Sistemas Operacionais

Problema dos Cuidados Infantis
(Child Care Problem)

Caio Ueno……753516
Gabriel Cheban……743535
João Augusto Leite……743551
Vinicius Carvalho……743602

Prof. Dr. Hélio Crestana Guardia
Profª. Drª. Kelen Cristiane Teixeira Vivaldini

UFSCar - Universidade Federal de São Carlos

São Carlos, 19 de Maio de 2019

1. Descrição do Problema

O problema ilustra o funcionamento de um centro de cuidados infantis (creche), na qual há a movimentação, entrada e saída, de crianças que frequentam e adultos que trabalham na creche. As crianças podem entrar somente se houver pelo menos um adulto que possa vigiá-la, porém, se não houver uma vaga, elas devem aguardar em uma fila. Os adultos podem entrar sem nenhuma restrição, entretanto, para que possam sair, é necessário que não haja crianças na creche ou que a quantidade de crianças seja de: no máximo 3 vezes a quantidade de adultos restantes após sua saída, pois cada adulto supervisiona até 3 crianças que estão dentro da creche. Caso um adulto queira sair, mas não pode devido às condições impostas, ele deve entrar na fila de espera para sair.

2. Abordagem para Solução do Problema

Utilizando *threads* para representar crianças e adultos, foram necessários semáforos e *mutexes* para controlá-los. A região crítica nesse caso, é o fluxo de pessoas. Um adulto não pode sair da creche caso sua saída inviabilize a estadia de outras crianças. Similarmente, uma criança não pode entrar na creche se não houver uma vaga disponível. Especificamente, deve-se utilizar um semáforo para o controle de entrada de crianças e outro para o controle de saída de adultos. Uma criança que deseja entrar, mas não pode (por indisponibilidade de vagas), deve aguardar em uma fila de *threads* bloqueadas até outra criança sair ou outro adulto entrar. O adulto, por sua vez, caso deseje sair e não seja possível, deve esperar, bloqueado, na fila de adultos até que saiam crianças suficientes para que sua presença não seja necessária. Além disso, será preciso utilizar um *mutex* para garantir que crianças e adultos não entrem e saiam ao mesmo tempo.

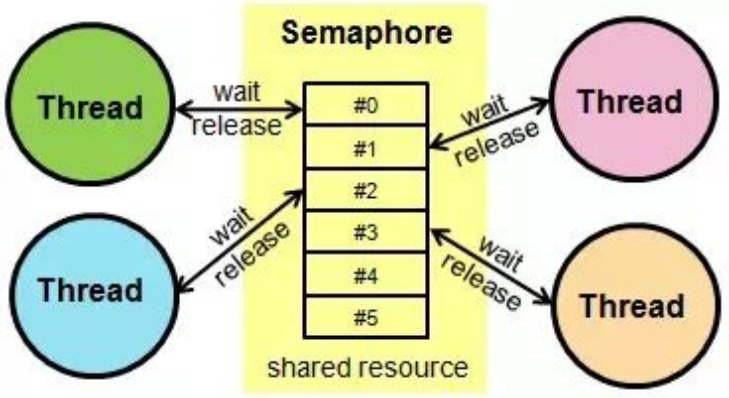

Figura 1: Semáforo

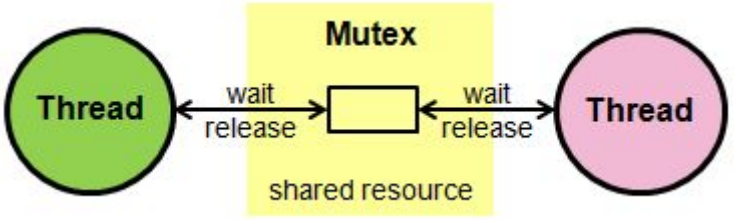

Figura 2: Mutex

3. Solução Completa

Para melhor compreensão do código, este será divido em partes: bibliotecas e variáveis, explicação da entradas e saídas das crianças e adultos, e por fim a estruturação da função *main*.

a. Bibliotecas e Variáveis

A seguir estão as bibliotecas que foram incluídas no programa e a definição de duas constantes *N_CRIANCAS* e *N_ADULTOS*, para caso o usuário não passar como parâmetros as quantidades de crianças e adultos que ele quer.

Além disso, há a inicialização de quatro variáveis globais *_criancas*, *_adultos*, *_criancas_esperando*, *_adultos_saindo*, que indicam quantas crianças estão dentro da creche, quantos adultos estão trabalhando na creche, quantas crianças estão na fila esperando para entrar na creche e a quantidade de adultos na fila para sair da creche, respectivamente.

Após as variáveis globais, foram instanciados dois semáforos e um *mutex*. O semáforo *fila_adultos* é utilizado para bloquear adultos em uma fila de bloqueados para saída. Já o *fila_criancas* controla a entrada das crianças, se uma delas tenta entrar e não pode, o semáforo põe ela na fila de bloqueados para entrar.

O mutex serve para acessar as regiões críticas do programa, as variáveis globais. Cada vez que uma criança ou um adulto precisa atualizar, ou atualizar, esses valores é necessário o uso do mutex.

Foi criada uma função *minimo*, dado dois argumentos inteiros, retorna o menor deles.

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

#define N_CRIANCAS 8
#define N_ADULTOS 3
```

```
int _criancas = 0, _adultos = 0, _criancas_esperando = 0,
_adultos_saindo = 0;  // Variaveis globais.

sem_t fila_adultos;
sem_t fila_criancas;
pthread_mutex_t mutex;

int minimo(int a, int b){
  if(a < b)
      return a;
  return b;
}
```

b. Entrada da criança

Ao entrar na região crítica, a thread trava o mutex, impedindo a entrada de outras threads. É feita a verificação se o número de crianças é menor que o número de adultos multiplicado por 3. Se sim, entra uma criança e a thread sai da região crítica, destravando-a para as demais. Senão, o contador de crianças de crianças na fila é incrementado, a thread sai da região crítica e a criança é colocada para dormir.

```
void * criancasThread() {

  pthread_mutex_lock(&mutex); // Entra na RC

  if(_criancas < 3 * _adultos){ // Se existe uma vaga
      _criancas++;                // Entra
      printf("Crianca %X entrou\n ===>Variaveis globais: _crianças
= %d; _adultos = %d; _criancas_esperando = %d; _adultos_saindo =
%d;\n\n",(int) pthread_self(), _criancas, _adultos,
_criancas_esperando, _adultos_saindo);
      pthread_mutex_unlock(&mutex); // Sai da RC
  }

  else{    // Nao tem vaga
      _criancas_esperando++;  // +1 crianca esperando para entrar
      printf("Crianca %X tentando entrar\n ===>Variaveis globais:
_crianças = %d; _adultos = %d; _criancas_esperando = %d;
_adultos_saindo = %d;\n\n",(int) pthread_self(), _criancas,
```

```
_adultos, _criancas_esperando, _adultos_saindo);
      sleep(2);
      pthread_mutex_unlock(&mutex); // Sai da RC
      sem_wait(&fila_criancas); // crianca dorme ate liberar uma
vaga
  }

  //Logica de espera
  sleep(8);
```

c. Entrada do adulto

De forma similar à entrada da thread criança, a thread adulto também deve travar o mutex ao entrar na região crítica. Nesse momento, o contador de adultos na creche é incrementado e é verificado se há crianças dormindo na fila de crianças. Caso haja, acorda-se até 3 crianças, decrementando o contador de crianças na fila e incrementando o contador de crianças na creche. Por fim, a thread adulto sai da região crítica, destravando o mutex.

```
void * adultosThread() {

  int n, i;

  pthread_mutex_lock(&mutex); // Entra na RC
  _adultos++; // +1 adulto na creche
  printf("Adulto %X entrou\n ===>Variaveis globais: _crianças =
%d; _adultos = %d; _criancas_esperando = %d; _adultos_saindo =
%d;\n\n",(int) pthread_self(), _criancas, _adultos,
_criancas_esperando, _adultos_saindo);
  if (_criancas_esperando){   // Se tem criancas esperando
      n = minimo(3, _criancas_esperando); // Pega n criancas da
fila, com n <= 3

      for (i = 0; i < n; i++) {
      sem_post(&fila_criancas); // Acorda n criancas
      _criancas_esperando--;  // -n criancas na fila
      _criancas++;  // +n criancas dentro
      printf("Crianca %X entrou\n ===>Variaveis globais: _crianças
= %d; _adultos = %d; _criancas_esperando = %d; _adultos_saindo =
%d;\n\n",(int) pthread_self(), _criancas, _adultos,
```

```
_criancas_esperando, _adultos_saindo);
      sleep(2);
      }

  }

  pthread_mutex_unlock(&mutex); // sai da RC

  //Logica de espera
  sleep(4);
```

d. Saída da criança

Caso alguma criança saia, a variável é decrementada. Depois, é verificada a possibilidade de um adulto ir embora, ou seja, se há um deles na fila e o número de crianças é menor que três vezes o número de adultos menos um. Se sim, o número de adultos na creche e na fila é decrementado, e, um é acordado.

Em seguida, se há alguma criança esperando para entrar e tem uma vaga, ela é acordada, entra, decrementa o número na fila e incrementa o total da creche. Por fim, a thread sai da região crítica.

```
pthread_mutex_lock(&mutex);
  _criancas--; //Sai
  printf("Crianca %X saiu\n ===>Variaveis globais: _crianças = %d;
_adultos = %d; _criancas_esperando = %d; _adultos_saindo =
%d;\n\n",(int) pthread_self(), _criancas, _adultos,
_criancas_esperando, _adultos_saindo);
  sleep(2);

  if(_adultos_saindo && (_criancas <= 3 * (_adultos - 1))){  // Se
tem adulto querendo sair e o nro de criancas tirando 1 adulto
ainda eh seguro
      _adultos_saindo--;  // Decrementa a fila de adultos querendo
sair
      _adultos--; //Adulto sai
      printf("Adulto %X saiu\n ===>Variaveis globais: _crianças =
%d; _adultos = %d; _criancas_esperando = %d; _adultos_saindo =
%d;\n\n",(int) pthread_self(), _criancas, _adultos,
_criancas_esperando, _adultos_saindo);
      sleep(2);
      sem_post(&fila_adultos);  // Acorda um adulto que queria sair
```

```
e ele sai
  }
  if(_criancas_esperando && (_criancas < 3 * _adultos)){ // Se tem
criancas esperando e tem espaco pra ela
      sem_post(&fila_criancas); // Ela entra
      _criancas_esperando--;  // -1 crianca esperando
      _criancas++;     // +1 crianca dentro
      printf("Crianca %X entrou\n ===>Variaveis globais: _crianças
= %d; _adultos = %d; _criancas_esperando = %d; _adultos_saindo =
%d;\n\n",(int) pthread_self(), _criancas, _adultos,
_criancas_esperando, _adultos_saindo);
      sleep(2);
  }
  pthread_mutex_unlock(&mutex); // Sai da RC

  pthread_exit(NULL);
}
```

e. Saída do adulto

Quando um adulto quer sair, ele deve travar o mutex para entrar na região crítica e verificar se a sua saída resulta em um estado seguro, ou seja, se mesmo com um adulto a menos, a creche ainda comporta a quantidade de crianças presentes. Em caso afirmativo, o adulto pode sair, decrementando o contador de adultos na creche. Senão, ele deve incrementar o contador de adultos querendo sair, destravar o mutex e ser colocado na fila de adultos para dormir.

```
 pthread_mutex_lock(&mutex); // Entra na RC
  if ((_criancas <= 3 * (_adultos - 1)) && ((_adultos > 1) ||
(_criancas_esperando == 0 && _criancas == 0))){
      _adultos--;
      printf("Adulto %X saiu\n ===>Variaveis globais: _crianças =
%d; _adultos = %d; _criancas_esperando = %d; _adultos_saindo =
%d;\n\n",(int) pthread_self(), _criancas, _adultos,
_criancas_esperando, _adultos_saindo);
      sleep(2);
      pthread_mutex_unlock(&mutex);
  }else{
      _adultos_saindo++;
      printf("Adulto %X tentando sair\n ===>Variaveis globais:
_crianças = %d; _adultos = %d; _criancas_esperando = %d;
_adultos_saindo = %d;\n\n",(int) pthread_self(), _criancas,
_adultos, _criancas_esperando, _adultos_saindo);
      sleep(2);
      pthread_mutex_unlock(&mutex);
      sem_wait(&fila_adultos);
  }

  pthread_exit(NULL);

}
```

f. Função main

Na função main, inicializa-se os semáforos e o mutex. É feita uma verificação dos argumentos passados na hora da execução do programa, caso nenhum seja passado então são utilizados os valores de adultos e crianças definidos no começo do programa, ou os respectivos valores passados pelo usuário. Caso o número de entrada seja incompatível, é emitida uma mensagem de erro e o processo é terminado. Por fim, são alocados dinamicamente vetores de threads para crianças e adultos.

```
int main(int argc, char const *argv[]) {

  sem_init(&fila_adultos, 0, 0);
  sem_init(&fila_criancas, 0, 0);
  pthread_mutex_init(&mutex, NULL);
```

```
  int n_criancas, n_adultos, retorno;

  if (argc == 1) { // Nao passou o numero de criancas e adultos
      n_criancas = N_CRIANCAS;
      n_adultos = N_ADULTOS;
  }

  if (argc > 3) { // Passou mais do que 2 parametros
      printf("Programa %s com número de entradas incompativel!
(%d)\n", argv[0], argc);
      exit(0);
  }

  if (argc == 3){
      n_criancas = atoi(argv[1]);
      n_adultos = atoi(argv[2]);
  }

  pthread_t* threads_criancas = (pthread_t*)
malloc(sizeof(pthread_t) * n_criancas);
  pthread_t* threads_adultos = (pthread_t*)
malloc(sizeof(pthread_t) * n_adultos);

  printf("crianca %d\n", n_criancas);
  printf("adulto %d\n", n_adultos);
```

São criadas as threads das crianças, caso haja erro na criação de alguma delas o processo é terminado. Depois são criadas as threads dos adultos. A função *pthread_join* é invocada duas vezes, uma para as threads das crianças e outras para os adultos. Por fim, a função *free* desaloca os vetores de threads.

```
  for (int i=0; i<n_criancas; i++){
      retorno = pthread_create(&threads_criancas[i], NULL,
criancasThread, NULL);
      if (retorno) {  // Erro na criacao da thread crianca
      printf("Erro na criação de uma thread criança!\n");
      exit(0);
      }
  }
```

```
  for (int i=0; i<n_adultos; i++){
      retorno = pthread_create(&threads_adultos[i], NULL,
adultosThread, NULL);
      if (retorno) { // Erro na criacao da thread adulto
      printf("Erro na criação de uma thread adulto!\n");
      exit(0);
      }
  }

  for (int i=0; i<n_adultos; i++)
      pthread_join(threads_adultos[i], NULL);
  for (int i=0; i<n_criancas; i++)
      pthread_join(threads_criancas[i], NULL);

  free(threads_criancas);
  free(threads_adultos);

  return 0;
}
```

4. Conclusão

A solução obtida, utilizando dois semáforos e um mutex, é satisfatória, evita o surgimento de *deadlocks* e inconsistências nos valores das variáveis compartilhadas. Dessa forma, a entrada/saída de crianças e adultos ocorre conforme o planejado.

5. Referências

[1] Allen B. D. The Little Book of Semaphores, Second Edition, 2016.

[2] Material de apoio do Prof. Dr. Hélio Crestana Guardia e da Profª. Drª. Kelen Cristiane Teixeira Vivaldini.

[3] Tanenbaum, A. S. *Modern Operating Systems*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2008.

[4] cs.indiana.edu (imagens).