

Laporan Praktikum

Sistem Operasi

Modul 3



Nama : Asep haryana saputra

NIM : 20230810043

Kelas : TINFC-2023-04

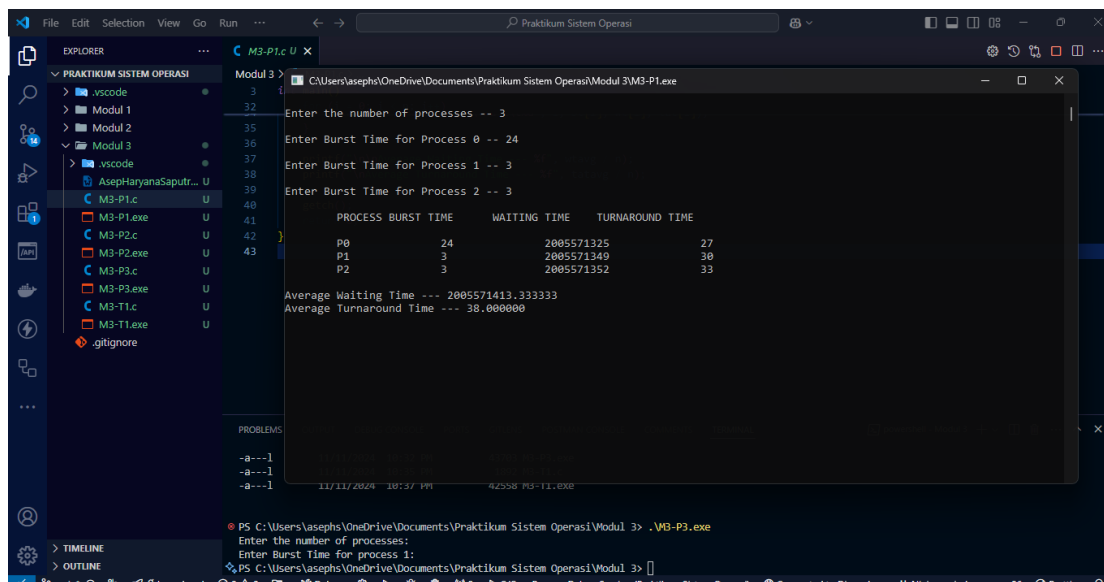
Teknik Informatika
Fakultas Ilmu Komputer
Universitas Kuningan

Pretest

1. Algoritma penjadwalan non-preemptive adalah algoritma yang tidak memungkinkan proses yang sedang berjalan diinterupsi sebelum proses tersebut selesai. Artinya, jika proses A sedang berjalan, maka proses B tidak bisa dijalankan sebelum proses A selesai. Proses yang sedang berjalan akan terus berjalan sampai selesai, meskipun ada proses lain yang memiliki prioritas lebih tinggi.
2. a. First-Come, First-Served (FCFS): Algoritma ini sangat sederhana. Proses yang tiba lebih dulu akan dijalankan lebih dulu. Proses ini akan terus berjalan hingga selesai atau mengalami masalah.
b. Shortest Job First (SJF): Algoritma ini menjalankan proses dengan waktu eksekusi terpendek terlebih dahulu. Hal ini dapat meminimalkan waktu tunggu rata-rata.

Praktikum

1.



```
Modul 3 >
3
Enter the number of processes -- 3
Enter Burst Time for Process 0 -- 24
Enter Burst Time for Process 1 -- 3
Enter Burst Time for Process 2 -- 3

PROCESS BURST TIME WAITING TIME TURNAROUND TIME
P0 24 2005571325 27
P1 3 2005571349 30
P2 3 2005571352 33

Average Waiting Time --- 2005571413.333333
Average Turnaround Time --- 38.000000
```

Kode:

<https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%203/M3-P1.c>

Penjelasan:

Program ini dirancang untuk menghitung dan menampilkan waktu tunggu (waiting time) dan waktu penyelesaian (turnaround time) dari sejumlah proses menggunakan algoritma penjadwalan First-Come, First-Served (FCFS). Pengguna diminta untuk memasukkan jumlah proses yang akan dihitung dan burst time untuk setiap proses. Burst time adalah waktu yang diperlukan oleh suatu proses untuk dieksekusi sepenuhnya di CPU.

Setelah data burst time dimasukkan, program mulai menghitung waktu tunggu untuk setiap proses. Proses pertama memiliki waktu tunggu sebesar nol karena tidak ada proses sebelumnya yang harus dieksekusi. Waktu tunggu proses berikutnya dihitung sebagai penjumlahan dari waktu tunggu proses sebelumnya dengan burst time proses sebelumnya. Dengan perhitungan ini, program memastikan bahwa setiap proses menunggu eksekusi proses-proses sebelumnya sebelum giliran eksekusi tiba.

Selanjutnya, program menghitung turnaround time untuk setiap proses, yang didefinisikan sebagai total waktu dari saat proses masuk ke dalam sistem hingga proses selesai dieksekusi. Turnaround time untuk suatu proses dihitung dengan menambahkan burst time dari proses tersebut ke turnaround time proses sebelumnya. Setelah semua perhitungan selesai, program menampilkan hasil berupa tabel yang mencantumkan proses, burst time, waiting time, dan turnaround time masing-masing proses. Selain itu, program menghitung dan menampilkan rata-rata waktu tunggu dan rata-rata turnaround time untuk semua proses yang diinputkan. Hasil akhir membantu memahami efisiensi penjadwalan proses dalam sistem tersebut.

2.

```

1  You, 9 minutes ago | 1 author (You)
2
3  C:\Users\asephs\OneDrive\Documents\Praktikum Sistem Operasi\Modul 3\M3-P2.exe
4
5  Enter The Number of Processes -- 4
6  Enter burst time for Process 0 -- 6
7  Enter burst time for Process 1 -- 8
8  Enter burst time for Process 2 -- 7
9  Enter burst time for Process 3 -- 3
10
11
12  PROCESS    BURST TIME    WAITING TIME    TURN AROUND TIME
13
14  p0         3             0               3
15  p1         6             3               9
16  p2         7             9              16
17  p3         8            16              24
18
19  Average Waiting Time -- 7.000000
20  Average Turn Around Time -- 13.000000
21
22  PROBLEM
23
24  -a---1
25  -a---1
26  -a---1
27
28  PS C:\Users\asephs\OneDrive\Documents\Praktikum Sistem Operasi\Modul 3> .\M3-P3.exe
29  Enter the number of processes:
30  Enter Burst Time for process 1:
31  PS C:\Users\asephs\OneDrive\Documents\Praktikum Sistem Operasi\Modul 3>

```

Kode:

<https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%203/M3-P2.c>

Penjelasan:

Program ini adalah implementasi dari algoritma penjadwalan **Shortest Job First (SJF)** dalam mode **non-preemptive**. Algoritma ini mengurutkan proses berdasarkan burst time dari yang terpendek ke yang terpanjang untuk meminimalkan rata-rata waktu tunggu. Pada program ini, pengguna diminta untuk memasukkan jumlah proses dan burst time untuk setiap proses.

Proses awal program adalah menginisialisasi array `p[]` untuk menyimpan indeks proses, serta array `bt[]` untuk menyimpan burst time. Program kemudian meminta input burst time dari pengguna dan menyimpannya di array `bt[]`. Selanjutnya, program mengurutkan burst time menggunakan algoritma bubble sort sederhana, di mana burst time yang lebih pendek dipindahkan ke posisi awal array. Proses ini memastikan bahwa proses dengan burst time terpendek dieksekusi lebih dulu, seperti yang diatur dalam algoritma SJF.

Setelah burst time diurutkan, program menghitung waktu tunggu (`wt[]`) untuk setiap proses. Proses pertama memiliki waktu tunggu sebesar nol, sementara proses-proses berikutnya dihitung sebagai penjumlahan waktu tunggu proses sebelumnya dengan burst time proses sebelumnya. Waktu penyelesaian (turnaround time) dihitung dengan menambahkan burst time ke waktu tunggu masing-masing proses.

Program kemudian mencetak tabel yang menampilkan setiap proses dengan burst time, waiting time, dan turnaround time-nya. Program juga menghitung dan menampilkan rata-rata

waiting time dan turnaround time untuk semua proses yang diinputkan. Ini memberi gambaran seberapa efisien proses-proses diatur dalam penjadwalan. Penjadwalan SJF yang digunakan di sini membantu meminimalkan rata-rata waktu tunggu, tetapi hanya cocok digunakan jika burst time proses sudah diketahui sebelumnya.

3.

```

Modul 3 > C M3-P3.c > main()
2  int main(){
24  while(1){
45      for (i = 0; i < n; i++){
46          wa[i] = tat[i] - ct[i];
47          att += tat[i];
48          awt += wa[i];
49      }
50
51      printf("\nThe Average Turnaround Time is: %f", att/n);
52      printf("\nThe Average Waiting Time is: %f", awt/n);
53      printf("\n\n\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");
54
55      for (i = 0; i < n; i++){
56          printf("\t%d\t%d\t%d\t%d\n", i + 1, ct[i], wa[i], tat[i]);
57      }
58      return 0;
59  }

```

The Average Turnaround Time is: 15.000000
The Average Waiting Time is: 5.000000

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	24	6	30
2	3	3	6
3	3	6	9

Kode:

<https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%203/M3-P3.c>

Penjelasan:

Program ini adalah implementasi dari algoritma **Round Robin** untuk penjadwalan proses dalam CPU. Algoritma Round Robin bekerja dengan membagi waktu eksekusi menjadi interval kecil yang disebut **time slice** atau **quantum**. Setiap proses dieksekusi dalam urutan melingkar selama waktu tertentu, dan jika proses belum selesai setelah satu quantum, ia ditempatkan kembali ke antrian untuk dilanjutkan pada giliran berikutnya.

Program dimulai dengan meminta pengguna untuk memasukkan jumlah proses (n) dan burst time untuk masing-masing proses. Burst time tersebut disimpan dalam array bu[], sementara array ct[] digunakan untuk menyimpan burst time asli, yang nantinya diperlukan dalam perhitungan waktu tunggu dan turnaround. Pengguna juga diminta untuk memasukkan ukuran time slice (t), yang menentukan berapa lama setiap proses dijalankan sebelum bergantian dengan proses berikutnya.

Program kemudian menghitung eksekusi proses dalam loop while yang berulang sampai semua burst time (bu[i]) habis. Jika burst time suatu proses lebih besar dari time slice, program mengurangnya dengan t dan menambahkan t ke waktu total (temp). Jika burst time proses lebih kecil atau sama dengan time slice, proses tersebut selesai, dan total waktu eksekusi (temp) ditambahkan ke tat[i] sebagai turnaround time proses tersebut. Loop berlanjut sampai semua proses selesai dijalankan, ditandai dengan variabel done yang diatur ke 1.

Setelah loop utama selesai, program menghitung waktu tunggu (wa[i]) untuk setiap proses sebagai selisih antara turnaround time dan burst time awal (ct[i]). Rata-rata waktu tunggu (awt) dan rata-rata turnaround time (att) dihitung dengan menjumlahkan nilai-nilai individual dan membaginya dengan jumlah proses.

Hasil akhirnya ditampilkan dalam bentuk tabel yang menunjukkan proses, burst time, waiting time, dan turnaround time. Program juga mencetak rata-rata waktu tunggu dan rata-rata turnaround time, yang memberikan gambaran efisiensi algoritma Round Robin dalam penjadwalan proses. Algoritma ini cocok digunakan dalam sistem di mana respon cepat dibutuhkan, seperti dalam **time-sharing systems**, karena semua proses mendapat giliran eksekusi secara adil.

PostTest

Dua jenis algoritma penjadwalan non-preemptive yang umum digunakan:

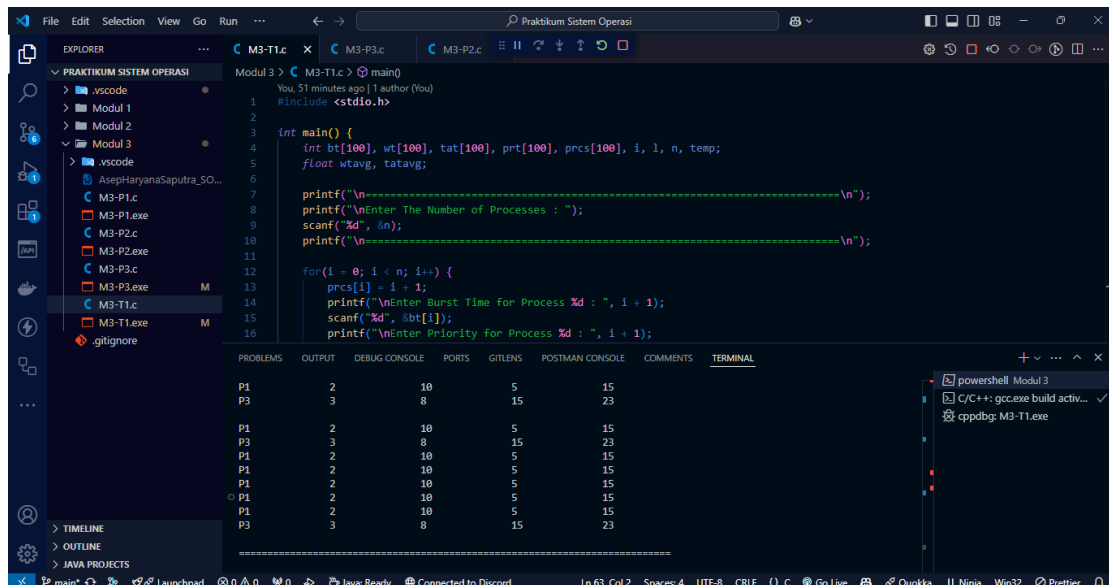
- **First-Come, First-Served (FCFS):** Algoritma ini bekerja berdasarkan urutan kedatangan proses. Proses yang tiba lebih dahulu akan dieksekusi terlebih dahulu hingga selesai sebelum proses lain dimulai. FCFS mudah diimplementasikan tetapi dapat menyebabkan **convoy effect**, di mana proses dengan waktu eksekusi panjang menyebabkan penundaan untuk proses lainnya.
- **Shortest Job First (SJF):** Algoritma ini memilih proses dengan waktu burst paling pendek untuk dieksekusi terlebih dahulu. Hal ini membantu meminimalkan waktu tunggu rata-rata. Namun, penjadwalan ini memerlukan pengetahuan sebelumnya tentang waktu eksekusi setiap proses, yang tidak selalu tersedia.

Kesulitan utama dalam menggunakan algoritma penjadwalan Shortest Job First (SJF):

- **Estimasi waktu eksekusi:** Algoritma SJF membutuhkan informasi tentang waktu burst dari setiap proses sebelum proses tersebut dimulai. Dalam banyak kasus, sulit untuk mengetahui waktu eksekusi proses secara akurat di awal, sehingga membuat penerapan SJF menjadi sulit.
- **Starvation (Kelaparan):** Proses dengan waktu burst yang lebih panjang mungkin terus ditunda jika selalu ada proses dengan waktu burst lebih pendek yang masuk ke dalam antrian, sehingga proses yang lebih panjang dapat mengalami penundaan yang signifikan atau bahkan tidak pernah dieksekusi.

Tugas

1



```
1 // You, 51 minutes ago | I author (You)
2 #include <stdio.h>
3
4 int main() {
5     int bt[100], wt[100], tat[100], prt[100], prcs[100], i, l, n, temp;
6     float wtavg, tatavg;
7
8     printf("\n-----\n");
9     printf("\nEnter The Number of Processes : ");
10    scanf("%d", &n);
11    printf("\n-----\n");
12
13    for(i = 0; i < n; i++) {
14        prcs[i] = i + 1;
15        printf("\nEnter Burst Time for Process %d : ", i + 1);
16        scanf("%d", &bt[i]);
17        printf("\nEnter Priority for Process %d : ", i + 1);
```

	PROBLEMS	OUTPUT	DEBUG CONSOLE	PORTS	GITLENS	POSTMAN CONSOLE	COMMENTS
P1	2	10	5	15			
P3	3	8	15	23			
P1	2	10	5	15			
P3	2	8	15	23			
P1	2	10	5	15			
P1	2	10	5	15			
P1	2	10	5	15			
P1	2	10	5	15			
P1	2	10	5	15			
P3	3	8	15	23			

Kode:

<https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%203/M3-T1.c>

Penjelasan:

Program ini adalah implementasi algoritma penjadwalan berbasis **prioritas non-preemptive**, di mana proses dengan prioritas tertinggi (nilai prioritas terendah) dieksekusi lebih dulu. Prioritas menentukan urutan eksekusi proses; semakin kecil angka prioritas, semakin tinggi prioritasnya.

Program dimulai dengan meminta pengguna untuk memasukkan jumlah proses (n) dan kemudian meminta input burst time serta prioritas untuk setiap proses. Setiap proses diberi label (prcs[i]) yang menunjukkan urutannya. Data burst time disimpan dalam array bt[], dan data prioritas dalam array prt[].

Setelah semua data diinput, program mengurutkan proses berdasarkan prioritas menggunakan algoritma bubble sort sederhana. Proses ini mengatur array prt[] (prioritas), bt[] (burst time), dan prcs[] (nomor proses) secara paralel sehingga proses dengan prioritas tertinggi (angka prioritas terendah) berada di awal array.

Selanjutnya, program menghitung waktu tunggu (wt[]) dan waktu penyelesaian (tat[]). Proses pertama memiliki waktu tunggu 0, sedangkan waktu tunggu untuk proses berikutnya dihitung sebagai jumlah waktu tunggu dari proses sebelumnya ditambah burst time dari proses sebelumnya. Waktu penyelesaian dihitung dengan menambahkan burst time ke waktu tunggu masing-masing proses. Program juga menghitung total waktu tunggu dan total turnaround time untuk semua proses agar dapat menghitung rata-rata waktu tunggu (wtavg) dan rata-rata turnaround time (tatavg).

Hasil akhir ditampilkan dalam bentuk tabel yang menunjukkan proses, prioritas, burst time, waktu tunggu, dan waktu penyelesaian. Program kemudian mencetak rata-rata waktu tunggu dan rata-rata turnaround time. Algoritma penjadwalan berbasis prioritas ini berguna untuk situasi di mana beberapa proses lebih penting daripada yang lain, dan prioritas menentukan urutan eksekusi mereka.

Kesimpulan

Kesimpulan dari laporan praktikum ini menunjukkan bahwa penjadwalan proses dalam sistem operasi merupakan aspek yang sangat penting yang memengaruhi efisiensi dan kinerja sistem. Berbagai algoritma penjadwalan non-preemptive, seperti First-Come, First-Served (FCFS), Shortest Job First (SJF), Round Robin, dan penjadwalan berbasis prioritas, telah diimplementasikan dan masing-masing memiliki karakteristik tersendiri. Algoritma FCFS, meskipun sederhana dan mudah diterapkan, dapat menyebabkan efek konvoi yang menghambat proses lainnya. Di sisi lain, SJF berfokus pada meminimalkan waktu tunggu dengan memprioritaskan proses yang memiliki waktu eksekusi terpendek, tetapi menghadapi tantangan dalam estimasi waktu dan risiko bagi proses yang memiliki waktu eksekusi lebih panjang. Algoritma Round Robin memberikan keadilan dalam eksekusi proses di sistem time-sharing dengan membagi waktu eksekusi dalam interval kecil. Sementara itu, penjadwalan berbasis prioritas memungkinkan proses yang lebih penting untuk dieksekusi lebih dahulu, sesuai dengan urgensi mereka. Melalui praktikum ini, pemahaman mengenai efisiensi penjadwalan proses dapat ditingkatkan, serta tantangan yang dihadapi dalam penerapannya dapat dikenali, sehingga membantu dalam pemilihan algoritma yang tepat berdasarkan kebutuhan spesifik dari sistem yang ada.