INF01151 – SISTEMAS OPERACIONAIS II N
ATIVIDADE DE PROGRAMAÇÃO: EXCLUSÃO MÚTUA IMPLEMENTADA EM SOFTWARE

Cauê Scotti Luciano Rocha - 00338597, Juliana Rodrigues de Vargas - 00337553

1. Objetivo

Implementar em software primitivas de exclusão mútua baseadas no Algoritmo de Lamport (Algoritmo da Padaria) e criar uma biblioteca que encapsule essas primitivas. Além disso, comparar o desempenho da implementação com as primitivas padrão da biblioteca POSIX (*pthread_mutex_lock* e *pthread_mutex_unlock)*.

2. Implementação

a. Ambiente

As principais configurações dos sistemas operacionais utilizados na atividade são mostradas na Tabela 1.

ambiente 1	Linux 5.15.0-124-generic #134-Ubuntu SMP Fri Sep 27 20:20:17 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
ambiente 2	Linux 5.15.153.1-microsoft-standard-WSL2 #1 SMP Fri Mar 29 23:14:13 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux

Tabela 1: Configurações de ambiente.

b. Biblioteca

O algoritmo de Lamport, implementado nos arquivos “lamport.c” e “lamport.h”, foi encapsulado na forma de uma biblioteca dinâmica, conforme os seguintes comandos:

```
gcc -fPIC -c lamport.c
gcc -shared -o liblamport.so lamport.o
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
gcc -o programa programa.c -L./ -llamport
./programa
```

Para auxiliar na compilação, foi criado também um arquivo *makefile*. É possível utilizar o comando “*make clean*” para remover quaisquer eventuais arquivos conflitantes e “*make*” para compilar a biblioteca e o programa principal. Em seguida, a execução deve ser feita com “./programa”.

As principais funções implementadas a partir do código “Algoritmo de Lamport - Algoritmo da Padaria” são:

- *lamport_mutex_init*

```
void lamport_mutex_init(){
    for(int i = 0; i<N; i++){
      choosing[i] = 0;
      ticket[i] = 0;
    }
}
```

- *lamport_mutex_lock*

```
void lamport_mutex_lock(int thread_num){
    choosing[thread_num] = 1;
    ticket[thread_num] = max_ticket () + 1;
    choosing[thread_num] = 0;

    for (int j = 0; j < N; j++) {
        while (choosing[j]) /* nao fazer nada */;
      while (ticket[j] != 0 && (
      (ticket[j] < ticket[thread_num]) || (ticket[j] == ticket[thread_num] && j < thread_num)
    )) /* nao fazer nada */;
    }
}
```

- *lamport_mutex_unlock*

```
void lamport_mutex_unlock(int thread_num){
    ticket[thread_num] = 0;
}
```

c. Programa principal

Baseia-se no código “Exemplo de Thread Simples”, disponibilizado pelo professor, que demonstra a condição de corrida através de somas concorrentes em uma variável compartilhada. Para este trabalho, cada thread incrementa a variável global 3.000.000 de vezes, diferentemente das 30 vezes do exemplo disponibilizado. As primitivas desenvolvidas na biblioteca definem quando uma thread entra na seção crítica, bloqueando outras threads de criar uma nova referência crítica. O trecho de código responsável por essa tarefa pode ser conferido a seguir:

- *em thread_start*

```
lamport_mutex_lock((tinfo->num)-1);
for (int i = 0; i < num_rep; i++)
    shared_var = shared_var + 1;
lamport_mutex_unlock((tinfo->num)-1);
```

d. Dificuldades encontradas

Ao usar os ids das threads (t_info->if) do programa principal como parâmetro para as funções da biblioteca do Algoritmo de Lamport, o seguinte erro ocorreu:

```
Hello! I'm thread 1, id 139782619268672!
Hello! I'm thread 2, id 139782610875968!
Segmentation fault (core dumped)
```

Ao utilizar o atributo o correto (t_info->num) o resultado esperado foi obtido:

```
Hello! I'm thread 3, id 140199094908480!
Hello! I'm thread 2, id 140199103301184!
Hello! I'm thread 1, id 140199111693888!
Joined with thread 1, id 140199111693888
Joined with thread 2, id 140199103301184
Joined with thread 3, id 140199094908480
Global var: 9000000
```

Maiores dificuldades não foram encontradas.

3. Resultados

a. Algoritmo da Padaria

Não foram encontradas inconsistências nas execuções do algoritmo em nenhum dos dois ambientes utilizados no trabalho.

```
Hello! I'm thread 1, id 140334778922560!
Hello! I'm thread 2, id 140334770529856!
Hello! I'm thread 3, id 140334762137152!
Joined with thread 1, id 140334778922560
Joined with thread 2, id 140334770529856
Joined with thread 3, id 140334762137152
Global var: 9000000
```

b. Tempos de execução

Para comparar tempos de execução foram feitas 10 execuções em dois ambientes de Sistema Operacional usando-se o seguinte *script bash*.

```
for i in {1..10}; do { time ./programa ; } 2>> resultados_lamport.txt; done

for i in {1..10}; do { time ./mutex ; } 2>> resultados_mutex.txt; done
```

Na tabela a seguir são considerados os tempos *real time* gerados pela função do *bash*.

Implementação	**Tempo médio em 10 execuções**		**Tempo máximo**	**Tempo mínimo**
Em software	***ambiente 1***	0,0482s	0,111s	0,036s
	ambiente 2	0,049s	0,086s	0,033s
Padrão POSIX: pthread_mutex_lock pthread_mutex_unlock	***ambiente 1***	0,0389s	0,057s	0,035s
	ambiente 2	0,033s	0,050s	0,031s

Tabela 2: Tempo de execução entre diferentes implementações