МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ

ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Заочный факультет

Кафедра информационных систем и технологий

**КОНТРОЛЬНАЯ РАБОТА   
ПО ПАРАЛЛЕЛЬНОМУ ПРОГРАММИРОВАНИЮ**

Вариант 22

Выполнил: студент группы *9651*

Сахипов Р.Р.

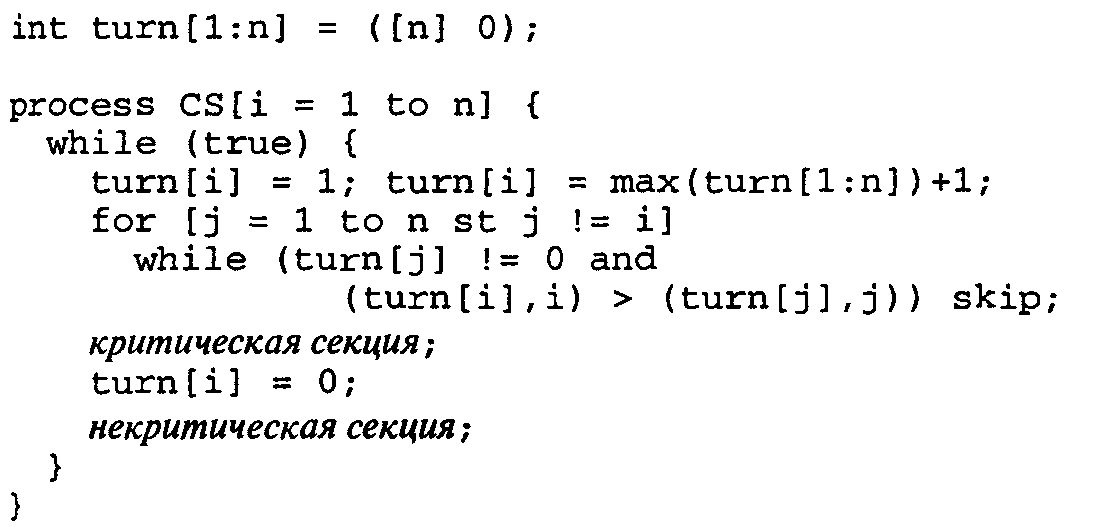
Проверила: Соловьева Яна Владимировна

Самара, *2016*

# Задача

В алгоритме поликлиники значение переменной turn не ограничены, если в критической секции постоянно находится хотя бы один процесс. Допустим, что чтение и запись неделимы. Можно ли изменить алгоритм так, чтобы значения turn всегда были ограниченными? Если да, предложите измененный алгоритм. Если нет, объясните почему.

Листинг 3.11



# Решение

Рассмотрим для примера реализацию алгоритма поликлиники для n потоков. Пусть turn[1:n] – массив целых чисел с начальными значениями 0. Протокол входа в КС (критическую секцию) следующий: поток CS[i] сначала присваивает переменной turn[i] значение, которое на 1 больше, чем максимальное среди текущих значений элементов массива turn. Затем поток ожидает, пока его значение turn[i] не станет наименьшим среди ненулевых значений массива turn. Выходя из КС поток CS[i] присваивает turn[i] значение 0.

Для мелкомодульного решения необходимо реализовать два неделимых действия в протоколе входа.

Требование неделимости операции поиска максимального элемента можно ослабить, если поток будет предупреждать о своем намерении войти в КС, присваивая своей переменной turn[i] значение 1. В результате несколько потоков могут получить одинаковые номера. Такие потоки можно, например, упорядочить по номеру потока i.

Второе неделимое действие можно реализовать циклом while с проверкой для потоков, ожидающих вход в КС, их значения очереди. Если поток имеет очередь не больше всех остальных потоков, то он выполняется, если его порядковый номер наибольший среди ожидающих потоков с таким же номером очереди. В результате мелкомодульное решение задачи КС с помощью алгоритма поликлиники можно записать следующим образом:

int turn[1:n]=([n] 0);

process CS[i = 1 to n] {

while (true) {

turn[i] = 1; turn[i] = max(turn[1 : n]) + 1;

for [j = 1 to n st j! = i]

while turn[j]! = 0 and (turn[i], i) > (turn[j], j) skip;

# либо turn[i] > turn[j], либо i > j, если turn[i] = turn[j]

[критическая секция];

turn[i] = 0;

[некритическая секция];

}

}

Заметим, что максимальное значение в массиве определяется считыванием всех его элементов и отысканием наибольшего. Поскольку эти действия не являются неделимыми, то точный результат не гарантируется. Но если несколько процессов получают одно и то же “значение очереди” (т.е. номер), то происходит их упорядочивание по правилу.

Данный алгоритм был реализован на языке C++. Исходный код программы представлен в приложении А.

Конструктором процессов является функция Thread в которой вызывается функция блокировки процесса (lock) и разблокировки (unlock):

void Thread(int i) {

while (true) {

lock(i);

/\* критическая секция \*/

COUNT\_CS[i + 1] = COUNT\_CS[i + 1] + 1;

printf("критическая секция потока %d\n", i + 1);

usleep(DURATION \* rand() / 2);

unlock(i);

/\* некритической секция \*/

}

}

Условие блокировки процесса выполнено следующим образом (функция lock):

void lock(int i) {  
 TURN[i] = 1;  
 TURN[i] = myMAX() + 1;  
  
 for (int j = 1; j <= NUM\_THREADS; j++) {  
 while ((TURN[j] != 0) &&  
 ((TURN[i] > TURN[j]) ||  
 ((TURN[i] == TURN[j]) && (i > j)))) {  
 /\* Ничего не делать \*/  
 }  
 }  
}

Поиск максимума реализован как отдельная функция myMAX:

int myMAX() {  
 int i = 0;  
 int MAXvalue = 0;  
  
 for (i = 0; i < NUM\_THREADS; i++) {  
 if ((TURN[i]) > MAXvalue) {  
 MAXvalue = TURN[i];  
 }  
 }  
  
 return MAXvalue;  
}

Теперь необходимо ответим на вопрос:

Можно ли изменить алгоритм так, чтобы значение turn всегда были ограниченными?

Алгоритм можно изменить таким образом, что значение turn не превышало определённого лимита, установленного заранее. Для этого необходимо: ввести новую переменную MyLIMIT, задающую лимит значений; проверять, не превышает ли максимальное значение заданный лимит.

Изменённый код выглядит следующим образом:

Дляблокировкипроцесса - функцияlock

void lock(int i) {  
 TURN[i] = 1;  
 int x = 0;  
  
 do {  
 x = TURN[i] + myMAX();  
 } while (x == 1);  
  
 TURN[i] = x;  
  
 for (int j = 1; j <= NUM\_THREADS; j++) {  
 while ((TURN[j] != 0) &&  
 ((TURN[i] > TURN[j]) ||  
 ((TURN[i] == TURN[j]) && (i > j)))) {  
 /\* Ничего не делать \*/  
 }  
 }  
}

Дляпоискамаксимума - функцияmyMAX

int myMAX() {  
 int i = 0;  
 int MAXvalue = 0;  
  
 for (i = 0; i < NUM\_THREADS; i++) {  
 if ((TURN[i]) > MAXvalue) {  
 MAXvalue = TURN[i];  
 }  
 }  
  
 if (MAXvalue < MyLIMIT) {  
 return MAXvalue;  
 } else {  
 return 0;  
 }  
}

Листинг изменённой программы представлен в приложении Б.

# Использованные источники

1. Эндрюс Г.Р. Основы многопоточного, параллельного и распределённого программирования. – М.: Издательский дом «Вильямс», 2003.

# Приложение А

Листинг разработанной программы

#include<pthread.h>  
#include<stdio.h>  
#include<unistd.h>  
  
volatileintNUM\_THREADS = 10; // количество потоков  
volatileintTURN[10] = {0}; // очередь  
volatileintCOUNT\_CS[10] = {0}; // счётчик  
volatileintDURATION = 10000; // задержка

/\* поиск максимального значения \*/  
intmyMAX() {  
inti = 0;  
intMAXvalue = 0;  
  
for (i = 0; i<NUM\_THREADS; i++) {  
if ((TURN[i]) >MAXvalue) {  
MAXvalue = TURN[i];  
 }  
 }  
  
returnMAXvalue;  
}  
  
voidlock(inti) {  
TURN[i] = 1;  
TURN[i] = myMAX() + 1;  
  
for (intj = 1; j<= NUM\_THREADS; j++) {  
while ((TURN[j] != 0) &&  
((TURN[i] >TURN[j]) ||  
 ((TURN[i] == TURN[j]) && (i>j)))) {  
 /\* Ничего не делать \*/  
 }  
 }  
}  
  
voidunlock(inti) {  
TURN[i] = 0;  
}  
  
voidThread(inti) {  
while (true) {  
lock(i);  
  
 /\* критическая секция \*/  
COUNT\_CS[i + 1] = COUNT\_CS[i + 1] + 1;  
printf("критическая секция потока %d\n", i + 1);  
  
usleep(DURATION \* rand() / 2);  
  
unlock(i);  
 /\* некритической секция \*/  
 }  
}  
  
intmain() {  
 /\* массив потоков \*/  
pthread\_tthreads[NUM\_THREADS];  
  
 /\* создание потоков \*/  
for (longt = 0; t<NUM\_THREADS; t++) {  
printf("Гланый поток: создание потока %ld\n", t + 1);  
pthread\_create(&threads[t], NULL, Thread, (void \*) t);  
 }  
  
usleep(DURATION \* 1000);  
  
 /\* завершение всех потоков \*/  
pthread\_exit(NULL);  
  
return 0;  
}

# Приложение Б

Листингизменённойпрограммы

#include <pthread.h>  
#include <stdio.h>  
#include <unistd.h>  
  
volatile int NUM\_THREADS = 10;  
volatile int TURN[10] = {0};  
volatile int COUNT\_CS[10] = {0};  
volatile int DURATION = 10000;  
volatile int MyLIMIT = 100;  
  
int myMAX() {  
 int i = 0;  
 int MAXvalue = 0;  
  
 for (i = 0; i < NUM\_THREADS; i++) {  
 if ((TURN[i]) > MAXvalue) {  
 MAXvalue = TURN[i];  
 }  
 }  
  
 if (MAXvalue < MyLIMIT) {  
 return MAXvalue;  
 } else {  
 return 0;  
 }  
}  
  
void lock(int i) {  
 TURN[i] = 1;  
 int x = 0;  
  
 do {  
 x = TURN[i] + myMAX();  
 } while (x == 1);  
  
 TURN[i] = x;  
  
 for (int j = 1; j <= NUM\_THREADS; j++) {  
 while ((TURN[j] != 0) &&  
 ((TURN[i] > TURN[j]) ||  
 ((TURN[i] == TURN[j]) && (i > j)))) {  
*/\* Ничего не делать \*/*}  
 }  
}  
  
void unlock(int i) {  
 TURN[i] = 0;  
}  
  
void Thread(int i) {  
 while (true) {  
 lock(i);  
  
*/\* критическая секция \*/*COUNT\_CS[i + 1] = COUNT\_CS[i + 1] + 1;  
 printf("критическая секция потока %d\n", i + 1);  
  
 usleep(DURATION \* rand() / 2);  
  
 unlock(i);  
*/\* некритической секция \*/*}  
}  
  
int main() {  
*/\* массив потоков \*/*pthread\_t threads[NUM\_THREADS];  
  
*/\* создание потоков \*/*for (long t = 0; t < NUM\_THREADS; t++) {  
 printf("Гланый поток: создание потока %ld\n", t + 1);  
 pthread\_create(&threads[t], NULL, Thread, (void \*) t);  
 }  
  
 usleep(DURATION \* 1000);  
  
*/\* завершение всех потоков \*/*pthread\_exit(NULL);  
  
 return 0;  
}