Работа 4. Векторные часы (логические часы)

В заданном примере реализованы функции поддержания векторных логических часов для MPI-процессов. Функция ds_send передает сообщение заданному процессы с вектором локального времени, а функция ds_recv принимает сообщение и корректирует локальный вектор времени на основе полученного.

Задание:

1. Реализовать передачу сообщения из процесса 0 во все остальные по схеме завершенного бинарного дерева (complete binary tree) – каждый процесс на основные своего номера *rank* и числа процессов *commsize* определяет свое положение в дереве.
2. Какой процесс по логическим часам завершает работу последним?

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

typedef struct ds {
    int *vclock;
    int nprocs;
    int rank;
    MPI_Comm comm;
} ds_t;

ds_t *ds_create(MPI_Comm comm)
{
    ds_t *ds = malloc(sizeof(*ds));
    if (NULL == ds) {
        return NULL;
    }
    ds->comm = comm;
    MPI_Comm_size(comm, &ds->nprocs);
    MPI_Comm_rank(comm, &ds->rank);
    ds->vclock = malloc(sizeof(*(ds->vclock)) * ds->nprocs);
    if (NULL == ds->vclock) {
        free(ds);
        return NULL;
    }
    for (int i = 0; i < ds->nprocs; i++)
        ds->vclock[i] = 0;
    return ds;
}

void ds_free(ds_t *ds)
{
    free(ds->vclock);
    free(ds);
}

void ds_print_clock(ds_t *ds)
{
    printf("proc %d clock: ", ds->rank);
    for (int i = 0; i < ds->nprocs; i++)
        printf("%d ", ds->vclock[i]);
    printf("\n");
}

void ds_event(ds_t *ds, char *event)
{
    ds->vclock[ds->rank]++;
    printf("proc %d event '%s' time %d\n", ds->rank, event, ds->vclock[ds->rank]);
}

int ds_send(ds_t *ds, void *buf, int count, MPI_Datatype datatype, int dest, int
tag)
{
    int userbuf_size, clockbuf_size, packbuf_size;
    MPI_Pack_size(count, datatype, ds->comm, &userbuf_size);
    MPI_Pack_size(ds->nprocs, MPI_INT, ds->comm, &clockbuf_size);
    packbuf_size = userbuf_size + clockbuf_size;
    char *outbuf = malloc(sizeof(*buf) * packbuf_size);
    if (NULL == outbuf) {
        fprintf(stderr, "error: no enough memory\n");
        MPI_Abort(ds->comm, EXIT_FAILURE);
    }
    int position = 0;
    MPI_Pack(buf, count, datatype, outbuf, packbuf_size, &position, ds->comm);
```

```
    MPI_Pack(ds->vclock, ds->nprocs, MPI_INT, outbuf, packbuf_size, &position, ds->comm);
    MPI_Send(outbuf, position, MPI_PACKED, dest, tag, ds->comm);
    free(outbuf);
}

int ds_recv(ds_t *ds, void *buf, int count, MPI_Datatype datatype, int src, int tag)
{
    int userbuf_size, clockbuf_size, packbuf_size, comm_size;
    MPI_Pack_size(count, datatype, ds->comm, &userbuf_size);
    MPI_Pack_size(ds->nprocs, MPI_INT, ds->comm, &clockbuf_size);
    packbuf_size = userbuf_size + clockbuf_size;
    char *inbuf = malloc(sizeof(*buf) * packbuf_size);
    int *vc = malloc(sizeof(*vc) * ds->nprocs);
    if (NULL == inbuf || NULL == vc) {
        fprintf(stderr, "error: no enough memory\n");
        MPI_Abort(ds->comm, EXIT_FAILURE);
    }

    MPI_Recv(inbuf, packbuf_size, MPI_PACKED, src, tag, ds->comm, MPI_STATUS_IGNORE);
    int position = 0;
    MPI_Unpack(inbuf, packbuf_size, &position, buf, count, datatype, ds->comm);
    MPI_Unpack(inbuf, packbuf_size, &position, vc, ds->nprocs, MPI_INT, ds->comm);
    free(inbuf);

    for (int i = 0; i < ds->nprocs; i++) {
        ds->vclock[i] = vc[i] > ds->vclock[i] ? vc[i] : ds->vclock[i];
    }
    ds->vclock[ds->rank]++;
    free(vc);
}

int main(int argc, char **argv)
{
    MPI_Comm comm = MPI_COMM_WORLD;
    int rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(comm, &rank);

    ds_t *ds = ds_create(comm);

    if (rank == 0) {
        ds_event(ds, "0-A");
        ds_event(ds, "0-B");
        ds_event(ds, "0-C");
        ds_print_clock(ds);
        int buf = rank;
        ds_send(ds, &buf, 1, MPI_INT, 1, 0);
    } else if (rank == 1) {
        int buf;
        ds_print_clock(ds);
        ds_recv(ds, &buf, 1, MPI_INT, 0, 0);
        ds_event(ds, "1-B");
        ds_print_clock(ds);
        ds_send(ds, &buf, 1, MPI_INT, 2, 0);
    } else if (rank == 2) {
        int buf;
        ds_print_clock(ds);
        ds_recv(ds, &buf, 1, MPI_INT, 1, 0);
        ds_event(ds, "2-B");
        ds_print_clock(ds);
    }
```

```
    ds_free(ds);

    MPI_Finalize();
    return  0;
}
```