```c
#define NFIBERS 4

ucontext_t fibers[NFIBERS];
int done[NFIBERS];
int current;

void fiber_yield(void) {
    int prev = current;

    for (int i = 0; i < NFIBERS; ++i) {
        current = (current + 1) % NFIBERS;

        if (!done[current]) {
            if (current != prev)
                swapcontext(&fibers[prev],
                    &fibers[current]);
            break;
        }
    }
}

void fiber_exit(void) {
    done[current] = 1;
    fiber_yield();
}

void fiber_fn(int start, int end) {
    int sum = 0;

    for (int i = start; i < end; ++i) {
        sum += i;
        if (i == (start + end) / 2) {
            printf("sum: %d (yield)\n", sum);
            fiber_yield();
        }
    }

    printf("sum: %d\n", sum);
    fiber_exit();
}

int main(void) {
    for (int i = 0; i < NFIBERS; ++i) {
        // Initialize fiber[i] and allocate stack
        getcontext(&fibers[i]);
        fibers[i].uc_stack.ss_sp = malloc(SIGSTKSZ);
        fibers[i].uc_stack.ss_size = SIGSTKSZ;
        fibers[i].uc_link = NULL;
        makecontext(&fibers[i],
            (void (*)())fiber_fn,
            2, 100 * i, 100 * (i + 1));
    }

    setcontext(&fibers[0]);
}
```

Fibers are a name for lightweight user level threads. `ucontext_t` holds a fiber context.

`makecontext(fiber, fn, argc, ...)` creates a fiber that runs the function `fn`, passing it `argc` arguments.

`swapcontext(prev, next)` suspends the `prev` fiber and activates `next`.

Question 1. Is this code with 4 fibers likely to run faster or slower than single-threaded code?

Question 2. How many context switches between fibers will there be?

Question 3. What is the expected output?

Question 4. Sketch how you would extend this to support multiple cores.