PROGRAMAÇÃO DE SISTEMAS

Relatório de Projeto

DISTRIBUTED CLIPBOARD

Identificação dos Alunos (Grupo 55):

Nome:Bernardo Bastos Número: 84012

Nome:Martim Pita Número: 84136

Nome do Docente: João Nuno de Oliveira e Silva

Índice

1 – Introdução

O projeto desenvolvido nesta unidade curricular de Programação de Sistemas, consiste na implementação de um clipboard distribuído, no qual um clipboard local tem capacidade para responder a pedidos de várias aplicações em simultâneo e também de se conectar a outros clipboards locais. A informação contida num determinado clipboard é a mesma que em todos os clipboards conectados.

Para a implementação deste clipboard distribuído foi criada uma biblioteca de funções que implementa uma API que permite a comunicação entre uma aplicação controlada pelo utilizador e o clipboard local. Este clipboard local também foi desenvolvido no âmbito deste projeto.

A realização deste projeto teve por base a consolidação e avaliação dos conhecimentos obtidos ao longo do semestre nesta unidade curricular, principalmente os protocolos de comunicação entre processos, a utilização de threads múltiplas e a sincronização das mesmas.

2 - Arquitetura do programa

Para melhor compreensão do programa, elaboraram-se vários fluxogramas relativos ao ficheiro “clipboard.c”. Os blocos representados a colorido representam threads. O fluxogramas elaborados foram:

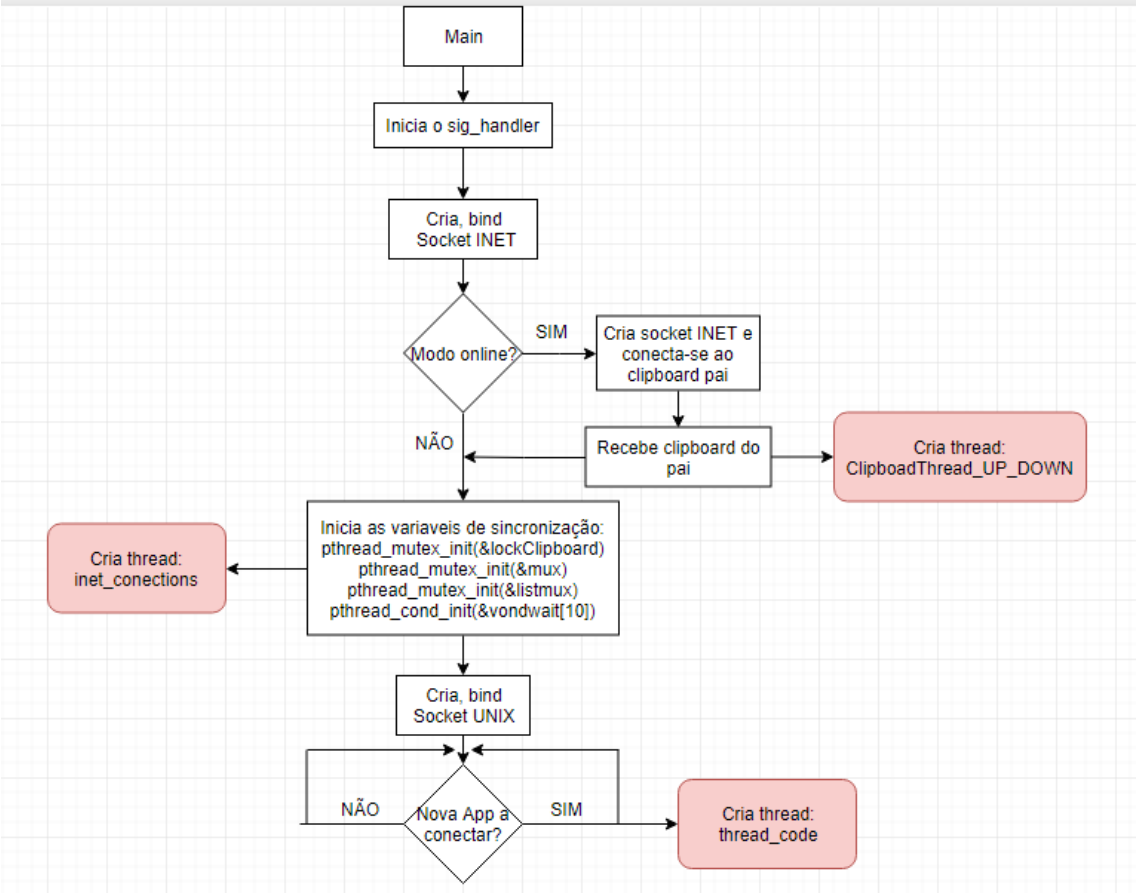

Figura 1 - Fluxograma função main

Observando o fluxograma da figura 1 verifica-se que o programa começa por criar uma socket INET, responsável pela comunicação com possíveis clipboards filhos.

Caso o clipboard esteja em modo ONLINE, cria-se uma nova socket INET responsável pela comunicação com o clipboard pai e a thread ClipboardThread_UP_DOWN, responsável pela receção de informação vinda do clipboard pai e se necessário, transferi-la aos filhos.

De seguida, cria-se a thread inet_connections, responsável por aceitar conexões vindas de clipboards filhos, e iniciam-se as vareáveis de condição e sincronização usadas ao longo do programa.

Finalmente, cria-se uma socket UNIX responsável pala comunicação entre o clipboard e as apps. De seguida, o programa entra num loop infinito onde está constantemente à

espera de novas conexões vindas de apps. Sempre que é feita uma conexão, é criada uma nova thread thread_code, que realiza as operações de copy, paste e wait.

Threads usadas:

inet_connections: É a primeira thread a ser criada. A sua principal função é aguardar por novas conexões vindas de clipboards filhos.

Caso seja feita a conexão, o clipboard atual é enviado para o filho e cria-se a thread ClipboardThread_DOWN_UP, responsável por receber informação do filho e transmitir para o pai caso o clipboard esteja em modo ONLINE. O client_id retornado pela função accept é guardado na lista destinada a tal.

Elaborou-se o fluxograma da figura 2.

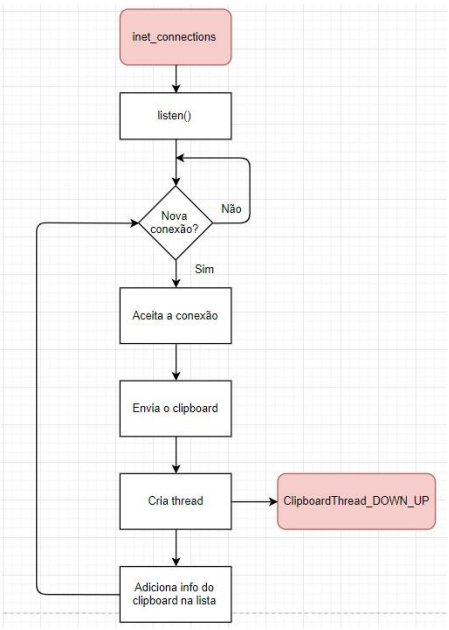

Figura 2 - Fluxograma thread inet_connections

ClipboardThread_DOWN_UP: Esta thread apenas é utilizada para receber informação vinda de clipboards filhos e transferi-la para os pais. Como tal, esta thread apenas é criada na função inet_connections assim que se conecta um clipboard filho.

Uma vez recebida a informação, verifica-se se se o clipboard está a atuar em modo ONLINE. Caso esteja é necessário continuar a subir a árvore, se não estiver, o clipboard e o vetor word_size são atualizados e inicia-se o processo de descida da árvore.

Elaborou-se um fluxograma para melhor compreensão da função:

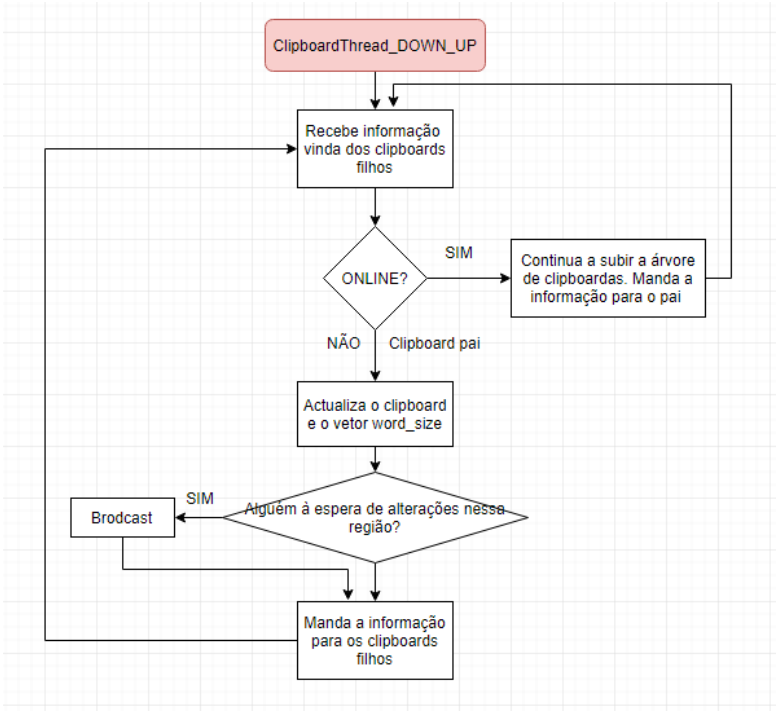

Figura 3 - fluxograma ClipboardThread_Down_Up

ClipboardThread_UP_DOWN: Esta thread apenas é criada quando o clipboard atua um modo ONLINE. Esta thread é responsável por receber a informação do pai, atualizar o clipboard e o vetor correspondente aos tamanhos das palavras e, caso não seja o último clipboard da árvore, transmitir a informação aos filhos.

Elaborou-se novamente um fluxograma:

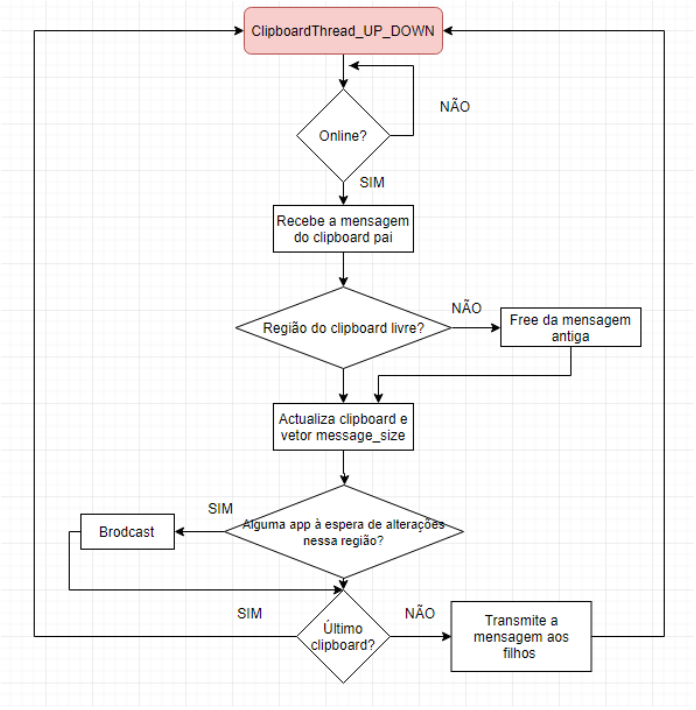

Figura 4 - Fluxograma ClipboardThread_UP_DOWN

2.1 – Protocolo de comunicação

Ao longo do programa são utilizadas várias formas de comunicação entre processos/programas. De acordo com a funcionalidade e tipo de comunicação pretendida (local ou entre maquinas diferentes), escolheram-se protocolos de comunicação diferentes.

Para a comunicação local, escolheram-se sockets domínio UNIX e variáveis globais.

Sockets UNIX: Usadas na comunicação entre a app local (cliente) e clipboard (servidor). A "ponte" de comunicação entre a app e o clipboard são as funções presentes no ficheiro "library.c". Estas funções comunicam com o clipboard recorrendo a uma socket domínio UNIX. A facilidade com este este protocolo de comunicação permite tanto ao cliente como ao servidor de enviar e receber informação, foi um dos fatores decisivos para a sua escolha.

Variáveis Globais: Usadas na comunicação entre threads e funções do clipboard. Para além de permitir a comunicação entre threads, o uso deste tipo de variáveis permite simplificar o código, uma vez que evita a passagem de variáveis como argumento nas funções. As variáveis globais usadas foram:

Tabela 1 - Variavéis globais usadas em clipboar.c

Variáveis Globais em "clipboard.c"	
int clipboard_type	Define se o clipboard está em modo online (filho) ou modo offline (pai)
int clipboard_last	Indica se o clipboard é o último da árvore
char *clipboard_vetor[10]	Vetor responsável por guardar a informação do clipboard
int word_size[10]	Guarda o tamanho da data do clipboard em cada região
int waiting[10]	Indica o numero de apps que estão à espera duma alteração numa determinada região
struct sockaddr_in server_adress	Endereço da socket INET (servidor)
struct sockaddr_in client_adress	Endereço da socket INET (cliente)
struct sockaddr_in connected_adress	Endereço da socket INET que comunica com o clipboard pai
socklen_t size_addr	Tamanho do endereço
int connected_socked	Guarda o file descriptor da socket INET que comunica som o clipboard pai
int sock_fd	Guarda o file descriptor duma socket UNIX que comunica com as app
int internet_socket	Guarda o file descriptor duma socket inet que comunica com os clipboards filhos

pthread_rwlock_t lockclipboard	Chaves de sincronização usadas ao longo do programa. A função de cada uma delas está explicada com maior detalhe na secção de sincronização
pthread_mutex_t mux	
pthread_mutex_t listmux	
pthread_cond_t condwait[10]	
clip_thread * thread_List	Cabeça da lista que guarda os endereços retornados pela função "accept", assim que se conecta um clipboard filho

Relativamente à comunicação entre clipboards em máquinas diferentes, recorreu-se a internet sockets (sockets do domínio INET). Este tipo de sockets permite a comunicação entre processos que correm em máquinas diferentes, no entanto, ligados à mesma rede de computadores.

Em ambos os tipos de comunicação (local e internet) usaram-se STREAM sockets visto que se pretendia transferir informação de forma ordeira e intacta.

2.2 – Fluxo de tratamento de pedidos

Como foi dito na secção anterior, a "ponte" de comunicação entre a app e a clipboard são as funções presentes no ficheiro "library.c". Isto é, sempre que o cliente (app) queira aceder ao servidor (clipboard), este vai recorrer às funções presentes no ficheiro "library.c". As funções e as suas respetivas funcionalidades encontram-se resumidas na seguinte tabela:

Tabela 2 - Funções presentes no ficheiro library.c

Funções "library.c"	
int clipboard_connect()	Função que faz a conexão entre a app e clipboard
int clipboard_copy()	Função que faz a copia de informação da app para o clipboard
int clipboard_paste()	Função que lê dados do clipboard e retorna-os para a app
int clipboard_wait()	Função que aguarda uma alteração numa dada região
int clipboard_close()	Função que fecha a socket que conecta com a app

Para compreender melhor o funcionamento do programa, analisou-se com mais detalhe as funções copy, paste e wait. Em qualquer uma destas funções foram usadas as seguintes variáveis:

Tabela 3 - Variaveis locais definidas nas funções copy, paste e wait

Variáveis usadas em copy, paste e wait	
message_sent message	Estrutura que contem a região do clipboard que se pretende aceder e o tamanho da mensagem a copiar/receber.
int numberofbytes	Número de bytes escritos/recebidos
int errorcheck	Variável que indica se ocorreu algum erro, tanto na library como no clipboard
int operation	Variável que informa o clipboard do tipo de operação a realizar

Nota: A estrutura message_sent encontra-se definida no ficheiro "clipboard.h".

Função copy

int clipboard_copy(int clipboard_id, int region, void *buf, size_t count)

Descrição dos parâmetros de entrada:

- clipboard_id valor retornado pelo clipboard conect
- region região para a qual se pretende copiar data
- buf ponteiro para a data que se pretende copiar
- count tamanho da data apontada pelo ponteiro buf

Primeiramente, a função envia para o clipboard, a variável operation com o valor 1, valor indicatitivo da operação copy. Desta forma o clipboard prepara-se para receber um copy.

De seguida, a função verifica se a região pedida pela app é válida, isto é, se está compreendida entre 0 e 9. Para além disso, também verifica se a mensagem a enviar é válida, ou seja, se o valor apontado pelo vetor buf é diferente de NULL. Se alguma destas condições se verificar, a operação copy não é válida e a variável errorcheck passa a ter o valor 1, e a operação copy é cancelada (return 0).

Tanto no caso em que a operação é valida (errorcheck = 0) como no caso em que é invalida (errorcheck =1), a variável errorcheck é sempre enviada para o clipboard. Assim, no caso da operação ser inválida, o clipboard deixa de realizar um copy e fica logo à espera de receber a próxima operação. No caso de ser válida, a operação continua normalmente.

Após feita a verificação, a estrutura message do tipo message_sent é atualizada com o tamanho e região da mensagem a enviar, e posteriormente enviada para a clipboard.

Uma vez enviada a estrutura, envia-se finalmente a mensagem a ser copiada para o clipboard.

Apesar de ser feita a verificação de erros na library, esta também é feita no clipboard. Esta dupla verificação de erros evita a implementação de uma biblioteca pirata que possa corromper o servidor.

Função paste

int clipboard_paste(int clipboard_id, int region, void *buf, size_t count)

Descrição dos parâmetros de entrada:

- clipboard_id valor retornado pelo clipboard conect
- region região da qual se pretende ler data
- buf ponteiro para a data que se pretende ler
- count tamanho da data apontada pelo ponteiro buf

A descrição da função paste é muito semelhante à função copy. A função envia a variável operation com o valor 0, indicativo da operação paste. Deste modo, o clipboard fica pronto para efetuar um paste.

De seguida, faz-se a verificação da região da mesma forma que na função copy. Se esta não tiver entre 0 e 9, errorcheck toma o valor 1 e a operação paste é cancelada (return 0). Se a operação for válida, errorcheck toma o valor 0 e a operação continua.

Após feita a verificação de erros, é enviada a estrutura message com a região da qual se pretende ler informação, e a quantidade de informação que se pretende ler.

Após a entrega do trabalho, verificou-se a ocorrência de um erro na elaboração da função paste, uma fez que a quantidade de informação enviada pelo clipboard corresponde sempre ao tamanho total da palavra. Assim, se o tamanho pedido pelo utilizador for inferior ao tamanho da palavra, é necessário enviar apenas o tamanho pedido pelo cliente. Deste modo, na linha 204, do “clipboard.c” deveria estar:

write(cliente_id, buf, mesage_sent_1.size);

Desta forma, a quantidade de data enviada pelo clipboard, corresponde à pedida pelo cliente.

Função wait

int clipboard_wait(int clipboard_id, int region, void *buf, size_t count)

Descrição dos parâmetros de entrada:

- clipboard_id valor retornado pelo clipboard conect
- region região da qual se pretende esperar uma alteração
- buf ponteiro para a data que foi alterada na região
- count tamanho da data apontada pelo ponteiro buf

Novamente a função envia a variável operation com o valor 2, indicativo da operação wait. De seguida, verifica se a região para a qual se pretende esperar uma alteração é válida. Caso não seja, checkerror = 1, o processo é cancelado (return 0), caso seja válido checkerror = 0, a operação continua.

De seguida, a estutura message é enviada para o clipboard com a zona que se pretende esperar a alteração e com a quantidade de data que se pretende receber.

Despois deste processo todo, a app fica em "stand by", aguardando um write por parte do clipboar com a alteração feita na região pretendida.

3 – Sincronização

A sincronização das diferentes tarefas realizadas pelo clipboard foi efetuada recorrendo tanto a read-write locks como a mutexes, de modo a proteger as regiões críticas do programa.

3.1 – Identificação de regiões críticas

Entende-se por região crítica como uma região que só pode ser executada por uma thread de cada vez, de modo a haver um funcionamento correto do programa. Deste modo foram identificadas as seguintes regiões críticas:

- Manipulação do conteúdo do clipboard – Uma thread não pode escrever novas informações na clipboard, enquanto outra thread lê do clipboard, e uma thread não pode ler nem escrever no clipboard enquanto outra thread escreve.
- Manipulação da lista de clipboards conectados – Apenas uma thread de cada vez pode efetuar alterações na lista ou percorrê-la.
- Espera por condição no wait – De modo a implementar a função wait do clipboard é necessária a utilização de uma variável de condição, que necessita de ser protegida através de um mutex, de forma a não ocorrer uma perda de sinal, ou seja, não ser executado um pthread_cond_broadcast() de uma variável antes de ser executado o pthread_cond_wait() dessa mesma variável.

3.2 - Implementação da exclusão mútua

Para a proteção das diferentes regiões críticas apresentadas anteriormente foi implementada a exclusão mútua, recorrendo a read-write locks e a mutexes.

Em primeiro lugar é efetuada a inicialização do read-write lock e dos mutexes, através da execução das funções pthread_rwlock_init() e pthread_mutex_init() respetivamente. Estas funções são chamadas no ficheiro clipboard.c dentro da função main().

Quanto ao read-write lock utilizado para proteger o conteúdo do clipboard a implementação é efetuada do seguinte modo: Antes de qualquer acesso para efetuar a leitura de conteúdo da clipboard é chamada a função pthread_rwlock_rdlock() de modo a bloquear a região para qualquer outra thread que tente adquirir uma write lock, e antes de qualquer acesso para escrita na clipboard é chamada a função pthread_rwlock_wrlock() de modo a bloquear a região para qualquer outra thread que tente adquirir uma read lock. Ou seja, enquanto a thread que possui o bloqueio de leitura não o libertar mais nenhuma outra thread poderá adquirir o bloqueio de escrita e deste modo alterar o conteúdo que poderia estar a ser lido por outra thread. De forma contrária, enquanto uma thread possuir o bloqueio de escrita, nenhuma outra thread poderá adquirir nem um bloqueio de leitura nem um bloqueio de escrita, o que não ocorre com o bloqueio de escrita. O bloqueio de leitura pode ser adquirido por múltiplas threads em simultâneo, enquanto o bloqueio de escrita só pode ser adquirido por uma thread de cada vez.

Após a realização da leitura ou escrita na clipboard, o programa sai da região crítica e é chamada a função pthread_rwlock_unlock() de modo a libertar o bloqueio.

Abaixo são apresentados exemplos do código do projeto.

```
pthread_rwlock_rdlock(&lockClipboard); //Acquires the read lock
memcpy(buf, clipboard_vector[message_sent_1.region], word_size[message_sent_1.region]); //copies the content of clipboard to the buffer
pthread_rwlock_unlock(&lockClipboard); // Releases the lock
```

Figura 5 - Read lock

```
pthread_rwlock_wrlock(&lockClipboard); // Acquires the writer lock
//Gets rid of old information
if(clipboard_vector[message_sent_1.region]!=NULL)
{
    free(clipboard_vector[message_sent_1.region]);
}

clipboard_vector[message_sent_1.region]=message; // Stores the new message
word_size[message_sent_1.region]=message_sent_1.message_size;
pthread_rwlock_unlock(&lockClipboard); //Releases the writer lock
```

Figura 6 - Write lock

Em relação ao mutex utilizado para proteger a manipulação da lista de clipboards, é chamada a função pthread_mutex_lock(&listmux) antes da inserção ou remoção de um novo elemento na lista e após esta inserção ou remoção é libertado o bloqueio

através da função pthread_mutex_unlock(&listmux). O mutex é implementado desta forma de modo a que não seja possível threads diferentes manipularem em simultâneo o conteúdo da lista. Na implementação desta exclusão mútua no projeto, não foi devidamente protegido com o mutex a distribuição de informação pelos diferentes clipboards conectados, através do varrimento da lista, podendo deste modo ocorrer erros, se por exemplo for conectado ou desconectado um clipboard enquanto a informação estiver a ser distribuída.

Para a implementação da função de wait, é realizada, como já referida uma espera por uma variável de condição, recorrendo às funções pthread_cond_wait() e pthread_cond_broadcast(). Estas só podem ser chamadas se estiverem inseridas dentro de um mutex. Deste modo é chamada a função pthread_mutex_lock(&mux) de modo a adquirir o bloqueio, que depois será desfeito com a chamada da função pthread_cond_wait(), onde o bloqueio será libertado (para que possa ser adquirido pelo mutex que precede a chamada da função pthread_cond_broadcast())e a thread ficará a aguardar um broadcast do sinal para continuar a execução do código. Após o broadcast a função pthread_cond_wait() adquire de novo o bloqueio que depois é libertado chamando a função pthread_mutex_unlock(&mux).

Apresentam-se exemplos do código referente a estas secções.

```
pthread_mutex_lock(&listmux); //Once you are adding an element in the list you can not change it
newthread->client_id = client_fd;
ThreadListAdd(newthread);
pthread_mutex_unlock(&listmux);
```

Figura 5 - Mutex manipulação da lista

```
pthread_mutex_lock(&mux); // Acquires the mutex lock

if(waiting[message_sent_1.region]!=0)
{
    pthread_cond_broadcast(&condwait[message_sent_1.region]); // Broadcasts the condition to all waiting threads
    waiting[message_sent_1.region]=0; // Resets the number of threads waiting for a certain region to zero
}

pthread_mutex_unlock(&mux); // Releases the mutex lock
```

Figura 6 - Mutex variável de condição

No fim da execução do programa dever-se-ia chamar as funções respetivas de destruição dos diferentes mutexes, o que não foi implementado.

4. Gestão de recursos e tratamento de erros

4.1 Gestão de recursos

Ao longo do projeto é alocada memória dinamicamente para as mensagens a guardar no clipboard e também para as estruturas que vão compor a lista de clipboards conectados. A memória é alocada de cada vez que chega uma nova mensagem para

ser guardada no clipboard, a mensagem é então guardada na região pretendida do clipboard. Esta memória será depois libertada quando for guardada uma nova mensagem na região, ou então quando o clipboard for fechada toda a memória é libertada num ciclo for que percorre todas as regiões.

Quando um novo clipboard se conecta é alocada memória para uma estrutura que contém informações sobre este clipboard e depois adiciona-se à lista. Quando o clipboard é desconectado é feita a libertação da memória da estrutura referente ao clipboard desconectado. Quando se encerra um clipboard que possui clipboards conectados, é percorrida a lista e liberta-se toda a memória alocada.

A gestão das sockets é efetuada de modo a que quando um clipboard é desconectado são chamadas as funções close() para todas as sockets, de modo a fechar todas as ligações abertas, e é chamada a função unlink() para eliminar o nome da socket e removê-la do sistema de ficheiros. Se uma app for desconectada é também chamada a função close().

As threads utilizadas são terminadas através de um return() que chama implicitamente a função pthread_exit().

4.2 Tratamento de Erros

O tratamento de erros que possam ser introduzidos pelo utilizador através das aplicações é efetuado tanto nas funções da library como no próprio clipboard, obtendo-se deste modo uma maior segurança no programa. É considerado um erro se o utilizador pretender aceder a uma região inexistente ou se pretender efetuar um copy de um vetor NULL. Deste modo é assegurado no clipboard que as informações referentes à região a que se pretende aceder está entre 0 e 9, e se o vetor que se pretende copiar é diferente de NULL. O mesmo ocorre nas funções presentes no ficheiro library.c. Após estas verificações o clipboard e a library comunicam entre si de modo a confirmar que não existe nenhum erro e continuar a operação desejada.

O outro tipo de erros tratados são os referentes ao sistema, por exemplo, falha na alocação de memória, falha na criação de sockets, falha numa tentativa de leitura ou escrita numa socket, entre outros. Quando estes ocorrem, o programa comporta-se sempre da mesma forma, envia para stdout uma mensagem perror e termina o programa através de um exit com valor -1.