

UNIVERSIDADE FEDERAL DE MINAS GERAIS



AUTOMAÇÃO EM TEMPO REAL – ELT127

Trabalho Prático - **Etapa 2**

Gustavo Santana Cavassani

PROFESSOR: Luiz Themystokliz S. Mendes

Sumario

1. Introdução.....	3
2. Descrição da Arquitetura	4
3. Descrição do Projeto e Escolhas de Implementação.....	6
4. Resultados.....	43
5. Conclusão/Problemas Encontrados	49

1. Introdução

O processo siderúrgico do aço compreende desde a etapa de processamento do minério de ferro passando pela fundição, lingotamento, conformação mecânica até chegar ao produto final acabado, tais como chapas, bobinas, vergalhões, arames, perfis, barras, etc.

A siderurgia de um modo geral apresentou evoluções tecnológicas, principalmente quanto à produtividade, rendimento metálico e espaço físico necessário. Etapas como redução, refino e conformação mecânica hoje podem ser realizadas em uma única usina, isto possibilita o ganho em escala fazendo com que o custo por unidade produzida caia. Nesse contexto, o **ferro gusa** é um tipo de ferro fundido que é produzido a partir de minério de ferro, coque e calcário em altos-fornos. É um material bruto e é usado principalmente como matéria-prima na produção de aço, e para tal, é necessária a passagem por um processo de remoção de enxofre, conhecido como dessulfuração.

Esse é o tema deste trabalho prático, o desenvolvimento de uma aplicação multithread de forma a realizar a integração dos processos envolvidos na dessulfuração, com intuito do aumento do desempenho da mesma.



Figura 1: Instalação de Dessulfuração

Fonte: <https://www.infomet.com.br/siderurgia-3a-dessulfuracao.php>

2. Descrição da Arquitetura

❑ ETAPA 1

Abaixo segue o diagrama de relacionamentos entre as tarefas passado na descrição do trabalho prático.

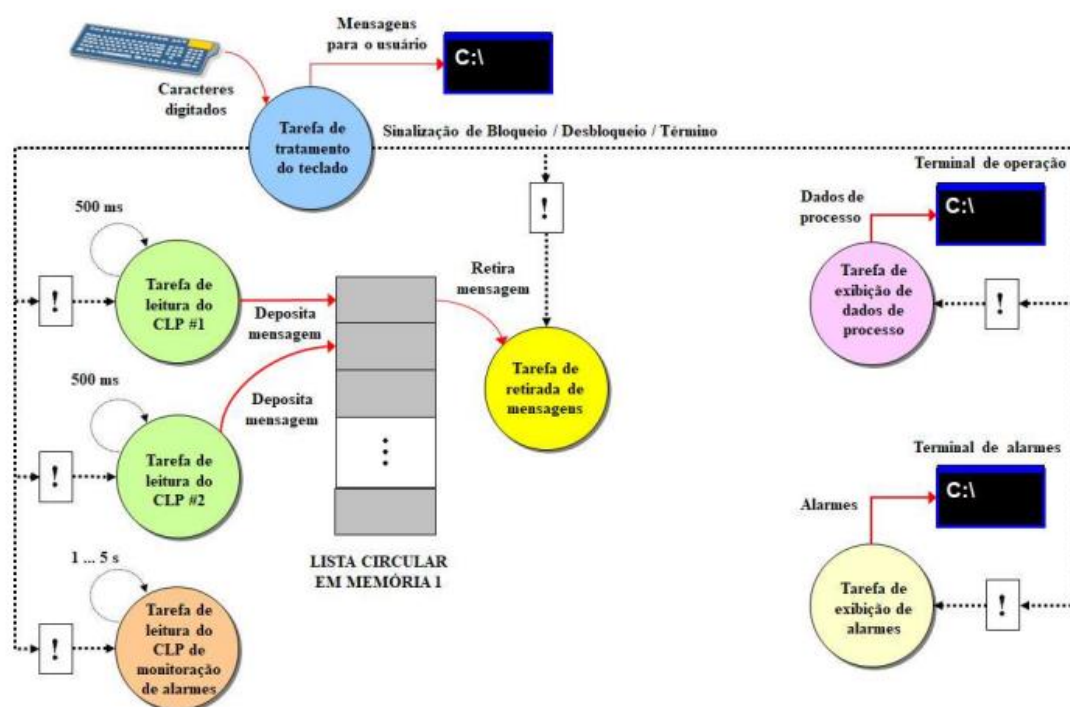


Figura 2: Diagrama de relacionamentos – Etapa 1
Fonte: Descrição do trabalho final fornecida pelo professor

A forma como ficou organizada a arquitetura é a seguinte:

As tarefas de **leitura do CLP#1**, **leitura do CLP#2**, **leitura do CLP de monitoração de alarmes**, **retirada de mensagens** e **tratamento do teclado** foram todas implementadas em um mesmo processo, chamado **“MainProcess”**, utilizado como o processo principal da aplicação, onde todas essas tarefas foram tratadas como threads secundárias, exceto a de retirada de mensagens que foi tratada na thread primária.

A tarefa de **exibição de dados do processo** foi implementada em um segundo processo, chamado **“DataViewProcess”**, e foi tratada na thread primária desse, sem a necessidade da criação de threads secundárias até o momento, na Etapa 1.

A tarefa de **exibição de alarmes**, similar a tarefa de exibição de dados do processo, foi também implementada em um segundo processo, chamado “**AlarmProcess**”, e foi tratada na thread primária desse, sem a necessidade da criação de threads secundárias até o momento, na Etapa 1.

Abaixo segue a imagem do diagrama de relacionamentos com as respectivas organizações feitas.

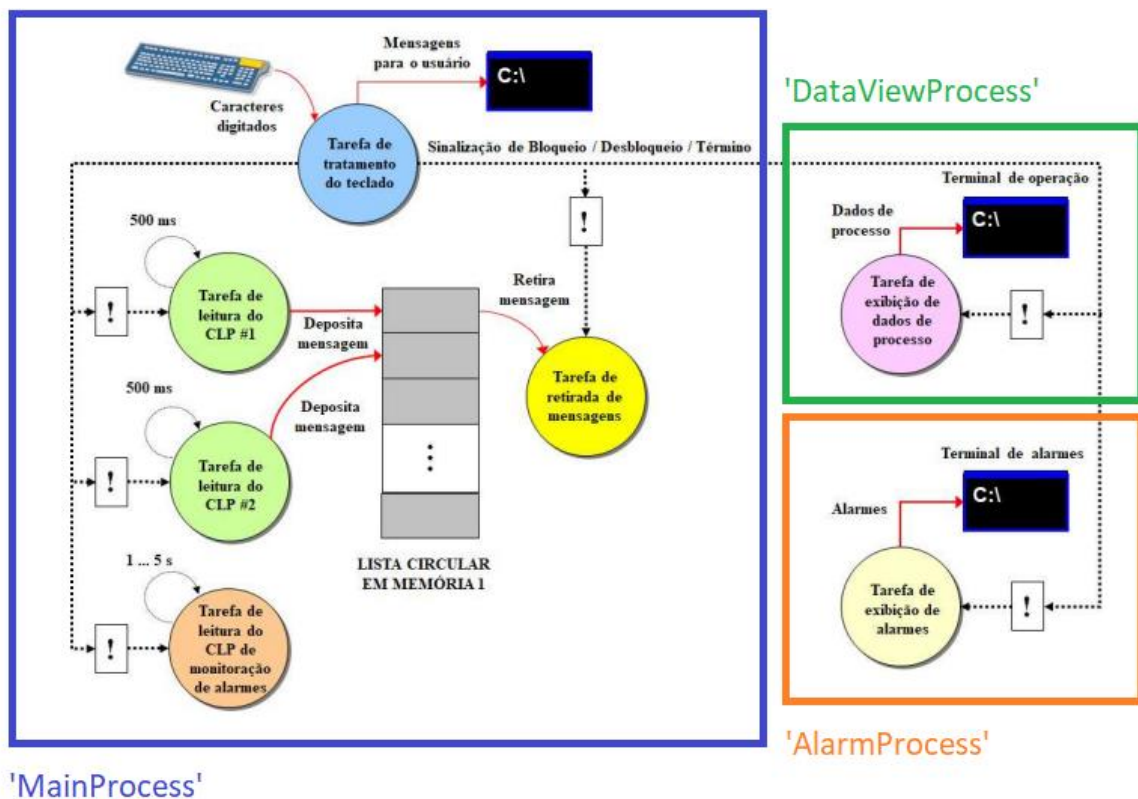


Figura 3: Diagrama de relacionamentos com as separações – Etapa 1

Fonte: Descrição do trabalho final fornecida pelo professor

❑ ETAPA 2

Para a segunda etapa, a organização das tarefas ficou da seguinte forma:

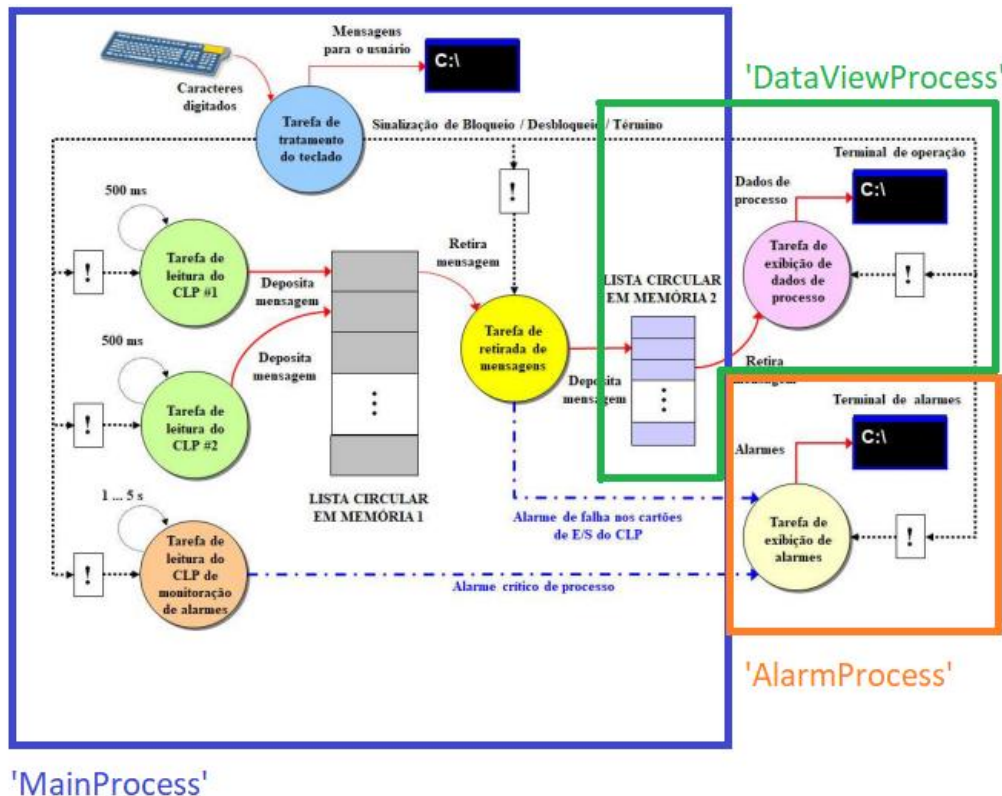


Figura 4: Diagrama de relacionamentos com as separações – Etapa 2

Fonte: Descrição do trabalho final fornecida pelo professor

Note que, *DataViewProcess* e *MainProcess* ‘dividem’ a lista circular 2, pois a mesma foi feita utilizando um arquivo mapeado em memória para o acesso compartilhado entre os dois processos, que será mais bem explicado nas seções abaixo.

3. Descrição do Projeto e Escolhas de Implementação

Para manter uma sequência e organização na explicação do código implementado, será explicado na sequência de linhas dos arquivos .cpp, e na seguinte sequência: *MainProcess*, *AlarmProcess*, *DataViewProcess*.

Como para todos os processos, existem declarações comuns e “repetitivas”, as mesmas serão explicadas apenas uma vez, além disso, alguns itens como include de bibliotecas, define de funções “padrão”, etc, serão ignorados, pois já é de conhecimento geral e desnecessária a sua explicação em específico. De resto, as

implementações que foram consideradas únicas serão todas detalhadas a seguir.

❏ ETAPA 1

- *MainProcess.cpp*:

```
struct previousValue {
int alarmid;
int diag;
int p_int;
int p_inj;
int temp;
};

previousValue Messages;

[...]

int main() {
[...]
Messages.alarmid = 0;
Messages.diag = 0;
Messages.p_int = 0;
Messages.p_inj = 0;
Messages.temp = 0;
[...]
}
```

Struct criada para armazenar os valores “prévios” de cada parâmetro da mensagem, na seção de problemas encontrados é explicado o porquê disso.

```
// Constantes definindo as teclas que serão digitadas e seus endereços em hexadecimal
#define ESC                0x1B           // Tecla para encerrar o programa
#define TECLA_1            0x31           // Tecla para notificação de leitura do CLP #1
#define TECLA_2            0x32           // Tecla para notificação de leitura do CLP #2
#define TECLA_m            0x6D           // Tecla para notificação de leitura do CLP de
Alarmes
#define TECLA_r            0x72           // Tecla para notificação de retirada de mensagens
#define TECLA_p            0x70           // Tecla para notificação de exibição de dados do
processo
#define TECLA_a            0x61           // Tecla para notificação de exibição de alarmes
```

Constantes utilizadas para definir os valores hexadecimais das teclas utilizadas na tarefa de tratamento do teclado.

```
// Declaração de objetos de sincronização
HANDLE hEscEvent, hList1Full;
HANDLE hNotificationEvents[6];
HANDLE hMutexforNSEQ;           // Mutex para exclusão mútua da variável NSEQ
```

```
HANDLE hSemforList1_IN;           // Semaforo para lógica de 'contagem' do acesso à
lista circular 1 para depositar mensagens
HANDLE hSemforList1_OUT;         // Semaforo para lógica de 'contagem' do acesso à lista
circular 1 para retirar mensagens
HANDLE hMutexforList1;           // Mutex para exclusão mútua do armazenamento
da mensagem e atualização da variável de posição livre da lista circular 1

[...]
```

```
int main(){
[...]
```

```
//----Criação dos eventos----
// 1.Evento da tecla '1':
hNotificationEvents[0] = CreateEvent(NULL, FALSE, FALSE, "Evento_1");
CheckForError(hNotificationEvents[0]);

(CONTINUA ABAIXO)
```



```

// 2.Evento da tecla '2':
hNotificationEvents[1] = CreateEvent(NULL, FALSE, FALSE, "Evento_2");
CheckForError(hNotificationEvents[1]);

// 3.Evento da tecla 'm':
hNotificationEvents[2] = CreateEvent(NULL, FALSE, FALSE, "Evento_m");
CheckForError(hNotificationEvents[2]);

// 4.Evento da tecla 'r':
hNotificationEvents[3] = CreateEvent(NULL, FALSE, FALSE, "Evento_r");
CheckForError(hNotificationEvents[3]);

// 5.Evento da tecla 'p':
hNotificationEvents[4] = CreateEvent(NULL, FALSE, FALSE, "Evento_p");
CheckForError(hNotificationEvents[4]);

// 6.Evento da tecla 'a':
hNotificationEvents[5] = CreateEvent(NULL, FALSE, FALSE, "Evento_a");
CheckForError(hNotificationEvents[5]);

// 7.Evento da tecla 'ESC'(Declarada fora do vetor de eventos de notificação, pois esse deve ser de
reset manual para notificar todos):
hEscEvent = CreateEvent(NULL, TRUE, FALSE, "Evento_ESC");
CheckForError(hEscEvent);

// 8.Evento de sinalização de lista 1 cheia:
hList1Full = CreateEvent(NULL, FALSE, FALSE, "Evento_L1Full");
CheckForError(hList1Full);

//---Criação de Mutexes e Semaforos---
hMutexforNSEQ = CreateMutex(NULL, FALSE, "MutexforNSEQ");
CheckForError(hMutexforNSEQ);
hSemforList1_IN = CreateSemaphore(NULL, 100, 100, "SemaphoreList1_IN");
CheckForError(hSemforList1_IN);
hSemforList1_OUT = CreateSemaphore(NULL, 0, 100, "SemaphoreList1_OUT");
CheckForError(hSemforList1_OUT);
hMutexforList1 = CreateMutex(NULL, FALSE, "MutexforList1");
CheckForError(hMutexforList1);
//Foram criados dois semáforos 'inversos', para sinalizar e armazenar a entrada e saída de
mensagens da lista circular
[...]
}

```

Handle's e as respectivas criações dentro do main() para os objetos de sincronização sendo esses o evento de tecla ESC (hEscEvent), o evento de sinalização que a lista circular 1 está cheia (hList1Full), um array contendo todos os eventos de notificação gerados por teclas com exceção do ESC (hNotificationEvents) , o mutex utilizado para exclusão mútua da variável NSEQ para geração de mensagens (hMutexforNSEQ), os semáforos para uma lógica de armazenamento da contagem do acesso à lista circular 1 (hSemforList1_IN e hSemforList1_OUT), de forma que, foi criado um semáforo com 100 posições livres e valor máximo 100 para sinalizar as entradas de mensagens na lista, e um segundo semáforo com 0 posições livres e valor máximo 100, de forma inversa ao primeiro, para sinalizar que há mensagens a serem lidas na lista circular 1, e por último, o mutex para exclusão mútua da escrita de mensagens e a adição da posição livre para ser escrita na lista circular 1 (hMutexforList1). Importante notar na criação do evento ESC, o segundo parâmetro foi passado como TRUE de forma a defini-lo como reset manual, assim sinalizando todas as threads simultaneamente.

```
// Declaração de uma struct para representar os estados das threads (FALSE -> Desbloqueado /  
TRUE -> Bloqueado)  
struct threadState {  
    BOOL BLOCKED;  
};  
  
threadState CLP1State, CLP2State, AlarmMState, MsgRmvState;  
  
[...]  
  
int main () {  
    [...]  
    CLP1State.BLOCKED = FALSE;           // Inicia desbloqueada  
    CLP2State.BLOCKED = FALSE;           // Inicia desbloqueada  
    AlarmMState.BLOCKED = FALSE;         // Inicia desbloqueada  
    MsgRmvState.BLOCKED = FALSE;         // Inicia desbloqueada  
    [...]  
}
```

Struct criada para armazenar e sinalizar o estado das threads, e a respectiva sinalização no main () de que todas threads iniciam desbloqueadas.

```
// Declaração de variáveis globais  
BOOL bStatus;  
DWORD dwRet;  
int NSEQ = 0;  
int NSEQ_Alarm = 0;  
string listaCircular1[100];             // Lista circular 1
```

```
int freePositionList1 = 0;           // Variável para armazenar a última posição livre da lista
circular a ser adicionada mensagem
int ocupPositionList1 = 0;          // Variável para armazenar a última posição ocupada da
lista circular a qual foi retirada mensagem
```

Variáveis globais utilizadas nas lógicas das threads, dentre elas, a variável que armazena o valor de NSEQ das mensagens de dados do processo (NSEQ), a variável que armazena o valor de NSEQ das mensagens de alarme, um array de strings de 100 posições utilizada como a lista circular 1 (listaCircular1), a variável que armazena a última posição livre na lista 1 a ser escrita mensagem (freePositionList1), e por último, a variável que armazena a última posição ocupada por uma mensagem na lista circular 1 a ser retirada pela thread de retirada de mensagens.

```
#include <locale>
```

```
[...]
```

```
int main (){
setlocale(LC_ALL, "");
[...]
}
```

Uma biblioteca e uma função importantes de citar é a função Locale, que por meio da chamada setlocale(LC_ALL, "") permite a exibição de caracteres especiais no console, como acentos, etc. Encontrei a dada função explicada e implementada em uma aplicação no stack overflow e foi de grande ajuda para a correta exibição dos textos.

```
int main () {
[...]

//----Criação dos processos----
// 1.Processo que engloba a exibição de dados de processo - DataViewProcess:
bStatus = CreateProcess("..\\MainProcess\\x64\\Debug\\DataViewProcess.exe",
NULL,
NULL,
NULL,
FALSE,
CREATE_NEW_CONSOLE,
NULL,
NULL,
&siDataViewProcess,
&piDataViewProcess);
CheckForError(bStatus);

// 2.Processo que engloba a exibição de alarmes - AlarmProcess:

bStatus = CreateProcess("..\\MainProcess\\x64\\Debug\\AlarmProcess.exe",
```

```

NULL,
NULL,
NULL,
FALSE,
CREATE_NEW_CONSOLE,
NULL,
NULL,
&siAlarmViewProcess,
&piAlarmViewProcess);
CheckForError(bStatus);

[...]
}

```

Funções padrão de criação de processos, com detalhe apenas no parâmetro “CREATE_NEW_CONSOLE”, que foi utilizado para que os dois processos “filhos” fossem criados cada um com seu respectivo prompt de comando.

```

int main (){
[...]

// Lógica de tratamento do teclado (Não criei uma thread secundária para isso, fiz uso da primária
como a thread de tratamento de teclado)
do {
printf("Digite uma tecla entre 1, 2, m, r, p, a ou ESC:\n");
    nTecla = _getch();
if (nTecla == TECLA_1) {
bStatus = SetEvent(hNotificationEvents[0]);
CheckForError(bStatus);
printf("Evento 'Tecla 1' setado!\n");
}
else if (nTecla == TECLA_2) {
bStatus = SetEvent(hNotificationEvents[1]);
CheckForError(bStatus);
printf("Evento 'Tecla 2' setado!\n");
}
else if (nTecla == TECLA_m) {
bStatus = SetEvent(hNotificationEvents[2]);
CheckForError(bStatus);
printf("Evento 'Tecla m' setado!\n");
}
else if (nTecla == TECLA_r) {
bStatus = SetEvent(hNotificationEvents[3]);
CheckForError(bStatus);
printf("Evento 'Tecla r' setado!\n");
}
else if (nTecla == TECLA_p) {
bStatus = SetEvent(hNotificationEvents[4]);
CheckForError(bStatus);
printf("Evento 'Tecla p' setado!\n");
}
}

```

```

}
else if (nTecla == TECLA_a) {
bStatus = SetEvent(hNotificationEvents[5]);
CheckForError(bStatus);
printf("Evento 'Tecla a' setado!\n");
}
} while (nTecla != ESC);

// Set do ESC feito fora da lógica de tratamento de teclas, pois se for digitado ESC, sai do loop e
finaliza os processos
bStatus = SetEvent(hEscEvent);
CheckForError(bStatus);
printf("Evento 'Tecla ESC' setado!\n");

[...]
}

```

Lógica de tratamento do teclado feita na thread primária, onde dentro de um loop uma variável `nTecla` recebe a tecla digitada e a mesma é usada em comparações com as constantes definidas anteriormente, caso a condição seja verdadeira o evento específico é setado. Observa-se que o evento de ESC é feito fora do loop, pois caso o mesmo ocorra, todas threads e processos devem se encerrar, inclusive a primária.

```

DWORD WINAPI CLP1_Read() {
HANDLE firstArrayOfObjects[3] = { hEscEvent, hNotificationEvents[0], hMutexforNSEQ };
HANDLE secondArrayOfObjects[3] = { hEscEvent, hNotificationEvents[0], hSemforList1_IN };
HANDLE thirdArrayOfObjects[3] = { hEscEvent, hNotificationEvents[0], hMutexforList1 };
HANDLE fourthArrayOfObjects[2] = { hEscEvent, hNotificationEvents[0] };
DWORD dwRet;
int nTipoEvento, NSEQAux;

printf("Estado 'CLP1_Read': Desbloqueada.\n");

do {
if (CLP1State.BLOCKED == FALSE) {
// Primeiro 'wait' para exclusão mútua e definição do NSEQ
dwRet = WaitForMultipleObjects(3, firstArrayOfObjects, FALSE, INFINITE);
nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (nTipoEvento == 1) {
printf("Evento de tecla 1 - Bloqueio thread 'CLP1_Read'.\n");
CLP1State.BLOCKED = TRUE;
printf("Estado 'CLP1_Read': Bloqueada.\n");
}
else if (nTipoEvento == 2) {
NSEQAux = NSEQ;
if (NSEQ == 99999) NSEQ = 0;
}
}
} while (1);
}

```

```
else  
NSEQ++;
```

```
bStatus = ReleaseMutex(hMutexforNSEQ);  
CheckForError(bStatus);  
}
```

(CONTINUA ABAIXO)

```
        // Segundo 'wait' para adiç o da mensagem   lista circular 1  
dwRet = WaitForMultipleObjects(3, secondArrayOfObjects, FALSE, 5);  
  
if (dwRet == WAIT_TIMEOUT) {  
    printf("Tempo excedido em 'wait' para adiç o de mensagem - Lista circular 1 cheia!\n");  
    bStatus = SetEvent(hList1Full);  
    CheckForError(bStatus);  
  
    dwRet = WaitForMultipleObjects(3, secondArrayOfObjects, FALSE, INFINITE);  
}  
// Tenta conquistar semaforo da lista circular 1, se passa de 5ms significa que a lista 1 est  cheia e  
fica bloqueada esperando at  ser retirada uma mensagem  
  
nTipoEvento = dwRet - WAIT_OBJECT_0;  
  
if (nTipoEvento == 0) {  
    break;  
}  
else if (nTipoEvento == 1) {  
    printf("Evento de tecla 1 - Bloqueio thread 'CLP1_Read'.\n");  
    CLP1State.BLOCKED = TRUE;  
    printf("Estado 'CLP1_Read': Bloqueada.\n");  
}  
else if (nTipoEvento == 2) {  
    dwRet = WaitForMultipleObjects(3, thirdArrayOfObjects, FALSE, INFINITE);    // Tenta conquistar  
    o mutex para exclus o m tua de lista circular 1  
  
    nTipoEvento = dwRet - WAIT_OBJECT_0;  
    if (nTipoEvento == 0) {  
        break;  
    }  
    else if (nTipoEvento == 1) {  
        printf("Evento de tecla 1 - Bloqueio thread 'CLP1_Read'.\n");  
        CLP1State.BLOCKED = TRUE;  
        printf("Estado 'CLP1_Read': Bloqueada.\n");  
    }  
    else if (nTipoEvento == 2) {  
        listaCircular1[freePositionList1] = Message_Creation(1, NSEQAux);  
  
        //cout << listaCircular1[freePositionList1] << endl;    // APENAS PARA VISUALIZA  O, REMOVER  
        DEPOIS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
  
        //freePositionList1 = freePositionList1 + 1;
```

```

freePositionList1 = (freePositionList1 + 1) % 100;
// Por algum motivo, ao implementar o somatório de posições da primeira forma
comentada acima gerava um erro de debug onde um null pointer apontava para um block de função
non-zero
// Pesquisei muito na internet e encontrei um exemplo de um programa onde ocorria o mesmo
erro em uma situação similar, e a solução encontrada pelo programador foi a segunda linha acima
// Fazer o somatório e armazenar o resto desse somatório dividido por 100, e funcionou, mas não
entendi o porque isso anula o erro apresentado anteriormente

bStatus = ReleaseMutex(hMutexforList1);
CheckForError(bStatus);
bStatus = ReleaseSemaphore(hSemforList1_OUT, 1, NULL);
CheckForError(bStatus);
}
}
}
else {
// Espera de desbloqueio da thread
dwRet = WaitForMultipleObjects(2, fourthArrayOfObjects, FALSE, INFINITE);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (nTipoEvento == 1) {
printf("Evento de tecla 1 - Desbloqueio thread 'CLP1_Read'.\n");
CLP1State.BLOCKED = FALSE;
printf("Estado 'CLP1_Read': Desbloqueada.\n");
}
}
Sleep(500); // Apenas para a etapa 1, remover na etapa 2
} while (nTipoEvento != 0);

printf("Encerrando thread 'CLP1_Read'.\n");

_endthreadex(0);

return 0;
}

```

Entrando agora nas funções executadas pelas threads secundárias, as threads CLP1_Read e CLP2_Read executam a mesma lógica, poderiam até ser implementadas em uma mesma função, mas para fim de facilitar na implementação, implementei funções separadas para cada.

Em cada uma das threads foram criados arrays de objetos de sincronização de forma a implementar os Wait's, onde todos contém os handles para os eventos de tecla ESC e teclas 1 (no CLP1_Read) ou 2 (no CLP2_Read), o que difere é que no array 1, é tratado o mutex para NSEQ, no array 2 é tratado o semaforo para

a lista 1 para entrada de mensagens e no array 3 é tratado o mutex para acesso à lista e atualização da posição livre.

As lógicas de tratamento foram feitas de forma que, caso a thread esteja desbloqueada definido pela comparação `CLPXState.BLOCKED == FALSE`, é feito um `WaitForMultipleObjects`, passando o array de objetos e armazenado em uma variável o retorno da função; com esse valor de retorno é feita a comparação do número do evento que foi sinalizado: caso o evento seja o ESC, o loop é termina e encerra a execução, caso o evento seja o da tecla 1 ou 2, ocorre a mudança do estado da thread para bloqueado, e no último caso, é feito o armazenamento do NSEQ da mensagem atual e o incremento da variável global, liberando o mutex para acesso da outra thread. O segundo Wait trata a condição de acesso à lista, nesse foi implementado um tempo de 5ms de espera para que, caso o valor de retorno seja `WAIT_TIMEOUT` significa que a lista está cheia e o mesmo deve esperar indefinidamente até que a mesma libere espaço. Caso haja espaço na lista, é adicionado a mensagem na lista na posição livre, indicada pela variável `freePositionList1`, e esse valor é incrementado, o mutex é liberado para uso em outra thread e o semáforo que serve para sinalização de que há mensagens a serem lidas na lista. Na comparação `CLPXState.BLOCKED == FALSE`, caso ela seja falsa, o else ocorre de forma que é feito um Wait apenas pelos eventos ESC e tecla 1 ou 2, de forma que a thread fica bloqueada, como descrito no enunciado do trabalho.

```
DWORD WINAPI Alarm_Monitor() {
HANDLE firstArrayOfObjects[2] = { hEscEvent, hNotificationEvents[2] };
DWORD dwRet;
int nTipoEvento, randomTimeout;

printf("Estado 'Alarm_Monitor': Desbloqueada.\n");

do {
if (AlarmMState.BLOCKED == FALSE) {
randomTimeout = rand() % 4001 + 1000; // Gera valor aleatório para
timeout de forma à disparar alarmes aleatoriamente entre intervalos de 1 a 5 segundos
dwRet = WaitForMultipleObjects(2, firstArrayOfObjects, FALSE, randomTimeout);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (dwRet == WAIT_TIMEOUT) {
// Geração de alarme
```



```

//Alarm_Creation();                                     // Criação da
mensagem de alarme

cout << "Alarme criado:" << Alarm_Creation() << endl; // APENAS PARA VISUALIZAÇÃO, REMOVER
DEPOIS !!!!!!!!!!!!!!!!!!!!!!!

if (NSEQ == 99999) NSEQ_Alarm = 0;
else
NSEQ_Alarm++;
// IMPLEMENTAR >MAILSLOT< PARA ENVIAR OS DADOS PARA PROCESSO DE EXIBIÇÃO DE ALARMES
!!!!!!!!!!!!!!!!!!!!!!
}
else if (nTipoEvento == 1) {
printf("Evento de tecla m - Bloqueio thread 'Alarm_Monitor'.\n");
AlarmMState.BLOCKED = TRUE;
printf("Estado 'Alarm_Monitor': Bloqueada.\n");
}
}
else {
dwRet = WaitForMultipleObjects(2, firstArrayofObjects, FALSE, INFINITE);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (nTipoEvento == 1) {
printf("Evento de tecla m - Desbloqueio thread 'Alarm_Monitor'.\n");
AlarmMState.BLOCKED = FALSE;
printf("Estado 'Alarm_Monitor': Desbloqueada.\n");
}
}
} while (nTipoEvento != 0);

printf("Encerrando thread 'Alarm_Monitor'.\n");

_endthreadex(0);

return 0;
}

```

Na função da thread de monitoramento de alarmes, é utilizado apenas um array de objetos, contendo apenas os eventos de tecla ESC e tecla m. Nesta thread, caso o estado da mesma seja desbloqueado, inicia a execução principal, e como é preciso gerar alarmes entre períodos aleatórios de 1 à 5 s, foi utilizada uma variável randomTimeout que armazena (por meio da função rand) um valor aleatório entre 1000 e 5000 e esse valor é passado como parâmetro para o TIMEOUT do Wait, assim, enquanto ESC ou m não são digitados, quando o timeout ocorre, a comparação do valor de retorno com WAIT_TIMEOUT se torna verdadeira, a thread entra na lógica de geração de alarme, e como apenas ela

faz acesso à NSEQ_Alarm, não é necessária a exclusão mútua do acesso à mesma. Caso o estado da thread esteja como bloqueado, o mesmo que é feito na thread de leitura 1 e 2 ocorre aqui, um Wait esperando o evento de desbloqueio pela tecla m.

```
DWORD WINAPI Messages_Removal() {
HANDLE firstArrayOfObjects[3] = { hEscEvent, hNotificationEvents[3], hSemforList1_OUT };
HANDLE secondArrayOfObjects[3] = { hEscEvent, hNotificationEvents[3] };
DWORD dwRet;
int nTipoEvento;

printf("Estado 'Messages_Removal': Desbloqueada.\n");

do {
if (MsgRmvState.BLOCKED == FALSE) {
dwRet = WaitForMultipleObjects(3, firstArrayOfObjects, FALSE, INFINITE);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (nTipoEvento == 1) {
printf("Evento de tecla r - Bloqueio thread 'Messages_Removal'.\n");
MsgRmvState.BLOCKED = TRUE;
printf("Estado 'Messages_Removal': Bloqueada.\n");
}

(CONTINUA ABAIXO)

else if (nTipoEvento == 2) {
// Lista circular 1 com mensagens à serem retiradas
// (A parte da retirada, também identificando o campo DIAG, para inserir em uma lista circular 2
será feita na etapa 2 do trabalho)
cout << listaCircular1[ocupPositionList1] << endl;

//ocupPositionList1 = ocupPositionList1 + 1;
ocupPositionList1 = (ocupPositionList1 + 1) % 100;
// Por algum motivo, ao implementar o somatório de posições de posições da primeira forma
comentada acima gerava um erro de debug onde un null pointer apontava para um block de função
non-zero
// Pesquisei muito na internet e encontrei um exemplo de um programa onde ocorria o mesmo
erro em uma situação similar, e a solução encontrada pelo programador foi a segunda linha acima
// Fazer o somatório e armazenar o resto desse somatório dividido por 100, e funcionou, mas não
entendi o porque isso anula o erro apresentado anteriormente

bStatus = ReleaseSemaphore(hSemforList1_IN, 1, NULL);      // Retirada de uma mensagem
CheckForError(bStatus);
}
}
else {
dwRet = WaitForMultipleObjects(2, secondArrayOfObjects, FALSE, INFINITE);
```

```

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (nTipoEvento == 1) {
printf("Evento de tecla r - Desbloqueio thread 'Messages_Removal'.\n");
MsgRmvState.BLOCKED = FALSE;
printf("Estado 'Messages_Removal': Desbloqueada.\n");
}
}
} while (nTipoEvento != 0);

printf("Encerrando thread 'Messages_Removal'.\n");

_endthreadex(0);

return 0;
}

```

Na função da thread de retirada de mensagens, o procedimento é o mesmo, caso o estado da mesma seja desbloqueado, um Wait ocorre esperando por ESC, tecla r ou a obtenção do semáforo responsável pela retirada de mensagens da lista. Se a thread obtém esse semáforo, é feita impressão no console a mensagem na última posição ocupada da lista, indicada pela variável `ocupPositionList1`, feito o incremento dessa variável e é feito um release no semáforo de mensagens a entrarem na lista. Nesse ponto é possível compreender como funciona a lógica dos semáforos, a obtenção de um realiza o release do outro, de forma a ocorrer corretamente o acesso, inserção e retirada na lista circular 1. Caso o estado da thread esteja como bloqueado, a thread espera pelo evento de tecla r, como as outras anteriores.

```

string NSEQ_Format(int _NSEQ) {
// Usando função de C++ (setfill()) que automaticamente preenche uma string com um caractere,
definindo a largura da string e passando a variável a ser armazenada
ostringstream stringAux;
string NSEQ_Format;

stringAux << setfill('0') << setw(5) << _NSEQ;
NSEQ_Format = stringAux.str();

return NSEQ_Format;
}

```

A função `NSEQ_format` serve para a formatação do parâmetro `NSEQ` da mensagem, de forma a criar uma string com 5 algarismos contendo o valor do número de sequência e sendo preenchido com 0 nos algarismos não definidos, para isso, foi usada a função `setfill()` que executa exatamente o que foi descrito,

juntamente com `setw()` que determina a “largura” dessa string, definindo o tamanho de 5 algarismos.

```
int alarmID_Format() {
    int randomNum;

    randomNum = rand() % 99;
    if (Messages.alarmid == 0 || randomNum != Messages.alarmid) {
        Messages.alarmid = randomNum;
        return randomNum;
    }
    else{
        alarmID_Format();
    }
}
```

A função `alarmID_Format` serve para formatação do ID dos alarmes, pois o mesmo é alfanumérico, diferente do ID das mensagens de processo, que seguem o número do CLP que as gerou, assim é retornado um valor aleatório gerado pela pelo resto da divisão da função `rand()` com 99, pois são dois dígitos.

```
int DIAG_Format() {
    int randomNum = rand() % 70;
    if (randomNum < 55) return randomNum;
    else
        return 55;
}
```

A função `DIAG_Format` serve para a formatação do parâmetro `DIAG`, gerando um número aleatório entre 0 à 55, caso esse valor seja maior ou igual a 55, a mensagem gerada, nos parâmetros de pressão interna, pressão de injeção e temperatura recebe 0.0;

```
string P_INT_Format() {
    int randomNum;
    float numDecimal;
    ostringstream stringAux;
    string P_INT_Format;

    randomNum = rand() % 2001 + 1000;
    if (randomNum % 10 == 0) randomNum++;
    numDecimal = randomNum / 10.0;
    stringAux << "0" << numDecimal;
    P_INT_Format = stringAux.str();

    return P_INT_Format;
}

string P_INJ_Format() {
    int randomNum;
    float numDecimal;
```

```

ostreamstream stringAux;
string P_INJ_Format;

randomNum = rand() % 2001 + 1000;
if (randomNum % 10 == 0) randomNum++;
numDecimal = randomNum / 10.0;
stringAux << "0" << numDecimal;
P_INJ_Format = stringAux.str();

return P_INJ_Format;
}

```

As funções P_INT_Format() e P_INJ_Format() são iguais e servem para gerar valores aleatórios de pressão na faixa de 100 à 300, e para a implementação correta de um valor decimal para as mesmas, a comparação com o resto da divisão por 10 ocorre de forma a ser somado de 1 caso o resto seja 0, para que haja valor na casa decimal.

```

string TEMP_Format() {
int randomNum;
float numDecimal;
ostreamstream stringAux;
string TEMP_Format;

randomNum = rand() % 10001 + 10000;
if (randomNum % 10 == 0) randomNum++;
numDecimal = randomNum / 10;
stringAux << numDecimal;
TEMP_Format = stringAux.str();

return TEMP_Format;
}

```

A função TEMP_Format serve para a formatação da temperatura, e funciona de forma similar à função de formatação das pressões, a única diferença é que o valor gerado é entre a faixa de 1000 à 2000.

```

string TIME_Format() {
SYSTEMTIME tempoAux;
ostreamstream stringAux;
string TEMP_Format;

GetSystemTime(&tempoAux);
stringAux << tempoAux.wHour << ":" << tempoAux.wMinute << ":" << tempoAux.wSecond;
TEMP_Format = stringAux.str();

return TEMP_Format;
}

```

A função TIME_Format serve para a formatação do parâmetro de tempo que a mensagem é gerada, e seu funcionamento é simples, pois usando a função

GetSystemTime(), é possível pegar o valor respectivo às horas, minutos e segundos do relógio do computador na hora em que a mensagem é gerada, e todos esses valores são armazenados em uma string que é retornada para a formação da mensagem.

Note que para as funções alarmID_Format(), DIAG_Format(), P_INT_Format(), P_INJ_Format() e TEMP_Format(), foi feita uma lógica recursiva, onde se comparava o valor aleatório gerado na chamada atual da função com o valor anterior da mensagem, e caso esses fossem iguais, a função chama a si mesma até que um valor diferente do anterior é gerado, na última seção é explicado o motivo para tal.

```
string Message_Creation(int ID, int NSEQAux) {
    ostringstream stringAux;
    string _NSEQ = NSEQ_Format(NSEQAux);
    int _DIAG = DIAG_Format();
    string _P_INT = P_INT_Format();
    string _P_INJ = P_INJ_Format();
    string _TEMP = TEMP_Format();
    string _TIME = TIME_Format();
    string mensagemRet;

    if (_DIAG == 55) {
        stringAux << _NSEQ << ";" << ID << ";" << _DIAG << ";" << "0000.0" << ";" << "0000.0" << ";" <<
        "0000.0" << ";" << _TIME;;
        mensagemRet = stringAux.str();
        return mensagemRet;
    }
    else
        stringAux << _NSEQ << ";" << ID << ";" << _DIAG << ";" << _P_INT << ";" << _P_INJ << ";" << _TEMP
        << ";" << _TIME;
        mensagemRet = stringAux.str();
        return mensagemRet;
    }

    string Alarm_Creation() {
        // As mensagens de alarme serão corretamente implementadas na etapa 2 do trabalho
        ostringstream stringAux;
        string _NSEQ = NSEQ_Format(NSEQ_Alarm);
        int _ID = alarmID_Format();
        string _TIME = TIME_Format();
        string mensagemRet;

        stringAux << _NSEQ << ";" << _ID << ";" << _TIME;
        mensagemRet = stringAux.str();
        return mensagemRet;
    }
}
```

As funções de criação de mensagens `Message_Creation()` e `Alarm_Creation` fazem uso de todas as funções explicadas anteriormente para a geração das respectivas mensagens, com diferença apenas em `Message_Creation`, que é feita uma comparação para saber se o valor de `DIAG` é igual a 55, que sinaliza a geração de uma mensagem de falha de hardware, passando 0.0 para as pressões a temperatura, explicado anteriormente.

- *AlarmProcess.cpp*:

Em AlarmProcess, não há threads secundárias, de forma que a única thread executando é a primária.

```
// Constantes definindo as teclas que serão digitadas e seus endereços em hexadecimal
#define ESC                0x1B          // Tecla para encerrar o programa
#define TECLA_1            0x31          // Tecla para notificação de leitura do CLP #1
#define TECLA_2            0x32          // Tecla para notificação de leitura do CLP #2
#define TECLA_m            0x6D          // Tecla para notificação de leitura do CLP de
Alarmes
#define TECLA_r            0x72          // Tecla para notificação de retirada de mensagens
#define TECLA_p            0x70          // Tecla para notificação de exibição de dados do
processo
#define TECLA_a            0x61          // Tecla para notificação de exibição de alarmes

// Declaração de objetos de sincronização
HANDLE hEscEvent, hEventA;

// Declaração de uma struct para representar os estados das threads (FALSE -> Desbloqueado /
TRUE -> Bloqueado)
struct threadState {
    BOOL BLOCKED;
};

threadState AlarmMState;
```

As declarações feitas são similares as feitas no MainProcess, com exceção de que apenas dois handles foram usados, um para o evento ESC e outro para o evento da tecla A, e que apenas uma variável da struct que armazena o estado da thread foi declarada.

```
int main() {
    setlocale(LC_ALL, "");
    printf("Processo de monitoramento de alarmes 'AlarmProcess' em execução.\n"); // Apenas para
etapa 1, remover na etapa 2

    AlarmMState.BLOCKED = FALSE;          // Inicia desbloqueado
    printf("Estado Inicial: Desbloqueado.\n\n");

    (CONTINUA ABAIXO)

    //----Abertura dos Eventos----
    hEscEvent = OpenEvent(EVENT_ALL_ACCESS, FALSE, "Evento_ESC");
    CheckForError(hEscEvent);

    hEventA = OpenEvent(EVENT_ALL_ACCESS, FALSE, "Evento_a");
    CheckForError(hEventA);

    // Recebimento dos eventos
```



```

HANDLE arrayOfObjects[2] = { hEscEvent, hEventA };
int nTipoEvento;
DWORD dwRet;

do {
    if (AlarmMState.BLOCKED == FALSE) {
        dwRet = WaitForMultipleObjects(2, arrayOfObjects, FALSE, INFINITE);
        //if (dwRet == WAIT_OBJECT_0) printf("Espera por eventos num 1 falhou (%d).\n", GetLastError());
        //CheckForError(dwRet);
        //CheckForError(dwRet == WAIT_OBJECT_0);

        nTipoEvento = dwRet - WAIT_OBJECT_0;
        if (nTipoEvento == 1) {
            printf("Evento de tecla a - Bloqueio thread 'AlarmProcess'.\n");
            AlarmMState.BLOCKED = TRUE;
            printf("Estado 'AlarmProcess': Bloqueada.\n");
        }
    }
    else {
        dwRet = WaitForMultipleObjects(2, arrayOfObjects, FALSE, INFINITE);
        //if (dwRet == WAIT_OBJECT_0) printf("Espera por eventos num 2 falhou (%d).\n", GetLastError());
        //CheckForError(dwRet);
        //CheckForError(dwRet == WAIT_OBJECT_0);

        nTipoEvento = dwRet - WAIT_OBJECT_0;
        if (nTipoEvento == 1) {
            printf("Evento de tecla a - Desbloqueio thread 'AlarmProcess'.\n");
            AlarmMState.BLOCKED = FALSE;
            printf("Estado 'AlarmProcess': Desbloqueada.\n");
        }
    }
} while (nTipoEvento != 0);

printf("Encerrando thread 'AlarmProcess'.\nDigite qualquer tecla para fechar a janela.\n");
_getch();

return EXIT_SUCCESS;
}

```

No main do processo, é feito a definição de estado inicial desbloqueado da mesma e a abertura (definição de apontador) para os eventos ESC e A. O restante é exatamente o que estava sendo feito nas threads do MainProcess, um loop onde, caso a thread esteja desbloqueada, fica esperando pelo evento de encerramento ou pelo evento de bloqueio, e caso ela esteja bloqueada, espera pelos eventos de encerramento ou desbloqueio, seu funcionamento é simples.

- *DataViewProcess.cpp*:

Em DataViewProcess também não há threads secundárias, apenas a primária.

```
HANDLE hEscEvent, hEventP;

struct threadState {
    BOOL BLOCKED;
};

threadState DataViewState;
```

Igual a AlarmProcess, apenas muda em suas declarações os handles dos objetos esperados, pois essa espera pelo evento da tecla p, e a criação de apenas uma variável de estado.

```
int main() {
    setlocale(LC_ALL, "");
    printf("Processo de monitoramento de alarmes 'DataViewProcess' em execução.\n"); // Apenas
    para etapa 1, remover na etapa 2

    DataViewState.BLOCKED = FALSE; // Inicia desbloqueado
    printf("Estado Inicial: Desbloqueado.\n\n");

    //----Abertura dos Eventos----
    hEscEvent = OpenEvent(EVENT_ALL_ACCESS, FALSE, "Evento_ESC");
    CheckForError(hEscEvent);

    (CONTINUA ABAIXO)

    hEventP = OpenEvent(EVENT_ALL_ACCESS, FALSE, "Evento_p");
    CheckForError(hEventP);

    // Recebimento dos eventos
    HANDLE arrayOfObjects[2] = { hEscEvent, hEventP };
    int nTipoEvento;
    DWORD dwRet;

    do {
        if (DataViewState.BLOCKED == FALSE) {
            dwRet = WaitForMultipleObjects(2, arrayOfObjects, FALSE, INFINITE);
            //if (dwRet == WAIT_OBJECT_0) printf("Espera por eventos num 1 falhou (%d).\n", GetLastError());
            //CheckForError(dwRet);
            //CheckForError(dwRet == WAIT_OBJECT_0);

            nTipoEvento = dwRet - WAIT_OBJECT_0;
            if (nTipoEvento == 1) {
                printf("Evento de tecla p - Bloqueio thread 'DataViewProcess'.\n");
                DataViewState.BLOCKED = TRUE;
                printf("Estado 'DataViewProcess': Bloqueada.\n");
            }
        }
    }
    else {
```

```

dwRet = WaitForMultipleObjects(2, arrayOfObjects, FALSE, INFINITE);
//if (dwRet == WAIT_OBJECT_0) printf("Espera por eventos num 2 falhou (%d).\n", GetLastError());
//CheckForError(dwRet);
//CheckForError(dwRet == WAIT_OBJECT_0);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 1) {
printf("Evento de tecla p - Desbloqueio thread 'DataViewProcess'.\n");
DataViewState.BLOCKED = FALSE;
printf("Estado 'DataViewProcess': Desbloqueada.\n");
}
}
} while (nTipoEvento != 0);

printf("Encerrando thread 'DataViewProcess'.\nDigite qualquer tecla para fechar a janela.\n");
_getch();
return EXIT_SUCCESS;
}

```

Muito similar à AlarmProcess, no main do processo, é feito a definição de estado inicial desbloqueado da mesma e a abertura (definição de apontador) para os eventos ESC e P. O restante é exatamente o que estava sendo feito nas threads do MainProcess também, um loop onde, caso a thread esteja desbloqueada, fica esperando pelo evento de encerramento ou pelo evento de bloqueio, e caso ela esteja bloqueada, espera pelos eventos de encerramento ou desbloqueio, seu funcionamento, como a anterior, é simples.

❏ ETAPA 2

Para otimizar o detalhamento do que foi feito, será explicado do código apenas o que difere da etapa 1 para a etapa 2.

- *MainProcess.cpp*:

```

[...]
// Declaração de objetos de sincronização
HANDLE hEscEvent, hMailslot;
HANDLE hNotificationEvents[6];
HANDLE hMutexforNSEQ; // Mutex para exclusão mútua da variável NSEQ
HANDLE hSemforList1_IN; // Semaforo para lógica de 'contagem' do acesso à
lista circular 1 para depositar mensagens
HANDLE hSemforList1_OUT; // Semaforo para lógica de 'contagem' do acesso à lista
circular 1 para retirar mensagens
HANDLE hSemforList2_IN; // Semaforo para lógica de 'contagem' do acesso à
lista circular 2 para depositar mensagens

```

```

HANDLE hSemforList2_OUT;           // Semaforo para lógica de 'contagem' do acesso à lista
circular 2 para retirar mensagens
HANDLE hMutexforList1;             // Mutex para exclusão mútua da atualização da
variável de posição livre da lista circular 1
HANDLE hMutexforMailslot;         // Mutex para exclusão mútua do mailslot
HANDLE hMailslotEvent;            // Evento para sincronização de entrada de dados
no mailslot
HANDLE hMailslotCriadoEvent; // Evento para sinalização da criação do mailslot
HANDLE hMappedSection;            // Arquivo mapeado em memória
HANDLE hMapFileWriteEvent;        // Evento de sinalização de escrita no arquivo mapeado
HANDLE hMapFileReadEvent;         // Evento de sinalização de leitura no arquivo mapeado
[...]
```

Foram declarados novos handles necessários para a implementação da lista circular 2, da comunicação IPC por meio de Mail Slot entre os processos e do arquivo mapeado em memória. O evento criado anteriormente List1Full foi descontinuado, pois foi utilizada outra lógica para sinalizar que as listas estão cheias.

```

[...]
```

```

// Declaração de struct para armazenar parametros das mensagens de alarme
struct alarmMsg {
int NSEQ;
int ID;
SYSTEMTIME TIME;
};

// Declaração de struct para armazenar parametros das mensagens de dados do processo
struct dataMsg {
int NSEQ;
int ID;
int DIAG;
float P_INT;
float P_INJ;
float TEMP;
SYSTEMTIME TIME;
};

dataMsg msgSize;           // Instanciando uma variável do tipo da struct apenas para passar o seu
sizeof() na função de CreateFileMapping

// Declaração de funções para 'formatar' as mensagens
int alarmID_Format();
int DIAG_Format();
float P_INT_Format();
float P_INJ_Format();
float TEMP_Format();
void Message_Creation(dataMsg& Message, int ID, int NSEQAux);
void Alarm_Creation(alarmMsg& Alarm, int NSEQAux, int ID);
[...]
```

Na primeira etapa, as mensagens estavam sendo geradas e tratadas como strings, mas após uma consulta com o professor viu-se que as mesmas poderiam ser geradas diretamente em structs que separam seus parâmetros, gerando uma maior facilidade no tratamento das mesmas, dessa forma algumas funções como a NSEQ_Format() foram descontinuadas, e as funções de criação de alarme e criação de mensagem foram alteradas para armazenar os valores diretamente em uma struct.

```
[...]
// Declaração de variáveis globais
BOOL bStatus;
DWORD dwRet;
DWORD dwBytesEnviados;
int NSEQ = 0;
int NSEQ_Alarm = 0;
dataMsg listaCircular1[100]; // Lista circular 1
int freePositionList1 = 0; // Variável para armazenar a última posição livre da lista
circular a ser adicionada mensagem
int ocupPositionList1 = 0; // Variável para armazenar a última posição ocupada da
lista circular a qual foi retirada mensagem
dataMsg listaCircular2[50]; // Lista circular 2
int freePositionList2 = 0; // Variável para armazenar a última posição livre da lista
circular a ser adicionada mensagem
int ocupPositionList2 = 0; // Variável para armazenar a última posição ocupada da
lista circular a qual foi retirada mensagem
dataMsg* lpImage; // Apontador para o arquivo mapeado
[...]
```

Foram declaradas novas variáveis globais para serem usados no arquivo mapeado em memória, a lista circular 2 e suas variáveis para manipulação, notando que anteriormente as listas eram de strings, mas foram alteradas para o tipo da struct de mensagem dataMsg, dessa forma, as listas circulares passaram a armazenar estruturas de dados com as mensagens.

```
[...]
// Criação do arquivo mapeado em memória para lista 2
hMappedSection = CreateFileMapping(INVALID_HANDLE_VALUE,
NULL,
PAGE_READWRITE,
0,
sizeof(msgSize) * 50,
"ArquivoMapeado");
CheckForError(hMappedSection);

lpImage = (dataMsg*)MapViewOfFile(hMappedSection,
FILE_MAP_WRITE,
0,
0,
sizeof(msgSize) * 50);
```

```
CheckForError(lpImage);  
[...]
```

Indo agora para o main, foi criado o arquivo mapeado em memória que é compartilhado com a thread DataViewProcess e atribuída a visão do arquivo para o ponteiro lpImage, onde o tamanho da visão passado é 50 vezes o tamanho da estrutura de dados criada, pois são 50 mensagens criadas na lista circular 2.

```
[...]  
// 8.Evento de que mailslot na exibição de alarmes foi criado:  
hMailslotCriadoEvent = CreateEvent(NULL, FALSE, FALSE, "Evento_MailslotCriado");  
CheckForError(hMailslotCriadoEvent);  
  
// 9.Evento de sinalização de mailslot (Envio de mensagens)  
hMailslotEvent = CreateEvent(NULL, FALSE, FALSE, "Evento_Mailslot");  
CheckForError(hMailslotEvent);  
  
// 10.Eventos de sinalização de leitura e escrita no arquivo mapeado em memória  
hMapFileWriteEvent = CreateEvent(NULL, FALSE, FALSE, "Evento_MapFileWrite");  
CheckForError(hMapFileWriteEvent);  
  
hMapFileReadEvent = CreateEvent(NULL, FALSE, FALSE, "Evento_MapFileRead");  
CheckForError(hMapFileReadEvent);  
[...]
```

Foram criados novos eventos para sincronização do mailslot e do acesso ao arquivo mapeado em memória.

```
[...]  
hSemforList2_IN = CreateSemaphore(NULL, 50, 50, "SemaphoreList2_IN");  
CheckForError(hSemforList2_IN);  
hSemforList2_OUT = CreateSemaphore(NULL, 0, 50, "SemaphoreList2_OUT");  
CheckForError(hSemforList2_OUT);  
[...]  
hMutexforMailslot = CreateMutex(NULL, FALSE, "MutexforMailslot");  
CheckForError(hMutexforMailslot);  
[...]
```

Foram criados os semáforos de entrada e saída de mensagens na lista circular 2 e um mutex para exclusão mútua do mailslot, pois o mesmo é utilizado pelas threads Message_Removal e Alarm_Monitor.

```
[...]  
// Wait pelo evento que sinaliza que o mailslot foi criado  
WaitForSingleObject(hMailslotCriadoEvent, INFINITE);  
// Criação do arquivo mailslot  
hMailslot = CreateFile("\\\\.\\mailslot\\MyMailslot",  
GENERIC_WRITE,  
FILE_SHARE_READ,  
NULL,  
OPEN_EXISTING,
```

```
FILE_ATTRIBUTE_NORMAL,  
NULL);  
CheckForError(hMailslot != INVALID_HANDLE_VALUE);  
[...]
```

O MainProcess espera que AlarmProcess crie o mailslot (pois é a thread que receberá as mensagens que o cria) e sinaliza para a mesma para que ela possa criar o arquivo que receberá as mensagens e será enviado para o outro processo.

```
[...]  
// Apaga o mapeamento em memória  
bStatus = UnmapViewOfFile(lpImage);  
CheckForError(bStatus);  
[...]
```

Ao término da execução, fecha-se o arquivo mapeado e fecham-se os handles, mas como é muito similar a etapa anterior não será inserido o código aqui.

```
DWORD WINAPI CLPX_READ(){  
[...]  
dataMsg msgCreated;  
[...]  
// Wait utilizado para a "temporização" da thread clp, na qual a mesma faz uma "espera" de  
500ms apenas para gerar a temporização de leitura dos CLPS  
dwRet = WaitForSingleObject(hEscEvent, 500);  
[...]  
  
// Segundo 'wait' para adição da mensagem à lista circular 1  
dwRet = WaitForMultipleObjects(3, secondArrayOfObjects, FALSE, 5);  
// Tenta conquistar semaforo da lista circular 1, se passa de 5ms significa que a lista 1 está cheia e  
fica bloqueada esperando até ser retirada uma mensagem  
if (dwRet == WAIT_TIMEOUT) {  
if (MsgRmvState.BLOCKED == FALSE) {  
printf("Listas Circulares 1 e 2 Cheias!\n");  
}  
else {  
printf("Lista Circular 1 Cheia!\n");  
}  
dwRet = WaitForMultipleObjects(3, secondArrayOfObjects, FALSE, INFINITE);  
}  
[...]
```

Nas threads de leitura dos CLP's, as únicas alterações feitas foi no tipo da mensagem criada, que passou a ser do tipo da struct, na geração da temporização que foi feita igual estava sendo feito para a thread de monitoramento de alarmes, utilizando o TIMEOUT da função Wait, e por último, na lógica de tratamento de lista cheia, onde era impresso apenas lista 1 cheia, mas nessa segunda etapa foi concluído que, caso o processo de exibição de dados esteja bloqueado, quando o time ocorre significa que as duas listas circulares estão cheias, pois a primeira lista enche apenas depois da segunda

lista encher, e caso quem esteja bloqueada seja a thread de remoção de mensagens, significa que o timeout ocorre quando apenas a lista 1 enche.

```
DWORD WINAPI Alarm_Monitor() {
[...]
alarmMsg Alarm;
[...]
// Geração de alarme
Alarm_Creation(Alarm, NSEQ_Alarm, 0);           // Criação da mensagem
de alarme
WaitForSingleObject(hMutexforMailslot, INFINITE); // Espera o mutex de escrita no mailslot
estar liberado
bStatus = WriteFile(hMailslot, &Alarm, sizeof(Alarm), &dwBytesEnviados, NULL);
CheckForError(bStatus);
SetEvent(hMailslotEvent);
ReleaseMutex(hMutexforMailslot);
[...]
```

Na thread de monitoramento de alarmes as únicas alterações foram na declaração de uma variável do tipo struct de alarme, e a implementação do envio da mensagem de alarme para o processo de exibição de alarmes, onde a mesma espera pelo mutex de escrita, escreve a mensagem no mailslot, ‘seta’ o evento de que algo foi escrito e libera o mutex para que outra thread possa acessar o mailslot.

```
DWORD WINAPI Messages_Removal() {
[...]
HANDLE thirdArrayOfObjects[3] = { hEscEvent, hNotificationEvents[3], hSemforList2_IN };
HANDLE fourthArrayOfObjects[3] = { hEscEvent, hNotificationEvents[3], hSemforList2_OUT };
HANDLE fifthArrayOfObjects[3] = { hEscEvent, hNotificationEvents[3], hMapFileReadEvent };
DWORD dwRet;
int nTipoEvento;
dataMsg Message;
alarmMsg Alarm;
[...]
else if (nTipoEvento == 2) {
// Lista circular 1 com mensagens à serem retiradas
Message = listaCircular1[ocupPositionList1];

if (Message.DIAG == 55 && Message.ID == 1) {
// Enviar mensagem de falha no CLP1 para monitoramento de alarme
Alarm_Creation(Alarm, Message.NSEQ, 1);           // Criação da mensagem
de alarme
WaitForSingleObject(hMutexforMailslot, INFINITE); // Espera o mutex de escrita no mailslot
estar liberado
bStatus = WriteFile(hMailslot, &Alarm, sizeof(Alarm), &dwBytesEnviados, NULL);
CheckForError(bStatus);
SetEvent(hMailslotEvent);
ReleaseMutex(hMutexforMailslot);
}
}
```



```

else if (Message.DIAG == 55 && Message.ID == 2) {
// Enviar mensagem de falha no CLP2 para monitoramento de alarme
Alarm_Creation(Alarm, Message.NSEQ, 2); // Criação da mensagem
de alarme
WaitForSingleObject(hMutexforMailslot, INFINITE); // Espera o mutex de escrita no mailslot
estar liberado
bStatus = WriteFile(hMailslot, &Alarm, sizeof(Alarm), &dwBytesEnviados, NULL);
CheckForError(bStatus);
SetEvent(hMailslotEvent);
ReleaseMutex(hMutexforMailslot);
}
else {
// Retirada da lista 1 e depósito na lista 2
dwRet = WaitForMultipleObjects(3, thirdArrayOfObjects, FALSE, 5);
nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (nTipoEvento == 1) {
printf("Evento de tecla r - Bloqueio thread 'Messages_Removal'.\n");
MsgRmvState.BLOCKED = TRUE;
printf("Estado 'Messages_Removal': Bloqueada.\n");
}
else if (nTipoEvento == 2) {
listaCircular2[freePositionList2] = Message;
freePositionList2 = (freePositionList2 + 1) % 50;
bStatus = ReleaseSemaphore(hSemforList2_OUT, 1, NULL);
CheckForError(bStatus);

dwRet = WaitForMultipleObjects(3, fourthArrayOfObjects, FALSE, INFINITE);
nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}
else if (nTipoEvento == 1) {
printf("Evento de tecla r - Bloqueio thread 'Messages_Removal'.\n");
MsgRmvState.BLOCKED = TRUE;
printf("Estado 'Messages_Removal': Bloqueada.\n");
}
else if (nTipoEvento == 2) {
*lpImage = listaCircular2[ocupPositionList2];
SetEvent(hMapFileWriteEvent);
ocupPositionList2 = (ocupPositionList2 + 1) % 50;
bStatus = ReleaseSemaphore(hSemforList2_IN, 1, NULL);
CheckForError(bStatus);

dwRet = WaitForMultipleObjects(3, fifthArrayOfObjects, FALSE, INFINITE); // Espera
DataViewProcess sinalizar que leu
nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break;
}

```

```

}
else if (nTipoEvento == 1) {
printf("Evento de tecla r - Bloqueio thread 'Messages_Removal'.\n");
MsgRmvState.BLOCKED = TRUE;
printf("Estado 'Messages_Removal': Bloqueada.\n");
}
else if (nTipoEvento == 2) {
// Read do arquivo mapeado feito e sinalizado
}
}
}
}

ocupPositionList1 = (ocupPositionList1 + 1) % 100;

bStatus = ReleaseSemaphore(hSemforList1_IN, 1, NULL);      // Retirada de uma mensagem
CheckForError(bStatus);
}
}
[...]
```

A thread de retirada de mensagens foi a que sofreu mais alterações. Foram declarados novos arrays de objetos para o tratamento de entrada e saída de mensagens na lista 2, e para o tratamento do arquivo mapeado, quando o mesmo recebe uma confirmação de leitura.

Quando há posições no semáforo de entrada na lista 2, uma variável armazena a mensagem da última posição ocupada da lista 1, e então são feitas comparações para tratar caso a mensagem tenha DIAG igual a 55, situação a qual deve gerar um alarme específico de falha no hardware e enviado por mailslot para o processo de monitoramento de alarmes; caso nenhuma das primeiras condições sejam atendidas, o que significa que o DIAG é diferente de 55, é feito um outro Wait esperando que haja posição no semáforo de entrada de mensagens na lista 2, e caso haja, a mensagem retirada da lista 1 é armazenada na lista circular 2 na última posição livre da mesma, e é feito um 'release' no semáforo de saída da lista 2, indicando que há mensagens a serem retiradas da lista 2, e logo depois, mais um Wait é feito esperando por esse mesmo semáforo, e quando há mensagens para serem retiradas, um ponteiro para a visão do arquivo mapeado em memória recebe a mensagem na última posição ocupada da lista 2 para que DataViewProcess tenha acesso à mensagem, o evento que sinaliza escrita no arquivo é 'setado' e o semáforo de entrada na lista 2 recebe um 'release', indicando que foi liberada uma posição na mesma, e o último Wait é feito esperando apenas a confirmação de que o processo de exibição de dados recebeu e fez a impressão correta da mensagem.

```

[...]  

int DIAG_Format() {  

    int randomNum;  
  

    randomNum = rand() % 99;    // Aumentado para 99 para a implementação correta, antes  

    estava até 70 pois demorava para gerar algum com DIAG igual a 55  

    if (Messages.diag == 0 || randomNum != Messages.diag) {  

        Messages.diag = randomNum;  

        return randomNum;  

        /*if (randomNum != 55) return randomNum;  

    else  

        return 55;*/  

    }  

    else {  

        DIAG_Format();  

    }  

}

float P_INT_Format() {  

    int randomNum;  

    float numDecimal;  
  

    randomNum = rand() % 2001 + 1000;  

    if (Messages.p_int == 0 || randomNum != Messages.p_int) {  

        Messages.p_int = randomNum;  

        if (randomNum % 10 == 0) randomNum++;  

        numDecimal = randomNum / 10.0;  
  

        return numDecimal;  

    }  

    else {  

        P_INT_Format();  

    }  

}

float P_INJ_Format() {  

    int randomNum;  

    float numDecimal;  
  

    randomNum = rand() % 2001 + 1000;  

    if (Messages.p_inj == 0 || randomNum != Messages.p_inj) {  

        Messages.p_inj = randomNum;  

        if (randomNum % 10 == 0) randomNum++;  

        numDecimal = randomNum / 10.0;  
  

        return numDecimal;  

    }  

    else {  

        P_INJ_Format();  

    }  

}

```

```

float TEMP_Format() {
int randomNum;
float numDecimal;

randomNum = rand() % 10001 + 10000;
if (Messages.temp == 0 || randomNum != Messages.temp) {
Messages.temp = randomNum;
if (randomNum % 10 == 0) randomNum++;
numDecimal = randomNum / 10;

return numDecimal;
}
else {
TEMP_Format();
}
}

void Message_Creation(dataMsg& Message, int ID, int NSEQAux) {
//int _NSEQ = NSEQ_Format(NSEQAux);
int _DIAG = DIAG_Format();
float _P_INT = P_INT_Format();
float _P_INJ = P_INJ_Format();
float _TEMP = TEMP_Format();
SYSTEMTIME _TIME;
GetLocalTime(&_TIME);

if (_DIAG == 55) {
Message.NSEQ = NSEQAux;
Message.ID = ID;
Message.DIAG = _DIAG;
Message.P_INT = 0;
Message.P_INJ = 0;
Message.TEMP = 0;
Message.TIME = _TIME;
}
else {
Message.NSEQ = NSEQAux;
Message.ID = ID;
Message.DIAG = _DIAG;
Message.P_INT = _P_INT;
Message.P_INJ = _P_INJ;
Message.TEMP = _TEMP;
Message.TIME = _TIME;
}
}

void Alarm_Creation(alarmMsg& Alarm, int NSEQAux, int ID) {
Alarm.NSEQ = NSEQAux;
GetLocalTime(&Alarm.TIME);
}

```

```

if (ID == 0) {
    Alarm.ID = alarmID_Format(); // Alarme gera ID aleatório
}
else {
    Alarm.ID = ID; // Alarme gerado por falha de funcionamento em CLP's
}
}
}
[...]
```

As funções para geração das mensagens e dos parâmetros não sofreram muitas mudanças, apenas deixaram de serem geradas em strings e passaram a serem geradas de acordo com o tipo do parâmetro (int, float, SYSTEMTIME). Uma observação é que, na primeira etapa, para pegar o tempo, estava sendo usada a função `GetSystemTime()`, mas notei que o tempo retornado não era o correto e ao pesquisar encontrei que o tempo retornado por essa função era em Coordinated Universal Time, fazendo a mudança para a função `GetLocalTime()`, é retornado corretamente o tempo do relógio do computador local o qual está sendo executado o programa.

- *AlarmProcess.cpp:*

```

[...]
```

// Declaração de objetos de sincronização

```

HANDLE hEscEvent, hEventA, hMailslot;
HANDLE hMailslotEvent; // Evento para sincronização de entrada de dados
no mailslot
HANDLE hMailslotCriadoEvent; // Evento para sinalização da criação do mailslot
[...]
```

// Declaração de struct para armazenar parametros das mensagens de alarme

```

struct alarmMsg {
    int NSEQ;
    int ID;
    SYSTEMTIME TIME;
};

// Declaração de funções
string Alarm_Format(alarmMsg& Alarm);
string textoAlarme(alarmMsg& Alarm);

BOOL bStatus;
[...]
```

No processo de exibição de alarmes, foram declarados novos handles para sincronização do mailslot, e sinalização de criação do mesmo. Foram declaradas também a struct do tipo de mensagem de alarme, utilizada também no `MainProcess`, e duas funções, uma para concatenar todos os parâmetros da mensagem com o respectivo texto da mensagem de alarme em uma string para

impressão, e outra para a formação de 10 textos de alarme comuns em processo de dessulfuração.

```
[...]
int main() {
int countOfMsgs = 0;    // Contador para contar quantas mensagens de alarme chegaram com a
thread bloqueada
    alarmMsg AlarmBuffer;
    DWORD dwBytesLidos;
[...]
```

Foram declaradas três variáveis, uma para armazenar a mensagem de alarme que chega pelo mailslot, outra para identificar o número de bytes lidos da mensagem, e um contador de mensagens que chegam enquanto o processo está bloqueada.

```
[...]
hMailslotCriadoEvent = OpenEvent(EVENT_MODIFY_STATE | SYNCHRONIZE, FALSE,
"Evento_MailslotCriado");
CheckForError(hMailslotCriadoEvent);

hMailslotEvent = OpenEvent(EVENT_MODIFY_STATE | SYNCHRONIZE, FALSE, "Evento_Mailslot");
CheckForError(hMailslotEvent);

//---Criação do Mailslot---
hMailslot = CreateMailslot(
"\\\\.\\mailslot\\MyMailslot",
0,
MAILSLOT_WAIT_FOREVER,
NULL);
CheckForError(hMailslot != INVALID_HANDLE_VALUE);

// Sinaliza que mailslot está pronto para receber mensagens
SetEvent(hMailslotCriadoEvent);

// Recebimento dos eventos
HANDLE arrayOfObjects[3] = { hEscEvent, hEventA, hMailslotEvent };
int nTipoEvento;
DWORD dwRet;
[...]
```

Foram abertos os eventos criados no processo principal para sinalização de criação e o de ‘uso’ do mailslot, e logo abaixo, foi criado o mailslot e ‘setado’ o evento que sinaliza para a thread principal que o mesmo foi criado, para que ela crie o arquivo do mailslot.

```
[...]
if (countOfMsgs > 0) {
for (int i = 0; i < countOfMsgs; i++) {
bStatus = ReadFile(hMailslot, &AlarmBuffer, sizeof(AlarmBuffer), &dwBytesLidos, NULL);
CheckForError(bStatus);
}
```

```

cout << Alarm_Format(AlarmBuffer) << endl;
}
countOfMsgs = 0;
}
dwRet = WaitForMultipleObjects(3, arrayOfObjects, FALSE, INFINITE);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break; // Evento ESC
}
else if (nTipoEvento == 1) {
printf("Evento de tecla a - Bloqueio thread 'AlarmProcess'.\n");
AlarmMState.BLOCKED = TRUE;
printf("Estado 'AlarmProcess': Bloqueada.\n");
}
else if (nTipoEvento == 2) {
bStatus = ReadFile(hMailslot, &AlarmBuffer, sizeof(AlarmBuffer), &dwBytesLidos, NULL);
CheckForError(bStatus);

cout << Alarm_Format(AlarmBuffer) << endl;
}
}
else {
dwRet = WaitForMultipleObjects(3, arrayOfObjects, FALSE, INFINITE);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 0) {
break; // Evento ESC
}
else if (nTipoEvento == 1) {
printf("Evento de tecla a - Desbloqueio thread 'AlarmProcess'.\n");
AlarmMState.BLOCKED = FALSE;
printf("Estado 'AlarmProcess': Desbloqueada.\n");
}
else if (nTipoEvento == 2) {
countOfMsgs++;
}
}
[...]
```

Dentro do loop, é feita uma espera pelo evento de mailslot sinalizado pelo processo principal, caso esse ocorra é feito um ReadFile armazenando no buffer a mensagem de alarme enviada e depois é impressa a mensagem formatada pela função Alarm_Format(). Na lógica de bloqueio da thread, foi inserido um tratamento para quando chegam mensagens e o processo está bloqueado: a variável countOfMsgs soma mais 1 a cada mensagem que chega, e quando ocorre o desbloqueio, se o número desse contador for maior que 0 indicando que chegaram mensagens, um loop é feito iterando até o número de mensagens que chegaram, realizando um ReadFile e a impressão de cada mensagem; isso

foi implementado pois, estava ocorrendo um erro de quando a thread estava bloqueada e sofria um desbloqueio, as mensagens eram lidas com atraso, pois o mailslot acumula as mensagens em fila e retira uma a cada Read feito.

```
string Alarm_Format(alarmMsg& Alarm) {
    ostringstream alarmAux;
    string alarmReturn;

    if (Alarm.ID == 1) {        // 1 - Falha em CLP 1 | 2 - Falha em CLP 2
        alarmAux << setw(2) << setfill('0') << Alarm.TIME.wHour << ":" << setw(2) << setfill('0') <<
        Alarm.TIME.wMinute << ":" << setw(2) << setfill('0') << Alarm.TIME.wSecond << " NSEQ: " <<
        setw(5) << setfill('0') << Alarm.NSEQ << " FALHA DE HARDWARE CLP No. 1";
        alarmReturn = alarmAux.str();
        return alarmReturn;
    }
    else if (Alarm.ID == 2) {
        alarmAux << setw(2) << setfill('0') << Alarm.TIME.wHour << ":" << setw(2) << setfill('0') <<
        Alarm.TIME.wMinute << ":" << setw(2) << setfill('0') << Alarm.TIME.wSecond << " NSEQ: " <<
        setw(5) << setfill('0') << Alarm.NSEQ << " FALHA DE HARDWARE CLP No. 2";
        alarmReturn = alarmAux.str();
        return alarmReturn;
    }
    else {
        alarmAux << setw(2) << setfill('0') << Alarm.TIME.wHour << ":" << setw(2) << setfill('0') <<
        Alarm.TIME.wMinute << ":" << setw(2) << setfill('0') << Alarm.TIME.wSecond << " NSEQ: " <<
        setw(5) << setfill('0') << Alarm.NSEQ << " ID: " << setw(2) << setfill('0') << Alarm.ID << " " <<
        textoAlarme(Alarm);
        alarmReturn = alarmAux.str();
        return alarmReturn;
    }
}

string textoAlarme(alarmMsg& Alarm) {
    if (Alarm.ID > 0 && Alarm.ID <= 10) {
        string texto = "Temperatura crítica atingida no reator de dessulfuração.";
        return texto;
    }
    else if (Alarm.ID > 10 && Alarm.ID <= 20) {
        string texto = "Detectado vazamento de gás sulfídrico.";
        return texto;
    }
    else if (Alarm.ID > 20 && Alarm.ID <= 30) {
        string texto = "Pressão fora dos limites seguros no reator de dessulfuração.";
        return texto;
    }
    else if (Alarm.ID > 30 && Alarm.ID <= 40) {
        string texto = "Sistema de injeção de reagente inoperante.";
        return texto;
    }
    else if (Alarm.ID > 40 && Alarm.ID <= 50) {
        string texto = "Baixo fluxo de gás detectado no sistema.";
    }
}
```



```

return texto;
}
else if (Alarm.ID > 50 && Alarm.ID <= 60) {
string texto = "Níveis elevados de subprodutos não desejados detectados.";
return texto;
}
else if (Alarm.ID > 60 && Alarm.ID <= 70) {
string texto = "Sistema de monitoramento ambiental offline.";
return texto;
}
else if (Alarm.ID > 70 && Alarm.ID <= 80) {
string texto = "Sobrecarga elétrica detectada no sistema de dessulfuração.";
return texto;
}
else if (Alarm.ID > 80 && Alarm.ID <= 90) {
string texto = "Sinais de corrosão detectados em componentes do sistema.";
return texto;
}
else {
string texto = "Níveis de efluentes fora dos padrões de segurança.";
return texto;
}
}

```

A função de formatação concatena os parâmetros em uma string e retorna essa string para o cout, e a função de geração dos textos, foi implementada uma lógica que gera mensagens de texto diferentes a cada múltiplo de 10 do ID, ou seja, de 0 a 10 é um texto, de 11 a 20 é outro, e assim por seguinte.

- *DataViewProcess.cpp:*

```

[...]
HANDLE hMappedSection;           // Arquivo mapeado em memória
HANDLE hMapFileWriteEvent;       // Evento de sinalização de escrita no arquivo mapeado
HANDLE hMapFileReadEvent;        // Evento de sinalização de leitura no arquivo mapeado

// Declaração de struct para armazenar parametros das mensagens de dados do processo
struct dataMsg {
int NSEQ;
int ID;
int DIAG;
float P_INT;
float P_INJ;
float TEMP;
SYSTEMTIME TIME;
};

dataMsg msgSize; // Instanciando uma variável do tipo da struct apenas para passar o seu
sizeof() na função de CreateFileMapping

```

[...]

No processo de exibição de dados, foram declarados novos handles, um para o arquivo mapeado em memória, uma para o evento de escrita e outro para o evento de leitura, no arquivo. Foi declarado também a struct do tipo da mensagem de alarme utilizada também no MainProcess, e foi instanciado uma variável do tipo da struct para ser passado como parâmetro no mapeamento da visão do arquivo.

```
int main(){
[...]
// Abertura do arquivo mapeado em memória para lista 2
hMappedSection = OpenFileMapping(FILE_MAP_ALL_ACCESS,
FALSE,
"ArquivoMapeado");
CheckForError(hMappedSection);

lpImage = (dataMsg*)MapViewOfFile(hMappedSection,
FILE_MAP_WRITE,
0,
0,
sizeof(msgSize) * 50);
CheckForError(lpImage);
[...]
```

```
hMapFileWriteEvent = OpenEvent(EVENT_MODIFY_STATE | SYNCHRONIZE, FALSE,
"Evento_MapFileWrite");
CheckForError(hMapFileWriteEvent);

hMapFileReadEvent = OpenEvent(EVENT_MODIFY_STATE | SYNCHRONIZE, FALSE,
"Evento_MapFileRead");
CheckForError(hMapFileReadEvent);
[...]
```

Indo para o main, foi aberto o arquivo mapeado em memória criado no processo principal e passada ao ponteiro a visão do arquivo, da mesma forma feita no MainProcess, e logo abaixo foram abertos os eventos de escrita e leitura nesse arquivo.

```
dwRet = WaitForMultipleObjects(3, secondArrayOfObjects, FALSE, INFINITE);

nTipoEvento = dwRet - WAIT_OBJECT_0;
if (nTipoEvento == 1) {
printf("Evento de tecla p - Bloqueio thread 'DataViewProcess'.\n");
DataViewState.BLOCKED = TRUE;
printf("Estado 'DataViewProcess': Bloqueada.\n");
}
else if (nTipoEvento == 2) {
ostringstream dataMsgAux;
string dataMsg;
```

```

dataMsgAux << setw(2) << setfill('0') << lplmage->TIME.wHour << ":" << setw(2) << setfill('0') <<
lplmage->TIME.wMinute << ":" << setw(2) << setfill('0') << lplmage->TIME.wSecond << " NSEQ: "
<< lplmage->NSEQ << " PR INT: " << lplmage->P_INT << " PR N2: " << lplmage->P_INJ << " TEMP: "
<< lplmage->TEMP;
dataMsg = dataMsgAux.str();
cout << dataMsg << endl;
SetEvent(hMapFileReadEvent);
}

```

A lógica da impressão das mensagens é simples, um Wait espera pelo evento de escrita sinalizado pelo MainProcess ao serem retiradas mensagens da lista 2, quando o mesmo ocorre, é concatenado em uma string os parâmetros da mensagem armazenada no ponteiro lplmag da visão do arquivo, essa string é então impressa, e por último, o evento de leitura pelo processo de exibição é 'setado' o que sinaliza ao processo principal que a mensagem atual já foi retirada e impressa.

4. Resultados

Nessa seção, serão colocados prints dos consoles do que foi pedido no enunciado.

❑ ETAPA 1

- Inicialização:

```

C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\x64\Debug\MainProcess.exe
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Estado 'CLP2_Read': Desbloqueada.
Estado 'CLP1_Read': Desbloqueada.
Estado 'Alarm_Monitor': Desbloqueada.
Estado 'Messages_Removal': Desbloqueada.
00001;1;41;0145.8;0133.1;1649;2:40:45
00000;2;55;0000.0;0000.0;0000.0;2:40:45
00002;2;44;0247.3;0234.4;1696;2:40:45
00003;1;55;0000.0;0000.0;0000.0;2:40:45
00005;2;34;0270.3;0113.1;1327;2:40:46
00004;1;12;0145.2;0270.3;1814;2:40:46
Alar me criado:00000;53;2:40:46
00006;1;41;0181.9;0295.7;1049;2:40:46
00007;2;27;0295.7;0149.1;1299;2:40:46
00008;2;42;0182.5;0243.4;1238;2:40:47
00009;1;55;0000.0;0000.0;0000.0;2:40:47

```

Figura 5: Console principal logo após iniciação

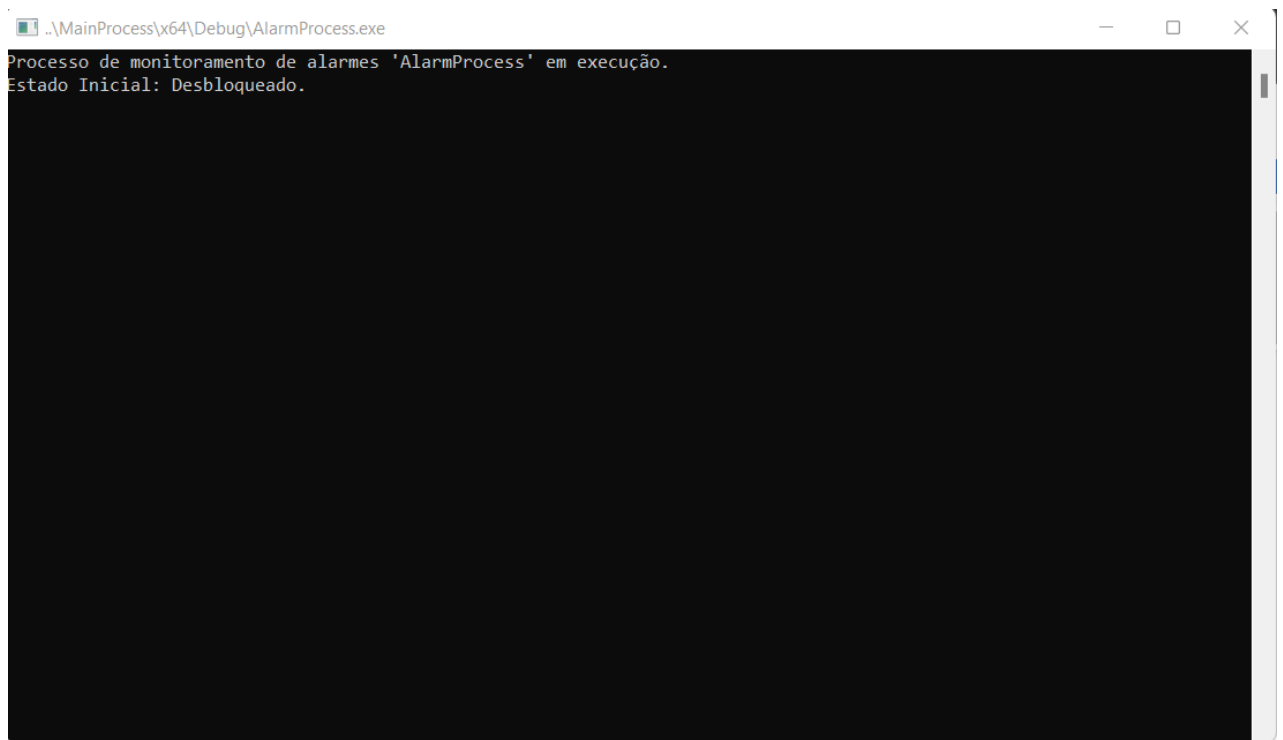


Figura 6: Console do processo de exibição de alarmes logo após iniciação

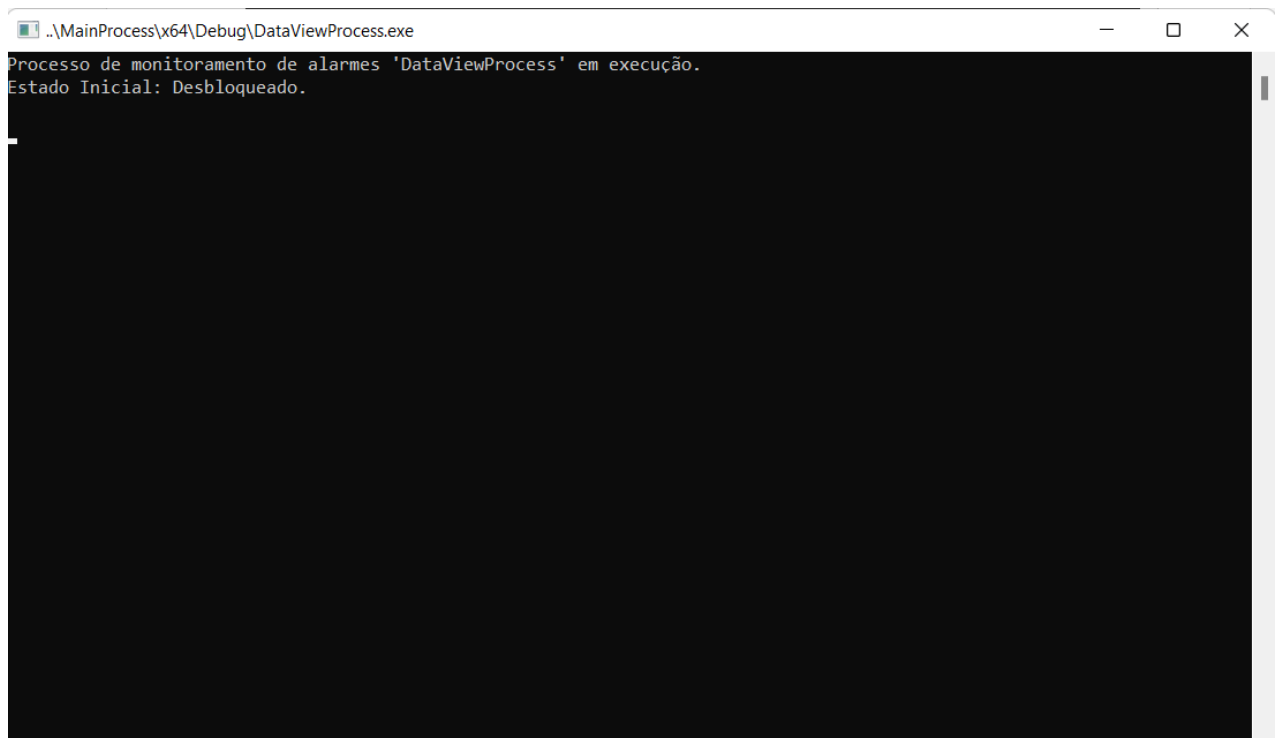


Figura 7: Console do processo de exibição de dados logo após iniciação

- Evento Teclas 1, 2, m e r:

```

C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\x64\Debug\MainProcess.exe
00144;2;32;0128.1;0207.5;1142;23:9:50
00145;1;32;0128.1;0207.5;1142;23:9:50
00146;2;55;0000.0;0000.0;0000.0;23:9:50
00147;1;55;0000.0;0000.0;0000.0;23:9:50
Evento 'Tecla 1' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla 1 - Bloqueio thread 'CLP1_Read'.
Estado 'CLP1_Read': Bloqueada.
00148;2;39;0115.3;0270.9;1718;23:9:51
00147;1;39;0115.3;0270.9;1718;23:9:51
Alarme criado:00003;54;23:9:51
00149;2;26;0231.4;0136.7;1869;23:9:51
00150;2;5;0155.1;0243.3;1654;23:9:52
00151;2;21;0250.8;0113.1;1805;23:9:52
00152;2;18;0275.2;0113.1;1242;23:9:53
Evento 'Tecla 2' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla 2 - Bloqueio thread 'CLP2_Read'.
Estado 'CLP2_Read': Bloqueada.
00152;2;39;0298.4;0167.9;1252;23:9:53
Evento 'Tecla r' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla r - Bloqueio thread 'Messages_Removal'.
Estado 'Messages_Removal': Bloqueada.
Alarme criado:00004;11;23:9:55
Evento 'Tecla m' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla m - Bloqueio thread 'Alarm_Monitor'.
Estado 'Alarm_Monitor': Bloqueada.

```

Figura 8: Console do principal logo após todos eventos

```

Selecionar C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\x64\Debug\MainProcess.exe
Estado 'Alarm_Monitor': Desbloqueada.
Alarme criado:00005;16;23:12:18
Evento 'Tecla r' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla r - Desbloqueio thread 'Messages_Removal'.
Estado 'Messages_Removal': Desbloqueada.
Alarme criado:00006;61;23:12:20
Evento 'Tecla 2' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla 2 - Desbloqueio thread 'CLP2_Read'.
Estado 'CLP2_Read': Desbloqueada.
00153;2;9;0242.9;0285.7;1294;23:12:21
Alarme criado:00007;25;23:12:22
00154;2;53;0218.4;0229.6;1041;23:12:22
00155;2;6;0109.7;0147.6;1628;23:12:22
00156;2;55;0000.0;0000.0;0000.0;23:12:23
00157;2;26;0166.1;0278.4;1226;23:12:23
00158;2;43;0134.3;0217.1;1005;23:12:24
00159;2;2;0180.3;0283.2;1094;23:12:24
00160;2;43;0274.3;0130.7;1955;23:12:25
00161;2;55;0000.0;0000.0;0000.0;23:12:25
00162;2;26;0121.2;0212.6;1216;23:12:26
Evento 'Tecla 1' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla 1 - Desbloqueio thread 'CLP1_Read'.
Estado 'CLP1_Read': Desbloqueada.
00163;2;5;0144.1;0216.8;1046;23:12:26
00164;1;26;0231.4;0136.7;1869;23:12:27
Alarme criado:00008;75;23:12:27

```

Figura 9: Console principal logo após todos eventos, uma segunda vez

- Eventos p e a:

```

C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\Debug\AlarmProcess.exe
00049;2;18;0108.1;0191.8;1654;23:16:31
Evento 'Tecla 1' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla 1 - Bloqueio thread 'CLP1_Read'.
Estado 'CLP1_Read': Bloqueada.
00048;1;3;0210.1;0163.7;1965;23:16:31
00050;2;3;0210.1;0163.7;1965;23:16:31
Evento 'Tecla 2' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Alarme criado:00003;54;23:16:31
Evento de tecla 2 - Bloqueio thread 'CLP2_Read'.
Estado 'CLP2_Read': Bloqueada.
00050;2;24;0292.6;0297.1;1230;23:16:32
Evento 'Tecla r' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla r - Bloqueio thread 'Messages_Removal'.
Estado 'Messages_Removal': Bloqueada.
Evento 'Tecla m' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento de tecla m - Bloqueio thread 'Alarm_Monitor'.
Estado 'Alarm_Monitor': Bloqueada.
Evento 'Tecla a' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento 'Tecla p' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento 'Tecla a' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento 'Tecla p' setado!
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
Evento 'Tecla p' setado!
Bloqueio thread 'DataViewProcess'.\n");
;
ess': Bloqueada.\n");

, arrayOfObjects, FALSE, INFINITE);
JECT_0);
T_0;
Desbloqueio thread 'DataViewProcess'.\n");
E;
ess': Desbloqueada.\n");

MainProcess\Debug\AlarmProcess.exe
Processo de monitoramento de alarmes 'AlarmProcess' em execução.
Estado Inicial: Desbloqueado.

Evento de tecla a - Bloqueio thread 'AlarmProcess'.
Estado 'AlarmProcess': Bloqueada.
Evento de tecla a - Desbloqueio thread 'AlarmProcess'.
Estado 'AlarmProcess': Desbloqueada.

MainProcess\Debug\DataViewProcess.exe
Processo de monitoramento de alarmes 'DataViewProcess' em execução.
Estado Inicial: Desbloqueado.

Evento de tecla p - Bloqueio thread 'DataViewProcess'.
Estado 'DataViewProcess': Bloqueada.
Evento de tecla p - Desbloqueio thread 'DataViewProcess'.
Estado 'DataViewProcess': Desbloqueada.

```

Figura 10: Consoles principal, exibição de dados e exibição de alarmes

Acima, fiz um print de todos consoles juntos, para poder ser observado que ao ser setado o evento no console principal, os outros processos recebem esse evento e alteram seu estado corretamente.

- Encerramento:

```

C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\x64\Debug\MainProcess.exe
00040;2;29;0161.7;0107.2;1995;0:16:28
00041;1;29;0161.7;0107.2;1995;0:16:28
00042;2;55;0000.0;0000.0;0000.0:0:16:28
00043;1;55;0000.0;0000.0;0000.0:0:16:28
00044;2;1;0129.5;0193.6;1243;0:16:29
00045;1;1;0129.5;0193.6;1243;0:16:29
00046;2;55;0000.0;0000.0;0000.0:0:16:29
00047;1;55;0000.0;0000.0;0000.0:0:16:29
00048;2;18;0108.1;0191.8;1654;0:16:30
00049;1;18;0108.1;0191.8;1654;0:16:30
00050;2;3;0210.1;0163.7;1965;0:16:30
00051;1;3;0210.1;0163.7;1965;0:16:30
Alarme criado:00003;54;0:16:31
00052;2;24;0292.6;0297.1;1230;0:16:31
00053;1;24;0292.6;0297.1;1230;0:16:31
00054;2;33;0137.5;0201.9;1874;0:16:31
00055;1;33;0137.5;0201.9;1874;0:16:31
Evento 'Tecla ESC' setado!
Encerrando thread 'Alarm_Monitor'.
Encerrando thread 'Messages_Removal'.
Encerrando thread 'CLP2_Read'.
Encerrando thread 'CLP1_Read'.
Thread 'CLP1_Read' terminou (0).
Thread 'CLP2_Read' terminou (0).
Thread 'Alarm_Monitor' terminou (0).
Thread 'Messages_Removal' terminou (0).

C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\x64\Debug\MainProcess.exe
Encerrado com o código 0.
Pressione qualquer tecla para fechar esta janela...

.\MainProcess\x64\Debug\AlarmProcess.exe
Processo de monitoramento de alarmes 'AlarmProcess' em execução.
Estado Inicial: Desbloqueado.

Encerrando thread 'AlarmProcess'.
Digite qualquer tecla para fechar a janela.

.\MainProcess\x64\Debug\DataViewProcess.exe
Processo de monitoramento de alarmes 'DataViewProcess' em execução.
Estado Inicial: Desbloqueado.

Encerrando thread 'DataViewProcess'.
Digite qualquer tecla para fechar a janela.

```

Figura 11: Consoles principal, exibição de dados e exibição de alarmes após evento ESC

ETAPA 2

- Inicialização:

```

C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\x64\Debug\MainProcess.exe
Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
1 -> Bloqueio/Desbloqueio leitura do CLP 1
2 -> Bloqueio/Desbloqueio leitura do CLP 2
m -> Bloqueio/Desbloqueio monitoramento de alarmes
r -> Bloqueio/Desbloqueio retirada de mensagens
p -> Bloqueio/Desbloqueio exibição de dados do processo
a -> Bloqueio/Desbloqueio exibição de alarmes
ESC -> Encerramento da execução
Estado 'CLP2_Read': Desbloqueada.
Estado 'Messages_Removal': Desbloqueada.
Estado 'CLP1_Read': Desbloqueada.
Estado 'Alarm_Monitor': Desbloqueada.

.\MainProcess\x64\Debug\AlarmProcess.exe
Processo de monitoramento de alarmes 'AlarmProcess' em execução.
Estado Inicial: Desbloqueado.

18:38:31 NSEQ: 00000 ID: 53 Níveis elevados de subprodutos não desejados detectados.
18:38:34 NSEQ: 00001 ID: 67 Sistema de monitoramento ambiental offline.
18:38:39 NSEQ: 00002 ID: 82 Sinais de corrosão detectados em componentes do sistema.
18:38:43 NSEQ: 00003 ID: 54 Níveis elevados de subprodutos não desejados detectados.

.\MainProcess\x64\Debug\DataViewProcess.exe
18:38:38 NSEQ: 30 PR INT: 203.4 PR N2: 117.9 TEMP: 1184
18:38:39 NSEQ: 32 PR INT: 109.1 PR N2: 203.6 TEMP: 1894
18:38:39 NSEQ: 33 PR INT: 203.6 PR N2: 193.8 TEMP: 1926
18:38:39 NSEQ: 34 PR INT: 243.3 PR N2: 279.4 TEMP: 1588
18:38:39 NSEQ: 35 PR INT: 163.7 PR N2: 243.3 TEMP: 1380
18:38:40 NSEQ: 37 PR INT: 172.6 PR N2: 135.8 TEMP: 1534
18:38:40 NSEQ: 36 PR INT: 135.8 PR N2: 234.3 TEMP: 1500
18:38:40 NSEQ: 38 PR INT: 138.1 PR N2: 254.7 TEMP: 1962
18:38:40 NSEQ: 39 PR INT: 208.6 PR N2: 138.1 TEMP: 1354
18:38:41 NSEQ: 40 PR INT: 161.7 PR N2: 107.2 TEMP: 1995
18:38:41 NSEQ: 41 PR INT: 107.2 PR N2: 294.5 TEMP: 1875
18:38:41 NSEQ: 43 PR INT: 196.4 PR N2: 237.3 TEMP: 1393
18:38:41 NSEQ: 42 PR INT: 283.5 PR N2: 196.4 TEMP: 1737
18:38:42 NSEQ: 44 PR INT: 129.5 PR N2: 193.6 TEMP: 1243
18:38:42 NSEQ: 45 PR INT: 193.6 PR N2: 142.3 TEMP: 1462
18:38:42 NSEQ: 46 PR INT: 253.5 PR N2: 252.8 TEMP: 1611
18:38:42 NSEQ: 47 PR INT: 231.8 PR N2: 253.5 TEMP: 1153
18:38:43 NSEQ: 49 PR INT: 108.1 PR N2: 191.8 TEMP: 1654
18:38:43 NSEQ: 48 PR INT: 191.8 PR N2: 153.3 TEMP: 1483
18:38:43 NSEQ: 51 PR INT: 163.7 PR N2: 264.4 TEMP: 1270
18:38:43 NSEQ: 50 PR INT: 210.1 PR N2: 163.7 TEMP: 1965
18:38:44 NSEQ: 52 PR INT: 297.1 PR N2: 130.5 TEMP: 1167
18:38:44 NSEQ: 53 PR INT: 292.6 PR N2: 297.1 TEMP: 1230
18:38:44 NSEQ: 54 PR INT: 201.9 PR N2: 173.1 TEMP: 1692
18:38:45 NSEQ: 55 PR INT: 137.5 PR N2: 201.9 TEMP: 1874
18:38:45 NSEQ: 56 PR INT: 206.3 PR N2: 126.7 TEMP: 1582
18:38:45 NSEQ: 57 PR INT: 126.7 PR N2: 282.7 TEMP: 1677
18:38:45 NSEQ: 59 PR INT: 209.5 PR N2: 150.4 TEMP: 1398
18:38:45 NSEQ: 58 PR INT: 256.6 PR N2: 209.5 TEMP: 1651

```

Figura 12: Consoles principal, exibição de dados e exibição de alarmes após inicialização

- Eventos de bloqueio/desbloqueio:

```
C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica(MainProcess)\x64\De...
MainProcess\x64\Debug\AlarmProcess.exe

Digite uma tecla entre 1, 2, m, r, p, a ou ESC:
1 -> Bloqueio/Desbloqueio leitura do CLP 1
2 -> Bloqueio/Desbloqueio leitura do CLP 2
m -> Bloqueio/Desbloqueio monitoramento de alarmes
r -> Bloqueio/Desbloqueio retirada de mensagens
p -> Bloqueio/Desbloqueio exibição de dados do processo
a -> Bloqueio/Desbloqueio exibição de alarmes
ESC -> Encerramento da execução

Estado 'CLP2_Read': Desbloqueada.
Estado 'Messages_Removal': Desbloqueada.
Estado 'CLP1_Read': Desbloqueada.
Estado 'Alarm_Monitor': Desbloqueada.
Evento 'Tecla 1' setado!
Evento de tecla 1 - Bloqueio thread 'CLP1_Read'.
Estado 'CLP1_Read': Bloqueada.
Evento 'Tecla 2' setado!
Evento de tecla 2 - Bloqueio thread 'CLP2_Read'.
Estado 'CLP2_Read': Bloqueada.
Evento 'Tecla m' setado!
Evento de tecla m - Bloqueio thread 'Alarm_Monitor'.
Estado 'Alarm_Monitor': Bloqueada.
Evento 'Tecla r' setado!
Evento de tecla r - Bloqueio thread 'Messages_Removal'.
Estado 'Messages_Removal': Bloqueada.
Evento 'Tecla p' setado!
Evento 'Tecla a' setado!

18:39:04 NSEQ: 00010 ID: 51 Níveis elevados de subprodutos não desejados detectados.
18:39:09 NSEQ: 00011 ID: 54 Níveis elevados de subprodutos não desejados detectados.
18:39:10 NSEQ: 00012 ID: 07 Temperatura crítica atingida no reator de dessulfuração.
18:39:10 NSEQ: 00158 FALHA DE HARDWARE CLP No. 1
18:39:13 NSEQ: 00013 ID: 05 Temperatura crítica atingida no reator de dessulfuração.
18:39:18 NSEQ: 00014 ID: 95 Níveis de efluentes fora dos padrões de segurança.
18:39:20 NSEQ: 00015 ID: 45 Baixo fluxo de gás detectado no sistema.
18:39:24 NSEQ: 00016 ID: 54 Níveis elevados de subprodutos não desejados detectados.
18:39:27 NSEQ: 00017 ID: 13 Detectado vazamento de gás sulfídrico.
18:39:29 NSEQ: 00018 ID: 64 Sistema de monitoramento ambiental offline.
18:39:31 NSEQ: 00019 ID: 93 Níveis de efluentes fora dos padrões de segurança.
18:39:33 NSEQ: 00020 ID: 51 Níveis elevados de subprodutos não desejados detectados.
18:39:37 NSEQ: 00021 ID: 39 Sistema de injeção de reagente inoperante.
18:39:40 NSEQ: 00022 ID: 11 Detectado vazamento de gás sulfídrico.
18:39:44 NSEQ: 00023 ID: 88 Sinais de corrosão detectados em componentes do sistema.
18:39:45 NSEQ: 00024 ID: 37 Sistema de injeção de reagente inoperante.
18:39:48 NSEQ: 00025 ID: 23 Pressão fora dos limites seguros no reator de dessulfuração.
18:39:49 NSEQ: 00312 FALHA DE HARDWARE CLP No. 2
18:39:50 NSEQ: 00026 ID: 87 Sinais de corrosão detectados em componentes do sistema.
18:39:51 NSEQ: 00027 ID: 88 Sinais de corrosão detectados em componentes do sistema.
18:39:52 NSEQ: 00028 ID: 39 Sistema de injeção de reagente inoperante.
18:39:57 NSEQ: 00029 ID: 85 Sinais de corrosão detectados em componentes do sistema.
18:39:58 NSEQ: 00030 ID: 65 Sistema de monitoramento ambiental offline.
18:40:01 NSEQ: 00031 ID: 60 Níveis elevados de subprodutos não desejados detectados.
18:40:03 NSEQ: 00032 ID: 10 Temperatura crítica atingida no reator de dessulfuração.
18:40:05 NSEQ: 00033 ID: 32 Sistema de injeção de reagente inoperante.
18:40:09 NSEQ: 00034 ID: 76 Sobrecarga elétrica detectada no sistema de dessulfuração.
Evento de tecla a - Bloqueio thread 'AlarmProcess'.
Estado 'AlarmProcess': Bloqueada.

18:40:01 NSEQ: 360 PR INT: 170.1 PR N2: 280.3 TEMP: 1923
18:40:01 NSEQ: 361 PR INT: 222.8 PR N2: 175.1 TEMP: 1520
18:40:02 NSEQ: 362 PR INT: 219.3 PR N2: 197.3 TEMP: 1153
18:40:02 NSEQ: 363 PR INT: 253.9 PR N2: 130.3 TEMP: 1142
18:40:02 NSEQ: 364 PR INT: 224.2 PR N2: 257.8 TEMP: 1364
18:40:02 NSEQ: 365 PR INT: 241.7 PR N2: 208.8 TEMP: 1124
18:40:03 NSEQ: 366 PR INT: 264.2 PR N2: 197.1 TEMP: 1786
18:40:03 NSEQ: 367 PR INT: 285.6 PR N2: 190.2 TEMP: 1187
18:40:03 NSEQ: 368 PR INT: 169.8 PR N2: 253.8 TEMP: 1867
18:40:03 NSEQ: 369 PR INT: 207.1 PR N2: 253.5 TEMP: 1871
18:40:04 NSEQ: 370 PR INT: 225.5 PR N2: 151.5 TEMP: 1398
18:40:04 NSEQ: 371 PR INT: 166.9 PR N2: 276.9 TEMP: 1526
18:40:04 NSEQ: 372 PR INT: 297.9 PR N2: 127.5 TEMP: 1854
18:40:04 NSEQ: 373 PR INT: 293.7 PR N2: 186.4 TEMP: 1593
18:40:05 NSEQ: 375 PR INT: 249.6 PR N2: 130.4 TEMP: 1786
18:40:05 NSEQ: 374 PR INT: 153.1 PR N2: 223.4 TEMP: 1550
18:40:05 NSEQ: 376 PR INT: 285.7 PR N2: 259.7 TEMP: 1832
18:40:05 NSEQ: 377 PR INT: 130.9 PR N2: 212.2 TEMP: 1447
18:40:06 NSEQ: 379 PR INT: 146.1 PR N2: 213.9 TEMP: 1508
18:40:06 NSEQ: 378 PR INT: 207.5 PR N2: 155.6 TEMP: 1976
18:40:07 NSEQ: 380 PR INT: 274.9 PR N2: 288.7 TEMP: 1710
18:40:07 NSEQ: 381 PR INT: 209.5 PR N2: 141.6 TEMP: 1352
18:40:07 NSEQ: 382 PR INT: 195.6 PR N2: 100.8 TEMP: 1556
18:40:07 NSEQ: 380 PR INT: 252.6 PR N2: 259.5 TEMP: 1696
18:40:08 NSEQ: 383 PR INT: 253.3 PR N2: 233.5 TEMP: 1288
18:40:08 NSEQ: 384 PR INT: 163.1 PR N2: 139.8 TEMP: 1646
18:40:09 NSEQ: 384 PR INT: 167.9 PR N2: 258.8 TEMP: 1134
Evento de tecla p - Bloqueio thread 'DataViewProcess'.
Estado 'DataViewProcess': Bloqueada.
```

Figura 13: Consoles principal, exibição de dados e exibição de alarmes após eventos

- Encerramento:

```

Console de Depuração do Microsoft Visual Studio
MainProcess(x64)\Debug\AlarmProcess.exe

Estado 'Messages_Removal': Bloqueada.
Evento 'Tecla p' setado!
Evento 'Tecla a' setado!
Evento 'Tecla 1' setado!
Evento de tecla 1 - Desbloqueio thread 'CLP1_Read'.
Estado 'CLP1_Read': Desbloqueada.
Evento 'Tecla 2' setado!
Evento de tecla 2 - Desbloqueio thread 'CLP2_Read'.
Estado 'CLP2_Read': Desbloqueada.
Evento 'Tecla m' setado!
Evento de tecla m - Desbloqueio thread 'Alarm_Monitor'.
Estado 'Alarm_Monitor': Desbloqueada.
Evento 'Tecla r' setado!
Evento de tecla r - Desbloqueio thread 'Messages_Removal'.
Estado 'Messages_Removal': Desbloqueada.
Evento 'Tecla p' setado!
Evento 'Tecla a' setado!
Evento 'Tecla ESC' setado!
Encerrando thread 'CLP2_Read'.
Encerrando thread 'Messages_Removal'.
Encerrando thread 'CLP1_Read'.
Encerrando thread 'Alarm_Monitor'.
Thread 'CLP1_Read' terminou (0).
Thread 'CLP2_Read' terminou (0).
Thread 'Alarm_Monitor' terminou (0).
Thread 'Messages_Removal' terminou (0).
0 C:\Users\Gustavo Santana\Desktop\ATR Projetos\TP_Siderurgica\MainProcess\AlarmProcess.exe
Encerrado com o código 0.
Digite qualquer tecla para fechar a janela.

MainProcess(x64)\Debug\DataViewProcess.exe

18:41:33 NSEQ: 403 PR INT: 274.1 PR N2: 291.4 TEMP: 1335
18:41:34 NSEQ: 404 PR INT: 249.2 PR N2: 157.5 TEMP: 1051
18:41:34 NSEQ: 405 PR INT: 175.4 PR N2: 194.2 TEMP: 1495
18:41:34 NSEQ: 406 PR INT: 233.1 PR N2: 107.1 TEMP: 1791
18:41:34 NSEQ: 407 PR INT: 171.1 PR N2: 239.9 TEMP: 1102
18:41:35 NSEQ: 408 PR INT: 218.1 PR N2: 293.8 TEMP: 1507
18:41:35 NSEQ: 409 PR INT: 198.6 PR N2: 153.8 TEMP: 1949
18:41:35 NSEQ: 410 PR INT: 159.9 PR N2: 116.1 TEMP: 1442
18:41:35 NSEQ: 411 PR INT: 251.1 PR N2: 295.6 TEMP: 1334
18:41:36 NSEQ: 412 PR INT: 170.2 PR N2: 161.6 TEMP: 1981
18:41:36 NSEQ: 413 PR INT: 290.5 PR N2: 113.4 TEMP: 1119
18:41:36 NSEQ: 414 PR INT: 107.8 PR N2: 155.7 TEMP: 1603
18:41:36 NSEQ: 415 PR INT: 206.1 PR N2: 141.6 TEMP: 1460
18:41:37 NSEQ: 416 PR INT: 212.7 PR N2: 179.9 TEMP: 1599
18:41:37 NSEQ: 417 PR INT: 141.7 PR N2: 138.8 TEMP: 1670
18:41:37 NSEQ: 418 PR INT: 164.9 PR N2: 292.5 TEMP: 1083
18:41:37 NSEQ: 419 PR INT: 279.8 PR N2: 236.6 TEMP: 1009
18:41:38 NSEQ: 420 PR INT: 235.5 PR N2: 200.5 TEMP: 1113
18:41:38 NSEQ: 421 PR INT: 102.8 PR N2: 172.9 TEMP: 1913
18:41:38 NSEQ: 422 PR INT: 283.3 PR N2: 161.8 TEMP: 1201
18:41:38 NSEQ: 423 PR INT: 299.2 PR N2: 125.2 TEMP: 1665
18:41:39 NSEQ: 424 PR INT: 118.4 PR N2: 268.1 TEMP: 1165
18:41:39 NSEQ: 425 PR INT: 182.3 PR N2: 147.2 TEMP: 1135
18:41:39 NSEQ: 426 PR INT: 262.4 PR N2: 188.8 TEMP: 1035
18:41:39 NSEQ: 427 PR INT: 213.1 PR N2: 122.1 TEMP: 1784
18:41:40 NSEQ: 428 PR INT: 244.6 PR N2: 100.1 TEMP: 1726
18:41:40 NSEQ: 429 PR INT: 101.1 PR N2: 162.1 TEMP: 1418
Encerrando thread 'DataViewProcess'.
Digite qualquer tecla para fechar a janela.
  
```

Figura 14: Consoles principal, exibição de dados e exibição de alarmes após evento ESC

5. Conclusão/Problemas Encontrados

❑ ETAPA 1

Nas seções acima, foram mostrados os usos das variáveis que indicam última posição livre e última posição ocupada na lista circular, freePositionList1 e ocupPositionList1, e na parte em que é feito seu incremento, eu encontrei um problema no qual não consegui identificar o motivo. Ao fazer o incremento dessas variáveis de 1 em 1, após 120 mensagens serem criadas e retiradas da lista, ocorria um erro de debug dizendo que um null pointer apontava para um block de função non-zero e parava de gerar e retirar mensagens, pesquisei na internet para tentar entender o porquê, mas apenas encontrei um programador

5. C

Nas seq
posiçã
ocupPo
proble
dessas
lista, oc
block d
interne
prograr
increm

com um problema similar, e a solução do mesmo foi fazer o incremento da variável pelo resto da divisão por 100, e implementando no meu código de forma que as linhas de comando ficaram 'freePositionList1 = (freePositionList1 + 1) % 100' e 'ocupPositionList1 = (ocupPositionList1 + 1) % 100', resolveu o problema, mas não fui capaz de entender o motivo do erro que estava ocorrendo e nem porque fazer o incremento de 0.01 em 0.01 corrige esse problema.

Outro problema que encontrei foi ao implementar os CheckForError's, que acabou por atrapalhar no encerramento correto do meu programa de início. Talvez eu tenha feito as comparações erradas nas chamadas da função, o que estava acusando erros de forma errada quando ocorre o evento de encerramento ESC, então resolvi deixar sem a checagem por erros para essa primeira etapa, e tentarei implementar corretamente até a entrega da etapa 2.

O último problema que encontrei foi no uso da função rand(), para geração de valores aleatórios na criação das mensagens. Por conta de uma condição natural da função, eram geradas duas mensagens "iguais" pelos CLP's 1 e 2, como mostrado na imagem abaixo.

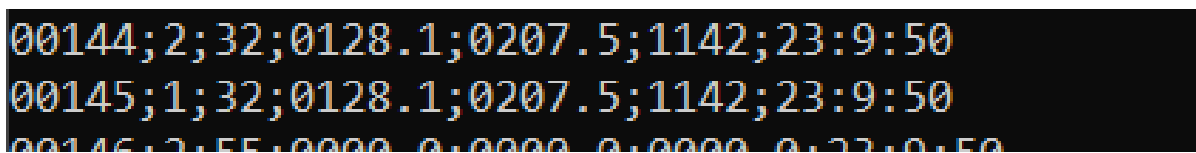


Figura 11: Erro da função rand()

A solução que encontrei foi criar uma forma de armazenar os valores antigos criados para geração da mensagem, que fiz por meio de uma struct "previousValue", e dentro das funções de formatação dos parâmetros da mensagem, uma comparação do valor aleatório gerado era feita com o valor anterior, de forma que caso esses fossem iguais, a função chama a si mesma e continua nessa recursão até que um valor novo, diferente do que foi gerado para a mensagem anterior, fosse criado.

Com exceção desses problemas citados, o programa executa toda lógica corretamente, exibindo todos alarmes e mensagens retiradas, e respondendo corretamente a todos eventos de teclas digitadas feitas pelo operador.

❑ ETAPA 2

Na segunda etapa foram corrigidos os erros encontrados na primeira, a checagem de erro não foi implementada em todas as funções por conta de

situações de erro estranhos que não consegui identificar, mas o funcionamento do programa se mantém normal.