Universidade Federal do Rio Grande do NorteCentro de Tecnologia
Departamento de Engenharia de Computação e Automação

DCA0107 – Sistemas Operacionais - Sincronização

GABRIEL SOUTO LOZANO BARBOSA - 20220080955

NATAL/RN – 2023

1. DESCRIÇÃO DE LINGUAGEM E BIBLIOTECAS

Na realização dessa atividade foi utilizado a linguagem C e as bibliotecas ‘windows.h’, para que o código possa ser executado em sistema operacional Windows, e a ‘pthreads.h’ para criação e gereciamento de threads, com isso é possível que as threads acessem e modifiquem uma variável compartilhada de forma segura, garantindo exclusão mútua e a sincronização por meio de mutex e variáveis de condição.

2. EXPLICAÇÃO DO PROGRAMA

Utilizando dos pseudo-códigos enviados pela atividade, tanto da parte da “Introdução” e da “1 Variável de condição a partir de um *mutex”*, com isso temos duas ‘struct’, a da *mutex* e a da *condvar*.

Figura 1: Estrutura inicial do código.

```
struct mutex {
    CRITICAL_SECTION lock;
};

struct condvar {
    CONDITION_VARIABLE condition;
};

void mutex_init(struct mutex *m) {
    InitializeCriticalSection(&m->lock);
}

void mutex_lock(struct mutex *m) {
    EnterCriticalSection(&m->lock);
}

void mutex_unlock(struct mutex *m) {
    LeaveCriticalSection(&m->lock);
}

void condvar_init(struct condvar *c) {
    InitializeConditionVariable(&c->condition);
}

void condvar_wait(struct condvar *c, struct mutex *m) {
    SleepConditionVariableCS(&c->condition, &m->lock, INFINITE);
}

void condvar_signal(struct condvar *c) {
    WakeConditionVariable(&c->condition);
}

void condvar_broadcast(struct condvar *c) {
    WakeAllConditionVariable(&c->condition);
}
```

Fonte: Autor.

Na “*struct mutex”* utiliza-se o “*CRITICAL_SECTION lock”* para criar o *mutex*, sendo parte da API de sincronização de threads do Windows e

responsável por controlar o acesso a recursos compartilhados. Já, em relação, a “*struct condvar”* utiliza-se o “*CONDITION_VARIABLE condition”* que representa uma variável de condição do Windows, a qual é utilizada para sincronização entre as threads, permitindo que elas esperem ou prossigam, baseado no que está acontecendo no momento.

Sobre as funções, dividimos em dois: as funções referentes ao *mutex* e referente ao *condvar*.

Iniciando pelo *mutex*, existe a função *mutex_init,* a qual inicializa o *“CRITICAL_SECTION”,* a função *mutex_lock*, a qual é responsável por bloquear o mutex garantindo que apenas uma thread acesse a seção crítica, e, por fim, temos a função *mutex_unlock*, a qual desbloqueia o mutex após a thread que anteriormente a ocupava sair.

Já pela *condvar*, existe a função *condvar_init,* responsável por iniciar a variável de condição, a função *condvar_wait,* responsável por aguardar a sinalização para continuar, ou seja, a thread vai liberar o mutex e aguarda na variável de condição, e quando for sinalizada a thread acorda e adquire o mutex novamente, a função *condvar_signal,* responsável por sinalizar uma thread em espera na variável de condição para continuar, caso tenha várias threads esperando, apenas uma delas será sinalizada, e, por fim, tem a função *condvar_broadcast,* a qual sinaliza todas as threads em espera na variável de condição para continuar.

Figura 2: Função “thread_function”

```c
struct shared_data {
    struct mutex mutex;
    struct condvar cond;
    int shared_variable;
};

void *thread_function(void *arg) {
    struct shared_data *data = (struct shared_data *)arg;

    for (int i = 0; i < 10; ++i) {
        mutex_lock(&data->mutex);
        data->shared_variable++; // Incrementa a variável compartilhada
        printf("Thread %ld - Valor compartilhado: %d\n", GetCurrentThreadId(), data->shared_variable);
        mutex_unlock(&data->mutex);

        // Espera por um curto período de tempo antes de incrementar novamente
        Sleep(100);
    }
    return NULL;
}
```

Fonte: Autor

A função ‘thread_function’ representa o comportamento que cada thread terá, incrementando uma variável compartilhada de forma segura utilizando um mutex para garantir a exclusão mútua, evitando possíveis condições de corrida, garantindo a ordem de execução.

Figura 3: Parte final do código.

```
int main() {
    pthread_t threads[MAX_THREADS];
    struct shared_data data;

    mutex_init(&data.mutex);
    condvar_init(&data.cond);
    data.shared_variable = 0;

    for (int i = 0; i < MAX_THREADS; ++i) {
        pthread_create(&threads[i], NULL, thread_function, &data);
    }

    for (int i = 0; i < MAX_THREADS; ++i) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Fonte: Autor.

Na ‘main’ ocorre a configuração e execução de múltiplas threads para modificar uma variável compartilhada de forma segura, como descrito anteriormente, garantindo a exclusão mútua por meio de um mutex e variável de condição.

O resultado desse código é o seguinte:

Figura 4: Resultado do código.

```
PS C:\Users\Usuario>  & 'c:\Users\Usuario\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-cx3305b3.bhh' '--stdout=Microsoft-MIEngine-Out-bs5caup0.p5x' '--stderr=Microsoft-MIEngine-Error-a0cq4vz4.1ez' '--pid=Microsoft-MIEngine-Pid-s0cqjkp5.2zr' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Thread 2756 - Valor compartilhado: 1
Thread 5812 - Valor compartilhado: 2
Thread 17640 - Valor compartilhado: 3
Thread 1840 - Valor compartilhado: 4
Thread 9468 - Valor compartilhado: 5
Thread 2756 - Valor compartilhado: 6
Thread 9468 - Valor compartilhado: 7
Thread 5812 - Valor compartilhado: 8
Thread 1840 - Valor compartilhado: 9
Thread 17640 - Valor compartilhado: 10
Thread 17640 - Valor compartilhado: 11
Thread 1840 - Valor compartilhado: 12
Thread 5812 - Valor compartilhado: 13
Thread 2756 - Valor compartilhado: 14
Thread 9468 - Valor compartilhado: 15
Thread 17640 - Valor compartilhado: 16
```

Fonte: Autor.

3. PROBLEMAS ENFRENTADOS

Os problemas enfrentados para realizar essa atividade se inicia pelo fato de estar pagando esse semestre de forma domiciliar, portanto, não tenho acesso as explicações do professor, o já se cria uma grande dificuldade para entender o assunto de forma individual de forma mais ágil.

Por ser a primeira vez que tenho que trabalhar com threads, ainda tenho certas dificuldades em implementar, mesmo já tendo trabalhado para realizar a

atividade de Threads, dificuldade essa que abrange o trabalho com mutex para essa atividade. Consequentemente, quando surge algum tipo de erro no código demoro mais entender o que pode estar dando errado e como posso corrigir o erro, juntando todas as dificuldades supracitadas, que atrasaram a resolução da atividade, acredito que consegui entregar o que foi solicitado de forma que eu tenha conseguido aprender mais sobre o uso das threads e aumentando sua complexidade com o uso de mutex.

4. GITHUB

https://github.com/KempesBarbosa/Sistemas-Operacionais/tree/main/sinc