Studi Kasus

1. Multitasking Fibonacci (Scheduler Test)
2.

A. Fungsi & Tujuan Pengujian

Mengukur:

- Kemampuan scheduler menjalankan banyak task secara bergantian.
- Menguji *sys_create_task()*, *sys_yield()*, *sys_time_us()*.
- Perbandingan kecepatan mini-OS vs Windows.

B. Tabel Input Uji

Parameter	Nilai Uji
Jumlah Task	2 Task
Task 1	hitung fib(35)
Task 2	hitung fib(38)
Mode Scheduler	Round Robin (Time Slice 10–50 ms)

C. Program C (bisa dijalankan di Windows VSCode)

```c
#include <stdio.h>
#include <windows.h>
#include <stdint.h>

uint64_t now_us() {
    LARGE_INTEGER f, c;
    QueryPerformanceFrequency(&f);
    QueryPerformanceCounter(&c);
    return (uint64_t)(c.QuadPart * 1000000 / f.QuadPart);
}

long long fib(int n) {
```

```
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}

DWORD WINAPI task_fib(LPVOID arg) {
    int n = (int)(intptr_t)arg;
    uint64_t t0 = now_us();
    long long r = fib(n);
    uint64_t t1 = now_us();
    printf("Task fib(%d) = %lld | time = %llu us\n", n, r, (t1 - t0));
    return 0;
}

int main() {
    HANDLE T1 = CreateThread(NULL, 0, task_fib, (void*)35, 0, NULL);
    HANDLE T2 = CreateThread(NULL, 0, task_fib, (void*)38, 0, NULL);

    WaitForSingleObject(T1, INFINITE);
    WaitForSingleObject(T2, INFINITE);

    return 0;
}
```

D. Ekspektasi output minimal

```
Task fib(35) = 9227465 | time ≈ 500000–650000 us
Task fib(38) = 39088169 | time ≈ 3500000–4500000 us
Scheduler berhasil menjalankan 2 task bergantian.
```

3. Studi Kasus Simulasi I/O 4KB Block

A. Fungsi & Tujuan Pengujian

Studi kasus ini dirancang untuk **mengukur performa operasi I/O blok** pada mini-OS dan membandingkannya dengan baseline di Windows.

Tujuan uji:

1. Menguji kinerja syscall I/O mini-OS

Yaitu fungsi seperti:

- *sys_write_block(block_index, buf, 4096)*
- *sys_read_block(block_index, buf, 4096)*
- *sys_time_us()* untuk pengukuran waktu.

2. Mengukur throughput & latency block device

Meliputi:

- waktu total write 1000 blok
- waktu total read 1000 blok
- latency rata-rata per blok
- throughput MB/s

3. Menguji konsistensi driver block device mini-OS

Dengan input terkontrol (pola data deterministik).

4. Membandingkan Mini-OS dengan Windows

Windows tidak memiliki block device mini-OS, maka dibuat **simulasi block device RAM** yang identik.

B. Parameter Input

Parameter	Nilai Uji	Penjelasan
Ukuran blok	4096 byte	ukuran standar blok storage
Jumlah blok	1000 blok	total data 4,000,000 bytes
Pola data	increment byte (0–255)	memastikan data valid
Mode akses	sequential	block 0 → 999
Simulasi device	array di RAM	identik dengan mini-OS RAM disk

Fungsi uji	write_block, read_block	simulasi syscalls

C. Program C

```c
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <windows.h>

// ================================================
// Timer resolusi tinggi (Windows)
// ================================================
uint64_t now_us() {
    LARGE_INTEGER f, c;
    QueryPerformanceFrequency(&f);
    QueryPerformanceCounter(&c);
    return (uint64_t)(c.QuadPart * 1000000 / f.QuadPart);
}

// ================================================
// Simulasi block device berbasis RAM
// ================================================
#define BLOCK_SIZE 4096
#define BLOCK_COUNT 1000

unsigned char block_device[BLOCK_SIZE * BLOCK_COUNT];

// menulis 1 blok ke device
int write_block(int block, const unsigned char *buf) {
    if (block < 0 || block >= BLOCK_COUNT) return -1;
    memcpy(&block_device[block * BLOCK_SIZE], buf, BLOCK_SIZE);
    return 0;
}

// membaca 1 blok dari device
int read_block(int block, unsigned char *buf) {
    if (block < 0 || block >= BLOCK_COUNT) return -1;
    memcpy(buf, &block_device[block * BLOCK_SIZE], BLOCK_SIZE);
    return 0;
}
```

```
// ================================================
// MAIN
// ================================================
int main() {
    unsigned char buf[BLOCK_SIZE];

    // isi buffer dengan pola data (0-255)
    for (int i = 0; i < BLOCK_SIZE; i++)
        buf[i] = (unsigned char)(i % 256);

    printf("=== Simulasi I/O 4KB Block (Windows Baseline) ===\n");

    // ------------------------------------
    // WRITE TEST
    // ------------------------------------
    uint64_t w0 = now_us();
    for (int block = 0; block < BLOCK_COUNT; block++)
        write_block(block, buf);
    uint64_t w1 = now_us();

    // ------------------------------------
    // READ TEST
    // ------------------------------------
    uint64_t r0 = now_us();
    for (int block = 0; block < BLOCK_COUNT; block++)
        read_block(block, buf);
    uint64_t r1 = now_us();

    // ------------------------------------
    // Hasil
    // ------------------------------------
    uint64_t write_time = w1 - w0;
    uint64_t read_time = r1 - r0;

    double MB_total = (BLOCK_SIZE * BLOCK_COUNT) / (1024.0 * 1024.0);

    double write_throughput = MB_total / (write_time / 1e6);
    double read_throughput  = MB_total / (read_time / 1e6);

    printf("Write time: %llu us | Throughput: %.2f MB/s\n", write_time,
write_throughput);
    printf("Read  time: %llu us | Throughput: %.2f MB/s\n", read_time,
read_throughput);
```

```
    return 0;
}
```

D. Ekspektasi Output Minimal

```
=== Simulasi I/O 4KB Block (Mini-OS) ===
Write time: 120000 us | Throughput: ~33 MB/s
Read  time:  95000 us | Throughput: ~42 MB/s

Latency rata-rata per blok:
Write: ~120 us/blok
Read : ~95 us/blok
```

4.