

UNIVERSIDADE FEDERAL DE MINAS GERAIS  
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA

**AUTOMAÇÃO EM TEMPO REAL**  
**TRABALHO PRÁTICO**

Leiza Souza - 2017102100

Gabriel Lara - 2017088182

BELO HORIZONTE

2021

## **SUMÁRIO**

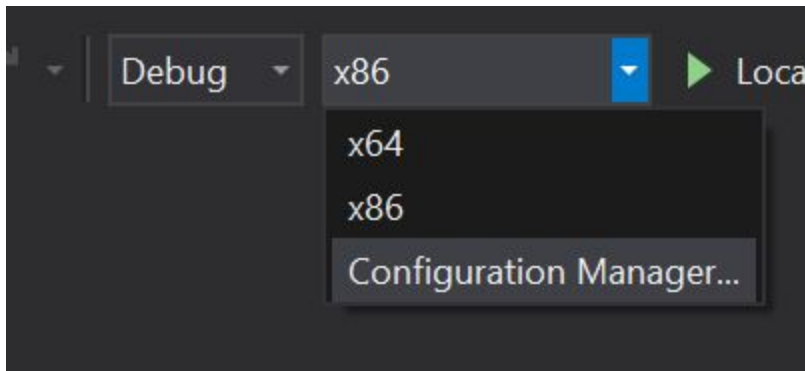
<b>INTRODUÇÃO</b>	<b>4</b>
<b>ESTRUTURA</b>	<b>5</b>
TAREFA DE LEITURA DO SISTEMA DE MEDIÇÃO	5
TAREFA DE LEITURA DE DADOS DO PROCESSO	5
TAREFA DE CAPTURA DE MENSAGENS	5
TAREFA DE EXIBIÇÃO DE DADOS DO PROCESSO	5
TAREFA DE ANÁLISE DE GRANULOMETRIA	5
TAREFA DE LEITURA DO TECLADO	6
PROCESSOS	6
COMUNICAÇÃO ENTRE PROCESSOS (IPC)	7
THREADS	10
DADOS COMPARTILHADOS	10
MENSAGENS	11
<b>CONTROLE</b>	<b>14</b>
SINCRONIZAÇÃO	15
<b>TEMPORIZAÇÃO</b>	<b>19</b>
TAREFA DE LEITURA DO SISTEMA DE MEDIÇÃO	19
TAREFA DE LEITURA DE DADOS DE PROCESSO	20
<b>ESPECIFICAÇÕES DE PROJETO</b>	<b>20</b>
ACESSO A LISTAS CIRCULARES	21
BLOQUEIO E DESBLOQUEIO DAS THREADS E ENCERRAMENTO	22
<b>TESTES DE EXECUÇÃO</b>	<b>24</b>
COMPILAÇÃO	24
DEPURAÇÃO	24

*Nota sobre execução: A solução do Visual Studio está dividida em três projetos.*

*A abertura da solução no Visual Studio deve ser feita carregando o arquivo  
"projeto/granulometria/granulometria.sln"*

*Dessa forma os outros projetos são carregados da maneira correta.*

*Verifique que ao lado de "Debug" está selecionado "x86":*



*Em seguida compila-se o programa com a função build.*

## 1. INTRODUÇÃO

O aço produzido nas siderúrgicas compõe vários materiais que estão presentes no dia a dia da população, sejam eles eletrodomésticos, veículos de locomoção, construções civis e até mesmo utensílios. Para obtê-lo, é necessário que o minério de ferro passe por um processo térmico com objetivo de transformá-lo em pellets, ou pelotas, o que atribui o nome à tecnologia: Pelotização. O processo surgiu no século XX e tem como produto pequenas esferas férricas com diâmetro entre 8mm e 18mm, após a transformação do minério nas etapas de moagem, espessamento, homogeneização, filtragem, pelotamento e queima, respectivamente.



Figura 1 - Comparação entre minério granulado e pelotas de ferro [1]

Nos altos-fornos das usinas siderúrgicas os pellets são transformados em ferro-gusa e, para que isso ocorra, é necessário que apresentem características que agreguem qualidade ao produto, sendo um dos principais indicativos de qualidade a granulometria: as pelotas devem ser pequenas, porém devem possuir tamanho suficiente para que haja lacunas entre elas, facilitando a circulação de ar e aumentando a superfície de contato. Para avaliar a granulometria de pelotas cruas, será desenvolvido um sistema a ser implantado nos discos de pelotização de uma usina, responsável por ler os dados do sistema de medição de granulometria e os dados do Controlador Lógico Programável (CLP) e apresentá-los em terminais dedicados, com o objetivo de automatizar o controle e a supervisão do processo. Os dados provenientes da aplicação de software serão analisados pelos operadores de processo no primeiro terminal e por especialistas de pelotização no segundo terminal, voltado à granulometria.

## 2. ESTRUTURA

A aplicação *multithread* desenvolvida no Ambiente de Desenvolvimento Integrado (IDE) *Microsoft Visual Studio Community Edition*, na linguagem C/C++, faz o uso API Win32 para garantir maior eficiência e facilidade de desenvolvimento, por se tratar de uma API nativa do Windows, e foi desenvolvida baseada nas seguintes tarefas:

### 2.1. TAREFA DE LEITURA DO SISTEMA DE MEDIÇÃO

Responsável por ler as mensagens originadas do sistema de medição online de granulometria das pelotas produzidas pela usina de mineração e depositá-las em uma primeira lista circular de memória RAM, que possui capacidade de 200 mensagens.

### 2.2. TAREFA DE LEITURA DE DADOS DO PROCESSO

Responsável por ler as mensagens originadas do CLP e depositá-las na mesma lista circular citada anteriormente.

### 2.3. TAREFA DE CAPTURA DE MENSAGENS

Responsável por consumir as mensagens da primeira lista circular de memória e enviá-las para a *tarefa de exibição de dados de processo*, caso sejam provenientes do CLP, ou depositá-las em uma segunda lista circular em memória, com capacidade de 100 mensagens, caso sejam provenientes do sistema de medição granulometria.

### 2.4. TAREFA DE EXIBIÇÃO DE DADOS DO PROCESSO

Responsável por receber as mensagens de dados de processo do CLP através da *tarefa de captura de mensagens* e exibi-las em uma tela dedicada na sala de controle.

### 2.5. TAREFA DE ANÁLISE DE GRANULOMETRIA

Responsável por consumir da segunda lista circular em memória as mensagens provenientes do sistema de medição de granulometria e exibi-las em uma segunda tela dedicada na sala de controle.

### 2.6. TAREFA DE LEITURA DO TECLADO

Responsável por aguardar comandos do operador (caracteres no teclado) e tratá-los, de acordo com o estado anterior de cada tarefa, bloqueando-a ou desbloqueando-a por meio dos seguintes objetos “evento” de sincronização do kernel:

2.6.1. <g>: referente à *tarefa de leitura do sistema de medição*;

2.6.2. <c>: referente à *tarefa de leitura de dados do processo*;

2.6.3. <r>: referente à *tarefa de captura de mensagens*;

- 2.6.4. <p>: referente à *tarefa de exibição de dados de processo*;
- 2.6.5. <a>: referente à *tarefa de análise de granulometria*;
- 2.6.6. <l>: responsável por notificar à *tarefa de exibição de dados de processo* que deve limpar a janela de console;
- 2.6.7. <ESC>: responsável por notificar todas as tarefas que devem encerrar a execução.

As tarefas indicadas nos itens 2.1, 2.2 e 2.3 devem se bloquear quando as respectivas listas em memória estiverem cheias e se desbloquear quando houver uma nova posição livre.

## 2.7. PROCESSOS

Um processo pode ser definido como um programa ou uma atividade em execução que possui um conjunto de recursos exclusivos gerenciados pelo Sistema Operacional, como registradores e espaços de endereçamento virtual, e é composto por diversos objetos relacionados à sua execução. Sempre que um processo cria um objeto do *kernel*, o S.O. retorna um *handle* através do qual é possível manipulá-lo. Os estados manipuláveis e possíveis nesta aplicação serão:

### 2.7.1. Pronto para executar

Aguardando a sinalização de um objeto do *kernel*, no caso um mutex, para possuir a CPU:

*WaitForSingleObject(sem\_livre, timeout)*

### 2.7.2. Executando

Possui a CPU durante um intervalo de tempo, após a sinalização do mutex.

### 2.7.3. Bloqueado

Aguardando a ocorrência de um evento de E/S para continuar, no caso a leitura de duas teclas (ESC ou o respectivo caractere sinalizador) ou a liberação de posição na lista circular:

*WaitForMultipleObjects(2, Events, FALSE, INFINITE)*

*WaitForMultipleObjects(2, buffer\_block\_objects, FALSE, INFINITE)*

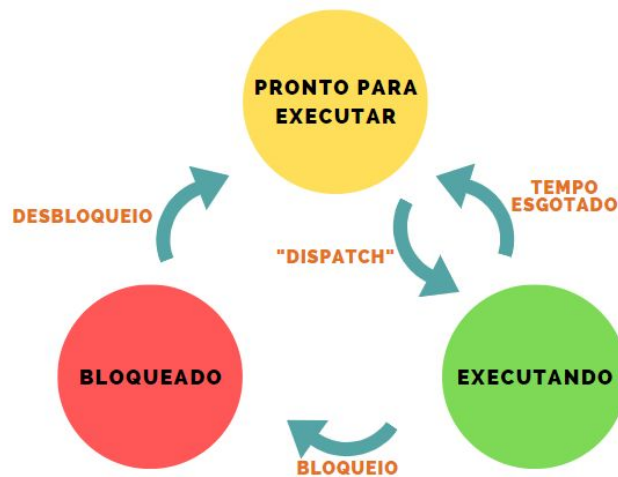


Figura 2 - Representação de estados

## 2.8. COMUNICAÇÃO ENTRE PROCESSOS (IPC)

A comunicação entre processos de uma mesma aplicação para realizar a troca de informações pode ocorrer duas formas:

- Síncrona, quando a troca de mensagens é executada de forma indivisível, ou seja, a *thread* emissora se bloqueia até que a *thread* receptora retire as informações e vice-versa, a *thread* receptora se bloqueia até que uma informação seja enviada pela *thread* emissora.
- Assíncrona, quando a troca de mensagens não é bloqueante, ou seja, após enviar ou aguardar a recepção de dados, a *thread* pode continuar sua execução.

Para este projeto, foram utilizados dois mecanismos de comunicação assíncronos, os arquivos mapeados em memória e os mailslots. Dessa forma foi possível evitar a possibilidade de ocorrência de deadlocks na troca de informações.

### 2.8.1. Arquivos compartilhados em memória

O método escolhido para estabelecer a comunicação entre a *tarefa de captura de mensagens* e a segunda lista circular foi a de arquivos mapeados em memória, que é um recurso da API Win32 com o objetivo de facilitar o acesso a arquivos em disco, em que sua manipulação é feita através de apontadores. Para esse tipo de comunicação as threads mapeiam diferentes “visões” que permitem troca de dados através de um mesmo arquivo.

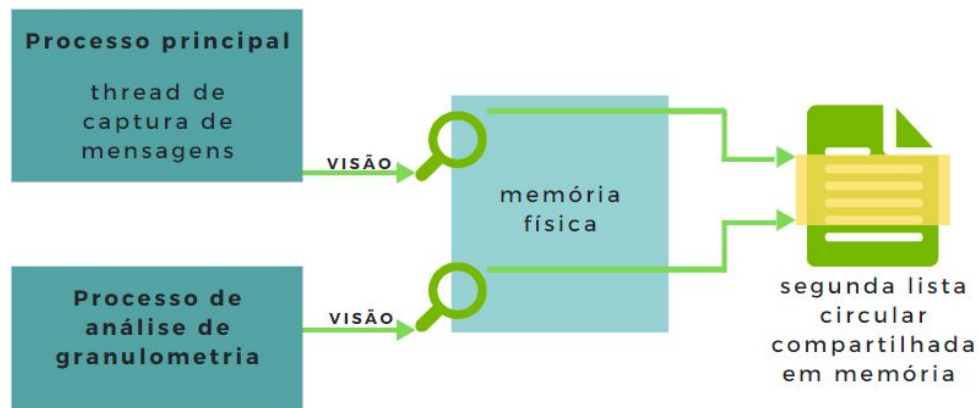


Figura 3 - Ilustração do funcionamento

### 2.8.2. Mailslots

Além de depositar mensagens na segunda lista em memória, a *tarefa de captura de mensagens* envia as mensagens provenientes do CLP diretamente para a *tarefa de exibição de dados*, que é um processo. Nesse caso, foi utilizado o mecanismo de mailslots que, como o próprio nome diz, simula o comportamento de enviar documentos (informações) a caixas de correio, para que o destinatário os recolha. Vale ressaltar que é um método unidirecional: a *thread* que cria o *mailslot* é chamada de servidora e é a única capaz de ler as mensagens do pseudo-arquivo, enquanto a *thread* que escreve no *mailslot* é chamada de cliente. No caso do software desenvolvido, a *thread* servidora é a *tarefa de exibição de dados*:

```

mailslot_mensagem = CreateMailslot(
    L"\\.\mailslot\lexibe_dados_mailslot_mensagem",
    0,
    0,
    NULL);
  
```

A *thread* cliente é a *tarefa de captura de mensagens*, no processo principal:

```

HANDLE mailslot = CreateFile(
    "\\.\mailslot\lexibe_dados_mailslot",
    GENERIC_WRITE,
    FILE_SHARE_READ,
  
```



```

NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL
);

```

Esse processo é utilizado para a sincronização do objeto evento “L”, responsável por limpar a tela de exibição de dados e para a exibição das mensagens de dados de processo. Vale ressaltar que apesar da comunicação ser assíncrona, a leitura e a escrita de dados é síncrona.

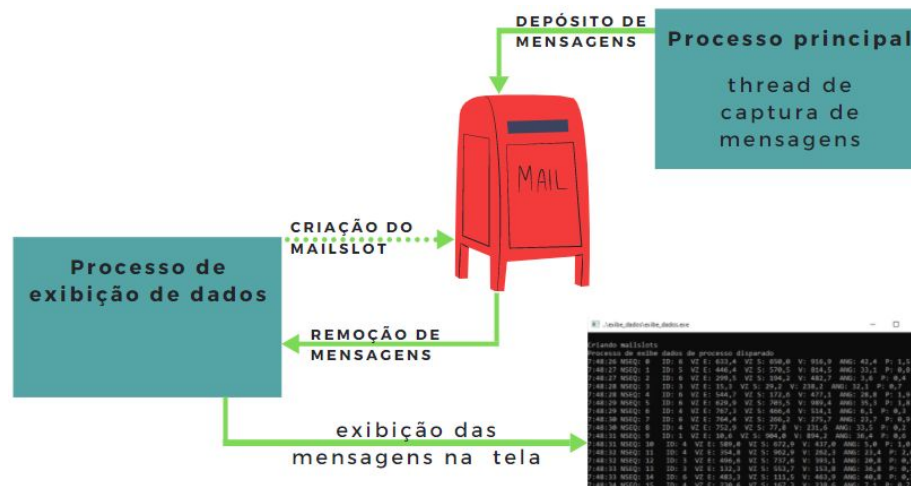


Figura 4 - Ilustração do funcionamento

## 2.9. THREADS

Threads são linhas de execução que compartilham todos recursos em comum do seu processo criador, promovendo simplicidade e rapidez de tratamento. Além disso, podem possuir os mesmos estados representados na Figura 5.



Figura 5 - Representação de processos e threads da aplicação

## 2.10. DADOS COMPARTILHADOS

### 2.10.1. Primeira lista circular em memória

A primeira lista circular em memória é compartilhada pelas tarefas de *leitura do sistema de medição*, de *leitura de dados de processo* e de *captura de mensagens*. Para acessar a lista, os *handles* dos semáforos de sincronização também são compartilhados entre as *threads*:

*HANDLE sem\_livre;*

*HANDLE sem\_ocupado;*

*HANDLE sem\_rw;*

Para encerrar o processo, outro *handle* é compartilhado (nesse caso, ele é sinalizado quando a tecla Esc é pressionada):

*HANDLE end\_event;*

Nota-se que o compartilhamento inter processo dos eventos é dado por meio da abertura do *handle* correspondente no processo secundário por meio da função *OpenHandle*. Para tal, os eventos compartilhados são nomeados.

### 2.10.2. Segunda lista circular em memória

A segunda lista circular em memória é compartilhada pelas tarefas de *captura de mensagens* e de *análise de granulometria* e, por se tratarem de *threads* de processos diferentes, foi utilizado um mecanismo de comunicação IPC (Interprocess Communication), detalhado no item 2.8. Os primeiros passos são a criação do arquivo pela *thread* de *captura de mensagens*, através da função *CreateFileMapping* e a abertura de uma “visão” do arquivo, através de *MapViewOfFile*.

```
CreateFileMapping( (HANDLE)0xFFFFFFFF, NULL, PAGE_READWRITE, 0,  
sizeof(Message) * buffer_2_size, "lista_2")
```

Feito isso, são criados dois apontadores para objetos dentro do arquivo com o objetivo de indicar as posições livres e ocupadas da lista e um apontador para indicação da mensagem transmitida. A *thread* de *análise de granulometria* abre o arquivo através de *OpenFileMapping* e mapeia suas visões dos indicadores de posições e das mensagens, através de *MapViewOfFile*. Para que isso ocorra, é aplicado um mecanismo de exclusão mútua baseado em semáforos para sinalizar os eventos de escrita e leitura na segunda lista (3.2)

```
OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, L"lista_2")
```

Observe que o espaço alocado em memória é relativo ao espaço da meta estrutura definida para a transmissão de mensagens, abordada na seção 2.11.3

## 2.11. MENSAGENS

A aplicação de software foi desenvolvida para permitir o controle e a supervisão dos dados representados nas mensagens provenientes do processo de produção e do sistema de medição que analisa as pelotas produzidas. Para simulá-las, foram criadas *structs* (também conhecidas como registros), dado que essas permitem o armazenamento de dados de tipos diferentes pertencentes a um mesmo elemento. Além disso, foram definidas três funções que permitem a manipulação dessas variáveis:

- *generate\_message\_gran* e *generate\_message\_clp*: responsáveis por gerar os números de composição dos dados e por registrar hora correspondente;

- `create_message`: referencia o nome da *struct* e atribui aos seus campos os dados gerados;
- `show_message`: formata e imprime as mensagens para exibição nos seguintes formatos para mensagens de granulometria e CLP, respectivamente:

HH:MM:SS NSEQ: ##### ID: ## GMED: ####.## GMAX: ####.## GMIN: ####.## SIG: ####.##

HH:MM:SS:MSS NSEQ: ##### ID:NN VZ E: #####.# VZ S:#####.# V:#####.# ANG:##.## P:#####.##

#### 2.11.1. Dados de processo

```
typedef struct{
    int tipo;
    int nseq;
    int id_disco;
    double vz_entrada;
    double vz_saida;
    double velocidade;
    double inclinacao;
    double potencia;
    Timestamp time;
} MessagePLC;
```

Mensagens geradas pela *tarefa de leitura de dados de processo* com o intuito de simular a leitura de informações provenientes de um CLP, seguindo o seguinte formato:

CAMPO	TAMANHO	TIPO	DESCRIÇÃO
TIPO	2	inteiro	Sempre "99"
NSEQ	4	inteiro	Número sequencial da mensagem [ 1 a 9999 ]
ID DISCO	2	inteiro	Identificador do disco de pelotamento [ 1 a 6 ]
VZ ENT	6	real	Vazão mássica de entrada da polpa de minério (kg/h) [ 0 a 1000 ]
VZ SAIDA	6	real	Vazão mássica de saída de pelotas cruas (kg/h) [ 0 a 1000 ]
V	6	real	Velocidade de rotação do disco (cm/s) [ 0 a 1000 ]
ANG	4	real	Ângulo de inclinação do disco (graus) [ 0 a 45 ]
POTENCIA	5	real	Potência consumida (kWh) [ 0 a 2 ]
TIMESTAMP	12	tempo	Hora, minuto, segundo e milissegundo

Tabela 1 - Formato dos dados de processo

#### 2.11.2. Dados de medição de granulometria

Mensagens geradas pela *tarefa de leitura do sistema de medição* com o intuito de simular a leitura de informações de análise de granulometria, seguindo o seguinte formato:

```
typedef struct{
    int tipo;
    int nseq;
    int id_disco;
    double gr_medio;
    double gr_max;
    double gr_min;
    double sigma;
    Timestamp time;
} MessageGranulometria;
```

CAMPO	TAMANHO	TIPO	DESCRIÇÃO
TIPO	2	inteiro	Sempre "00"
NSEQ	4	inteiro	Número sequencial da mensagem [ 1 a 9999 ]
ID DISCO	2	inteiro	Identificador do disco de pelotamento [ 1 a 2 ]
GR MED	6	real	Valor médio da granulometria das pelotas [ 0,0 a 100 mm ]
GR MAC	6	real	Valor máximo da granulometria das pelotas [ 0,0 a 100 mm]
GR MIN	6	real	Valor médio da granulometria das pelotas [ 0,0 a 100 mm]
SIGMA	6	real	Desvio padrão da granulometria [ 0,0 a 100 mm]
TIMESTAMP	8	tempo	Hora, minuto e segundo

Tabela 2 - Formato dos dados de granulometria

### 2.11.3. Metadados

A necessidade de comunicar dois tipos de dados distintos entre processos diferentes por meio de uma única lista, implementada como um *array*, demanda que a lista comporte esses dois tipos diferentes de dados. Em um contexto de orientação a objetos isso é implementado naturalmente por meio de superclasses e recursos de polimorfismo, mas a implementação das mensagens em classes foi evitada a fim de manter a complexidade do sistema um mínimo. Para contornar a limitação resultante de que a lista pode comportar apenas um tipo de mensagem criou-se uma meta estrutura responsável por armazenar como atributo os dois tipos de mensagens e com um identificador especial para sinalizar qual tipo de mensagem cada instância possui. A

comunicação das mensagens de fato é realizada transferindo essas meta estruturas e a identificação de qual mensagem é contida em cada meta estrutura é feita analisando o campo identificador. Como o enunciado sugere, esse identificador é 99 para mensagens provenientes da tarefa de leitura de dados de processo e 00 para mensagens provenientes da tarefa de leitura de granulometria. Ao recuperar a meta mensagem, esse campo é examinado para que a rotina de impressão correta seja invocada. Essa estrutura é definida como a seguir:

```
typedef struct{  
    MessagePLC plc;  
    MessageGranulometria granulometria;  
    int type;  
} Message;
```

### 3. CONTROLE

O controle da execução e acesso aos recursos compartilhados das tarefas foi feito por meio de objetos de sincronização que se resumem a semáforos contadores, semáforos binários e eventos. Optou-se por usar mecanismos simples e confiáveis para garantir boa interpretabilidade e desempenho. Observa-se que pela multiplicidade de maneiras possíveis e experimentadas de atender as especificações do projeto as soluções propostas foram semelhantes entre si, baseadas no mecanismo de *timeout* das funções de espera por objetos. Uma breve discussão sobre a motivação dessa escolha é travada na seção 5. Documentação e análise desse tipo de método foi encontrada no material “*Intel Guide For Developing Multithreaded Applications*” [4], principalmente as seções “*Choosing appropriate synchronization primitives to minimize overhead*” e “*Use non blocking locks when possible*”.

#### 3.1. SINCRONIZAÇÃO

O problema de sincronização se resume a coordenar a produção e consumo de dados nas listas circulares em memória. Ambas interações são identicamente análogas com exceção do número de produtores; a primeira lista possui dois e a segunda apenas um.

Essa diferença, no entanto, não interfere no método de sincronização de forma que a solução para ambas interações é idêntica. Seu detalhamento será feito portanto para a dinâmica da primeira lista apenas, observando que nada muda, esquematicamente, para a segunda lista circular. Ao fim desta seção há notas relevantes sobre as diferenças na sincronização da segunda lista.

Nessa interação há dois produtores (tarefa de leitura do sistema de medição e tarefa de leitura de dados do processo) e um consumidor (tarefa de captura de mensagens). As seguintes subseções descrevem por meio de quais objetos e como seu comportamento foi controlado. Nota-se que a solução descrita é muito semelhante à uma das que foram implementadas para o problema de produtores e consumidores no material didático [5].

#### 3.1.1. Objetos

A função de cada objeto, assim como o nome de sua variável e os valores de inicialização, serão listados nesta subseção. O processo usou duas variáveis globais responsáveis por armazenar a próxima posição livre para depósito de informações por parte dos dois produtores e a próxima posição ocupada para retirada de informações por parte do consumidor:

*int p\_livre = 0;*

*int p\_ocupado = 0;*

Os valores foram assim atribuídos por que inicialmente não há dados no buffer, portanto a próxima posição livre e ocupada é a primeira. Note que a inicialização dos semáforos foi documentada de forma simplificada, omitindo a declaração dos *handles* correspondentes.

Para controle do processo de escrita e leitura foi usado:

- Um semáforo binário, agindo efetivamente como mutex, para permitir apenas um processo a executar sua ação desejada por vez:

*sem\_rw = CreateSemaphore(NULL, 1, 1, NULL);*

O valor máximo do semáforo é 1 porque ele é binário e seu valor inicial é 1 porque a primeira *thread* que desejar escrever no banco deve conseguir (caso contrário ocorreria um *deadlock*);

- Um par de semáforos contadores para controlar a população de informações na lista, impedir que informações sejam sobrescritas ou lidas de maneira repetida e garantir que só sejam consumidas ou depositadas informações quando for necessário. Ao produzir um dado os produtores consomem uma sinalização do semáforo *sem\_livre*, efetivamente comunicando que o número das posições livres para depósito reduziu em 1. De maneira exatamente análoga, o único consumidor da lista, ao consumir um dado, consome uma sinalização do semáforo *sem\_ocupado* comunicando que o número de posições ocupadas, ou em que se encontram dados prontos para leitura, reduziu em 1:

*sem\_livre = CreateSemaphore(NULL, buffer\_size, buffer\_size, NULL);*

*sem\_ocupado = CreateSemaphore(NULL, 0, buffer\_size, NULL);*

Ambos semáforos devem ter valor máximo correspondente ao tamanho do *buffer* porque é possível que ocorram consecutivos depósitos ou consumos até o número de algum desses eventos se igualar a esse valor. O valor inicial de cada um é diferente, no entanto; como o *buffer* é inicializado sem dados, o número inicial de posições livres é igual ao seu tamanho e não há posições ocupadas.

### 3.1.2. Métodos

Semáforos são objetos de sincronização do *kernel* relacionados ao sincronismo entre *threads* e à comunicação entre processos, que implementam uma fila do tipo “*First In, First Out*”, associada a um contador, e são executados de forma atômica. O contador, por sua vez, indica o número de instâncias livres do recurso:

- *contador = 0*: podem existir N *threads* aguardando na fila para tomarem posse do semáforo;
- *contador > 1*: a fila está vazia.

Sendo assim, quando uma *thread* instancia um semáforo (correspondente a *Wait*), seu valor de contagem é decrementado em 1. Analogamente, quando uma *thread* libera o





posições livre e ocupada são mapeados em memória da mesma maneira que a segunda lista circular, especificada na seção 2.10. Observa-se que pelo fato do mapeamento ocorrer na mesma página um offset é realizado para impedir sobreposição indesejado entre os apontadores e a segunda lista de memória, como o trecho abaixo demonstra:

```
HANDLE mapped_memory_p_ocupado = CreateFileMapping(
    (HANDLE)0xFFFFFFFF,
    NULL,
    PAGE_READWRITE,
    0,
    sizeof(int),
    "p_ocupado");

int second_p_ocupado_offset = sizeof(Message) * buffer_2_size + 100;

int second_p_ocupado = (int)MapViewOfFile(
    mapped_memory_p_ocupado,
    FILE_MAP_WRITE,
    0,
    second_p_ocupado_offset,
    sizeof(int));
```

Como já observado o método de sincronização em si se mantém como descrito na seção 3.1.2

#### **4. TEMPORIZAÇÃO**

A temporização das tarefas é realizada por meio das propriedades de *time out* da função de *WaitForMultipleObjects* de maneira a aproveitar que o mesmo mecanismo é utilizado para uma série de requisitos de projeto como abordado nas seções 5.1 e 5.2. Além de poupar a implementação de objetos adicionais de temporização, aumentando

a simplicidade e interpretabilidade do sistema, esse método permite realizar as temporizações de maneira dinâmica de forma natural por meio do último parâmetro da função de espera.

#### 4.1. TAREFA DE LEITURA DO SISTEMA DE MEDIÇÃO

A periodicidade da simulação de leitura de mensagens dessa tarefa é um valor aleatório entre 1 e 5 segundos, controlada através da função *WaitForMultipleObjects*. Para isso, é gerado um número aleatório entre 1 e 5 e o valor é passado em segundos como *timeout* do método.

```
int rand_timeout = rand() % 5;
```

```
ret = WaitForMultipleObjects(2, Events, FALSE, rand_timeout * 1000);
```

Neste caso, a *thread* aguarda até o tempo limite a sinalização de dois objetos do kernel responsáveis pelo seu bloqueio (item 5.2) e, caso não sejam sinalizados, prossegue com a lógica.

```
if (ret == WAIT_TIMEOUT) { continue; }
```

Observe que o intervalo de tempo é determinado em segundos inteiros.

#### 4.2. TAREFA DE LEITURA DE DADOS DE PROCESSO

A periodicidade da simulação de leitura de mensagens dessa tarefa é fixo e igual a 500 ms, controlada através da função *WaitForMultipleObjects*. Como o *timeout* é especificado em milissegundos, não há a necessidade de conversão de unidade.

```
int local_timeout = 500;
```

```
ret = WaitForMultipleObjects(2, Events, FALSE, local_timeout);
```

Neste caso, a *thread* aguarda até o tempo limite a sinalização de dois objetos do kernel responsáveis pelo seu bloqueio (item 5.2) e, caso não sejam sinalizados, prossegue com a lógica.

```
if (ret == WAIT_TIMEOUT) { continue; }
```

## 5. ESPECIFICAÇÕES DE PROJETO

Essa seção aborda a maneira que as especificações de projeto foram atendidas, sendo elas o bloqueio de tarefas produtoras ao tentar depositar dados no *buffer* se ele estiver cheio, o bloqueio e desbloqueio de cada tarefa e o encerramento simultâneo de todas as tarefas por um mesmo sinal. Optou-se por utilizar ao máximo as propriedades de temporização das funções de *WaitForSingleObjects* e *WaitForMultipleObjects*, explorando as possibilidades de controle com o mecanismo de *timeout*, pelos seguintes motivos:

- Como as funções de espera por sinalização são usadas de qualquer maneira, está no interesse de simplicidade e eficiência usá-las para temporização também;
- O mecanismo usado é essencialmente o mesmo, tornando a compreensão do código como um todo mais simples;
- A temporização resultante é mais flexível por ser essencialmente parametrizável em tempo de execução;
- A natureza *soft real time* da aplicação permite que a granularidade e precisão da temporização seja a natural do sistema operacional.

### 5.1. ACESSO A LISTAS CIRCULARES

O acesso dos produtores ao *buffer* circular é mediado, necessariamente, pelos semáforos binários *sem\_livre* (subseção 3.1.1). Na API WIN32 a espera pela sinalização desse semáforo é feita pela função *WaitForSingleObject(x, y)* onde *x* corresponde ao objeto cuja sinalização é aguardada e *y* ao tempo da espera antes da função desistir e retornar um código de erro, sinalizando que o tempo de espera se esgotou. Propõe-se explorar a seguinte propriedade da produção de dados analisada: há dois estados possíveis para o contador do semáforo *sem\_livre* (que indica o número de posições livres no buffer), 0 e diferente de 0. Se o contador for igual a zero, o buffer está necessariamente cheio e o produtor que faz a tentativa de depósito vai esperar por um intervalo de tempo não nulo. Se o contador for diferente de zero, há espaço na lista e a *thread* não aguarda por tempo algum; a função *WaitForSingleObject* simplesmente

consome uma sinalização e a *thread* prossegue com sua execução. Conclui-se portanto que se o tempo de espera da função *WaitForSingleObject* for desprezível para a execução do programa como um todo, o sinal de *timeout* pode ser usado para indicar que o *buffer* estava cheio no momento da chamada da função, porque caso contrário não ocorreria espera alguma.

A *thread* executa uma tentativa de acesso à lista da seguinte maneira:

```
ret = WaitForSingleObject(sem_livre, timeout);
```

Em que o valor de *timeout* é 100 ms. Se o retorno da função informar que ocorreu o *timeout*, isso é, que não haviam posições imediatamente livres no *buffer*, conclui-se que o *buffer* estava cheio no momento da chamada.

No caso em que o *buffer* está cheio a *thread* executa uma chamada de *WaitForMultipleObjects* com tempo de espera infinito aguardando sinalização do evento de término ou do semáforo *sem\_livre* para prosseguir com sua execução, cada caso sendo tratado da maneira apropriada.

No caso em que o *buffer* possui posições livres a *thread* “passa reto” da chamada e deposita o dado na posição adequada.

## 5.2. BLOQUEIO E DESBLOQUEIO DAS THREADS E ENCERRAMENTO

O bloqueio e desbloqueio assim como o encerramento de todas as tarefas é realizada da mesma maneira, semelhante em conceito ao que foi discutido na subseção anterior. Inicializa-se um par de eventos:

```
end_event = CreateEvent(NULL, TRUE, FALSE, TEXT("end_event"));
```

```
leitura_dados_toggle_event = CreateEvent(NULL, FALSE, FALSE, NULL);
```

Observa-se que o segundo desses eventos é criado para cada uma das tarefas, com o intuito de bloquear e desbloquear cada uma individualmente. Uma diferença relevante

entre os eventos é seu tipo, cujo evento de encerramento (primeira linha do exemplo acima) é do tipo *reset* manual justamente porque é desejado que sua sinalização encerre *todas* as tarefas, sendo necessário que o sinal de encerramento não seja consumido em sua recepção. Já o evento de bloqueio e desbloqueio deve ter sua sinalização consumida, uma vez que é desejado que o estado da *thread* receptora seja alterado. Além disso, os eventos que são compartilhados entre os três processos da aplicação (como o evento de encerramento) foram criados de maneira nomeada e os que foram usados apenas pelo processo principal (como o exemplificado acima) foram criados de maneira não nomeada.

Dessa maneira o controle do bloqueio é feito por meio da seguinte chamada:

```
ret = WaitForMultipleObjects(2, Events, FALSE, timeout);
```

Em que *Events* é um vetor que armazena ambos eventos comentados no parágrafo anterior. As *threads* tratam o retorno *ret* de forma a identificar e reagir aos três possíveis casos das seguintes maneiras:

- Se ocorrer *timeout*: Caso normal para a *thread*. Pula para o próximo laço de sua execução, continuando seu comportamento esperado;
- Se for recebido o sinal do evento de bloqueio a *thread* executa uma outra chamada de *WaitForMultipleObjects* que se diferencia da anterior por aguardar os sinais indefinidamente. Se a *thread* se despertar nesse ponto por sinalização do evento de bloqueio e desbloqueio a execução continua normalmente;
- Se for recebido o sinal de encerramento a *thread* escapa do laço infinito e encerra sua execução.

Dessa forma, a função *WaitForMultipleObjects* funciona mais como uma verificação do que uma espera de fato, conferindo se desde a última passagem da execução naquele ponto houve sinalização pelos objetos de interesse. Se não, continua-se como se nada tivesse ocorrido (de fato nada ocorreu!).

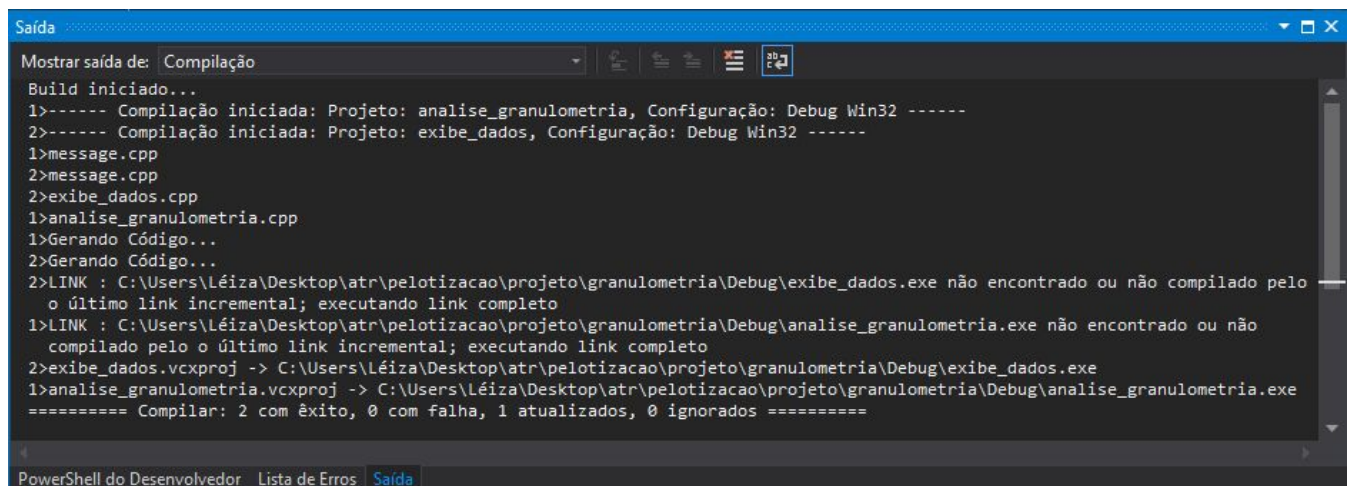
O encerramento é realizado pelo evento correspondente, cujo sinal é disparado pela tecla “Esc”. A única observação pertinente sobre seu comportamento é que foi necessário ter o cuidado de incluir uma observação desse evento e consequente reação à sua sinalização sempre que uma função de espera (*WaitFor\**) com tempo de espera infinito for chamada, para garantir que não ocorresse um *deadlock* resultante de uma *thread* aguardando por um sinal que não pode ser disparado devido ao encerramento da *thread* principal.

Recomenda-se que seja recorrido ao código e seus minuciosos comentários para a compreensão dos detalhes mais finos da lógica descrita

## 6. TESTES DE EXECUÇÃO

Observação importante: Após os testes, foi reparado que a função “*show\_message*” não exibia na tela de exibição de dados de processo o tempo com milissegundos (o formato exibido era HH:MM:SS, enquanto deveria ser HH:MM:SS:MSS), **apesar de registrá-lo corretamente na *struct***. Por não interferir na lógica de funcionamento, os testes apresentados abaixo permaneceram no formato antigo de forma a evitar retrabalho.

### 6.1. COMPILAÇÃO



```
Saída
Mostrar saída de: Compilação
Build iniciado...
1>----- Compilação iniciada: Projeto: analise_granulometria, Configuração: Debug Win32 -----
2>----- Compilação iniciada: Projeto: exibe_dados, Configuração: Debug Win32 -----
1>message.cpp
2>message.cpp
2>exibe_dados.cpp
1>analise_granulometria.cpp
1>Gerando Código...
2>Gerando Código...
2>LINK : C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\exibe_dados.exe não encontrado ou não compilado pelo
o último link incremental; executando link completo
1>LINK : C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\analise_granulometria.exe não encontrado ou não
compilado pelo o último link incremental; executando link completo
2>exibe_dados.vcxproj -> C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\exibe_dados.exe
1>analise_granulometria.vcxproj -> C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\analise_granulometria.exe
===== Compilar: 2 com êxito, 0 com falha, 1 atualizados, 0 ignorados =====
```

Figura 8 - Compilação

### 6.2. DEPURAÇÃO

#### 6.2.1. Inicialização dos processos



```
..\analise_granulometria\analise_granulometria.exe

Processo analise de granulometria disparado
7:48:26 NSEQ: 0 ID: 2 GMED: 74,00 GMAX: 84,67 GMIN: 63,34 SIG: 6500,00
7:48:30 NSEQ: 1 ID: 1 GMED: 54,18 GMAX: 93,58 GMIN: 14,78 SIG: 6962,00
7:48:34 NSEQ: 2 ID: 2 GMED: 57,13 GMAX: 81,45 GMIN: 32,81 SIG: 6827,00
7:48:35 NSEQ: 3 ID: 2 GMED: 24,68 GMAX: 29,95 GMIN: 19,42 SIG: 4827,00
7:48:36 NSEQ: 4 ID: 2 GMED: 42,53 GMAX: 46,04 GMIN: 39,02 SIG: 153,00
7:48:38 NSEQ: 5 ID: 1 GMED: 80,69 GMAX: 87,16 GMIN: 74,21 SIG: 9718,00
7:48:38 NSEQ: 6 ID: 2 GMED: 32,48 GMAX: 47,71 GMIN: 17,26 SIG: 1538,00
7:48:42 NSEQ: 7 ID: 1 GMED: 59,83 GMAX: 62,99 GMIN: 56,67 SIG: 7035,00
7:48:46 NSEQ: 8 ID: 2 GMED: 25,66 GMAX: 38,11 GMIN: 13,22 SIG: 333,00
7:48:49 NSEQ: 9 ID: 1 GMED: 64,26 GMAX: 77,11 GMIN: 51,41 SIG: 8253,00
7:48:52 NSEQ: 10 ID: 2 GMED: 51,53 GMAX: 76,44 GMIN: 26,62 SIG: 2757,00
7:48:54 NSEQ: 11 ID: 2 GMED: 92,32 GMAX: 97,41 GMIN: 87,23 SIG: 7529,00
7:48:57 NSEQ: 12 ID: 1 GMED: 26,13 GMAX: 30,35 GMIN: 21,90 SIG: 1842,00
7:49:0 NSEQ: 13 ID: 1 GMED: 89,91 GMAX: 90,40 GMIN: 89,42 SIG: 9264,00
7:49:3 NSEQ: 14 ID: 1 GMED: 48,48 GMAX: 58,90 GMIN: 38,05 SIG: 6729,00
7:49:3 NSEQ: 15 ID: 1 GMED: 30,54 GMAX: 50,06 GMIN: 11,01 SIG: 4393,00
```

Figura 9 - Processos analise\_granulometria

```
..\exibe_dados\exibe_dados.exe

Criando mailslots
Processo de exibe dados de processo disparado
7:48:26 NSEQ: 0 ID: 6 VZ E: 633,4 VZ S: 650,0 V: 916,9 ANG: 42,4 P: 1,5
7:48:27 NSEQ: 1 ID: 5 VZ E: 446,4 VZ S: 570,5 V: 814,5 ANG: 33,1 P: 0,8
7:48:27 NSEQ: 2 ID: 6 VZ E: 299,5 VZ S: 194,2 V: 482,7 ANG: 3,6 P: 0,4
7:48:28 NSEQ: 3 ID: 3 VZ E: 15,3 VZ S: 29,2 V: 238,2 ANG: 32,1 P: 0,7
7:48:28 NSEQ: 4 ID: 6 VZ E: 544,7 VZ S: 172,6 V: 477,1 ANG: 28,8 P: 1,9
7:48:29 NSEQ: 5 ID: 6 VZ E: 629,9 VZ S: 703,5 V: 989,4 ANG: 35,3 P: 1,8
7:48:29 NSEQ: 6 ID: 4 VZ E: 767,3 VZ S: 466,4 V: 514,1 ANG: 6,1 P: 0,3
7:48:30 NSEQ: 7 ID: 6 VZ E: 764,4 VZ S: 266,2 V: 275,7 ANG: 23,7 P: 0,9
7:48:30 NSEQ: 8 ID: 4 VZ E: 752,9 VZ S: 77,8 V: 231,6 ANG: 33,5 P: 0,2
7:48:31 NSEQ: 9 ID: 1 VZ E: 10,6 VZ S: 904,0 V: 894,2 ANG: 36,4 P: 0,6
7:48:31 NSEQ: 10 ID: 4 VZ E: 589,0 VZ S: 672,9 V: 437,0 ANG: 5,0 P: 1,0
7:48:32 NSEQ: 11 ID: 4 VZ E: 354,8 VZ S: 962,9 V: 262,3 ANG: 23,4 P: 2,0
7:48:32 NSEQ: 12 ID: 3 VZ E: 496,6 VZ S: 737,6 V: 393,1 ANG: 20,8 P: 0,9
7:48:33 NSEQ: 13 ID: 3 VZ E: 132,3 VZ S: 553,7 V: 153,8 ANG: 36,8 P: 0,1
7:48:33 NSEQ: 14 ID: 6 VZ E: 483,3 VZ S: 111,5 V: 463,9 ANG: 40,8 P: 0,7
7:48:34 NSEQ: 15 ID: 4 VZ E: 230,6 VZ S: 167,3 V: 238,6 ANG: 7,1 P: 0,7
```

Figura 10 - Processos exibe\_dados



```
C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\granulometria.exe

thread leitura de medicao criada com id = 299c
thread leitura dados criada com id = e90
thread captura mensagens criada com id = 3660

SISTEMA DE CONTROLE E SUPERVISÃO

Processo de pelotização e medição de granulometria
.....
Bloqueio/retomada da execução de tarefas:
Tecla g: Tarefa de leitura de dados de medição
Tecla c: Tarefa de leitura de dados de processo
Tecla r: Tarefa de captura de mensagens
Tecla p: Tarefa de exibição de dados de processo
Tecla a: Tarefa de análise de granulometria
.....
Tecla l: Limpar tela de monitor de exibição de dados de processo
.....
Para encerrar o processo: ESC
```

Figura 11 - Processo principal

```
C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\granulometria.exe

thread leitura de medicao criada com id = 299c
thread leitura dados criada com id = e90
thread captura mensagens criada com id = 3660

SISTEMA DE CONTROLE E SUPERVISÃO

Processo de pelotização e medição de granulometria
.....
Bloqueio/retomada da execução de tarefas:
Tecla g: Tarefa de leitura de dados de medição
Tecla c: Tarefa de leitura de dados de processo
Tecla r: Tarefa de captura de mensagens
Tecla p: Tarefa de exibição de dados de processo
Tecla a: Tarefa de análise de granulometria
.....
Tecla l: Limpar tela de monitor de exibição de dados de processo
.....
Para encerrar o processo: ESC

.....
Tarefa CAPTURA DE MENSAGENS *BLOQUEADA*. Para desbloquear, tecle <r>.
.....
```

Figura 12 - Bloqueio da thread consumidora “captura\_mensagens”

```
..\exibe_dados\exibe_dados.exe
Criando mailslots
Processo de exibe dados de processo disparado
7:54:24 NSEQ: 0 ID: 6 VZ E: 633,4 VZ S: 650,0 V: 916,9 ANG: 42,4 P: 1,5
7:54:24 NSEQ: 1 ID: 5 VZ E: 446,4 VZ S: 570,5 V: 814,5 ANG: 33,1 P: 0,8
7:54:25 NSEQ: 2 ID: 6 VZ E: 299,5 VZ S: 194,2 V: 482,7 ANG: 3,6 P: 0,4
7:54:25 NSEQ: 3 ID: 3 VZ E: 15,3 VZ S: 29,2 V: 238,2 ANG: 32,1 P: 0,7

..\exibe_dados\exibe_dados.exe
7:57:7 NSEQ: 321 ID: 5 VZ E: 106,9 VZ S: 515,4 V: 352,9 ANG: 5,2 P: 0,1
7:57:7 NSEQ: 322 ID: 5 VZ E: 224,4 VZ S: 84,4 V: 304,9 ANG: 16,8 P: 0,1
7:57:8 NSEQ: 323 ID: 5 VZ E: 877,3 VZ S: 847,0 V: 973,1 ANG: 44,7 P: 1,5
7:57:8 NSEQ: 324 ID: 5 VZ E: 49,8 VZ S: 710,3 V: 735,2 ANG: 2,9 P: 0,1
7:57:9 NSEQ: 325 ID: 3 VZ E: 108,8 VZ S: 156,3 V: 583,4 ANG: 40,0 P: 1,0
7:57:9 NSEQ: 326 ID: 6 VZ E: 949,2 VZ S: 565,1 V: 858,0 ANG: 17,7 P: 1,6
7:57:10 NSEQ: 327 ID: 3 VZ E: 522,0 VZ S: 461,5 V: 234,8 ANG: 24,8 P: 0,2
7:57:10 NSEQ: 328 ID: 4 VZ E: 288,3 VZ S: 166,2 V: 890,2 ANG: 36,2 P: 1,4
7:57:11 NSEQ: 329 ID: 3 VZ E: 927,3 VZ S: 584,1 V: 268,6 ANG: 8,8 P: 1,9
7:57:11 NSEQ: 330 ID: 3 VZ E: 369,8 VZ S: 126,7 V: 174,9 ANG: 6,5 P: 0,4
7:57:12 NSEQ: 331 ID: 6 VZ E: 608,1 VZ S: 200,3 V: 913,0 ANG: 42,8 P: 0,6
7:57:12 NSEQ: 332 ID: 5 VZ E: 863,0 VZ S: 917,2 V: 986,4 ANG: 10,7 P: 0,3
7:57:13 NSEQ: 333 ID: 6 VZ E: 849,0 VZ S: 661,0 V: 617,7 ANG: 2,1 P: 1,2
7:57:13 NSEQ: 334 ID: 4 VZ E: 684,0 VZ S: 763,3 V: 703,7 ANG: 4,3 P: 0,6
7:57:14 NSEQ: 335 ID: 4 VZ E: 878,2 VZ S: 220,3 V: 746,1 ANG: 19,0 P: 1,7
```

Figura 13 - Início e término do teste de bloqueio (exibe\_dados)

**Comentário:** A tarefa de captura de mensagens foi bloqueada, o que resultou na parada de transferência de mensagens para os dois processos secundários. Esse teste foi disparado às 7h 54min e 24s e interrompido às 7h 54min e 14s (170 segundos de execução), como representando pela Figura 13. Como foram geradas e transmitidas 335 mensagens durante esse período, é possível perceber que a periodicidade da leitura de mensagens provenientes do CLP foi igual a 500ms.

**Comentário:** Por outro lado, no processo de exibição de dados de granulometria da Figura 14 nota-se que foram geradas e transmitidas 77 mensagens, um valor aceitável, dado que a periodicidade da tarefa de leitura de dados do sistema de medição é aleatória entre 1s e 5s, ou seja, por não ser tendenciosa, é esperado que a periodicidade média esteja próxima à 2,5 s (neste caso, foi igual a 2,2 s).

**Comentário:** Como a thread consumidora foi bloqueada, as tarefas de leitura de dados continuaram a depositar informações na primeira lista em memória, fazendo com que sua capacidade máxima fosse alcançada, como representado na Figura 15, em que as



duas *threads* tentam depositar informação novamente mas são alertadas e bloqueadas. Após o desbloqueio, os processos secundários voltam a exibir os dados.

```

..\analise_granulometria\analise_granulometria.exe

Processo analise de granulometria disparado
7:54:24 NSEQ: 0 ID: 2 GMED: 74,00 GMAX: 84,67 GMIN: 63,34 SIG: 6500,00
7:54:28 NSEQ: 1 ID: 1 GMED: 54,18 GMAX: 93,58 GMIN: 14,78 SIG: 6962,00
7:54:32 NSEQ: 2 ID: 2 GMED: 57,13 GMAX: 81,45 GMIN: 32,81 SIG: 6827,00
7:54:33 NSEQ: 3 ID: 2 GMED: 24,68 GMAX: 29,95 GMIN: 19,42 SIG: 4827,00

..\analise_granulometria\analise_granulometria.exe
7:56:50 NSEQ: 67 ID: 2 GMED: 9,94 GMAX: 13,16 GMIN: 6,71 SIG: 5786,00
7:56:53 NSEQ: 68 ID: 2 GMED: 27,70 GMAX: 43,55 GMIN: 11,85 SIG: 53,00
7:56:55 NSEQ: 69 ID: 1 GMED: 13,89 GMAX: 18,32 GMIN: 9,45 SIG: 4313,00
7:56:56 NSEQ: 70 ID: 2 GMED: 66,02 GMAX: 95,58 GMIN: 36,46 SIG: 7982,00
7:56:57 NSEQ: 71 ID: 1 GMED: 17,09 GMAX: 31,96 GMIN: 2,22 SIG: 7129,00
7:56:58 NSEQ: 72 ID: 2 GMED: 8,12 GMAX: 11,73 GMIN: 4,50 SIG: 466,00
7:57:2 NSEQ: 73 ID: 2 GMED: 63,66 GMAX: 64,39 GMIN: 62,92 SIG: 7253,00
7:57:6 NSEQ: 74 ID: 1 GMED: 71,28 GMAX: 95,10 GMIN: 47,45 SIG: 649,00
7:57:7 NSEQ: 75 ID: 2 GMED: 62,48 GMAX: 80,22 GMIN: 44,74 SIG: 2168,00
7:57:10 NSEQ: 76 ID: 2 GMED: 89,31 GMAX: 99,05 GMIN: 79,58 SIG: 7391,00
7:57:12 NSEQ: 77 ID: 2 GMED: 54,45 GMAX: 64,77 GMIN: 44,14 SIG: 9314,00

```

Figura 14 - Início e término do teste de bloqueio (analise\_granulometria)

```

C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\granulometria.exe

Processo de pelotização e medição de granulometria
.....
Bloqueio/retomada da execução de tarefas:
Tecla g: Tarefa de leitura de dados de medição
Tecla c: Tarefa de leitura de dados de processo
Tecla r: Tarefa de captura de mensagens
Tecla p: Tarefa de exibição de dados de processo
Tecla a: Tarefa de análise de granulometria
.....
Tecla l: Limpar tela de monitor de exibição de dados de processo
.....
Para encerrar o processo: ESC

.....
Tarefa CAPTURA DE MENSAGENS *BLOQUEADA*. Para desbloquear, tecle <r>.
.....

*****
Capacidade máxima da primeira lista circular em memória atingida.
Thread de leitura de dados do processo tentou depositar informação e está se bloqueando até livrar posição.
*****

*****
Capacidade máxima da primeira lista circular em memória atingida.
Thread de leitura do sistema de medição tentou depositar informação e está se bloqueando até livrar posição.
*****

```

Figura 15 - Capacidade máxima da primeira lista circular atingida

**Comentário:** Para observar o bloqueio da segunda lista circular em memória (Figura 16) foi realizado o bloqueio da *tarefa de análise de granulometria*, consumidora.

```
C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\granulometria.exe
thread leitura de medicao criada com id = 1fa0
thread leitura dados criada com id = 2884
thread captura mensagens criada com id = 1924

SISTEMA DE CONTROLE E SUPERVISÃO

Processo de pelotização e medição de granulometria
.....
Bloqueio/retomada da execução de tarefas:
Tecla g: Tarefa de leitura de dados de medição
Tecla c: Tarefa de leitura de dados de processo
Tecla r: Tarefa de captura de mensagens
Tecla p: Tarefa de exibição de dados de processo
Tecla a: Tarefa de análise de granulometria
.....
Tecla l: Limpar tela de monitor de exibição de dados de processo
.....
Para encerrar o processo: ESC

*****
Capacidade máxima da segunda lista circular em memória atingida.
Thread de captura de dados tentou depositar informação e está se bloqueando até livrar posição.
*****
```

Figura 16 - Capacidade máxima da segunda lista circular atingida

..analise_granulometria\analise_granulometria.exe						
7:58:25	NSEQ: 113	ID: 1	GMED: 48,65	GMAX: 96,17	GMIN: 1,12	SIG: 2717,00
7:58:26	NSEQ: 114	ID: 2	GMED: 42,32	GMAX: 44,23	GMIN: 40,41	SIG: 4129,00
7:58:30	NSEQ: 115	ID: 2	GMED: 77,45	GMAX: 89,32	GMIN: 65,59	SIG: 2296,00
7:58:30	NSEQ: 116	ID: 2	GMED: 52,73	GMAX: 69,62	GMIN: 35,84	SIG: 9734,00
7:58:34	NSEQ: 117	ID: 1	GMED: 29,13	GMAX: 43,69	GMIN: 14,57	SIG: 2532,00
8:20:11	NSEQ: 118	ID: 2	GMED: 16,97	GMAX: 24,83	GMIN: 9,11	SIG: 1635,00
8:20:13	NSEQ: 119	ID: 1	GMED: 38,06	GMAX: 46,75	GMIN: 29,38	SIG: 2223,00
8:20:15	NSEQ: 120	ID: 1	GMED: 46,26	GMAX: 65,11	GMIN: 27,41	SIG: 175,00
8:20:19	NSEQ: 121	ID: 2	GMED: 55,45	GMAX: 78,70	GMIN: 32,21	SIG: 1626,00

Figura 17 - Bloqueio das tarefas de leitura - análise temporal

**Comentário:** Na figura acima é possível perceber que não foram gerados dados enquanto as tarefas de leituras ficaram bloqueadas. O mesmo ocorre no processo “exibe\_dados”.

**Comentário:** Na figura abaixo é possível perceber que ao bloquear a *tarefa de leitura do sistema de medição*, apenas dados do processo provenientes do CLP são lidos e transmitidos ao processo de exibição. O mesmo acontece para a *tarefa de leitura de dados do processo*.



```
..\analise_granulometria\analise_granulometria.exe
8:27:22 NSEQ: 290 ID: 1 GMED: 38,37 GMAX: 58,34 GMIN: 18,40 SIG: 1497,00
8:27:22 NSEQ: 291 ID: 2 GMED: 51,73 GMAX: 88,05 GMIN: 15,40 SIG: 8791,00
8:27:24 NSEQ: 292 ID: 1 GMED: 25,64 GMAX: 35,49 GMIN: 15,78 SIG: 6979,00
8:27:25 NSEQ: 293 ID: 2 GMED: 1,33 GMAX: 1,93 GMIN: 0,73 SIG: 1620,00
8:27:27 NSEQ: 294 ID: 1 GMED: 55,33 GMAX: 97,90 GMIN: 12,76 SIG: 6582,00
8:27:30 NSEQ: 295 ID: 2 GMED: 34,53 GMAX: 64,89 GMIN: 4,18 SIG: 159,00
8:27:34 NSEQ: 296 ID: 1 GMED: 47,26 GMAX: 90,72 GMIN: 3,80 SIG: 7008,00
8:27:36 NSEQ: 297 ID: 1 GMED: 74,90 GMAX: 85,03 GMIN: 64,77 SIG: 5370,00

..\exibe_dados\exibe_dados.exe
8:29:5 NSEQ: 1400 ID: 4 VZ E: 762,9 VZ S: 111,8 V: 11,1 ANG: 15,2 P: 0,3
8:29:6 NSEQ: 1401 ID: 4 VZ E: 294,2 VZ S: 449,7 V: 918,8 ANG: 22,2 P: 0,8
8:29:6 NSEQ: 1402 ID: 6 VZ E: 534,8 VZ S: 283,1 V: 855,0 ANG: 39,6 P: 1,7
8:29:7 NSEQ: 1403 ID: 2 VZ E: 411,7 VZ S: 632,4 V: 390,6 ANG: 17,1 P: 0,2
8:29:7 NSEQ: 1404 ID: 3 VZ E: 440,9 VZ S: 246,9 V: 588,1 ANG: 3,9 P: 0,1
8:29:8 NSEQ: 1405 ID: 6 VZ E: 15,4 VZ S: 668,5 V: 405,0 ANG: 29,3 P: 1,2
8:29:8 NSEQ: 1406 ID: 3 VZ E: 800,8 VZ S: 193,4 V: 872,7 ANG: 12,2 P: 1,7
8:29:9 NSEQ: 1407 ID: 3 VZ E: 497,2 VZ S: 265,0 V: 500,8 ANG: 41,9 P: 1,6
8:29:9 NSEQ: 1408 ID: 2 VZ E: 357,3 VZ S: 370,5 V: 464,4 ANG: 15,8 P: 1,4
8:29:10 NSEQ: 1409 ID: 2 VZ E: 793,4 VZ S: 875,2 V: 898,3 ANG: 42,4 P: 1,0

Tarefa LEITURA DO SISTEMA DE MEDIÇÃO *BLOQUEADA*. Para desbloquear, tecle <g>.
```

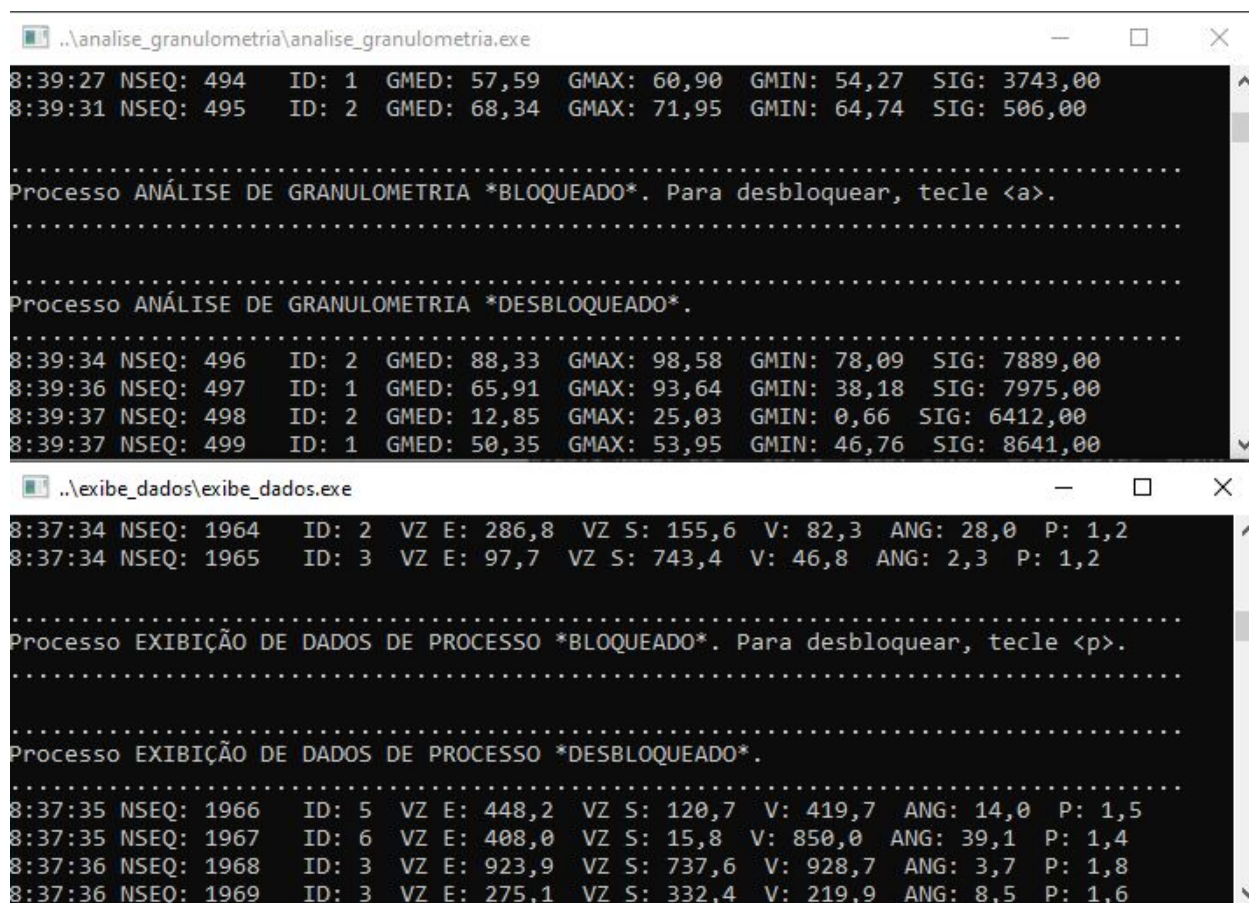
Figura 18 - Bloqueio da tarefa de leitura do sistema de medição

```
..\analise_granulometria\analise_granulometria.exe
8:34:14 NSEQ: 334 ID: 2 GMED: 93,03 GMAX: 99,26 GMIN: 86,80 SIG: 5678,00
8:34:14 NSEQ: 335 ID: 2 GMED: 45,80 GMAX: 49,61 GMIN: 41,99 SIG: 855,00
8:34:18 NSEQ: 336 ID: 1 GMED: 10,67 GMAX: 15,61 GMIN: 5,73 SIG: 3245,00
8:34:21 NSEQ: 337 ID: 1 GMED: 29,52 GMAX: 43,53 GMIN: 15,50 SIG: 1181,00
8:34:23 NSEQ: 338 ID: 2 GMED: 83,77 GMAX: 86,43 GMIN: 81,10 SIG: 7465,00
8:34:25 NSEQ: 339 ID: 2 GMED: 60,47 GMAX: 99,81 GMIN: 21,12 SIG: 3476,00
8:34:26 NSEQ: 340 ID: 2 GMED: 67,81 GMAX: 68,90 GMIN: 66,71 SIG: 8805,00
8:34:28 NSEQ: 341 ID: 1 GMED: 66,55 GMAX: 93,20 GMIN: 39,89 SIG: 3165,00
8:34:29 NSEQ: 342 ID: 1 GMED: 42,49 GMAX: 72,06 GMIN: 12,93 SIG: 6578,00
8:34:32 NSEQ: 343 ID: 1 GMED: 76,69 GMAX: 81,66 GMIN: 71,71 SIG: 3396,00

..\exibe_dados\exibe_dados.exe
8:33:21 NSEQ: 1885 ID: 4 VZ E: 650,9 VZ S: 41,0 V: 573,4 ANG: 26,6 P: 2,0
8:33:21 NSEQ: 1886 ID: 5 VZ E: 587,8 VZ S: 995,1 V: 285,9 ANG: 33,7 P: 1,0
8:33:22 NSEQ: 1887 ID: 2 VZ E: 593,7 VZ S: 206,0 V: 61,0 ANG: 44,7 P: 0,1
8:33:22 NSEQ: 1888 ID: 4 VZ E: 476,3 VZ S: 868,3 V: 70,4 ANG: 34,9 P: 1,0
8:33:23 NSEQ: 1889 ID: 3 VZ E: 520,6 VZ S: 592,7 V: 691,6 ANG: 15,8 P: 0,6
8:33:23 NSEQ: 1890 ID: 5 VZ E: 560,9 VZ S: 487,8 V: 370,1 ANG: 14,9 P: 0,3
8:33:24 NSEQ: 1891 ID: 4 VZ E: 355,5 VZ S: 40,1 V: 593,9 ANG: 11,5 P: 1,0
8:33:24 NSEQ: 1892 ID: 4 VZ E: 251,9 VZ S: 962,1 V: 52,4 ANG: 42,8 P: 0,0
8:33:25 NSEQ: 1893 ID: 5 VZ E: 379,3 VZ S: 868,7 V: 112,1 ANG: 44,5 P: 0,2
8:33:25 NSEQ: 1894 ID: 4 VZ E: 802,8 VZ S: 93,0 V: 64,0 ANG: 28,9 P: 1,2

Tarefa LEITURA DE DADOS DO PROCESSO *BLOQUEADA*. Para desbloquear, tecle <c>.
```

Figura 19 - Bloqueio da tarefa de leitura de dados do processo



The image contains two screenshots of command-line windows. The top window, titled '..\analise\_granulometria\analise\_granulometria.exe', shows a list of process data with columns for time, NSEQ, ID, GMED, GMAX, GMIN, and SIG. It displays a transition from a blocked state ('\*BLOQUEADO\*') to an unblocked state ('\*DESBLOQUEADO\*') after a user input. The bottom window, titled '..\exibe\_dados\exibe\_dados.exe', shows a similar list of process data with columns for time, NSEQ, ID, VZ E, VZ S, V, ANG, and P. It also shows a transition from a blocked state ('\*BLOQUEADO\*') to an unblocked state ('\*DESBLOQUEADO\*') after a user input.

```
8:39:27 NSEQ: 494 ID: 1 GMED: 57,59 GMAX: 60,90 GMIN: 54,27 SIG: 3743,00
8:39:31 NSEQ: 495 ID: 2 GMED: 68,34 GMAX: 71,95 GMIN: 64,74 SIG: 506,00

.....
Processo ANÁLISE DE GRANULOMETRIA *BLOQUEADO*. Para desbloquear, tecle <a>.
.....

.....
Processo ANÁLISE DE GRANULOMETRIA *DESBLOQUEADO*.
.....

8:39:34 NSEQ: 496 ID: 2 GMED: 88,33 GMAX: 98,58 GMIN: 78,09 SIG: 7889,00
8:39:36 NSEQ: 497 ID: 1 GMED: 65,91 GMAX: 93,64 GMIN: 38,18 SIG: 7975,00
8:39:37 NSEQ: 498 ID: 2 GMED: 12,85 GMAX: 25,03 GMIN: 0,66 SIG: 6412,00
8:39:37 NSEQ: 499 ID: 1 GMED: 50,35 GMAX: 53,95 GMIN: 46,76 SIG: 8641,00

.....
8:37:34 NSEQ: 1964 ID: 2 VZ E: 286,8 VZ S: 155,6 V: 82,3 ANG: 28,0 P: 1,2
8:37:34 NSEQ: 1965 ID: 3 VZ E: 97,7 VZ S: 743,4 V: 46,8 ANG: 2,3 P: 1,2

.....
Processo EXIBIÇÃO DE DADOS DE PROCESSO *BLOQUEADO*. Para desbloquear, tecle <p>.
.....

.....
Processo EXIBIÇÃO DE DADOS DE PROCESSO *DESBLOQUEADO*.
.....

8:37:35 NSEQ: 1966 ID: 5 VZ E: 448,2 VZ S: 120,7 V: 419,7 ANG: 14,0 P: 1,5
8:37:35 NSEQ: 1967 ID: 6 VZ E: 408,0 VZ S: 15,8 V: 850,0 ANG: 39,1 P: 1,4
8:37:36 NSEQ: 1968 ID: 3 VZ E: 923,9 VZ S: 737,6 V: 928,7 ANG: 3,7 P: 1,8
8:37:36 NSEQ: 1969 ID: 3 VZ E: 275,1 VZ S: 332,4 V: 219,9 ANG: 8,5 P: 1,6
```

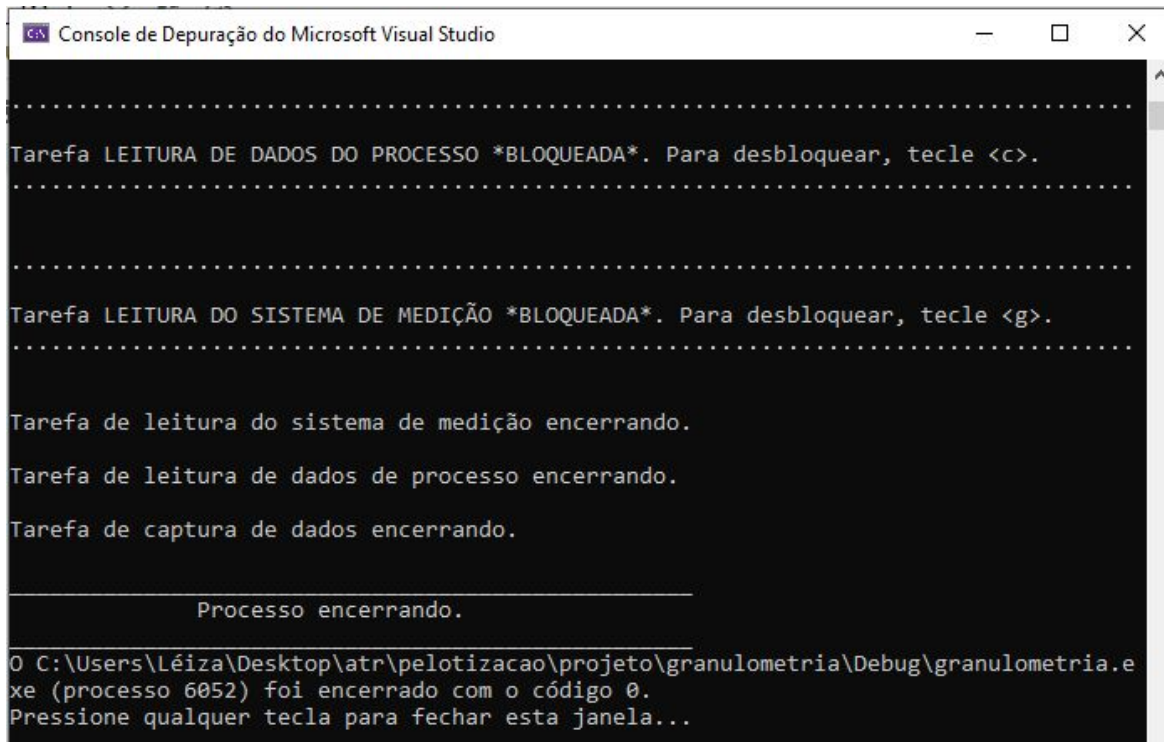
Figura 20 - Bloqueio e desbloqueio dos processos secundários

**Comentário:** Mesmo com os processos bloqueados durante alguns segundos, após efetuar o desbloqueio as mensagens informam o momento em que foram geradas, e não exibidas ao operador, representado na figura acima.



Figura 21 - Limpeza dos dados da tarefa de exibição de dados





```
.....
Tarefa LEITURA DE DADOS DO PROCESSO *BLOQUEADA*. Para desbloquear, tecle <c>.
.....

Tarefa LEITURA DO SISTEMA DE MEDIÇÃO *BLOQUEADA*. Para desbloquear, tecle <g>.
.....

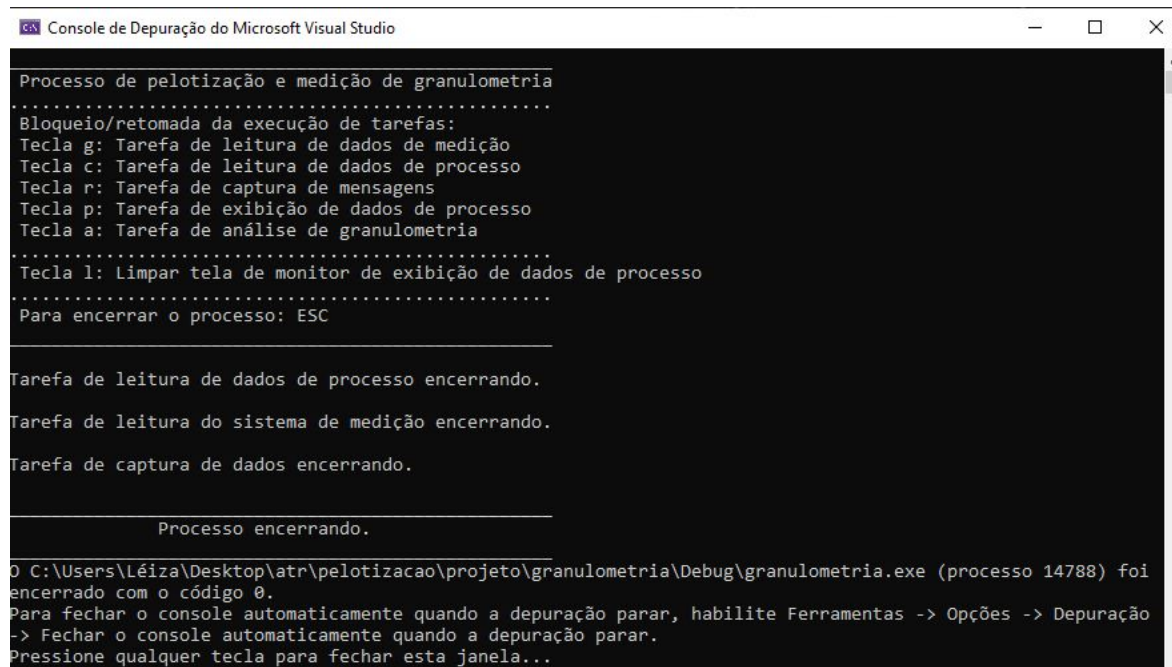
Tarefa de leitura do sistema de medição encerrando.
Tarefa de leitura de dados de processo encerrando.
Tarefa de captura de dados encerrando.

Processo encerrando.

O C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\granulometria.exe (processo 6052) foi encerrado com o código 0.
Pressione qualquer tecla para fechar esta janela...
```

Figura 22 - Encerramento do processo principal após bloqueio

**Comentário:** Para o teste representado acima, todas as tarefas foram bloqueadas (inclusive as dos processos secundários) e encerradas com o comando “ESC”. Os processos secundários foram fechados imediatamente.



```
Processo de pelotização e medição de granulometria
.....
Bloqueio/retomada da execução de tarefas:
Tecla g: Tarefa de leitura de dados de medição
Tecla c: Tarefa de leitura de dados de processo
Tecla r: Tarefa de captura de mensagens
Tecla p: Tarefa de exibição de dados de processo
Tecla a: Tarefa de análise de granulometria
.....
Tecla l: Limpar tela de monitor de exibição de dados de processo
.....
Para encerrar o processo: ESC

Tarefa de leitura de dados de processo encerrando.
Tarefa de leitura do sistema de medição encerrando.
Tarefa de captura de dados encerrando.

Processo encerrando.

O C:\Users\Léiza\Desktop\atr\pelotizacao\projeto\granulometria\Debug\granulometria.exe (processo 14788) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas -> Opções -> Depuração -> Fechar o console automaticamente quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...
```

Figura 23 - Encerramento do processo principal sem condição especial

## 7. REFERÊNCIAS

[1]

POLICARPO, Flávio F. **Minério de ferro: desafios para as indústrias mineral e siderúrgica**. 2012. 64 páginas. Universidade Federal de Minas Gerais - UFMG, Belo Horizonte. Disponível em: <[https://repositorio.ufmg.br/bitstream/1843/BUOS-9CAH4A/1/monografia\\_\\_\\_vers\\_o\\_final\\_\\_\\_fl\\_vio\\_ferreira\\_policarpo.pdf](https://repositorio.ufmg.br/bitstream/1843/BUOS-9CAH4A/1/monografia___vers_o_final___fl_vio_ferreira_policarpo.pdf)>. Acesso em: 20 de fev. de 2021.

[2]

VOCÊ sabe o que é pelotização? **Vale**. Disponível em: <<http://www.vale.com/brasil/PT/aboutvale/news/Paginas/voce-sabe-o-que-e-pelotizacao.aspx#:~:text=Eles%20são%20todos%20feitos%20de,dá%20nas%20usinas%20de%20pelotização>>. Acesso em: 20 de fev. de 2021.

[3]

ENTENDA como funciona o processo de pelotização. **Vale**. Disponível em: <<http://www.vale.com/brasil/pt/aboutvale/news/paginas/entenda-funciona-processo-pelotizacao-usinas.aspx>>. Acesso em: 20 de fev. de 2021.

[4]

INTEL Guide for Developing Multithreaded Applications. **Intel Software Dispatch**. Disponível em: <[http://runge.math.smu.edu/Courses/Math6370\\_Spring11/intel\\_multithreading\\_guide1.pdf](http://runge.math.smu.edu/Courses/Math6370_Spring11/intel_multithreading_guide1.pdf)>. Acesso em: 22 de fev. de 2021.

[5]

MENDES, Luiz Themystokliz. **Sincronismo entre Threads - Parte II**. 35 slides.

[6]

MENDES, Luiz Themystokliz. **Multithreading na plataforma Windows I**. 23 slides.

[7]

MENDES, Luiz Themystokliz. **Multithreading na plataforma Windows IV**. 32 slides.