**Реализация широковещательного маркерного алгоритма Сузуки-Касами**

**Лычева Екатерина Олеговна**

**421 группа**

1. **Описание алгоритма**

Алгоритм Сузуки-Касами – широковещательный маркерный алгоритм синхронизации в распределённых системах.

*Структура данных и обозначения:*

* Массив целых чисел RN[1…N].

Процесс Si хранит RNi[1…N], где RNi[j] - наибольший номер последовательности, полученный из запроса от процесса Sj.

* Массив целых чисел LN[1…N].

Этот массив используется маркером, LN[j] - номер последовательности запроса, который недавно был выполнен процессом Sj.

* Очередь Q.

Эта структура данных используется маркером для записи идентификаторов процессов, ожидающих маркер.

*Алгоритм:*

- Вход в критическую секцию:

Когда процесс Si хочет войти в критическую секцию и у него нет маркера, то он увеличивает свой номер последовательности RNi[i] = SN и отправляет запрос маркера REQUEST(i, SN) остальным процессам.

Когда процесс Sj получает запрос REQUEST(i, SN) от процесса Si, он устанавливает RNj[i] = max(RNj[i],SN). После обновления RNj[i] процесс Sj отправляет маркер процессу Si, если маркер у него и RNj[i] = LN[i] + 1.

* Выполнение критической секции:

Процесс Si выполняет критическую секцию, если маркер у него.

* Выход из критической секции:

После завершения выполнения критической секции процесс Si выходит из нее и выполняет следующее:

Устанавливает LN[i] = RNi[i], чтобы указать, что его запрос на критическую секцию RNi[i] был выполнен.

Для каждого процесса Sj, чей ID отсутствует в очереди маркера Q, добавляет его ID в Q, если RNi[j] = LN[j] + 1, чтобы указать, что у процесса Sj есть необработанный запрос.

После этих обновлений, если очередь Q не пуста, извлекает ID процесса из Q и отправляет маркер процессу с этим ID.

Если очередь Q пуста, он сохраняет маркер.

1. **Детали реализации**

Считается, что процесс может находиться в одном из трёх состояний – свободен, запросил маркер, получил маркер. Все процессы изначально находятся в состоянии «свободен».

В состоянии «свободен» процесс периодически проверяет, имеются ли новые запросы от других процессов. Если имеются, обновляет RN. Если процесс является владельцем маркера, то в ответ на запрос он отправляет маркер при RNj[i] = LN[i] + 1. Также процесс следит за сообщениями об обновлении глобального счётчика (по условию задачи все процессы проходят критическую секцию ровно один раз => ждём, когда все выполнятся).

Если процесс ещё не входил в критическую секцию, то он отправляет всем остальным процессам запрос в соответствии с алгоритмом, описанным выше, и переходит в состояние «запросил маркер».

Если глобальный счётчик достиг значения, равного числу процессов, то все процессы завершают работу.

В состоянии «запросил маркер» процесс также следит за запросами и обновлениями счётчика. Также процесс ожидает маркер, и когда получает его, переходит в состояние «получил маркер».

В состоянии «получил маркер» процесс действует согласно алгоритму, представленному выше, и затем обновляет глобальный счётчик процессов, выполнивших критическую секцию, и отправляет его остальным процессам. После этого процесс возвращается в состояние «свободен».

**Временная оценка**

Время отправки сообщения рассчитывается по формуле Tn = Ts + Tb\*N.

Будем считать, что размер int равен 4 байтам.

- 0 процесс выполнит критическую секцию бесконечно быстро

- следующие процессы разошлют по 24 сообщения REQUEST (4 байта) + получат один ответ с токеном (204 байт)

- для синхронизации счётчика каждый процесс посылает 24 сообщения (4 байт)

Тогда T = 24\*(24\*(100+1\*4) + 100 + 204\*1) + 25\*24\*(100 + 4\*1) = 189600.

В общем случае:

- на получение маркера нужно не более (N – 1) сообщения REQUEST и 1 сообщения с токеном

- время прохода критической секции считаем бесконечно малым

- для синхронизации каждый процесс отправляет (N - 1) сообщение с новым значением счётчика.

В силу особенностей задачи и реализации первый процесс (владелец маркера) сразу переходит к выполнению критической секции => не выполняет запрос маркера. Тогда:

T = (N-1)\*((N-1)\*(Ts+int\_size\*Tb) + Ts + token\_size\*Tb) + N\*(N-1)\*(Ts + int\_size)