```
typedef struct hfsc_class {
    struct hfsc_class *parent;
    struct hte_ring *children[MAX_CHILDREN];
    int num_children;
    bool is_leaf;

    struct rte_ring *q;

    service_curve_t rsc;
    service_curve_t fsc;
    service_curve_t usc;

    runtime_sc_t deadline;
    runtime_sc_t eligible;
    runtime_sc_t virtual;
    runtime_sc_t ulimit;

    uint64_t cumul;
    uint64_t total;

    uint64_t cl_e;
    uint64_t cl_d;
    uint64_t cl_vt;
    uint64_t cl_vtadj;
    uint64_t cl_myf;
    uint64_t cl_f;                 // KEEP THIS - used in code
    /* REMOVE THIS - never used */
    // uint64_t cl_cfmin;          // DELETE THIS LINE

    uint32_t vtperiod;
    uint32_t parentperiod;

    bool active;
    uint64_t last_time;

    /* ADD THESE FIELDS for tracking active children */
    struct hfsc_class *active_children[MAX_CHILDREN];
    int num_active_children;
} hfsc_class_t;



/* ================ ADD THIS FUNCTION ================ */
static void compute_cl_f(hfsc_class_t *cl) {
    if (cl->parent == NULL) {
        /* Root class: cl_f = cl_myf */
        cl->cl_f = cl->cl_myf;
        return;
    }

    /* Find minimum cl_f among active siblings */
    uint64_t min_sibling_f = UINT64_MAX;
    bool found_sibling = false;

    for (int i = 0; i < cl->parent->num_children; i++) {
        hfsc_class_t *sibling = cl->parent->children[i];
        if (sibling != cl && sibling->active) {
            if (sibling->cl_f < min_sibling_f) {
                min_sibling_f = sibling->cl_f;
                found_sibling = true;
            }
        }
    }

    if (!found_sibling) {
        /* No active siblings */
        cl->cl_f = cl->cl_myf;
    } else {
        /* cl_f = max(cl_myf, min_sibling_f) */
        cl->cl_f = (cl->cl_myf > min_sibling_f) ? cl->cl_myf : min_sibling_f;
    }
}



static void hfsc_activate(hfsc_class_t *cl, uint64_t now) {
    if (cl->active) return;
```

```
cl->active = true;
cl->last_time = now;

/* ADD: Initialize active children tracking */
cl->num_active_children = 0;

/* ADD: Add to parent's active children list */
if (cl->parent) {
    cl->parent->active_children[cl->parent->num_active_children++] = cl;
}

double now_sec = cycles_to_sec(now);

// Real-time
if (cl->rsc.m1 > 0 || cl->rsc.m2 > 0) {
    init_runtime_curve(&cl->deadline, now_sec, cl->cumul,
                       cl->rsc.m1, cl->rsc.m2, cl->rsc.d);
    cl->eligible = cl->deadline;

    if (cl->rsc.m1 <= cl->rsc.m2) {
        cl->eligible.dx = 0;
        cl->eligible.dy = 0;
        cl->eligible.sm1 = bytes_per_sec_to_per_cycle(cl->rsc.m2);
        cl->eligible.sm2 = cl->eligible.sm1;
    }
    uint32_t next_len = peek_next_len(cl->q);
    cl->cl_e = (uint64_t)(rtsc_y2x(&cl->eligible, cl->cumul) * rte_get_tsc_hz());
    cl->cl_d = (uint64_t)(rtsc_y2x(&cl->deadline, cl->cumul + next_len) * rte_get_tsc_hz());
}

// Link-sharing
if (cl->fsc.m1 > 0 || cl->fsc.m2 > 0) {
    init_runtime_curve(&cl->virtual, now_sec, cl->total,
                      cl->fsc.m1, cl->fsc.m2, cl->fsc.d);
    cl->cl_vt = (uint64_t)(rtsc_y2x(&cl->virtual, cl->total) * rte_get_tsc_hz());
}

// Upper limit
if (cl->usc.m1 > 0 || cl->usc.m2 > 0) {
    init_runtime_curve(&cl->ulimit, now_sec, cl->total,
                      cl->usc.m1, cl->usc.m2, cl->usc.d);
    cl->cl_myf = (uint64_t)(rtsc_y2x(&cl->ulimit, cl->total) * rte_get_tsc_hz());
} else {
    cl->cl_myf = UINT64_MAX;
}

/* ADD: Compute cl_f */
compute_cl_f(cl);

cl->vtperiod++;
if (cl->parent) cl->parentperiod = cl->parent->vtperiod;
```

```
    if (cl->parent) hfsc_activate(cl->parent, now);
}



/* ================ ADD THIS FUNCTION ================ */
static void hfsc_deactivate(hfsc_class_t *cl) {
    if (!cl->active) return;

    /* Remove from parent's active children list */
    if (cl->parent) {
        for (int i = 0; i < cl->parent->num_active_children; i++) {
            if (cl->parent->active_children[i] == cl) {
                /* Shift remaining elements */
                for (int j = i; j < cl->parent->num_active_children - 1; j++) {
                    cl->parent->active_children[j] = cl->parent->active_children[j + 1];
                }
                cl->parent->num_active_children--;
                break;
            }
        }

        /* Recompute cl_f for siblings */
        for (int i = 0; i < cl->parent->num_children; i++) {
            hfsc_class_t *sibling = cl->parent->children[i];
            if (sibling->active) {
                compute_cl_f(sibling);
            }
        }
    }

    cl->active = false;
    cl->num_active_children = 0;  /* Clear active children list */
}



struct rte_mbuf *hfsc_packet_out(void) {
    uint64_t now = now_cycles();
    if (!root->active) return NULL;

    hfsc_class_t *cl = hfsc_rt_select(now);
    bool is_realtime = (cl != NULL);

    if (!cl)
        cl = hfsc_ls_select(root, now);

    if (!cl || !cl->is_leaf) return NULL;

    struct rte_mbuf *m;
    if (rte_ring_dequeue(cl->q, (void **)&m) < 0)
        return NULL;
```

```
uint32_t len = rte_pktmbuf_pkt_len(m);


cl->total += len;

if (is_realtime)

    cl->cumul += len;


double now_sec = cycles_to_sec(now);


rtsc_min(&cl->virtual, now_sec, cl->total,

        bytes_per_sec_to_per_cycle(cl->fsc.m1),

        bytes_per_sec_to_per_cycle(cl->fsc.m2), 0);

cl->cl_vt = (uint64_t)(rtsc_y2x(&cl->virtual, cl->total) * rte_get_tsc_hz());


if (cl->usc.m1 > 0 || cl->usc.m2 > 0) {

    rtsc_min(&cl->ulimit, now_sec, cl->total,

            bytes_per_sec_to_per_cycle(cl->usc.m1),

            bytes_per_sec_to_per_cycle(cl->usc.m2), 0);

    cl->cl_myf = (uint64_t)(rtsc_y2x(&cl->ulimit, cl->total) * rte_get_tsc_hz());

}


if (cl->rsc.m1 > 0 || cl->rsc.m2 > 0) {

    uint32_t next_len = peek_next_len(cl->q);

    rtsc_min(&cl->deadline, now_sec, cl->cumul,

            bytes_per_sec_to_per_cycle(cl->rsc.m1),

            bytes_per_sec_to_per_cycle(cl->rsc.m2),
            (double)cl->rsc.d / 1000000.0);

    cl->eligible = cl->deadline;
    if (cl->rsc.m1 <= cl->rsc.m2) {
        cl->eligible.dx = 0;
        cl->eligible.dy = 0;
    }

    cl->cl_e = (uint64_t)(rtsc_y2x(&cl->eligible, cl->cumul) * rte_get_tsc_hz());
    cl->cl_d = (uint64_t)(rtsc_y2x(&cl->deadline, cl->cumul + next_len) * rte_get_tsc_hz());
}

/* ADD: Recompute cl_f after service */
compute_cl_f(cl);

/* ADD: Recompute cl_f for siblings */
if (cl->parent) {
    for (int i = 0; i < cl->parent->num_children; i++) {
        hfsc_class_t *sibling = cl->parent->children[i];
        if (sibling->active && sibling != cl) {
            compute_cl_f(sibling);
        }
    }
}

if (rte_ring_empty(cl->q)) {
        /* CHANGE: Use deactivate instead of just setting active=false */
        hfsc_deactivate(cl);
        cl->vtperiod++;
    }

    return m;
}



void hfsc_init(void) {
    root = calloc(1, sizeof(*root));
    /* ADD: Initialize new fields */
    root->num_active_children = 0;

    /* ... rest of your initialization ... */

    /* ADD: Initialize cl_f for all classes */
    root->cl_f = root->cl_myf;
    site1->cl_f = site1->cl_myf;
    site2->cl_f = site2->cl_myf;
    udp1->cl_f = udp1->cl_myf;
    tcp1->cl_f = tcp1->cl_myf;
    udp2->cl_f = udp2->cl_myf;
    tcp2->cl_f = tcp2->cl_myf;
}
```