**BLOCO DE CONSTRUÇÃO 4**

**FUNÇÃO DE ENTRADA DO THREAD URGENTE**

**"void Urgent\_thread\_entry(ULONG thread\_input)” :** Cria uma função chamada “Ugernt\_thread\_entry” que recebe a variável “thread\_input” do tipo “ULONG” (unsigned integer) como parâmetro. Essa função será responsável pela entrada de uma thread urgente.

**“while (1)”:** Cria um loop while dentro da função criada anteriormente. Esse loop recebe como parâmetro “ (1) ” que representa uma condição verdadeira, ou seja, o 1 é utilizado para criar um loop infinito que só parará caso haja o uso de um “break”.

**“tx\_thread\_sleep(4);”:** Nesse trecho a thread é colocada em espera por quatro unidades de tempo, simulando o processamento.

**“tx\_mutex\_get(&my\_mutex, TX\_WAIT\_FOREVER);”:** Nessa linha de código a thread tenta receber a mutex “&my\_mutex” e caso ela não consiga ela entra em espera por limite de tempo indeterminado até que ela consiga adquirir essa mutex.

**“tx\_thread\_sleep(3);”:** Após obter o mutex, a thread entrará em espera por mais 3 unidades de tempo, simulando o processamento.

**“tx\_mutex\_put(&my\_mutex);”:** Por fim, a thread libera a mutex permitindo que outras threads acessem ela para usar o recurso.

O código define uma função de entrada para uma thread de alta prioridade que executa um loop infinito. Dentro do loop, a thread introduz atrasos, adquire um mutex, executa algum processamento dentro da seção crítica protegida pelo mutex e, em seguida, libera o mutex. Esse padrão se repete indefinidamente.

**FUNÇÃO DE ENTRADA DE THREAD IMPORTANTE**

**“void Important\_thread\_entry(ULONG thread\_input)”:** Função que não retorna nenhum valor que recebe a entrada de uma thread de nível importante (médio) de prioridade.

**“ULONG kount;”:** É criada uma variável do tipo unsigned integer que servirá como contador.

**“char  letter;”:** Cria uma variável do tipo caractere que receberá uma letra como valor.

**“while (1)”:** Inicia um loop infinito.

**“tx\_thread\_sleep(3);”:** Simula um processamento com o tempo de espera sendo de 3 unidades.

**“for (kount = 1; kount < 100000000; kount++) letter = 'A';”:** Essa linha de código representa um loop “for” que se repetirá até que o contador atinja o valor 100000000, e a cada ciclo o valor “A” será atribuído para a variável “letter”. Essa linha simula uma tarefa sendo realizada pela thread importante (média prioridade).

Basicamente, o código define uma função de entrada para uma thread que executa um loop infinito. Dentro do loop, a thread faz uma pausa de 3 unidade de tempo do sistema e, em seguida, executa um loop intensivo de CPU, simulando um trabalho intenso. Isso é feito para simular uma tarefa sendo realizada por uma thread de prioridade média. A thread importante acessa o mesmo recurso que a de rotina e a de urgência, causando assim uma inversão de prioridade pois ela acessara o recurso enquanto a de urgência terá que aguardar. Isso ocorre pois ela não está em seção crítica, diferente das threads de prioridade baixa e alta, que estão.

**FUNÇÃO DE ENTRADA DE THREAD DE ROTINA**

**“void Routine\_thread\_entry(ULONG thread\_input)”:** Cria uma função que não retorna nenhum valor e recebe a entrada de uma thread de rotina (baixa prioridade).

**“CHAR letter;”:** Cria uma variável do tipo caractere que receberá uma letra como valor.

**“ULONG kount;”:** É criada uma variável do tipo unsigned integer que servirá como contador.

**“while (1)”:** Cria um loop while dentro da função criada anteriormente. Ele recebe como parâmetro “ (1) ” criando um loop infinito.

**“tx\_mutex\_get(&my\_mutex, TX\_WAIT\_FOREVER);”:** Nessa linha de código a thread tenta receber a mutex “&my\_mutex” e caso ela não consiga ela entra em espera por limite de tempo indeterminado até que ela consiga adquirir essa mutex.

**“for (kount = 1; kount < 100000000; kount++) letter = 'A';”:** Essa parte simula uma tarefa sendo realizada pela thread de rotina, no caso a mesma tarefa que já foi explicada anteriormente na seção da thread importante.

**“tx\_mutex\_put(&my\_mutex);”:** Essa parte representa a liberação da mutex para que outra thread possa adquiri-la.

**“tx\_thread\_sleep(1);”:** Simula um processamento com uma unidade de tempo.

Resumindo, o código define uma função de entrada para uma thread com a menor prioridade. Dentro do loop infinito, a thread adquire um mutex, executa um trabalho intensivo de CPU dentro da seção crítica protegida pelo mutex, libera o mutex e faz uma pequena pausa antes de repetir o ciclo. Isso cria uma situação em que a thread de rotina compete com outras threads (como a thread "Urgent") pelo acesso ao mutex.

**FUNÇÃO DE TEMPORIZADOR DE APLICATIVO PRINT\_UPDATE**

**“void print\_update(ULONG invalue)”:** Cria uma função que não retorna nada e recebe como entrada a variável “invalue” que é do tipo unsigned integer. Essa função será usada para imprimir informações sobre o experimento.

**“current\_time = tx\_time\_get();”:** Cria uma variável que recebe o tempo do sistema.

**“printf("Priority Inversion Experiment--Time Remaining:  %lu timer ticks ...\n", DISPLAY\_INTERVAL - 1 - current\_time);”:** Essa linha imprimi o intervalo de tempo até o próximo intervalo de tempo para a exibição do experimento. Esse intervalo é calculado pela subtração do DISPLAY\_INTERVAL (constante com o valor de 5001) – 1 - currrent\_time (tempo do sistema).

Portanto, a função print\_update é usada para imprimir uma mensagem que informa sobre o intervalo de tempo do experimento.

**FUNÇÃO DE TEMPORIZADOR DE APLICATIVO PRINT\_STATS**

**“void print\_stats(ULONG invalue)”:** cria a função que será responsável por exibir status relacionado ao experimento de inversão de prioridade.

**“tx\_thread\_performance\_info\_get(&Routine\_thread, &resumptions\_Routine, &suspensions\_Routine, &solicited\_preemptions\_Routine, TX\_NULL, TX\_NULL, X\_NULL, TX\_NULL, TX\_NULL, TX\_NULL, TX\_NULL);”:** Neste trecho, são coletadas informações de desempenho sobre a thread chamada Routine\_thread e essas informações são armazenadas nas variáveis resumptions\_Routine, suspensions\_Routine e solicited\_preemptions\_Routine. Isso inclui o número de vezes que a thread foi retomada (resumptions), o número de vezes que foi suspensa (suspensions) e o número de preempções solicitadas pela thread (solicited preemptions). Essas informações de desempenho são obtidas por meio da função tx\_thread\_performance\_info\_get.

**“tx\_thread\_performance\_info\_get(&Urgent\_thread, &resumptions\_Urgent, &suspensions\_Urgent, TX\_NULL, TX\_NULL, TX\_NULL, TX\_NULL, TX\_NULL, TX\_NULL, TX\_NULL, TX\_NULL);”:** Nessa linha de código, são coletadas informações de desempenho sobre a thread chamada Urgent\_thread e essas informações são armazenadas nas variáveis resumptions\_Urgent e suspensions\_Urgent que serão usadas para exibir essas informações posteriormente.

**“tx\_mutex\_performance\_info\_get(&my\_mutex, &mutex\_puts, &mutex\_gets, TX\_NULL, TX\_NULL, TX\_NULL, TX\_NULL);”:** Aqui, são coletadas informações de desempenho sobre o mutex chamado my\_mutex, e essas informações são armazenadas nas variáveis mutex\_puts e mutex\_gets. Isso inclui o número de vezes que o mutex foi colocado (mutex puts) e o número de vezes que foi obtido (mutex gets).

**“printf("\nProjectPriorityInversion: 3 threads, 1 byte pool, 1 mutex, and 2 timers\n\n");”:** Imprime informações sobre o experimento como o número de threads, número de byte pool, mutex e temporizadores.

**“printf("   Routine thread resumptions: %lu\n", resumptions\_Routine);”:** Exibe o número de vezes que a thread de rotina foi retomada.

**“printf("   Routine thread suspensions: %lu\n", suspensions\_Routine);”:** Exibe o número de vezes que a thread de rotina foi suspensa.

**“printf("Routine solicited\_preemptions: %lu\n\n", solicited\_preemptions\_Routine);”:** imprime o número de preempções solicitadas pelas threads.

**“printf("    Urgent thread resumptions: %lu\n", resumptions\_Urgent);”:** Exibe o número de vezes que a thread de urgência foi retomada.

**“printf("    Urgent thread suspensions: %lu\n\n", suspensions\_Urgent);”:** Exibe o número de vezes que a thread de urgência foi suspensa.

**“printf("                   mutex puts: %lu\n", mutex\_puts);”:** Imprime o número de vezes a mutex foi colocada.

**“printf("                   mutex gets: %lu\n\n", mutex\_gets**);”: Exibi o número de vezes que a mutex foi obtida.

**“tx\_timer\_deactivate(&update\_timer);”:** Por fim, o timer de update criado no bloco de construção três é desativado.

A função print\_stats é usada para imprimir estatísticas de desempenho relacionadas a threads e mutexes, bem como informações sobre o contexto do sistema.