Черепнина Мария, гр 427

Распределенные системы

Задание 1

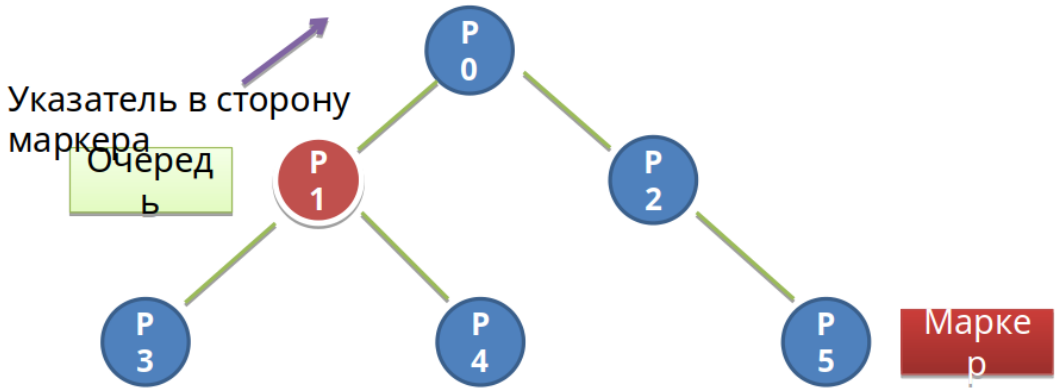
Условие:
Все 25 процессов, находящихся на разных ЭВМ сети, одновременно выдали запрос на вход в критическую секцию. Реализовать программу, использующую древовидный маркерный алгоритм для прохождения всеми процессами критических секций.
Критическая секция:

```
<проверка наличия файла “critical.txt”>;
if (<файл “critical.txt” существует>) {
<сообщение об ошибке>;
<завершение работы программы>;
} else {
<создание файла “critical.txt”>;
Sleep (<случайное время>);
<уничтожение файла “critical.txt”>;
}
```

Для передачи маркера использовать средства MPI.

Алгоритм:

Алгоритм древовидный маркерный (Raymond)



Поведение процесса в соответствии с древовидным маркерным алгоритмом реализует класс Node(), знающий свой номер, номер своего родителя и левого и правого потомка.

```
class Node {
    int current_process;
    int parent_process;
    int left_process;
    int right_process;
}
```

Node проверяет, владеет ли он маркером. Если не владеет, то ожидает получения маркера от родителя. После получения маркера Node работает по следующему алгоритму:

1. Выполняет действия в критической секции
2. Передает маркер левому потомку
3. Передает маркер правому потомку
4. Возвращает маркер родителю

Передача маркера реализована с помощью средств MPI

```
MPI_Send(&number, 1, MPI_INT, left_process, 0, MPI_COMM_WORLD);
            MPI_Recv(&number, 1, MPI_INT, left_process, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Временная оценка работы:

Время работы = 100 *[время старта]* + 2*(2^h - 2) *[2* кол-во ребер]* - h *[можно не возвращать маркер в корень]*

Запуск:

```
$ mpic++ -o run main.cpp  # build
$ mpiexec -np 25 --map-by :OVERSUBSCRIBE ./run  # run
```

```
process 22 wait the marker from 10
process 2 wait the marker from 0
process 5 wait the marker from 2
process 6 wait the marker from 2
process 0 got the marker
process 4 wait the marker from 1
process 1 wait the marker from 0
process 3 wait the marker from 1
process 12 wait the marker from 5
process 8 wait the marker from 3
process 23 wait the marker from 11
process 15 wait the marker from 7
process 16 wait the marker from 7
process 11 wait the marker from 5
process 13 wait the marker from 6
process 9 wait the marker from 4
process 7 wait the marker from 3
process 10 wait the marker from 4
process 19 wait the marker from 9
process 24 wait the marker from 11
process 17 wait the marker from 8
process 14 wait the marker from 6
process 21 wait the marker from 10
process 20 wait the marker from 9
process 18 wait the marker from 8
process 1 got the marker
process 3 got the marker
process 7 got the marker
```

```
process 15 got the marker
process 16 got the marker
process 8 got the marker
process 17 got the marker
process 18 got the marker
process 4 got the marker
process 9 got the marker
process 19 got the marker
process 20 got the marker
process 10 got the marker
process 21 got the marker
process 22 got the marker
process 2 got the marker
process 5 got the marker
process 11 got the marker
process 23 got the marker
process 24 got the marker
process 12 got the marker
process 6 got the marker
process 13 got the marker
process 14 got the marker
```

Задание 2

Условие:
Доработать MPI-программу, реализованную в рамках курса “Суперкомпьютеры и параллельная обработка данных”. Добавить контрольные точки для продолжения работы программы в случае сбоя. Реализовать один из 3-х сценариев работы после сбоя: а) продолжить работу программы только на “исправных” процессах; б) вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов; в) при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

Задача: Red-Black2D

Алгоритм:

- Каждый из K процессов итеративно выполняет обработку N//K строк входной матрицы A, где N -- размерность матрицы
- После каждой итерации сохраняется резервная копия обрабатываемой матрицы A. (Это необходимо, т.к. в случае возникновения ошибки в одном из процессов во время обработки матрицы, значения этой матрицы останутся не консистентными)
- В случае возникновения ошибки в одном из обрабатывающих потоков (сигнала SIGKILL), вызывается обработчик `verbose_errhandler()`.
    1. Обработчик перераспределяет вычисления между оставшимися `K-1` процессами так, чтоб каждый процесс обрабатывал `N//(K-1)` строк матрицы A
    2. Обработчик восстанавливает матрицу A из резервной копии
    3. Обработчик с помощью передачи управления по `longjmp` запускает заново вычисления с прерванной итерации

Запуск:

```
$ mpicc -std=c99 -o run_1 redb_2d.c -lm  # build
$ mpirun -np 4 --mca shmem posix --mca opal_event_include poll
--map-by :OVERSUBSCRIBE --with-ft ulfm ./run_1
  # running on 4 processors
```

Результат:

Программа продолжает выполнение, даже если один из процессов прекратил работу

```
$ mpicc -std=c99 -o run_1 redb_2d.c -lm && mpirun -np 6 --mca
shmem posix --mca opal_event_include poll --map-by
:OVERSUBSCRIBE --with-ft ulfm ./run_1

num_workers: 6, rank 3: first_row 31, last_row 41
num_workers: 6, rank 5: first_row 51, last_row 65
num_workers: 6, rank 0: first_row 1, last_row 11
num_workers: 6, rank 4: first_row 41, last_row 51
num_workers: 6, rank 2: first_row 21, last_row 31
num_workers: 6, rank 1: first_row 11, last_row 21
KILL
num_workers: 5, rank 2: first_row 25, last_row 37
num_workers: 5, rank 0: first_row 1, last_row 13
```

```
num_workers: 5, rank 1: first_row 13, last_row 25
num_workers: 5, rank 3: first_row 37, last_row 49
num_workers: 5, rank 4: first_row 49, last_row 65
  S = 49848.574219
time taken for thread=5, N=66: 0.009994 seconds
```