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 da atividade foram utilizados 5 bibliotecas para o correto funcionamento do código:

- stdio.h: biblioteca padrão de entrada e saída da linguagem C.
- stdlib.h: biblioteca que contém funções envolvendo alocação de memória, entre outras utilitarias.
- pthread.h: biblioteca que fornece suporte para criação e manipulação de threads.
- stdatomic.h: ela é usada para garantir que operações de leitura e escrita em variáveis compartilhadas ocorram de maneira atômica, sem interferência de outras threads.
- stdint.h: define tipos inteiros de tamanho fixo. É utilizada no código para garantir um tamanho específico para a variável utilizada no semáforo

2. EXPLICAÇÃO DO PROGRAMA

Figura 1: Mutex

```
7. // Estrutura do Mutex
8. struct mutex {
9.     atomic_uint v;
10.};
11.
12.// Funções Mutex
13.void mutex_init(struct mutex *m)
14.{
15.    m->v = 0;
16.}
17.
18.void mutex_travar(struct mutex *m)
19.{
20.    for (;;) {
21.        uint32_t v = 0;
22.        if (atomic_compare_exchange_strong(&m->v, &v, 1)) {
23.            break;
24.        }
25.    }
26.}
27.
28.void mutex_destravar(struct mutex *m)
29.{
30.    atomic_store(&m->v, 0);
31.}
```

Fonte: Autor.

Nessa parte do código tem definida a estrutura do ‘mutex’, que contém a variável atômica ‘v’ que representará o estado do mutex. Mostra também a inicialização, que define a variável atômica como 0, temos a função de travar o

mutex, utilizando de um loop para realizar a troca atômica, caso esse valor seja 0 a função sai do loop, e uma para destravar o mutex, que define a variável como 0.

Figura 2: Estrutura semaforo.

```
struct sem {
    pthread_mutex_t mutex;
    pthread_cond_t condition;
    unsigned valor;
};
```

Fonte: Autor

Define uma estrutura chamada sem que contém um mutex (pthread_mutex_t), uma variável de condição (pthread_cond_t), e uma variável valor para representar o valor do semáforo, após ele sair da estado crítico.

Figura 3: Funções semaforo

```
void sem_init(struct sem *s, unsigned val) {
    pthread_mutex_init(&(s->mutex), NULL);
    pthread_cond_init(&(s->condition), NULL);
    s->valor = val;
}

void sem_inc(struct sem *s) {
    mutex_travar((struct mutex *)s); // Utiliza um mutex para garantir
atomicidade
    s->valor++;
    pthread_cond_signal(&(s->condition)); // Acorda uma thread que esteja
esperando
    mutex_destravar((struct mutex *)s);
}

void sem_dec(struct sem *s) {
    mutex_travar((struct mutex *)s); // Utiliza um mutex para garantir
atomicidade
    while (s->valor == 0) {
        // Aguarda até que o semáforo seja incrementado por outra thread
        pthread_cond_wait(&(s->condition), &(s->mutex));
    }
    s->valor--;
    mutex_destravar((struct mutex *)s);
}
```

Fonte: Autor

O sem_init inicializa a estrutura do semáforo. Usa as funções pthread_mutex_init e pthread_cond_init para inicializar o mutex e a variável de condição, respectivamente.

O sem_inc incrementa o valor do semáforo, usando funções definidas anteriormente (mutex_travar e mutex_destravar) para garantir atomicidade e o

sem_dec decrementa o valor do semáforo. Se o valor é zero, espera até que outro thread incremente o semáforo usando pthread_cond_wait.

Figura 4: Função da thread

```
void *thread_function(void *arg) {
    int thread_id = *((int *)arg);

    // Aguarda para entrar na seção crítica usando semáforo
    printf("Thread %d: Aguardando para entrar na seção crítica.\n",
thread_id);

    // Entra na seção crítica
    sem_dec(&axSemaforo);
    printf("Thread %d: Entrou na seção crítica.\n", thread_id);

    // Sai da seção crítica
    axSemaforo.valor++;
    printf("Thread %d: Saindo da seção crítica. \nValor atual do
semáforo: %u\n", thread_id, axSemaforo.valor);
    sem_inc(&axSemaforo);

    return NULL;
}
```

Fonte: Autor

Define a função que será executada por cada thread. Cada thread tenta entrar em uma seção crítica usando o semáforo, realiza uma operação simulada na seção crítica, e depois libera o semáforo, e ao sair será somado um valor (+1) representando a sua saída.

Figura 5: Main

```
int main() {
    int i, num_threads = 3;
    pthread_t threads[num_threads];
    int thread_ids[num_threads];

    // Inicializa o mutex e o semáforo
    mutex_init(&axMutex);
    sem_init(&axSemaforo, 1); // Inicia com valor 1 para permitir a
entrada na seção crítica

    // Cria threads
    for (i = 0; i < num_threads; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, thread_function, (void
*)&thread_ids[i]);
    }

    // Aguarda o término das threads
    for (i = 0; i < num_threads; i++) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Fonte: Autor

Na função principal, são inicializadas as instâncias do mutex e do semáforo, e em seguida, threads são criadas para executar a função thread_function. A função principal aguarda a conclusão dessas threads usando pthread_join.

Figura 6: Resposta do código

```
Thread 0: Aguardando para entrar na seção crítica.
Thread 0: Entrou na seção crítica.
Thread 0: Saindo da seção crítica.
Valor atual do semáforo: 1
Thread 2: Aguardando para entrar na seção crítica.
Thread 2: Entrou na seção crítica.
Thread 2: Saindo da seção crítica.
Valor atual do semáforo: 2
Thread 1: Aguardando para entrar na seção crítica.
Thread 1: Entrou na seção crítica.
Thread 1: Saindo da seção crítica.
Valor atual do semáforo: 3
```

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.

Após a reunião com o professor, decide realizar o exercício sobre semáforos por já ter uma base de comparação mais fácil localizada no sigaa, sendo assim, foi mais fácil e ágil, atingir uma resolução sem mudar a ideia inicial de fazer uma thread entrar e contar a sua saída, e enquanto essa ação é feita as outras threads ficam na fila aguardando para executar a mesma ação.

4. GITHUB

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