UNIVERSIDADE FEDERAL DE MINAS GERAIS

Automação em tempo real

Trabalho prático - Etapa 2

Professor Luiz Luiz Themystokliz Sanctos Mendes

Estevão Coelho Kiel de Oliveira - 2016119416
Italo José Dias - 2017002121

30 de agosto de 2021 - 2021/1

Conteúdo

1 Introdução

O trabalho desenvolvido tem como contexto uma industria de papel e celulose. O objetivo do mesmo é desenvolver uma aplicação de software *multithread* responsável pela leitura de dados tanto de um Sistema Digital de Controle Distribuído (SDCD) quanto de um *Plant Information Management System* (PIMS) fictícios. Os mesmos serão apresentados em dois terminais de vídeo para os operadores da planta. O primeiro ira exibir os dados do processo de fabricação (TERMINAL A) e o segundo apresentara previsores de falhas operacionais geradas pelo PIMS (TERMINAL B).O mesmo foi dividido em duas entregas distintas, Etapa 1 e Etapa 2. Este relatório diz respeito a Etapa 2 do projeto.

2 Desenvolvimento

2.1 Visão global

A Figura 1 a seguir é uma representação visual da aplicação desenvolvida. Nela podemos ver as tarefas representadas por círculos, a lista circular em memória RAM representada pelos quadrados azuis numerados e suas posições livre e ocupado retratadas por setas das cores verde e vermelho respectivamente. Também pode-se observar os terminais ilustrados por monitores e a imagem de um teclado demostrando as entradas que um operador faria no sistema. Na Etapa 2 foram acrescentadas a memória em disco, que guarda os valores dos dados do processo escritos pela *Thread* Captura de dados do processo e que posteriormente são lidos pelo Processo Exibição de dados do processo, e a comunicação entre processos, representada pelas setas verdes. Já as setas em vermelho de retirada e deposito de mensagens representam um acesso direto à memória da nossa aplicação. As linhas em azul ligadas aos monitores, ao teclado e a memória em disco representam operações de entrada e saída em disco, vídeo ou teclado. Por fim, as linha pontilhada que liga as tarefas entre si diz respeito ao sincronismo entre os mesmo via eventos e via semáforos (veremos mais a frente que os dois objetos foram utilizados para isso).

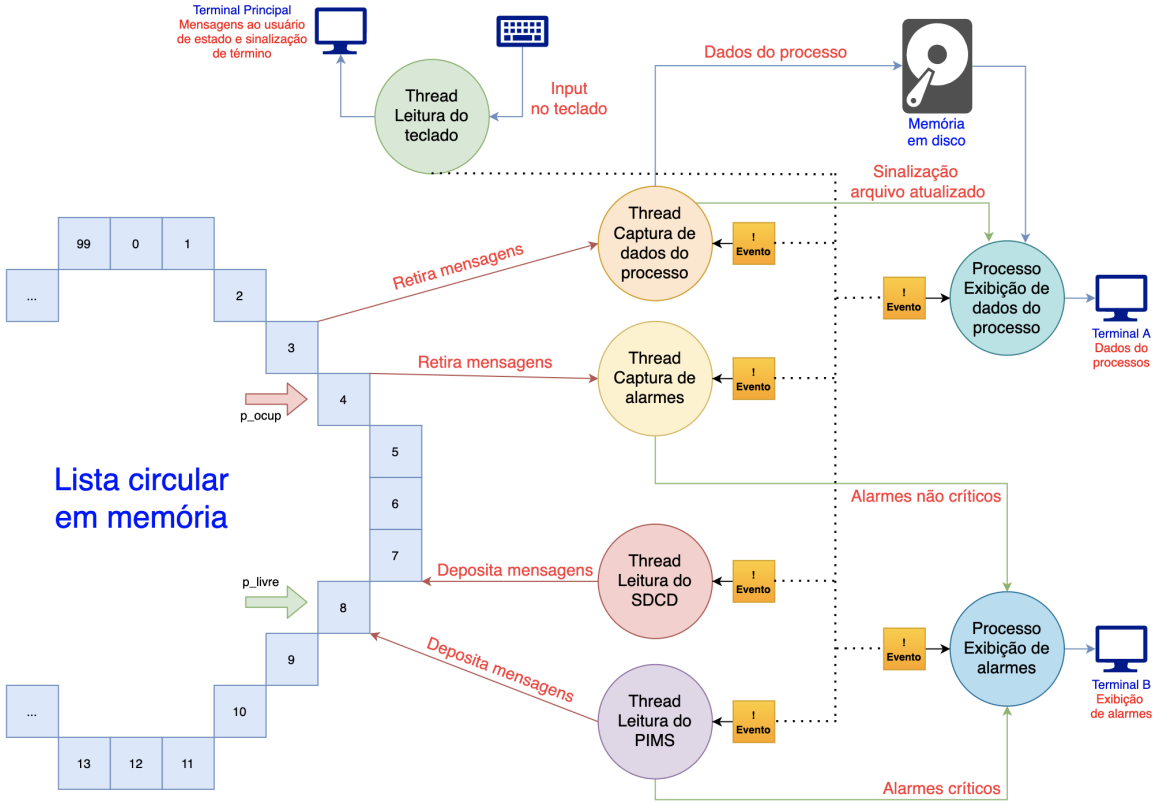

Figura 1: Representação visual da aplicação.

Nossa aplicação foi dividida em 7 diferentes tarefas, o funcionamento mais aprofundado das mesma será visto mais a frente. Um breve resumo das mesmas e de como elas interagem entre si pode ser visto logo a seguir.

- **Leitura do teclado:** Responsável por ler os comandos digitados pelo operador e sinalizar ações especificas em outras tarefas. Essas ações dizem respeito a bloqueio, desbloqueio e encerramento.
- **Leitura do SDCD:** Na teoria essa tarefa seria responsável por recolher dados, porém para simularmos esse efeito iremos gerar valores aleatórios com uma certa formatação determinada e com periodicidade de 500 em 500 milissegundos. A tarefa gera esses valores para mensagens do tipo 1 e os mesmos são salvos em uma lista circular na memória RAM do computador. Além disso, os dados do tipo SDCD estão armazenados em uma arquivo em disco, sendo possível realizar a comunicação entre diferentes processos através desses arquivo.
- **Leitura do PIMS:** É uma tarefa com funções semelhantes a de Leitura do SDCD, porém a mesma produz mensagens do tipo 2 (não críticas) e tipo 9 (críticas) e com um padrão diferente. Apensa as mensagens do tipo 2 serão gravadas na lista circular. Os alarmes críticos de segurança do tipo 9, assim que gerados, são diretamente enviados para o Processo de Exibição de alarmes, essa comunicação entre processos se dá via *mailslots*. Os dados do tipo dois são produzidos com intervalos que variam de 1 a 5 segundos, já os de tipo nove de 3 a 8 segundos.
- **Captura de dados do processo:** Responsável por recolher as mensagens armazenadas na memória pela tarefa de Leitura do SDCD, grava-las no arquivo circular em disco e sinalizar a atualização do arquivo para o Processo Exibição de dados do processo.
- **Captura de alarmes:** É a tarefa responsável apenas por recolher da memória circular os valores do tipo 2 produzidos pela tarefa de Leitura do PIMS e envia-las, via *mailslots*, para o Processo Exibição de alarmes.
- **Exibição de dados do processo:** Exibi no TERMINAL A sua mudança de estado (bloqueada ou desbloqueada), retira do arquivo circular em disco os dados do processo anteriormente gravados assim que houver a sinalização da atualização do arquivo e exibe-os no TERMINAL A com uma formatação específica.
- **Exibição de alarmes:** Exibi no TERMINAL B sua mudança de estado (bloqueada ou desbloqueada), lê do servidor de *mailslot* os alarmes enviados anteriormente e exibe-os no TERMINAL B com uma formatação específica.

A nossa aplicação multithread foi desenvolvido no Visual Studio Community da Microsoft nas linguagens C e C++. Ela consiste de uma solução com três projetos diferentes. Do ponto de vista do usuário apenas um arquivo é executado para o disparo da aplicação completa.

O projeto ExibicaoDados e ExibicaoAlarmes dizem respeito as tarefas Exibição de dados do processo e Exibição de alarmes respectivamente, eles foram pensados para serem processos e terem seus próprios terminais (TERMINAL A e TERMINAL B). O processo ExibicaoDados é bloqueado ou desbloqueado de acordo com seu estado anterior quando o operador digita a tecla 'o' no teclado, já para o processo ExibicaoAlarmes a tecla é a 'c'. Ambos os processos exibem suas mudanças de estado nos respectivos terminais dedicados e são finalizados independente de seus estados quando o operador digita a tecla 'Esc'. Enquanto o ExibicaoDados mostra no TERMINAL A todas as mensagens de dados do processo o ExibicaoAlarmes mostra no TERMINAL B todos os alarmes gerados.

O último projeto, com nome TrabalhoPratico, é composto de um processo com cinco *threads*, uma primária e quatro secundárias. Ele possuiu terminal próprio que é chamado de TERMINAL PRINCIPAL, nele são exibidos os estados das *threads*, os avisos de memória cheia e de inicialização e finalização.

A *thread* primária (Leitura do teclado) é responsável pela criação dos objetos, lista em disco, *threads* secundárias, processos filhos e pela tarefa de Leitura do teclado.

As *threads* secundárias produtoras Leitura do SDCD e Leitura do PIMS são responsáveis por gerarem mensagens aleatórias com padrões e tipos definidos. Os dados do processo produzidos pelo SDCD e os alarmes não críticos produzidos pelo PIMS são gravados na lista circular que serve como um *buffer* para nossa aplicação. Os alarmes críticos gerados são enviados diretamente para o Processo de Exibição de alarmes via *mailslots*.

Já as *threads* secundárias consumidoras Captura de dados do processo e Captura de alarmes recolhem as mensagens depositadas na memória circular pelas *threads* Leitura do SDCD e Leitura do PIMS respectivamente.

Para acesso da lista circular e controle das posições disponíveis dois objetos do *kernel* foram utilizados para sincronismo e proteção da seção crítica. Foi-se feito uso de *mutex* e semáforos para esse tipo de controle. Para a sinalização de inputs do teclado utilizamos objetos do tipo evento, todos os objetos fazem parte da API-Win32. Já para a memória em disco foi utilizado um objeto de sincronização do tipo semáforo, que sinaliza a inserção de novos dados e utilização desses mesmos dados. Também para a memória em disco foi utilizado um objeto do tipo evento, que sinaliza quando o arquivo está cheio.

Caso a lista circular, que age como um *buffer* para a aplicação, estiver cheia no momento de gravação de uma mensagem pela *threads* produtoras as mesmas se bloqueiam e avisam o ocorrido no TERMINAL PRINCIPAL. A mesma se desbloqueia quando houver uma posição livre disponível novamente.

Caso a memória em disco em disco fique cheia, um evento será sinalizado e a escrita ao arquivo será bloqueado e um aviso em texto será exibido no terminal principal.

A nossa aplicação faz uso da biblioteca *Pthreads-Win32* para a criação das *threads* secundárias e também faz uso da biblioteca *CheckForError* dos autores Constantino Seixas Filho e Marcelo *Szuster* que auxilia na identificação de problemas que possam ocorrer durante a execução do programa.

A seguir temos uma explicação mais detalhada do funcionamento das tarefas.

2.2 Tarefa de leitura do teclado

É a thread primária do projeto TrabalhoPratico. Após a criação dos objetos, lista em disco, das threads secundárias e dos processos filhos a mesma entra em loop e espera que o operador digite uma tecla. Cada um dos inputs válidos sinaliza um evento com reset automático, exceto pela tecla 'Esc' que dispara um evento de reset manual, por meio da função SetEvent(). Podemos ver as teclas e suas funções na Tabela 1.

Tecla	Função
s	Notificar a tarefa de Leitura do SDCD que a mesma deve bloquear-se ou desbloquear-se de acordo com o estado anterior.
p	Notificar a tarefa de Leitura do PIMS que a mesma deve bloquear-se ou desbloquear-se de acordo com o estado anterior.
d	Notificar a tarefa de Captura de Dados do Processo que a mesma deve bloquear-se ou desbloquear-se de acordo com o estado anterior.
a	Notificar a tarefa de Captura de Alarmes que a mesma deve bloquear-se ou desbloquear-se de acordo com o estado anterior.
o	Notificar a de Dados do Processo que a mesma deve bloquear-se ou desbloquear-se de acordo com o estado anterior.
c	Notificar a de Alarmes que a mesma deve bloquear-se ou desbloquear-se de acordo com o estado anterior.
Esc	Notifica todas as tarefas que as mesmas devem se encerrar inclusive a de Leitura do teclado.

Tabela 1: Teclas e suas funções na aplicação.

Todas as tarefas são finalizadas ao se apertar a tecla 'Esc' independentemente dos seus estados. Quando as mesmas entram em modo de finalização todos os *HANDLES* abertos são fechados e as tarefas são finalizadas.

2.3 Tarefa de leitura do SDCD

Diz respeito a uma *thread* secundária do tipo produtora, ela está inserida no projeto TrabalhoPratico. A mesma começa seu loop de execução gerando mensagens aleatórias com os seguintes campos separados por '|'.

NSEQ: É um número inteiro sequencial de 1 até 999999 com um tamanho igual a 6.
Exemplo: "000001"

TIPO: É um número inteiro que sempre vale 1 de tamanho igual a 1.
Exemplo: "1"

TAG: É uma string de um indicador alfanumérico da variável de processo, ele é separado por '-' na posição 3 e 6 e tem tamanho igual a 10.
Exemplo: "A4T-04-B01".

VALOR: É um valor real de uma variável de processo de tamanho 8 (contando com o ponto).
Exemplo: "87654.31".

UE: É composto por uma string de tamanho 8 com uma unidade de engenharia selecionada de um banco de dados.
Exemplo: "kg ".

MODO: É composto por um caracter de tamanho 1 e representa o modo de operação associado (A)utomático ou (M)anual.
Exemplo: "A".

TIMESTAMP: Campo relacionado a hora corrente com precisão de milissegundos, tamanho igual a 12.
Exemplo: "11:30:53.777".

Exemplo completo de mensagem aleatória gerada:
"000001|1|A4T-04-B01|87654.31|kg |A|11:30:53.777"

Após a mensagem ter sido gerada a *thread* aguarda por um semáforo contador chamado *hSemLivre* ou por um evento chamado *hEventKeyEsc* por meio da função *WaitForMultipleObjects()*. Caso exista uma posição para escrita disponível na memória a tarefa avança pois o contador do sinal é positivo e está em estado sinalizado, caso a tecla 'Esc' seja pressionada pelo operador a tarefa entra em modo de finalização.

É importante ressaltar que todas as operações realizadas nessa etapa de semáforos e *mutex* serão realizadas em conjunto com o evento *hEventKeyEsc* por meio da função *WaitForMultipleObjects()* que pode der sinalizada com apenas a ocorrência de um dos dois eventos. Por esse motivo a parte que diz respeito a explicação desse comportamento será suprimida daqui em diante. Essa operação casada tem o objetivo de, mesmo que a tarefa esteja em um estado bloqueado, finalizar as *threads* quando o operador digitar a tecla 'Esc' no teclado.

Após passar pelo semáforo a mesma espera para a conquista de um *mutex* chamado *hMutexBuffer*. Esse objeto tem como função proteger a seção crítica da nossa aplicação. A *thread* que conquistar o *mutex* acessa a seção crítica de maneira exclusiva, evitando assim conflitos quando existe o chaveamento de tarefas realizado pelo SO. No caso das *threads* produtoras os valores da mensagem gerados são armazenados na lista circular.

A seção crítica da aplicação diz respeito a operações realizadas na lista circular em memória RAM, nos seus apontadores p_livre e p_ocup e impressão das mensagens na console.

Por fim, a tarefa libera os objetos mutex e semáforo e espera determinado tempo antes de começar o loop de execução novamente.

A temporização da tarefa foi feita como a utilização da função fornecida pela API-win32 *WaitForSingleObject(handle, tempo_em_ms)* que aguarda por determinado tempo a sinalização de um objeto do tipo *handle*, que nesse caso nunca será sinalizado (hTimeOut). A resolução desse método de temporização é de $15ms$ o que, para a periodicidade requerida é de $500ms$, irá gerar uma perda da precisão na temporização em torno de 3%. Como a nossa aplicação é do tipo *soft real-time* podemos considerar esse valor um ótimo resultado acompanhado de uma boa eficiência.

2.4 Tarefa de leitura do PIMS

A tarefa de Leitura do PIMS também é uma thread secundária produtora e tem as mesmas características e modo de funcionamento da tarefa de Leitura do SDCD, porém o padrão da mensagem produzida é diferente.

Ela produz dois tipos de alarmes os não críticos (tipo 2) e os críticos (tipo 9). Apenas as mensagens do tipo 2 são gravadas na lista circular pois os alarmes do tipo 9 são diretamente enviadas via mailslots para o Processo de Exibição de alarmes.

Sua temporização é realizada com o mesmo método utilizado na Tarefa de leitura do SCDC. Entretanto, a periodicidade dos alarmes não críticos e críticos é, respectivamente, de 1 a 5 segundos e 3 a 8 segundos. Durante a geração dos alarmes o tipo que estiver mais próximo de vencer o seu tempo máximo de repetição é produzido e um tempo aleatório de delay é acrescentado ao mesmo.

As mensagens aleatórias são compostas pelos seguintes campos que são separados por '|'.

NSEQ: É um número inteiro sequencial de 1 até 999999 com um tamanho igual a 6.
Exemplo: "000001"

TIPO: É um número inteiro que pode valer 2 (Alarmes não-críticos) ou 9 (Alarmes críticos) de tamanho igual a 1.
Exemplo: "2"

ID ALARME: É um número inteiro de tamanho 4 que vai de 1 até 9999 e representa o identificador da condição anormal.
Exemplo: "0717".

GRAU: É um número inteiro de tamanho 2 que vai de 1 até 99 e representa o grau de impacto da condição anormal.
Exemplo: "04".

PREV: É um número inteiro de tamanho 5 que vai de 1 até 14440 e representa o número de minutos previstos até a ocorrência da condição.
Exemplo: "00120".

TIMESTAMP: Campo relacionado a hora corrente com precisão de segundos, tamanho igual a 8.
Exemplo: "11:30:53".

Exemplo completo de mensagem aleatória gerada:
"000001|2|0717|04|00120|11:30:53"

2.5 Tarefa de captura de dados de processo

Diz respeito a uma *thread* secundária do tipo consumidora, que também está inserida no projeto Trabalho-Pratico. Ela é responsável por retirar as mensagens produzidas pela tarefa de Leitura do SDCD e grava-las no arquivo em disco.

A *thread* começa sua execução aguardando por um semáforo contador chamado *hSemOcupado*. Caso exista uma posição ocupada disponível na memória a tarefa avança pois o contador do sinal é positivo e está em estado sinalizado.

Após passar pelo semáforo a mesma espera para a conquista de um *mutex* chamado *hMutexBuffer*. No caso das *threads* consumidoras os valores da mensagem são selecionados de acordo com o tipo, são lidas da lista circular em memória RAM e depois gravadas em arquivo.

Os dados lidos são gravados no arquivo compartilhado em memória. O arquivo é protegido pelas funções de *lock* e *unlock* e sinalizado por um semáforo e por um evento, o evento agaurda a sinalização, informando que o arquivo não está mais cheio de dados. Além disso, a tarefa de captura de dados do processo possui dois objetos de sincronização do tipo semáforo, baseados na logica de implementação do algoritmo de produtores e consumidores, onde os produtos depositam os dados em um *buffer* e os consumidores aguardam a existência dos

dados, aguardo sinalização por meio do semáforo. Por fim, o *buffer* é protegido pelo por um *mutex*, garantindo exclusão mutua na manipulação de seus dados.

Por fim, a tarefa libera os objetos *mutex* e semáforo.

2.6 Tarefa de captura de alarmes

É uma *thread* secundária do tipo consumidora que possui as mesmas características e modo de funcionamento da tarefa de Captura de dados do processo. Ela também se encontra inserida no projeto TrabalhoPratico. A mesma é responsável por retirar da lista circular as mensagens gravadas de alarmes não-críticos produzidas pela tarefa de Leitura do PIMS e então envia-las via *mailslot* para a Tarefa de Exibição de alarmes.

2.7 Tarefa de exibição de dados do processo

Como já foi dito, a tarefa de Exibição de dados do processo é o processo do projeto ExibicaoDados e tem seu próprio console, o TERMINAL A. Ele exibe as mudanças de estados bloqueado ou desbloqueado de acordo com seu valor anterior quando o operador digita a tecla 'o' no teclado. O mesmo é finalizado independente do seu estado atual quando o operador digita a tecla 'Esc'.

A principal função da tarefa de exibição de dados do processo é a de recolher e exibir os dados recolhidos do arquivo em disco no TERMINAL A. O processo responsável pela geração do terminal também faz a leitura do arquivo. Dentro do processo existe funções de *lockfile* e *unlockfile* para proteção do arquivo, além de sinalizar mostrando que o arquivo não se encontra mais cheio, podendo ser usado para escrever novamente, possui, também, um semáforo contador que protege a manipulação dos dados.

2.8 Tarefa de exibição de alarmes

O mesmo é valido para a tarefa de Exibição de alarmes. Ele é o processo do projeto ExibicaoAlarmes e tem seu próprio console o TERMINAL B. Ele exibe as mudanças de estados bloqueado ou desbloqueado de acordo com seu valor anterior quando o operador digita a tecla 'c' no teclado. O mesmo é finalizado independente do seu estado atual quando o operador digita a tecla 'Esc'.

A principal função da tarefa de exibição de alarmes é a de ler e exibir no TERMINAL B os alarmes passados a ela pelas outras tarefas via *mailslots*.

2.9 Arquivo em disco

O arquivo em disco foi feito através das funções da API Win32, usando as funções referentes ao tratamento de arquivo. Inicialmente o arquivo é criado na pasta do projeto, esse arquivo será acessado pela tarefa de captura de dados, que abre o mesmo arquivo. Como dois processos trabalham com o mesmo arquivo, é necessário proteger o documento, de forma que não seja possível realizar as ações de escrita/leitura quando o arquivo estiver em uso, em outras palavras, garantir a exclusão mutua. Para isso, foram utilizado os recurso de *lock* e unlock *file*. Primeiramente é realizado um *lock* do acesso ao arquivo e posteriormente um *unlock*.

Além disso, o arquivo possui tamanho máximo de dados que podem ser gravados, assim, é necessário contar o número de dados inseridos no texto e bloquear caso o número máximo seja atingido. Nesse sentido, pela própria natureza do objeto de sincronização, foi utilização um semáforo contador, que decresce sua contagem até o valor de 0, indicando que o arquivo está totalmente cheio. Por fim, para fins de praticidade, quando o arquivo se encontra cheio, um evento é disparado que impede que novos dados sejam inseridos.

Do ponto de vista de construção, muitos dados ruidosos, lixo, eram gerados e a identificação da causa não foi possível. Para correção, o tamanho dos dados de lixo foi diminuído do argumento de dados a serem lidos da função *WriteFile* de forma manual.

3 Como executar o programa

O arquivo trabalho_atr_estevao_e_italo.zip deve ser descompactado, e dentro da pasta a solução Trabalho-Pratico.sln deve ser aberta.

ATENÇÃO!!! Certifique-se de que o seu projeto esta como x86. Do lado de Debug na barra superior é possível alterar esse parâmetro.

Para que o programa compile e funcione corretamente é necessário seguir as instruções para a utilização da biblioteca pthread e checkforerror incluídas no início dos códigos TrabalhoPratico.cpp, ExibicaoDados.cpp e ExibicaoAlarmes.cpp.

Após isso deve-se compilar (Ctrl + Shift + B) e rodar (Ctrl + F5) o arquivo.

4 Resultados

O TERMINAL PRINCIPAL, TERMINAL A e TERMINAL B podem ser vistos em execução na Figura 2. O TERMINAL PRINCIPAL exibe os dados de inicialização e finalização da aplicação, também exibe as mudanças de estados das tarefas e avisos de memória cheia, tanto para a lista circular, quanto para o arquivo em disco. O TERMINAL A exibe os dados do processo lidos diretamente da memória em disco e as mudanças de estado da tarefa de exibição de dados do processo. O TERMINAL B exibe respectivamente os alarmes não críticos (em branco) e críticos (em vermelho) vindos da Thread Captura de alarmes e Thread Leitura do PIMS e as mudanças de estado da tarefa de exibição de alarmes. É possível observar, de forma simplista, o correto funcionamento da aplicação.

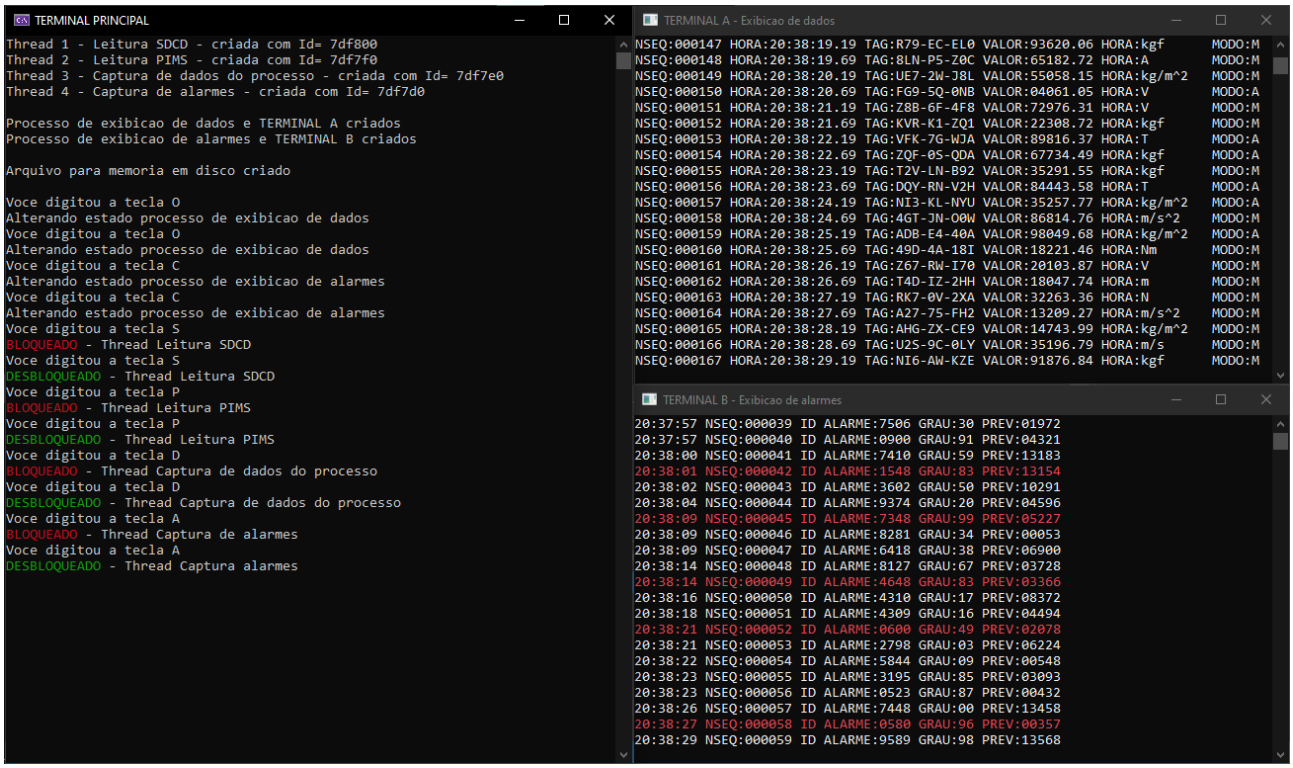

Figura 2: Resultados obtidos

5 Conclusão

Todas as especificações do projeto foram atendidas e estão descritas ao longo do trabalho. A tarefa consumidora Captura dos dados do processo lê os dados produzidos e gravados pela tarefa produtora Leitura do SDCD na lista circular em memória e as grava na memória em disco. Esses dados em disco são lidos e exibidos pela tarefa de exibição de dados do processo. A tarefa produtora Leitura do PIMS produz e repassa alarmes críticos e não críticos para a tarefa de exibição de alarmes, via comunicação entre processos (Mailslot), e lista circular respectivamente. A tarefa consumidora Captura de alarmes retira os alarmes não críticos gravados na memória e os repassa para a tarefa de exibição de alarmes que exibe tanto os alarmes críticos quanto os não críticos.