IST_A.emf

**Clipboard distribuído**

Programação de Sistemas

**Alunos:**

Nome: André Cavalheiro Número: 84000

Nome: John Mendonça Número: 84106

Nome do Docente: João Nuno De Oliveira e Silva

**Índice**

[1 Arquitetura 3](#_Toc515211479)

[2 Gestão de Threads 3](#_Toc515211480)

[3 Sincronização 3](#_Toc515211481)

[3.1 Clipboard 3](#_Toc515211482)

[3.2 Envio de dados para o exterior 3](#_Toc515211483)

[*3.3* Criação de *threads* 4](#_Toc515211484)

[4 Comunicação Local 4](#_Toc515211485)

[5 Comunicação Remota 5](#_Toc515211486)

[6 Conclusões Finais 5](#_Toc515211487)

# Arquitetura

# Gestão de Threads

# Sincronização

O Clipboard Distribuído tem no decorrer da sua execução diversas *threads* que necessitam de aceder e/ou alterar a mesma informação. De modo a garantir o funcionamento pretendido do programa será necessário garantir a sincronização das mesmas, de modo a evitar problemas (como por exemplo as *race conditions)* e manter a integridade dos dados.

## Clipboard

Podemos considerar o Clipboard Distribuído na sua essência como um problema de Produtor-Consumidor, em que diversas entidades (*threads*) necessitam de aceder a uma dada região do Clipboard para ler(*paste*) ou escrever (*copy*). Visto que se pretende manter coerência no conteúdo em todos os clipboard ligados, será necessário garantir que o processo de *paste* não interfira com o processo de *copy* e vice-versa e, portanto, estamos perante uma zona crítica.

O clipboard está subdivido em várias regiões, cada uma com a sua própria região crítica. Esta região crítica está, por sua, vez implementada utilizando dois métodos em simultâneo: o *read-write* *lock* e o *mutex*. Tratando-se de um problema de produtor-consumidor, o tipo de implementação mais adequado para esta região crítica é o *read-write lock*, pois permite a leitura do conteúdo por parte de várias *threads* em simultâneo. No entanto, de modo a acomodar a função *wait*, que é um requisito do cliente local, bem como das conexões exteriores (de modo a manter a sincronização global) foi necessário introduzir um mutex, de modo a podermos utilizar a implementação POSIX da espera condicional. A utilização da espera condicional de novos dados em detrimento da espera ativa (while loop) conduz a uma maior eficiência do projeto. Na figura seguinte está ilustrado a zona crítica de uma região do clipboard.

## Envio de dados para o exterior

Após aquisição de novos dados, a *thread* dedicada ao envio de nova informação ao exterior da dada região necessita de percorrer uma lista ligada dedicada para o efeito. Esta lista, sendo partilhada entre as threads das várias regiões constitui uma região crítica, pois podem ocorrer novas ligações ou fecho de outras em tempo real. Deste modo, será necessário proteger esta região com exclusividade(*mutex)* pois todos os processos com acesso a esta lista têm a possibilidade de eliminarem elementos da lista, caso contrário um read-write lock seria o ideal. O mecanismo de atualização da lista após o fecho de uma ligação consiste na *thread* responsável pelo envio da informação eliminar o elemento quando deteta no envio pelo *socket*, que a ligação se encontra encerrada.

## Criação de *threads*

Ao longo de todo o processo existem situações em que são necessários criar *threads* semelhantes, nomeadamente as *threads* de envio de informação ao exterior e de *handling* de conexões com clientes locais, em que estas reutilizam as mesmas variáveis (pois elas só contêm informação necessária para a *thread*) e portanto existe uma *race condition*. De modo a evitar este problema, utilizou-se um *mutex* comum para todo o processo denominado setup\_mutex. Cada nova thread só liberta a zona crítica quando tiver todos os parâmetros em questão em memória própria.

# Comunicação Local

A comunicação remota entre os utilizadores e o próprio clipboard é feita a partir de sockets da família AF\_UNIX e do tipo SOCK\_STREAM. A escolha entre os diversos tipos de sockets foi feita levando em consideração a natureza dinâmica das mensagens a ser transmitidas.

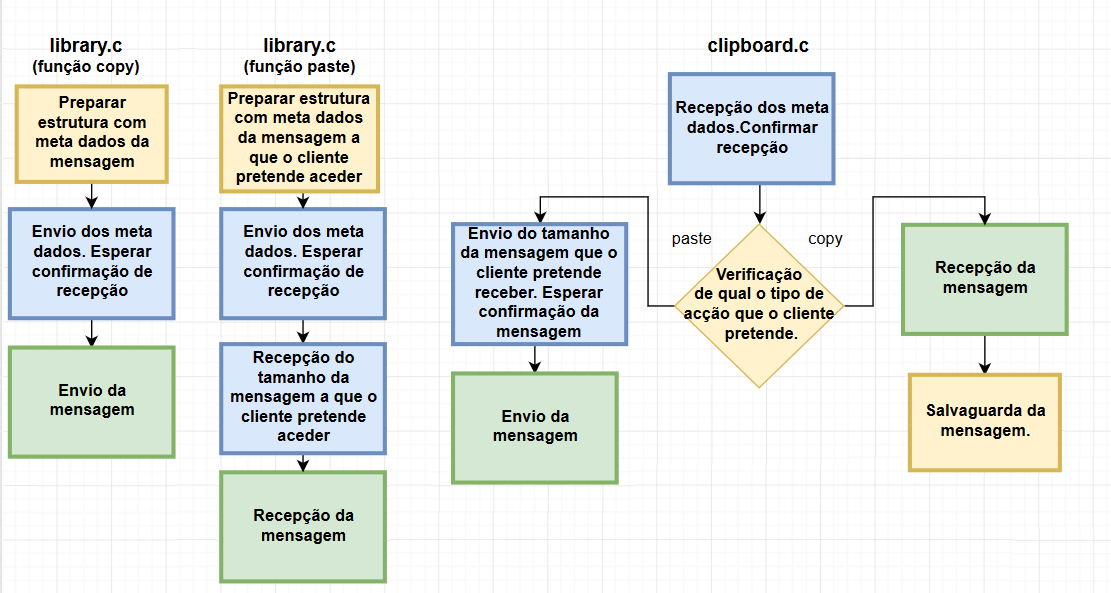
Tendo isto em mente o protocolo de comunicação entre os dois programas segue o seguinte padrão:

* É feita uma primeira comunicação para informar ambos os lados da informação necessária à transmissão da mensagem. Este diálogo ocorre sempre com o uso da estrutura representada na figura …, a partir das funções *handShake* e *handleHandShake*.
* A mensagem é transmitida de um lado para o outro a partir de um ciclo while, garantindo sempre o envio total da mensagem ou o lançamento de um erro. Esta transmissão é feita usando ponteiros do tipo *void\** e as funções *sendData* e *receiveData*.

Meter aqui print da estrutura struct metaData.

Os diagramas representativos das funções mais importantes na comunicação local encontram-se representados na figura ….

* Os retângulos representados a azul representam as funções handShake / handleHandShake consoante se trata do envio dos meta dados ou receção dos mesmos respetivamente.
* Os retângulos a verde representam as funções *sendData* */* *receiveData* consoante se trata do envio da mensagem ou da receção da mesma respetivamente.
* Os retângulos a amarelos não representam qualquer função, mas sim passos intermédios e necessários ao funcionamento do programa.



(falar aqui de como é lançada uma thread para cada usuário local do clipboard ou isso fica tudo explicito na secção 2?)

# Comunicação Remota

# Conclusões Finais

