Universidade do Algarve

Faculdade de Ciências e Tecnologia

Licenciatura em Engenharia Informático

Sistemas Operativos

Relatório 3º Guião Prático

Discente: Hugo Paixão, nº 64514

Docentes: Amine Berqia e José Bastos

16 de Maio de 2021

Índice

1. Introdução

1.1 Posix

POSIX (Portable Operating System Interface) é uma família de padrões especificados pela IEEE Computer Society para manter a compatibilidade entre sistemas operativos. POSIX define a interface de programação de aplicativos (API), junto com a linha de comandos shell e interfaces de utilitários, para compatibilidade de software do Unix e outros sistemas.

1.2 Threads

Thread é o termo inglês para "linha de execução", pelo que se trata da forma pelo qual um processo se divide em duas ou mais tarefas que podem ser executadas sequencialmente.

Ao longo do seu tempo de vida, as tarefas mudam de estado. Quando uma tarefa é criada inicialmente (start), ela torna-se pronta a executar (runnable), mas ainda não esta a ser executada até que seja enviada pelo escalonador. Estando em execução (running), a tarefa pode ser libertada pelo processador (end) ou pode voltar ao estado de "pronta a

executar" (runnable). Para alem disso, uma tarefa em execução (running) pode ter que aguardar (waiting) por um certo acontecimento, até que possa voltar a estar pronta a executar (runnable) outra vez.

- pthread_create – Cria uma thread.
- pthread_exit – Terminam a thread em execução.
- pthread_join – Espera que a thread executada termine.
- pthread_cancel – Pede que a thread seja cancelada.
- pthread_mutex – É um dispositivo de exclusão mútua e é útil para proteger estruturas de dados compartilhadas de modificações simultâneas e implementar secções.
- sem_t – semaphore é uma construção de sincronização de thread que pode ser usada para enviar sinais entre threads para evitar sinais perdidos ou para proteger uma seção critica.

2. Concurso P3

Exercício 1.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>

pthread_mutex_t mutex;

char *message = {"Mensagem da thread nº "};


void *printMessage(void *value) {
  pthread_mutex_lock(&mutex);

  long num = (long) value;

  for(int i = 0; i < strlen(message); i++) {
    printf("%c", message[i]);
  }
```

```c
  printf("%ld\n", num);

  pthread_mutex_unlock(&mutex);

  return NULL;
}


int main() {
  pthread_t thread[5];
  pthread_mutex_init(&mutex, NULL);

  for(long i = 0; i < 5; i++)
    pthread_create(&thread[i], NULL, printMessage, (void *)i);

  for(long i = 0; i < 5; i++)
    pthread_join(thread[i], NULL);

  pthread_mutex_destroy(&mutex);

  return 0;
}
```

Exercício 2.

2.1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *work(void *id) {
  long num = (long) id;

  printf("Id: %ld.\n", num);

  sleep(rand() % 1000);

  printf("Thread %ld awake.\n\n", num);

  return NULL;
}

int main() {
  pthread_t thread;
```

```
  long num = 1;

  pthread_create(&thread, NULL, work, (void *)num);

  pthread_join(thread, NULL);

  return 0;
}
```

2.2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>

pthread_mutex_t mutex;


void *work(void *id) {

  pthread_mutex_lock(&mutex);

  long num = (long) id;

  printf("Id: %ld.\n", num);

  sleep(rand() % 5);

  printf("Thread %ld awake.\n\n", num);

  pthread_mutex_unlock(&mutex);

  return NULL;
}

void createAndWait(int n) {
  pthread_t threads[n];

  pthread_mutex_init(&mutex, NULL);

  for(long i = 0; i < n; i++)
    pthread_create(&threads[i], NULL, work, (void *)i);

  for(int i = 0; i < n; i++)
    pthread_join(threads[i], NULL);
```

```
  pthread_mutex_destroy(&mutex);

}

int main() {
  int n;
  scanf("%d", &n);
  createAndWait(n);
  return 0;
}
```

Exercício 3.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>

pthread_t thread, thread2;

char character;

sem_t semaphore, semaphore2;

void *printChar() {

  while(1) {

    sem_wait(&semaphore2);

    printf("%c", character);

    sem_post(&semaphore);
  }
}

void *input() {
  while(1) {
    sem_wait(&semaphore);
    scanf("%c", &character);

    if(character == 'X') {
      pthread_cancel(thread);
      pthread_cancel(thread2);
    }
    sem_post(&semaphore2);
  }
```

```
}

int main() {

  sem_init(&semaphore, 0, 1);
  sem_init(&semaphore2, 0, 0);

  pthread_create(&thread, NULL, input, NULL);

  pthread_create(&thread2, NULL, printChar, NULL);

  pthread_join(thread, NULL);
  pthread_join(thread2, NULL);

  sem_destroy(&semáforo);
  sem_destroy(&semaphore2);

  return 0;
}
```

Exercício 4.

a) O problema apresenta uma condição de corrida porque temos um grande número de clientes a tentar usar um menor número de máquinas, ou dito de outra forma temos varias threads a tentar aceder ao mesmo pedaço de memória e precisamos de algo que meta ordem no acesso há memoria, devemos utilizar os semaphore na resolução deste problema.

b)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
#ifndef NUM_THREADS
#define NUM_THREADS 10
#endif

#define NUM_MACHINE 2

pthread_t thread[NUM_THREADS];

sem_t full, empty;
```

```c
void *machineUnUse(int client) {
    int machine = client % NUM_MACHINE;
    sem_wait(&full);
    printf("Cliente  %d terminou o uso da maquina %d\n.", client, machine
);
    sem_post(&empty);
}

void *machineUse(int client) {
    int machine = client % NUM_MACHINE;
    sem_wait(&empty);
    printf("Cliente %d ésta a usar maquina %d.\n", client, machine);
    sem_post(&full);
}

void *store(void *thread_id) {
    long id = (long) thread_id;

    machineUse(id);
    sleep(1);
    machineUnUse(id);
}

int main() {

    sem_init(&empty, 0, NUM_MACHINE);
    sem_init(&full, 0, 0);

    for(long i = 0; i < NUM_THREADS; i++) {
        pthread_create(&thread[i], NULL, store, (void *)i+1);

        printf("Cliente %ld chegou.\n", i+1);
    }

    for(int i = 0; i < NUM_THREADS; i++)
        pthread_join(thread[i], NULL);

    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;
}
```

Exercício 5.

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

#define NUM_THREADS 2
#define NUM_LINES 4
#define NUM_COLUMNS 4

pthread_t threads[NUM_THREADS];

sem_t mutex;

int matrix_a[NUM_LINES][NUM_COLUMNS] = {{1, 1, 1, 1},
                                        {1, 1, 1, 1},
                                        {1, 1, 1, 1},
                                        {1, 1, 1, 1}};

int matrix_b[NUM_LINES][NUM_COLUMNS] = {{1, 2, 3, 4},
                                        {5, 6, 7, 8},
                                        {9, 10, 11, 12},
                                        {13, 14, 15, 16}};

int result[NUM_LINES][NUM_COLUMNS];

void *matrix_solve(void *arg) {
    sem_wait(&mutex);
    long idx = (long) arg;
    for(int i = 0; i < NUM_LINES; i++) {
        if(idx == i % NUM_THREADS)
            for(int j = 0; j < NUM_COLUMNS; j++) {
                result[i][j] = 0;
                for(int k = 0; k < NUM_COLUMNS; k++)
                    result[i][j] += matrix_a[i][k] * matrix_b[k][j];
        }
    }
    sem_post(&mutex);
}

int main() {

    sem_init(&mutex, 0, 1);

    for(long i = 0; i < NUM_THREADS; i++) {
        pthread_create(&threads[i], NULL, matrix_solve, (void *)i);
    }

    for(long i = 0; i < NUM_THREADS; i++)
        pthread_join(threads[i], NULL);
```

```
    for(int i = 0; i < NUM_LINES; i++) {
        for(int  j = 0; j < NUM_COLUMNS; j++)
            printf("%d  ", result[i][j]);
        printf("\n");
    }

    sem_destroy(&mutex);
    return 0;

}
```

3. Conclusão

Com a elaboração deste relatório foram introduzidos alguns conceitos fundamentais principalmente acerca de processo e a sua manipulação. Neste trabalho prático também adquiri alguma experiência no desenvolvimento de programas que utilizem mecanismos de sincronização e comunicação entre processos, com base na programação em C. Tive diversas dificuldades em alguns dos exercícios. Mas, contudo, pesquisei sobre estas e obtive a informação necessária para a realização dos exercícios e para a conclusão deste trabalho prático. Em suma, com a elaboração do relatório prático, permitiu-me introduzir alguns conceitos, e adquirir alguma experiência na manipulação e criação e threads.

Webgrafia:

https://www.youtube.com/watch?v=nVESQQg-Oiw&t=195s&ab_channel=EngineerMan

https://www.youtube.com/watch?v=oq29KUy29iQ&ab_channel=CodeVault

https://www.youtube.com/watch?v=l6zkaJFjUbM&t=1148s&ab_channel=CodeVault

https://linuxhint.com/posix-semaphores-with-c-programming/