Progresif: Jurnal Ilmiah Komputer
Jl. Ahmad Yani, K.M. 33,5 - Kampus STMIK Banjarbaru
Loktabat – Banjarbaru (Tlp. 0511 4782881), e-mail: puslit.stmikbjb@gmail.com
e-ISSN: 2685-0877
p-ISSN: 0216-3284

Analisis Perbandingan Penjadwalan Proses Menggunakan Algoritma *Round Robin* dan *Priority Preemptive*

Wisnu Widiarto[1*], Rakhmadiani Adinda Chaerunnisa[2], Regina Aurellia Tsaqif[3], Salsabila Sekar Nadia[4]
Program Studi Sains Data, Universitas Sebelas Maret, Surakarta, Indonesia
*e-mail *Corresponding Author:* wisnu.widiarto@staff.uns.ac.id

Abstract

An algorithm is a sequence of steps that have been arranged in an orderly manner to solve a particular problem. To plan an algorithm, something called pseudocode has been used, which is similar to a program code. One of the algorithms that is often used is Round Robin, which has divided processor time fairly between various running processes. On the other hand, there is also a scheduling method using the Priority Preemptive method. In this research, the application of algorithms that have been able to increase efficiency and effectiveness in managing various processes has been discussed. The selection of the right scheduling algorithm has depended on the needs and characteristics of the system that has been used. The Round Robin algorithm has been suitable for systems that prioritize fairness and equality of process time. While the Priority Preemptive algorithm has been more suitable for systems that require fast handling with high priority. In Round Robin scheduling, an Average Waiting Time of 14.6 and an Average Turn Around Time of 22.8 have been obtained. While in Priority Preemptive scheduling, an Average Waiting Time of 10.6 and an Average Turn Around Time of 18.8 have been obtained.
Keywords: *Algorithms; Pseudocode; Round robin; Priority scheduling*

Abstrak

Algoritma merupakan serangkaian langkah-langkah yang telah disusun secara teratur untuk menyelesaikan suatu masalah tertentu. Untuk merencanakan suatu algoritma digunakanlah *pseudocode* yang merupakan kode program. Salah satu algoritma yang sering digunakan adalah *Round Robin* yang membagi waktu prosesor secara adil pada berbagai proses yang sedang berjalan. Di sisi lain, terdapat pula metode penjadwalan dengan menggunakan metode *Priority Preemptive*. Pada penelitian ini dibahas mengenai penerapan algoritma yang mampu meningkatkan efisiensi dan efektivitas dalam mengelola berbagai proses. Pemilihan algoritma penjadwalan yang tepat bergantung pada kebutuhan dan karakteristik sistem yang digunakan. Algoritma *Round Robin* cocok digunakan untuk sistem yang mengutamakan kewajaran dan kesetaraan waktu proses. Sedangkan algoritma *Priority Preemptive* lebih cocok digunakan untuk sistem yang membutuhkan penanganan cepat dengan prioritas tinggi. Pada penjadwalan *Round Robin* diperoleh *Average Waiting Time* sebesar 14,6 dan *Average Turn Around Time* sebesar 22,8. Sementara pada penjadwalan *Prioritas Preemptif*, Waktu Tunggu Rata-rata 10,6 dan Waktu Penyelesaian Rata-rata 18,8 telah diperoleh.
Kata kunci*:* *Algoritma; Pseudocode; Round robin; Penjadwalan priority*

1. Pendahuluan

Algoritma merupakan serangkaian instruksi logis yang dipersiapkan secara sistematis untuk menyelesaikan masalah tertentu. Dalam domain komputasi, algoritma memegang peranan penting dalam menangani berbagai masalah pemrograman. Tanpa adanya algoritma yang didesain secara cermat, proses pengembangan perangkat lunak bisa berisiko menghasilkan kode yang tidak akurat, tidak berfungsi, atau bahkan tidak efisien. Dengan menjadi pedoman yang terstruktur, algoritma membantu para pengembang perangkat lunak untuk mengelola proses penyelesaian masalah dengan lebih terorganisir dan efisien, sehingga dapat menghasilkan solusi yang optimal tanpa banyak membuang sumber daya komputer [1].

Sebuah algoritma merupakan kumpulan instruksi yang bertujuan untuk menyelesaikan masalah tertentu. Dengan instruksi-instruksi tersebut, maka dapat diinterpretasikan secara berurutan dari awal hingga akhir [2][3][4]. Sedangkan *pseudocode* menyerupai dengan kode program sebenarnya, meskipun disusun dengan menggunakan bahasa pemrograman tertentu. *Pseudocode* mereplikasi atau meniru kode program yang sebenarnya melalui penggunaan bahasa pemrograman tertentu [3].

Penjadwalan *Round Robin* merupakan penjadwalan yang berjalan dengan cara bersambung-berputar, dengan batas waktu proses yang tertentu. Jika proses berjalan melewati batas waktu, maka proses diberhentikan sementara dan dimasukkan dalam antrian paling belakang. Algoritma ini juga bersifat *starvation-free*, yaitu dapat menjalankan semua proses secara bergantian tanpa memprioritaskan proses tertentu [5]. Sedangkan penjadwalan Priority merupakan penjadwalan berprioritas, yaitu mendahulukan proses yang memiliki prioritas terbesar. Proses yang memiliki nilai prioritas lebih besar akan berada pada urutan lebih awal dibandingkan proses yang memiliki nilai priority lebih rendah. Jika beberapa proses memiliki prioritas yang sama maka pengerjaan dilakukan berdasarkan urutan kedatangan, yaitu pertama datang maka pertama dikerjakan.

Algoritma penjadwalan sangat penting dalam sistem operasi, karena memastikan proses-proses berjalan dengan efisien dan efektif. Dua algoritma penjadwalan yang banyak digunakan adalah *Round Robin* dan *Priority Preemptive*, yang masing-masing memiliki keunggulan dan aplikasi tertentu. Algoritma *Round Robin* membagi waktu CPU secara adil di antara semua proses yang sedang berjalan, sehingga memastikan setiap proses memperoleh kesempatan yang sama dan mengurangi waktu respons. Algoritma *Round Robin* sangat cocok untuk sistem time-sharing yang mengutamakan distribusi waktu CPU yang merata di antara semua proses, serta mencegah dominasi oleh satu proses saja [6]. Di sisi lain, *algoritma Priority Preemptive* memberikan prioritas yang berbeda pada setiap proses, sehingga memungkinkan proses dengan prioritas tinggi untuk menginterupsi proses dengan prioritas lebih rendah. Algoritma *Priority Preemptive* ideal untuk sistem yang membutuhkan penanganan cepat untuk proses-proses yang sangat penting atau kritis.

Dalam penelitian ini, dipaparkan bagaimana penerapan kedua algoritma yang dapat meningkatkan efisiensi dan efektivitas dalam mengelola berbagai proses. Setelah kedua algoritma diterapkan, kemudian dilakukan analisis terhadap kinerja masing-masing algoritma berdasarkan studi kasus tertentu, seperti waktu tunggu (*Waiting Time*), dan waktu penyelesaian (*Turn Around Time*). Pemilihan algoritma penjadwalan yang sesuai sangat tergantung pada kebutuhan dan karakteristik sistem yang digunakan, sehingga bisa dipahami kelebihan dan kekurangan dari algoritma masing-masing.

2. Tinjauan Pustaka

Algoritma *Round Robin*, yang digunakan dalam sistem operasi, merupakan bentuk penjadwalan *real-time*. Algoritma ini beroperasi secara *preemptive*, menetapkan potongan waktu yang disebut quantum waktu (*Time Quantum*) kepada setiap proses yang menunggu eksekusi. Setelah waktu quantum pada proses tersebut habis, proses di-preempted dan dipindahkan ke akhir antrian proses yang menunggu eksekusi. Metode ini sering digunakan dalam sistem operasi *real-time* dan *time-sharing* untuk memastikan penggunaan CPU yang adil antara proses-proses, sehingga meminimalkan waktu respon. Namun, algoritma *Round Robin* konvensional memiliki beberapa kekurangan, termasuk *throughput* yang berkurang, waktu turn around yang berkepanjangan, periode menunggu yang substansial, dan frekuensi pergantian yang tinggi. Salah satu upaya penyelesaian yang disarankan untuk masalah-masalah ini adalah dengan melalui potongan waktu dinamis. Potongan waktu yang dirancang khusus untuk arsitektur *Round Robin* dalam sistem operasi *real-time*, mewakili iterasi pada pendekatan penjadwalan *Round Robin* [7][8].

Dalam penjadwalan Priority, setiap proses diberikan prioritas. Proses yang memiliki prioritas yang sama diurutkan secara FCFS. *Shortest Job First* (SJF) merupakan contoh dari pendekatan ini, dimana prioritas ditentukan oleh panjang burst CPU, dengan memihak pada proses yang lebih pendek. Prioritas dapat ditetapkan secara internal atau eksternal, dengan prioritas internal dihitung berdasarkan parameter yang dapat diukur. Algoritma ini mengurutkan proses dalam antrian berdasarkan tingkat prioritas, dan juga memengaruhi proses mana yang diprioritaskan saat berjalan [9][10]. Dalam algoritma penjadwalan prioritas *preemptive*, CPU menjalankan proses yang baru tiba jika prioritasnya lebih tinggi. Hal ini dilakukan terhadap

proses yang sedang berjalan setelah dibandingkan [8]. Algoritma ini memprioritaskan proses yang memiliki prioritas yang lebih tinggi. Faktor-faktor yang dapat menentukan prioritas tersebut meliputi batasan waktu, kebutuhan memori, dan tingkat pentingnya proses [11].

Waktu yang dibutuhkan oleh setiap proses untuk menyelesaikan eksekusinya disebut sebagai *turn around time* (TAT). Waktu yang dimaksud adalah waktu yang dihabiskan dalam system [12]. *Turn arround time* = waktu eksekusi (*burst time*) + waktu menunggu (*waiting time*). Ketika proses dieksekusi di CPU, beberapa proses memasuki antrian untuk dieksekusi. Namun demikian pada satu waktu tertentu, hanya satu proses yang dapat dieksekusi di CPU. Proses yang tersisa harus menunggu di antrian, siap untuk dieksekusi. Total waktu menunggu yang dihabiskan oleh suatu proses dalam antrian yang tersedia dianggap sebagai *waiting time* proses tersebut [12].

Sistem operasi memberikan waktu kepada proses untuk menyelesaikan tugas, dan sejumlah proses yang selesai dalam waktu yang ditentukan disebut sebagai *throughput*. Semua algoritma penjadwalan berusaha meminimalkan *throughput*, dan algoritma penjadwalan *Round Robin* memiliki *throughput* yang lebih rendah. Jika pergantian konteks tinggi, *throughput* rendah, dan sebaliknya [12]. Pada penelitian [5][13] menerapkan algoritma *Round Robin* dalam mengatur urutan prioritas untuk penjadwalan proses, dengan cara pengurutan eksekusi berdasar waktu. Sedangkan [14][15] menerapkan algoritma *Priority* berdasarkan prioritas yang lebih tinggi, dengan tujuan agar lebih produktif dan efisien dalam penjadwalan proses.

Penelitian [8] mengkaji penerapan algoritma *Round Robin* untuk penjadwalan CPU. Waktu CPU didistribusikan secara merata kepada setiap proses. Dua studi kasus dianalisis guna menilai performa algoritma dengan berbagai durasi waktu quantum. Penelitian ini mengandalkan studi literatur untuk mengevaluasi algoritma *Round Robin*. Algoritma *Round Robin* yang menggunakan waktu quantum lebih panjang menunjukkan kinerja lebih baik dalam hal mengurangi jumlah perpindahan konteks, mengurangi rata- rata waktu tunggu, dan mempercepat rata-rata *Turn Around Time*.

Penelitian [11] membahas tentang pengembangan sistem penjadwalan iklan di Radio Kiss FM dengan menggunakan algoritma penjadwalan Priority. Algoritma penjadwalan *Priority* dipilih karena dinilai efektif dalam menangani penjadwalan yang berfokus pada kepentingan waktu, sedangkan prioritas ditentukan berdasarkan urgensi iklan. Penjadwalan *Priority* memprioritaskan proses dengan prioritas tertinggi. Sistem dapat menjalankan penjadwalan secara *preemptive*. Proses dengan prioritas lebih tinggi dapat menggantikan proses yang sedang berjalan. Algoritma penjadwalan Priority terbukti efektif dalam mengelola penjadwalan iklan, memastikan iklan dengan prioritas tertinggi ditayangkan sesuai dengan ketentuan yang ditetapkan.

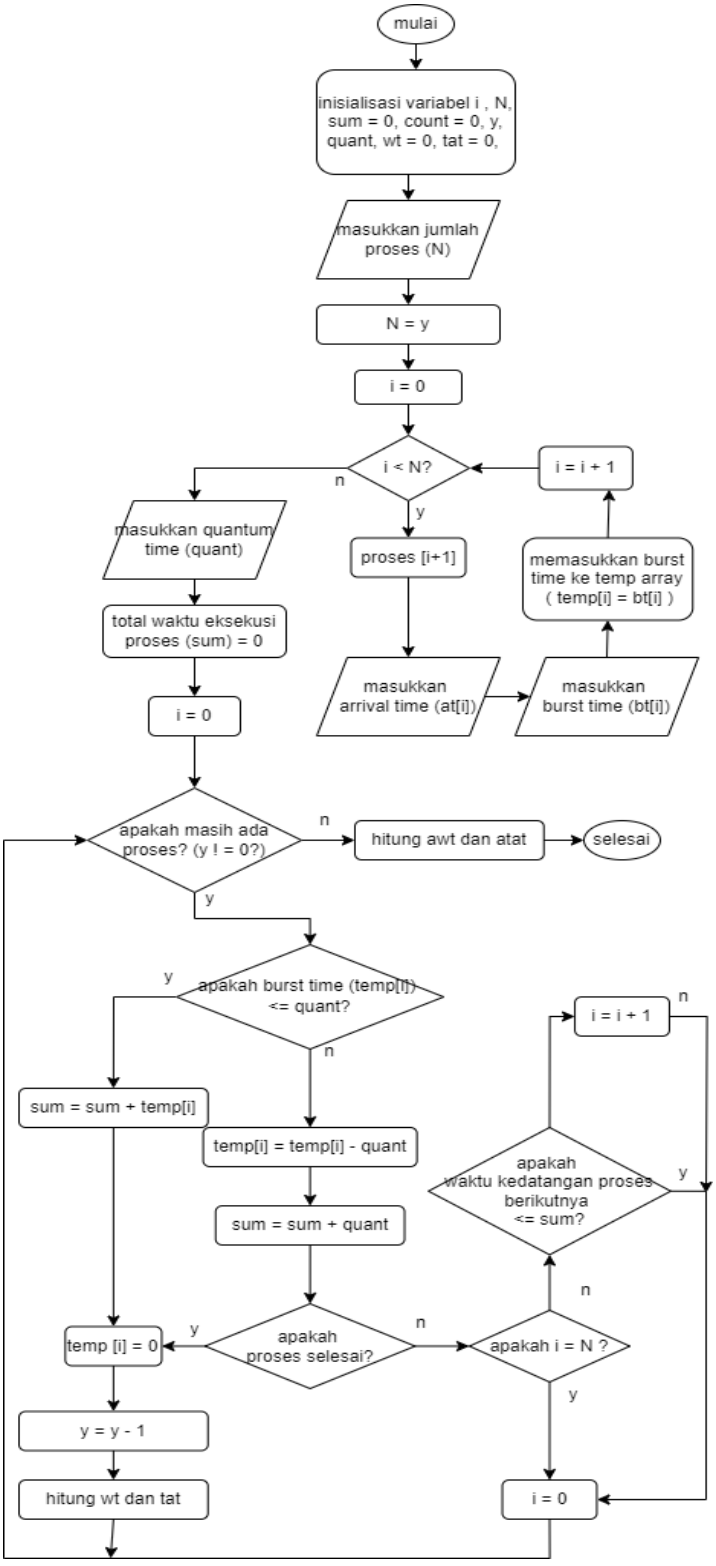
Pada penelitian [16] membahas tentang penjadwalan mata kuliah untuk proses belajar mengajar setiap semester menggunakan penjadwalan Round Robin. Proses ini melibatkan seluruh dosen dan mahasiswa sehingga jadwal mata kuliah yang disusun harus dapat memfasilitasi kepentingan dosen dan mahasiswanya. Penelitian [17] membahas tentang penjadwalan prioritas dengan cara mendahulukan proses yang berprioritas terbesar, nilai prioritas lebih besar berada pada urutan lebih awal. Pada kasus dimana beberapa proses memiliki prioritas yang sama, maka urutan kedatangan menjadi dasar dalam urutan pengerjaan, yaitu pertama datang maka pertama dikerjakan.

Penjadwalan CPU mempunyai aturan dan mekanisme dalam menentukan urutan pekerjaan CPU yang harus dilakukan. Yang melakukan tugas ini adalah sistem operasi dengan menggunakan algoritma penjadwalan [18]. Dengan menggunakan algoritma penjadwalan yang diusulkan, maka algoritma diimplementasikan dan dibandingkan. Algoritma yang diusulkan dibandingkan dengan algoritma lainnya, menghasilkan waktu tunggu rata-rata (AWT) dan waktu penyelesaian rata-rata (ATAT).

3. Metodologi

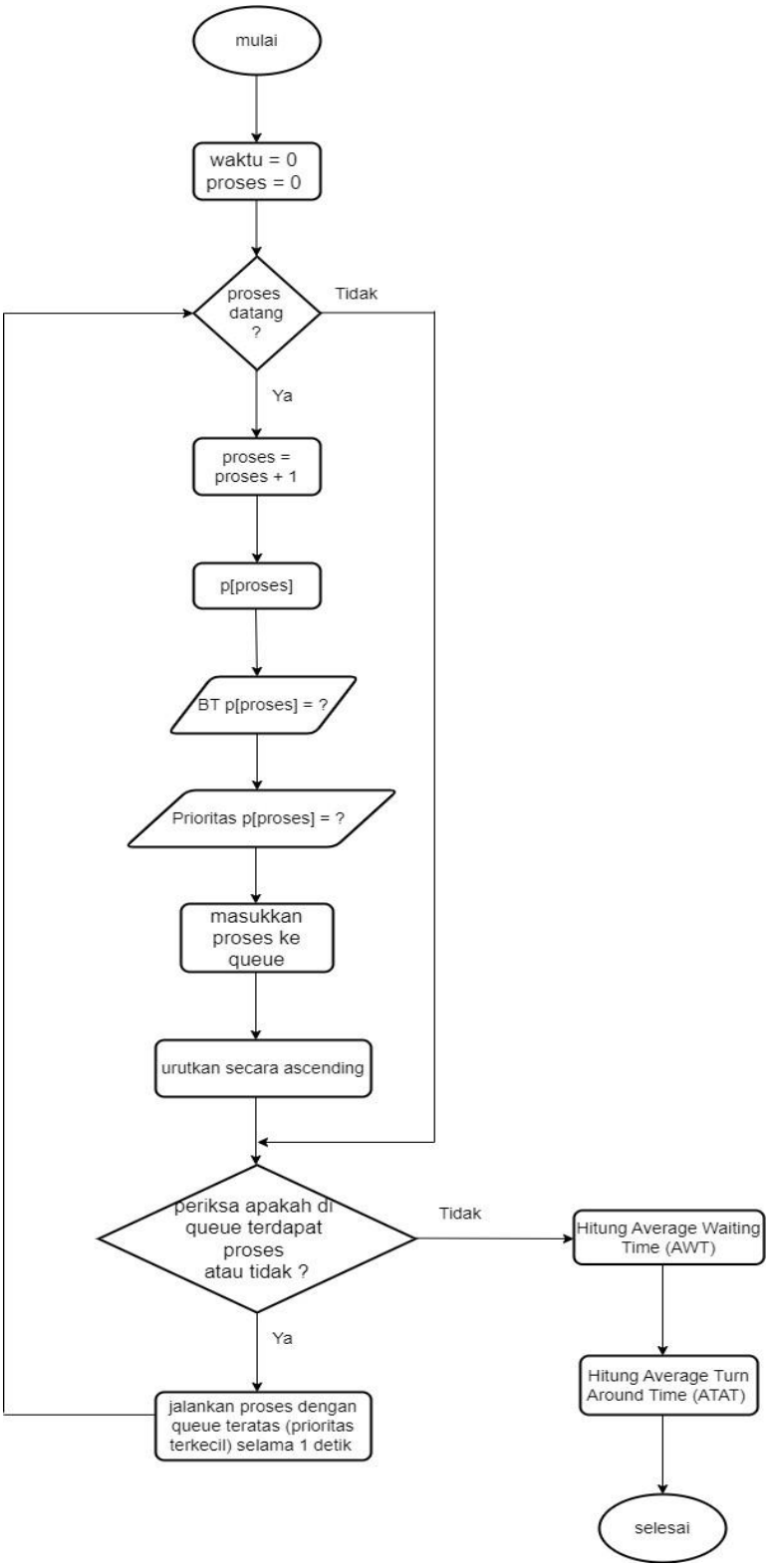
Flowchart adalah sebuah bentuk penggambaran dari langkah-langkah dan urutan suatu prosedur dalam sebuah program [19]. F*lowchart* mampu mempermudah pembaca untuk dapat menyelesaikan permasalahan. Bagan alir (*flowchart*) digunakan sebagai sarana untuk menjelaskan aspek-aspek pada sistem informasi secara jelas dan logis. Dalam pembuatan flowchart membutuhkan simbol-simbol yang memiliki makna masing-masing. Untuk mempermudah menyelesaikan permasalahan dengan algoritma round robin dan algoritma

priority preemptive maka dibuat sebuah flowchart. Pada Gambar 1 merupakan *flowchart round robin* sedangkan pada Gambar 2 untuk melihat *flowchart priority preemptive*.



Gambar 1. *Flowchart* algoritma *Round Robin*

Hal yang perlu dilakukan setelah mendapatkan gambaran umum dan pemahaman alur logika dasar pada sebuah flowchart adalah membuat pseudocode, yang mereplikasi atau meniru kode program yang sebenarnya melalui penggunaan bahasa pemrograman tertentu [3]. Pseudocode algoritma *Round Robin* dan pseudocode untuk *priority preemptive* dapat dilihat pada Tabel 1.



Gambar 2. *Flowchart* algoritma *Priority Preemptive*

Table 1. *Pseudocode* Algoritma *Round Robin* dan Algoritma *Priority Preemptive*

Algoritma Round Robin	Algoritma Priority Preemptive

begin
initialize variables i, N, count = 0, sum = 0, y, quant, wt = 0, tat = 0
input the number of processes (N)
 	
for i = 0 to N-1 do
input burst time (bt[i]) input arrival time (at[i]) temp[i] = bt[i]
end for
 	
input quantum time (quant)
 	
total execution time (sum) = 0 i = 0
 	
while (count != N) do
if (temp[i] <= quant and temp[i] > 0) then
sum = sum + temp[i] temp[i] = 0
count = count + 1
wt = wt + (sum - at[i] - bt[i]) tat = tat + (sum - at[i])
else if (temp[i] > 0) then temp[i] = temp[i] - quant sum = sum + quant
end if
 	
if (i == N-1) then i = 0
else if (at[i+1] <= sum) then i = i + 1
else
i = 0 end if
end while
 	
output average waiting time (awt = wt / N)
output average turnaround time (atat = tat / N)
 	
end
	begin
	time = 0
	process_count = 0
	initialize an empty priority queue (process_queue)
	
	while true do
	if (new process arrives at time) then
	process_count = process_count + 1
	p[process_count] = new process
	BT[p[process_count]] = burst time of new process
	Prioritas[p[process_count]] = priority of new process
	add process to process_queue sort process_queue in ascending order of priority end if
	
	if (process_queue is not empty) then
	current_process = process_queue.peek()
	execute current_process for 1 second
	BT[current_process] = BT[current_process] - 1
	if (BT[current_process] == 0) then
	remove current_process from process_queue
	end if end if
	
	time = time + 1
	
	if (all processes have arrived and process_queue is empty) then
	break end if
	end while
	
	calculate Average Waiting Time (AWT)
	calculate Average Turnaround Time (ATAT)
	end

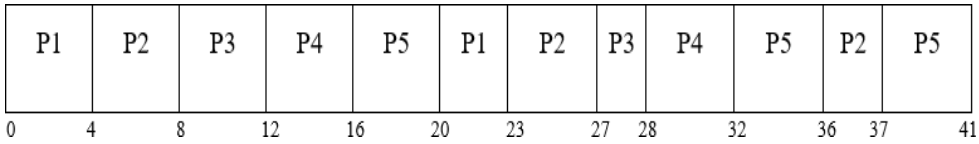
4. Hasil dan Pembahasan

Pada artikel ini, digunakan kasus pada Tabel 2 yang sama untuk diimplementasikan dalam algoritma round robin dan *priority preemptive*. Pada algoritma *Round Robin*, *quantum time* diatur sama untuk semua proses, yaitu 4, sebagaimana Tabel 3. Pada *priority preemptive*, nilai prioritynya dapat dilihat pada Tabel 4. Artikel ini menunjukkan bagaimana perbandingan hasil algoritma *Round Robin* dan *priority preemptive* dengan kasus yang sama menggunakan perhitungan manual dan bahasa pemrograman C. Output yang dihasilkan adalah *waiting time, turn around time*, serta rata rata dari keduanya.

Table 2. Proses beserta rinciannya

Proses	Arrival time	Burst Time
P1	0	7
P2	3	9
P3	5	5
P4	9	8
P5	11	12

4.1. Proses Penjadwalan *Round Robin*



Gambar 3. *Gantt Chart* Algoritma *Round Robin* dengan quantum = 4

Sebagaimana *gantt chart* pada Gambar 3, proses dieksekusi sesuai dengan waktu kedatangannya. Urutan kedatangan proses sama dengan nama proses. P1 datang paling awal, P5 datang paling akhir. Cara dan hasil perhitungan waiting time dan turn around time ditunjukkan sebagaimana pada Tabel 3.

Table 3. Perhitungan *Waiting Time* dan *Turn Around Time Algoritma Round Robin*

Proses	Waiting Time	Turn Around Time
P1	11 - 0 - 7 = 4	4 + 7 = 11
P2	29 - 3 - 9 = 17	17 + 9 = 26
P3	28 - 5 - 5 = 18	18 + 5 = 23
P4	33 - 9 - 8 = 16	16 + 8 = 24
P5	41 - 11 - 12 =18	18 + 12 = 30

Dari Tabel 3, dapat dihitung *average waiting time* dan *average turn around time.* Average *waiting time* adalah (4 + 17 + 18 + 16 + 18) / 5 = 14,6 detik, dan *average turn around time* (11 + 26 + 23 + 24 + 20) / 5 = 22,8 detik. Penjelasan sebelumnya merupakan perhitungan algoritma *round robin*. Sedangkan implementasi algoritma round robin dalam bahasa pemrograman C dibuat sesuai dengan flowchart (Gambar 1) dan *pseudocode* (Tabel 1). Source code yang telah dibuat dapat dilihat pada Gambar 4.

Hal pertama yang muncul saat menjalankan *source code* adalah pengguna diminta untuk memasukkan jumlah proses. Program melakukan looping untuk menerima input arrival time dan *burst time* yang terus berjalan sebanyak jumlah proses. Setelah selesai input proses beserta rinciannya, program meminta input *quantum time*. Program dijalankan dan menghasilkan *output waiting time, around time, beserta averagenya*. Cuplikan hasil program dapat ditunjukkan pada gambar 5.

```c
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

struct P {
    int AT, BT, ST[20], WT, FT, TAT, pos;
};

int quant;

int main() {
    int n, i, j;

    // Taking Input
    printf("enter the total process: ");
    scanf("%d", &n);
    struct P p[n];

    printf("Enter the quantum: ");
    scanf("%d", &quant);

    printf("Enter the process name: \n");
    for(i = 0; i < n; i++)
        scanf("%d", &(p[i].pos));
    printf
    ("Enter the Arrival time of processes: \n");
    for(i = 0; i < n; i++)
```

Gambar 4a. Implementasi *round robin* pada bahasa C

```c
    scanf("%d", &(p[i].AT));

printf
("Enter the Burst time of processes: \n");
for(i = 0; i < n; i++)
    scanf("%d", &(p[i].BT));

// Declaring variables
int c = n, s[n][20];
float time = 0, mini = INT_MAX, b[n], a[n];

// Initializing burst and arrival time arrays
int index = -1;
for(i = 0; i < n; i++) {
    b[i] = p[i].BT;
    a[i] = p[i].AT;
    for(j = 0; j < 20; j++) {
        s[i][j] = -1;
    }
}

int tot_wt = 0, tot_tat = 0;
bool flag = false;

while(c != 0) {
    mini = INT_MAX;
    flag = false;

    for(i = 0; i < n; i++) {
        float p = time + 0.1;
        if(a[i] <= p && mini > a[i] && b[i] > 0) {
            index = i;
            mini = a[i];
            flag = true;
        }
    }

    if(!flag) {
        time++;
        continue;
    }

    // calculating start time
    j = 0;
    while(s[index][j] != -1) {
        j++;
    }
    if(s[index][j] == -1) {
        s[index][j] = time;
        p[index].ST[j] = time;
    }

    if(b[index] <= quant) {
        time += b[index];
        b[index] = 0;
    } else {
        time += quant;
        b[index] -= quant;
    }

    if(b[index] > 0) {
        a[index] = time + 0.1;
    }

    // calculating arrival, burst, final times
    if(b[index] == 0) {
        c--;
        p[index].FT = time;
        p[index].WT = p[index].FT -
          p[index].AT - p[index].BT;
        tot_wt += p[index].WT;
        p[index].TAT = p[index].BT +
          p[index].WT;
        tot_tat += p[index].TAT;
    }
}

// Printing output
printf
("Process number\tWait Time\tTurnAround Time\n");
```

Gambar 4b. Implementasi *round robin* pada bahasa C

```
    // Printing output
    printf
    ("Process number\tWait Time\tTurnAround Time\n");

    for(i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\n", p[i].pos,
        p[i].WT, p[i].TAT);
    }

    // Calculating average waiting time
    //and average turnaround time
    double avg_wt = tot_wt / (float)n;
    double avg_tat = tot_tat / (float)n;

    // Printing average wait time
    //and turnaround time
    printf("The average wait time is:
      %lf\n", avg_wt);
    printf("The average TurnAround time is:
      %lf\n", avg_tat);

    return 0;
}
```

Gambar 4c. Implementasi *round robin* pada bahasa C

```
enter the total process: 5
Enter the quantum: 4
Enter the process name:
1 2 3 4 5
Enter the Arrival time of processes:
0 3 5 9 11
Enter the Burst time of processes:
7 9 5 8 12
Process number  Wait Time       TurnAround Time
1               4               11
2               17              26
3               18              23
4               16              24
5               18              30
The average wait time is: 14.600000
The average TurnAround time is: 22.800000
```

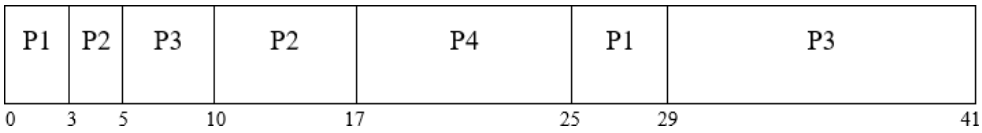
Gambar 5. Cuplikan Hasil Program *Round Robin* dalam Bahasa C

4.2. Proses Penjadwalan *Priority Preemptive*

Table 4. Rincian Proses dengan *Priority*

Proses	Arrival time	Burst Time	Priority
P1	0	7	4
P2	3	9	2
P3	5	5	1
P4	9	8	3
P5	11	12	5

Algoritma priority preemptive mengeksekusi proses dengan nilai priority paling kecil, meskipun proses tersebut datang saat proses dengan nilai priority lebih besar sedang dieksekusi.



Gambar 6. *Gantt Chart* Algoritma *Priority Preemptive*

Berdasarkan gantt chart pada Gambar 6, terlihat bahwa pemrosesan P1 ditunda pada detik ke 3 dikarenakan pada saat itu, datang P2 dengan nilai priority lebih utama. Hal tersebut juga dirasakan oleh P2 yang prosesnya ditunda akibat P3 yang memiliki nilai prioritas lebih kecil muncul. P3 dan P4 tidak pernah disela karena P1 memiliki nilai prioritas paling utama yaitu 1, sedangkan P4 meskipun nilai prioritasnya 3, nilai prioritasnya paling utama diantara sisa proses yang ada. Hasil perhitungan waiting time dan turn around time dapat dilihat pada Tabel 5.

Table 5. Perhitungan *Waiting Time* dan *Turn Around Time* Algoritma *Priority Preemptive*

Proses	Waiting Time	Turn Around Time
P1	29 - 0 - 7 = 22	22 + 7 = 29
P2	17 - 3 - 9 = 5	5 + 9 = 14
P3	10 - 5 - 5 = 0	0 + 5 = 5
P4	25 - 9 - 8 = 8	8 + 8 = 16
P5	41 - 11 - 12 =28	18 + 12 = 30

Dari Tabel 6, dapat dihitung *average waiting time* yaitu (22 + 5 + 0 + 8 + 28) / 5 = 10,6 detik dan *average turn around* time sebesar (29 + 14 + 5 + 16 + 30) / 5 = 18,8 detik. Untuk implementasi algoritma *priority preemptive* pada bahasa pemrograman C dapat dilihat pada Gambar 7.

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

typedef struct {
    int pid;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int priority;
    int waiting_time;
    int turnaround_time;
    int start_time;
} Process;

void sort_by_priority(Process *queue[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (queue[i]->priority > queue[j]->priority)
            {
                Process *temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
}
```

Gambar 7. *Source code priority preemptive* dalam bahasa C

Sama seperti program round robin, program *priority preemptive* saat dijalankan pertama kali meminta jumlah proses dari pengguna. Lalu, program melakukan looping sebanyak jumlah proses untuk menerima input *arrival time, burst time*, dan *priority* dari proses. Setelah itu, proses dijalankan dan program memberi *output waiting time, turn around time*, dan rata rata dari keduanya. Hasil running program dan perhitungan dapat dilihat pada Gambar 8.

```
Enter the number of processes: 5
Enter arrival time for process 1: 0
Enter burst time for process 1: 7
Enter priority for process 1: 4
Enter arrival time for process 2: 3
Enter burst time for process 2: 9
Enter priority for process 2: 2
Enter arrival time for process 3: 5
Enter burst time for process 3: 5
Enter priority for process 3: 1
Enter arrival time for process 4: 9
Enter burst time for process 4: 8
Enter priority for process 4: 3
Enter arrival time for process 5: 11
Enter burst time for process 5: 12
Enter priority for process 5: 5
Process 1: Waiting Time = 22, Turnaround Time = 29
Process 2: Waiting Time = 5, Turnaround Time = 14
Process 3: Waiting Time = 0, Turnaround Time = 5
Process 4: Waiting Time = 8, Turnaround Time = 16
Process 5: Waiting Time = 18, Turnaround Time = 30
Average Waiting Time: 10.60
Average Turnaround Time: 18.80
```

Gambar 8. Hasil program *priority preemptive*

4.3. Perbandingan *Round Robin* dengan *Priority Preemptive*

Table 6. Perbandingan *Round Robin* dengan *Priority Preemptive*

Jenis penjadwalan	Average Waiting Time	Average Turn Around Time
Round Robin	14,6	22,8
Priority Preemptive	10,6	18,8

Pada penjadwalan *Round Robin* diperoleh *Average Waiting Time* dan *Average Turn Around Time* masing-masing sebesar 14,6 dan 22,8. Pada penjadwalan Prioritas *Preemptif*, diperoleh *Average Waiting Time* dan *Average Turn Around Time* masing-masing sebesar 10,6 dan 18,8 (Tabel 6). Dalam kasus ini, algoritma priority preemptive lebih efektif untuk dipakai karena mengolah proses dengan AWT dan ATAT lebih rendah daripada *Round Robin*. Hal ini terjadi karena priority *preemptive* mengurangi *waiting time* untuk program dengan prioritas tinggi, sedang *Round Robin* sering terjadi perpindahan proses sehingga *waiting time* lebih lama.

5. Simpulan

Artikel ini mengevaluasi dan membandingkan dua algoritma penjadwalan *Round Robin* dan *Priority Preemptive.* Kedua algoritma ini memiliki kelebihan dan kekurangan dalam mengelola proses dan alokasi waktu CPU. Algoritma *Round Robin* mendistribusikan waktu CPU secara merata di antara semua proses yang berjalan, memastikan setiap proses mendapatkan kesempatan eksekusi yang adil. Algoritma ini sangat sesuai untuk sistem time-sharing dan real-time, tetapi eksekusi proses yang terlalu lama. Namun, algoritma *Round Robin* dapat menghasilkan throughput yang lebih rendah dan waktu tunggu yang lebih lama jika waktu quantum tidak diatur dengan tepat. Sebaliknya, algoritma *Priority Preemptive* menjadwalkan proses berdasarkan prioritas, memungkinkan proses dengan prioritas lebih tinggi dieksekusi lebih cepat. Algoritma ini efektif dalam situasi di mana beberapa proses memerlukan penanganan segera karena lebih penting. Namun, algoritma ini juga dapat menyebabkan *starvation* bagi proses dengan prioritas rendah jika tidak diatur dengan baik.

Secara keseluruhan, pemilihan algoritma penjadwalan yang tepat sangat bergantung pada kebutuhan dan karakteristik sistem yang digunakan. Algoritma *Round Robin* cocok untuk sistem yang mengutamakan keadilan dan kesetaraan waktu proses, sedangkan algoritma *Priority Preemptive* lebih sesuai untuk sistem yang membutuhkan penanganan cepat bagi proses-proses prioritas tinggi. Memahami kelebihan dan kekurangan masing-masing algoritma membantu dalam mengoptimalkan kinerja sistem operasi dan mencapai tujuan yang diinginkan.

Daftar Referensi

[1] Khairunnisa dan N. Wulan, Perancangan Intelligent Tutoring System Sebagai Upaya Inovatif Pada Pembelajaran Algoritma dan Struktur Data, *ALGORITMA: Jurnal Ilmu Komputer dan Informatika*, Vol. 4, No. 2, pp. 34-42, 2020

[2] S. Akgun dan C. Greenhow, Artificial Intelligence (AI) in Education: Addressing Societal and Ethical Challenges in K-12 Settings, *AI and Ethics*, Vol. 2, No. 3, pp. 431-440, 2022, https://doi.org/10.1007/s43681-021-00096-7

[3] I. Tarsini dan R. Anggraeni, Explore flowchart and pseudocode concepts in algorithms and programming, *Indonesian Journal of Multidisciplinary Science*, Vol. 3, No. 5, pp. 1-8, 2024

[4] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, dan J. Weglarz, *Scheduling Computer and Manufacturing Processes*, 2nd Edition, Kindle Edition, Springer, June 29, 2013

[5] G. Lumbantoruan, 2016, Modifikasi Algoritma Round Robin dengan Dynamic Quantum Time dan Pengurutan Proses Secara Ascending, *Journal Information System Development (ISD)*, Vol. 2, No. 2, pp. 44-55, 2016

[6] A.A. Alsheikhy, R.A. Ammar, dan R.S. Elfouly, An improved dynamic Round Robin scheduling algorithm based on a variant quantum time, 2015 11th International Computer Engineering Conference (ICENCO), Cairo, Egypt, 29-30 December 2015, pp. 98-104, 2015

[7] K.O. Hoger, H.J. Kamal dan F.H. Shalau, Comparative Analysis of The Essential CPU Scheduling Algorithms, *Bulletin of Electrical Engineering and Informatics*, Vol. 10, No. 5, pp. 2742-2750, 2021

[8] T.D. Putra dan R. Purnomo, Simulation of Priority Round-Robin Scheduling Algorithm, *Sinkron : Jurnal dan Penelitian Teknik Informatika*, Vol. 6, No. 4, pp. 2170-2181, 2022

[9] L.H. Karteri dan A. Shehu, Preemptive and Non Preemptive Priority Scheduling, *International Journal of Computer Science & Management Studies*, Vol. 19, No. 1. p. 1, 2015

[10] T.D. Putra, Analysis of Priority Preemptive Scheduling Algorithm: Case Study, *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 11, No. 1, pp. 27-30, 2022

[11] A.M. Simarmata dan M. Harahap, Sistem Penjadwalan Iklan Menggunakan Metode Priority Schedulling pada PT. Kidung Indah Selaras Suara (Radio Kiss FM) untuk Efektivitas dan Efisiensi Produksi Siaran, *Jurnal Penelitian Teknik Informatika Universitas Prima Indonesia (UNPRI) Medan*, Vol. 2, No. 1, pp. 337-347, 2019

[12] Sakshi, C. Sharma, S. Sharma, S. Kautish, S.A.M. Alsallami, E.M. Khalil, dan A.W. Mohamed, A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time, *Alexandria Engineering Journal*, Vol. 61, No. 12, pp. 10527-10538, 2022, https://doi.org/10.1016/j.aej.2022.04.006

[13] C. Effendy dan G. Gusrianty, Application of Round Robin in Scheduling in Web-Based Wedding Organizers, *Journal of Applied Business and Technology*, Vol. 5, No. 2, pp. 90-95, 2024

[14] A.A. Rohmah dan D. Gunawan, Implementasi Algoritma Priority Scheduling Sistem Informasi Pelayanan Administrasi Kependudukan Desa, *Jurnal Informatika: Jurnal pengembangan IT (JPIT)*, Vol. 8, No. 3, pp. 181-187, 2023

[15] S. Eje, L. Ochei, dan C.B. Marcus, Clinical Surgery Scheduling System: A Novel Approach to Enhancing Hospital Efficiency Using Priority Algorithms, *The International Journal of Engineering and Science (IJES)*, Vol. 13, No. 2, pp. 26-53, 2024

[16] A. Wijaya dan Gunawan, Implementasi Algoritma Round Robin Pada Sistem Penjadwalan Mata Kuliah (Studi Kasus: Universitas Muhammadiyah Bengkulu), *JURNAL INFORMATIKA UPGRIS*, Vol. 4, No. 1, pp. 64-71,2018

[17] R. Setyawati dan A.B. Maulachela, Penerapan Algoritma Dynamic Priority Scheduling pada Antrian Pencucian Mobil, *JTIM : Jurnal Teknologi Informasi Dan Multimedia*, Vol. 2, No. 1, pp. 29–35, 2020, https://doi.org/10.35746/jtim.v2i1.85

[18] A. Abdulrahim, S.E. Abdullahi, dan J.B. Salahu, A New Improved Round Robin (NIRR) CPU Scheduling Algorithm, *International Journal of Computer Aplication*, Vol. 90, No. 4, pp. 27-33, 2014

[19] Indrajani, *Pengantar Sistem Basis Data Case Study All In One*, Elex Media Komputindo, Jakarta, ISBN 978-602-02-4903-2, 2014