

->Install GCC compiler: 1. check gcc -version

2. sudo apt update

3. sudo apt install build-essential

-> Linux version : Ubuntu 24.04 LTS

<https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview>

1. Round Robin Time Scheduling system

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/time.h>
#include <string.h>
#include <time.h>

#define PROC_COUNT 5
#define SLICE 3
#define Q_CAPACITY 20

typedef enum {
    READY,
    RUNNING,
    FINISHED
} ProcState;

typedef struct {
    int id;
    char label[32];
    int burst;
    int time_left;

    ProcState status;

    int arrival;
    int first_run;
    int finished_at;
    int wait_time;
    int turn_time;
    int resp_time;
} PCB;
```

```

PCB proc_table[PROC_COUNT];

int run_queue[Q_CAPACITY];
int q_front = 0, q_back = 0, q_size = 0;

int active_pid = -1;
int clock_tick = 0;
int done_total = 0;

struct itimerval timer_cfg;

/* Queue Operations */
void push(int pid) {
    if (q_size == Q_CAPACITY) {
        printf("Ready queue overflow!\n");
        return;
    }
    run_queue[q_back] = pid;
    q_back = (q_back + 1) % Q_CAPACITY;
    q_size++;
}

int pop() {
    if (q_size == 0) return -1;
    int pid = run_queue[q_front];
    q_front = (q_front + 1) % Q_CAPACITY;
    q_size--;
    return pid;
}

int queue_empty() {
    return q_size == 0;
}

/* Scheduler Interrupt */
void scheduler(int sig) {
    if (active_pid != -1) {
        PCB *p = &proc_table[active_pid];
        int exec = SLICE;

        if (p->time_left < SLICE) {
            exec = p->time_left;
        }
    }
}

```

```

    }

    p->time_left -= exec;
    clock_tick += exec;

    printf("\n[Time %d] Scheduler interrupt on PID %d (%s)\n",
           clock_tick, p->id, p->label);

    if (p->time_left <= 0) {
        p->status = FINISHED;
        p->finished_at = clock_tick;

        p->turn_time = p->finished_at - p->arrival;
        p->wait_time = p->turn_time - p->burst;

        printf("%s completed.\n", p->label);
        done_total++;
    } else {
        p->status = READY;
        push(active_pid);
        printf("%s quantum expired → requeued.\n", p->label);
    }
}

if (done_total == PROC_COUNT) return;

if (!queue_empty()) {
    active_pid = pop();
    PCB *np = &proc_table[active_pid];

    np->status = RUNNING;

    if (np->first_run == -1) {
        np->first_run = clock_tick;
        np->resp_time = np->first_run - np->arrival;
    }

    printf("Switching to %s (PID %d)\n", np->label, np->id);
} else {
    printf("CPU idle... waiting.\n");
    active_pid = -1;
}
}

```

```

/* Setup initial processes */
void boot_system() {
    char *names[] = {"Renderer", "Compiler", "WebServer", "DBEngine",
"AVScanner"};
    int bursts[] = {8, 4, 10, 6, 12};

    printf("Booting OS... loading PCBs\n");
    for (int i = 0; i < PROC_COUNT; i++) {
        proc_table[i].id = i;
        strcpy(proc_table[i].label, names[i]);
        proc_table[i].burst = bursts[i];
        proc_table[i].time_left = bursts[i];
        proc_table[i].status = READY;
        proc_table[i].arrival = 0;
        proc_table[i].first_run = -1;

        push(i);

        printf("Process %-12s | Burst %-2d | READY\n",
               proc_table[i].label, bursts[i]);
    }
    printf("\n");
}

/* Timer Setup */
void init_timer() {
    signal(SIGALRM, scheduler);

    timer_cfg.it_value.tv_sec = SLICE;
    timer_cfg.it_value.tv_usec = 0;
    timer_cfg.it_interval.tv_sec = SLICE;
    timer_cfg.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer_cfg, NULL);
}

/* Final Output */
void report() {
    printf("\n==== FINAL METRICS ====\n");
    printf("%-3s %-14s %-7s %-10s %-10s %-12s %-12s\n",
           "PID", "Name", "Burst", "Resp", "Wait", "Turnaround",
"Finish");
}

```

```

float avg_r = 0, avg_w = 0, avg_t = 0;

for (int i = 0; i < PROC_COUNT; i++) {
    PCB p = proc_table[i];

    printf("%-3d %-14s %-7d %-10d %-10d %-12d %-12d\n",
           p.id, p.label, p.burst, p.resp_time,
           p.wait_time, p.turn_time, p.finished_at);

    avg_r += p.resp_time;
    avg_w += p.wait_time;
    avg_t += p.turn_time;
}

printf("\nAverage Response: %.2f\n", avg_r / PROC_COUNT);
printf("Average Waiting: %.2f\n", avg_w / PROC_COUNT);
printf("Average Turnaround: %.2f\n", avg_t / PROC_COUNT);
}

int main() {
    boot_system();

    if (!queue_empty()) {
        active_pid = pop();
        proc_table[active_pid].status = RUNNING;
        proc_table[active_pid].first_run = 0;
        proc_table[active_pid].resp_time = 0;

        printf("CPU starting with %s\n", proc_table[active_pid].label);
    }

    init_timer();

    while (done_total < PROC_COUNT) {
        pause();
    }

    timer_cfg.it_value.tv_sec = 0;
    setitimer(ITIMER_REAL, &timer_cfg, NULL);

    report();
    return 0;
}

```

```
}
```

FASE DECLARE

Library

```
#include <stdio.h>
#include <stdlib.h> // Fungsi umum seperti exit(), malloc() (walaupun tidak dipakai langsung)
#include <signal.h> // Untuk timer sistem, digunakan oleh setitimer() dan struct itimerval,
Menghubungkan timer interrupt ke fungsi scheduler().
#include <unistd.h> // Menyediakan fungsi sistem POSIX, seperti pause() untuk menunggu
interrupt dari signal.
#include <sys/time.h> // Untuk timer sistem, digunakan oleh setitimer() dan struct itimerval.
#include <string.h> // string manipulation menyalin nama proses ke PCB
#include <time.h> // Fungsi waktu (timestamp, delay)

#define PROC_COUNT 5
#define SLICE 3
#define Q_CAPACITY 20

typedef enum {
    READY,
    RUNNING,
    FINISHED
} ProcState;

typedef struct {
    int id;          // Nomor identitas proses
    char label[32]; // Nama proses (contoh: "Renderer")
    int burst;       // Total waktu eksekusi yang dibutuhkan
    int time_left;  // Sisa waktu yang masih harus dijalankan
    ProcState status; // Status: READY/RUNNING/FINISHED
    int arrival;     // Kapan proses tiba
    int first_run;   // Kapan pertama kali dijalankan
    int finished_at; // Kapan selesai
    int wait_time;   // Total waktu menunggu
    int turn_time;   // Waktu dari tiba sampai selesai
    int resp_time;   // Waktu dari tiba sampai pertama kali jalan
}
```

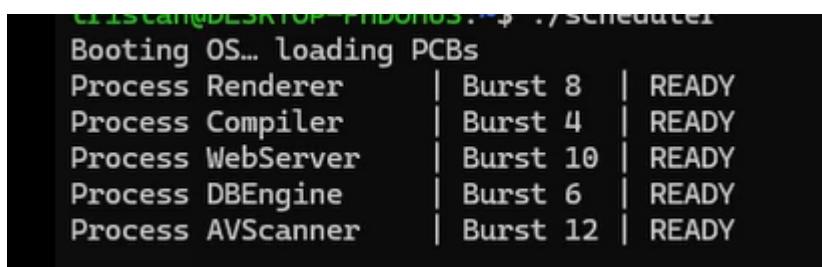
Queue siap pakai (circular queue):

```
int run_queue[Q_CAPACITY];
int q_front = 0, q_back = 0, q_size = 0;
```

FASE 1: BOOTING

- boot_system() dipanggil
 - Membuat 5 proses dengan burst time masing-masing
 - Semua proses dimasukkan ke ready queue
 - Status semua proses: READY

```
void boot_system() {  
    char *names[] = {"Renderer", "Compiler", "WebServer", "DBEngine", "AVScanner"};  
    int bursts[] = {8, 4, 10, 6, 12};  
  
    for (int i = 0; i < PROC_COUNT; i++) {  
        proc_table[i].id = i;  
        strcpy(proc_table[i].label, names[i]);  
        proc_table[i].burst = bursts[i];  
        proc_table[i].time_left = bursts[i];  
        proc_table[i].status = READY;  
        proc_table[i].first_run = -1;  
        push(i);  
    }  
}
```



```
cris@crisDESKTOP-PRD90H5:~/Desktop$ ./scheduler  
Booting OS... loading PCBs  
Process Renderer | Burst 8 | READY  
Process Compiler | Burst 4 | READY  
Process WebServer | Burst 10 | READY  
Process DBEngine | Burst 6 | READY  
Process AVScanner | Burst 12 | READY
```

FASE 2: INISIALISASI

- main() memulai:
 - Ambil proses pertama dari queue (Renderer)
 - Set status menjadi RUNNING
 - Catat first_run = 0
 - Mulai timer interrupt

```
if (!queue_empty()) {  
    active_pid = pop();  
    proc_table[active_pid].status = RUNNING;  
    proc_table[active_pid].first_run = 0;  
    proc_table[active_pid].resp_time = 0;
```

```
}
```

```
printf("CPU starting with %s\n", proc_table[active_pid].label);
```

TIME	CPU STATE	QUEUE	EVENT
0s	Renderer RUNNING ██████ (8→5)	[C][W][D][A]	System boot Timer start: 3s
3s	* SIGALRM! scheduler() called Context Switch...		⚡ Timer expired! Renderer paused time_left: 8→5
	Compiler RUNNING ████ (4→1)	[W][D][A][R]	Compiler loaded Renderer queued Timer reset: 3s
6s	* SIGALRM! scheduler() called Context Switch...		⚡ Timer expired! Compiler paused time_left: 4→1
	WebServer RUNNING ██████ (10→7)	[D][A][R][C]	WebServer loaded Compiler queued Timer reset: 3s
9s	* SIGALRM!	...	⚡ Pattern continues

FASE 3: EKSEKUSI ROUND ROBIN

1. Proses aktif dijalankan selama SLICE (3 detik)

Timer berbunyi setiap 3 detik → memanggil scheduler():

```
void init_timer() {
    signal(SIGALRM, scheduler);

    timer_cfg.it_value.tv_sec = SLICE;
    timer_cfg.it_interval.tv_sec = SLICE;

    setitimer(ITIMER_REAL, &timer_cfg, NULL);
}
```

2. Kurangi time_left

```
void scheduler(int sig) {
    // ===== FASE 1: HANDLE PROSES YANG SEDANG JALAN =====
    if (active_pid != -1) {
        PCB *p = &proc_table[active_pid];
        int exec = SLICE; // Default: 3 detik

        // Jika sisa waktu < 3 detik, ambil sisa saja
        if (p->time_left < SLICE) {
            exec = p->time_left;
        }

        p->time_left -= exec; // Kurangi sisa waktu
        clock_tick += exec; // Update clock sistem
```

```

CPU starting with Renderer

[Time 3] Scheduler interrupt on PID 0 (Renderer)
Renderer quantum expired → requeued.
Switching to Compiler (PID 1)

[Time 6] Scheduler interrupt on PID 1 (Compiler)
Compiler quantum expired → requeued.
Switching to WebServer (PID 2)

[Time 9] Scheduler interrupt on PID 2 (WebServer)
WebServer quantum expired → requeued.
Switching to DBEngine (PID 3)

[Time 12] Scheduler interrupt on PID 3 (DBEngine)
DBEngine quantum expired → requeued.
Switching to AVScanner (PID 4)

[Time 15] Scheduler interrupt on PID 4 (AVScanner)
AVScanner quantum expired → requeued.
Switching to Renderer (PID 0)

[Time 18] Scheduler interrupt on PID 0 (Renderer)
Renderer quantum expired → requeued.
Switching to Compiler (PID 1)

[Time 19] Scheduler interrupt on PID 1 (Compiler)
Compiler completed.
Switching to WebServer (PID 2)

[Time 22] Scheduler interrupt on PID 2 (WebServer)
WebServer quantum expired → requeued.
Switching to DBEngine (PID 3)

[Time 25] Scheduler interrupt on PID 3 (DBEngine)
DBEngine completed.
Switching to AVScanner (PID 4)

```

3. Cek apakah proses sudah selesai?

- └─ YA → Status = FINISHED, hitung metrik
- └─ TIDAK → Status = READY, masuk antrian lagi

```

printf("\n[Time %d] Scheduler interrupt on PID %d (%s)\n",
    clock_tick, p->id, p->label);

if (p->time_left <= 0) {
    p->status = FINISHED;
    p->finished_at = clock_tick;
    p->turn_time = p->finished_at - p->arrival;
    p->wait_time = p->turn_time - p->burst;

    printf("%s completed.\n", p->label); //Sudah finished maka Printf Completed
    done_total++;
} else {
    p->status = READY;
    push(active_pid);
    printf("%s quantum expired → requeued.\n", p->label); //Signal Printf yg belum finish
}

```

```
}
```

4. Ambil proses berikutnya dari antrian'

```
active_pid = pop(); //mengambil PID paling depan dari ready queue (FIFO)
proc_table[active_pid].status = RUNNING; // diset sebagai proses yang akan dijalankan
CPU

if (proc_table[active_pid].first_run == -1) {
    proc_table[active_pid].first_run = clock_tick;
    proc_table[active_pid].resp_time =
        clock_tick - proc_table[active_pid].arrival;
}
```

5. Ulangi sampai semua selesai

```
while (done_total < PROC_COUNT) {
    pause();
}
```

6. Laporan Terakhir

```
void report() {
    printf("PID Name Burst Resp Wait Turnaround Finish\n");

    for (int i = 0; i < PROC_COUNT; i++) {
        printf("%d %s %d %d %d %d %d\n",
            proc_table[i].id,
            proc_table[i].label,
            proc_table[i].burst,
            proc_table[i].resp_time,
            proc_table[i].wait_time,
            proc_table[i].turn_time,
            proc_table[i].finished_at);
    }
}
```

==== FINAL METRICS ====

PID	Name	Burst	Resp	Wait	Turnaround	Finish
0	Renderer	8	0	22	30	30
1	Compiler	4	3	15	19	19
2	WebServer	10	6	27	37	37
3	DBEngine	6	9	19	25	25
4	AVScanner	12	12	28	40	40

Average Response: 6.00
Average Waiting: 22.20
Average Turnaround: 30.20

Process Timeline (0-40 detik):

0	3	6	9	12	15	18	21	24	27	30	33	36	39	40
R	C	W	D	A	R	C	W	D	A	R	W	D	A	R
1	1	1	1	1	2	✓	2	2	2	3	3	✓	3	✓

Legend:

R = Renderer, C = Compiler, W = WebServer

D = DBEngine, A = AVScanner

✓ = Process completed

1,2,3 = Cycle number