**LAPORAN PRAKTIKUM**

**SISTEM OPERASI**

(**DOSEN PENGAMPU : IWAN LESMANA. S.KOM., M.KOM)**

**MODUL 3**



DISUSUN OLEH :

NAMA : MOHAMAD ABAN SY’BANA

NIM : 20230810012

KELAS : TINFC-2023-04

**TEKNIK INFORMATIKA**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS KUNINGAN**

**2024**

**PreTest**

1. Jelaskan apa yang di makasud dengan algortima penjadwalan non-preemptive ?

**Jawab :**

Pada penjadwalan non preemptive ketika sebuah proses diberi jatah waktu pemroses maka proses tersebut akan dieksekusi hingga menyelesaikan eksekusi jobnya yang terakhir. Secara normal pemroses tidak dapat diambil alih proses lain.

1. Sebutkan dan jelaskan dua jenis algoritma penjadwalan non-preemptive yang umum di gunakan ?

**Jawab :**

1. First-Come, First-Served (FCFS) Algoritma FCFS adalah algoritma penjadwalan yang sederhana di mana proses yang datang pertama kali akan dilayani terlebih dahulu. FCFS dianggap adil, tetapi memiliki kelemahan yaitu *convoy effect*, di mana proses yang membutuhkan waktu lama dapat menyebabkan proses lain yang lebih singkat harus menunggu lama, sehingga meningkatkan waktu tunggu rata-rata.
2. Shortest Job First (SJF) Algoritma SJF memilih proses dengan waktu eksekusi atau *burst time* terpendek terlebih dahulu. Algoritma ini mengurangi waktu tunggu rata-rata secara signifikan dibandingkan dengan FCFS, karena proses yang lebih singkat tidak harus menunggu proses yang lebih panjang. Namun, kelemahan SJF adalah bahwa ia membutuhkan estimasi atau pengetahuan awal tentang durasi setiap proses, dan bisa mengakibatkan ketidakadilan terhadap proses yang lebih panjang.

**Praktikum 1**

#include <stdio.h>

#include <conio.h>

int main(){

int bt[20], wt[20], tat[20], i, n;

float wtavg, tatavg;

// Input jumlah proses

printf("\Enter the number of processess -- ");

scanf("%d", &n);

// Input burst time untuk setiap proses

for(i = 0; i < n; i++){

printf("\nEnter burst time for process %d -- ", i);

scanf("%d", &bt[i]);

}

// Inisialisasi waktu tunggu dan turnaround

wt[0] = 0;

wtavg = 0;

tat[0] = bt[0];

tatavg = bt[0];

// Hitung waktu tunggu dan turnaround untuk setiap proses

for(i = 1; i < n; i++){

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg += wt[i];

tatavg += tat[i];

}

// Tampilan hasil

printf("\n\tProcess\tBurst Time\tWaiting Time\tTurnAround Time\n");

for(i = 0; i < n; i++){

printf("\n\tP%d\t\t%d\t\t%d\t\t%d", i, bt[i], wt[i], tat[i]);

}

// Hitung dan tampilkan rata rata waktu tunggu dan turnaround

printf("\n\nAverage Waiting Time -- %f", wtavg/n);

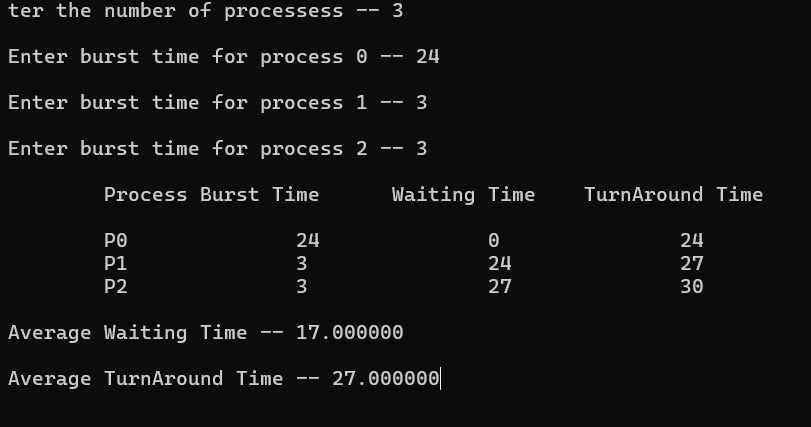
printf("\n\nAverage TurnAround Time -- %f", tatavg/n);

getch();

return 0;

}

**Hasil RUN Praktikum 1**

****

**Penjelasan :**

Gambar di atas menunjukkan hasil penjadwalan CPU menggunakan algoritma First-Come, First-Served (FCFS) untuk tiga proses. Setiap proses memiliki *burst time* masing-masing: P0 = 24, P1 = 3, dan P2 = 3. Pada algoritma FCFS, proses yang datang lebih awal akan dieksekusi lebih dulu. Hal ini menghasilkan *waiting time* (waktu tunggu) untuk setiap proses sebagai berikut: P0 = 0, P1 = 24, dan P2 = 27. Sedangkan, *turnaround time* (waktu penyelesaian) adalah P0 = 24, P1 = 27, dan P2 = 30. Rata-rata waktu tunggu dihitung menjadi 17, dan rata-rata waktu penyelesaian menjadi 27. Algoritma ini kurang optimal dalam kasus seperti ini, di mana proses dengan waktu eksekusi pendek harus menunggu lama jika ada proses dengan waktu eksekusi panjang di awal.

**Praktikum 2**

#include <stdio.h>

#include <conio.h>

int main() {

int p[20], bt[20], wt[20], tat[20], i, k, n, temp; // Deklarasi array dan variabel

float wtavg, tatavg; // Variabel untuk rata-rata waiting time dan turnaround time

// Meminta user untuk memasukkan jumlah proses

printf("\nEnter the number of processes -- ");

scanf("%d", &n);

// Loop untuk memasukkan burst time untuk setiap proses

for (i = 0; i < n; i++) {

p[i] = i;

printf("Enter Burst Time for Process %d -- ", i);

scanf("%d", &bt[i]);

}

// Sorting burst time dengan metode bubble sort (ascending) agar sesuai dengan algoritma SJF

for (i = 0; i < n; i++) {

for (k = i + 1; k < n; k++) {

if (bt[i] > bt[k]) { // jika burst time ke-i lebih besar dari burst time ke-k

// Tukar burst time

temp = bt[i];

bt[i] = bt[k];

bt[k] = temp;

// Tukar juga nomor proses agar tetap sesuai dengan burst time

temp = p[i];

p[i] = p[k];

p[k] = temp;

}

}

}

// Inisialisasi waktu tunggu (wt) dan turnaround time (tat) untuk proses pertama

wt[0] = wtavg = 0; // Proses pertama tidak memiliki waktu tunggu

tat[0] = tatavg = bt[0]; // Turnaround time untuk proses pertama sama dengan burst time-nya

// Loop untuk menghitung waktu tunggu dan turnaround time untuk proses lainnya

for (i = 1; i < n; i++) {

wt[i] = wt[i - 1] + bt[i - 1]; // Waktu tunggu dihitung berdasarkan waktu tunggu dan burst time proses sebelumnya

tat[i] = tat[i - 1] + bt[i]; // Total turnaround time

wtavg = wtavg + wt[i]; // Total waktu tunggu untuk rata-rata

tatavg = tatavg + tat[i]; // Total turnaround time untuk rata-rata

}

// Output tabel hasil: menampilkan proses, burst time, waiting time, dan turnaround time

printf("\nPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

for (i = 0; i < n; i++) {

printf("\nP%d\t\t%d\t\t%d\t\t%d", p[i], bt[i], wt[i], tat[i]);

}

// Menampilkan rata-rata waktu tunggu dan turnaround time

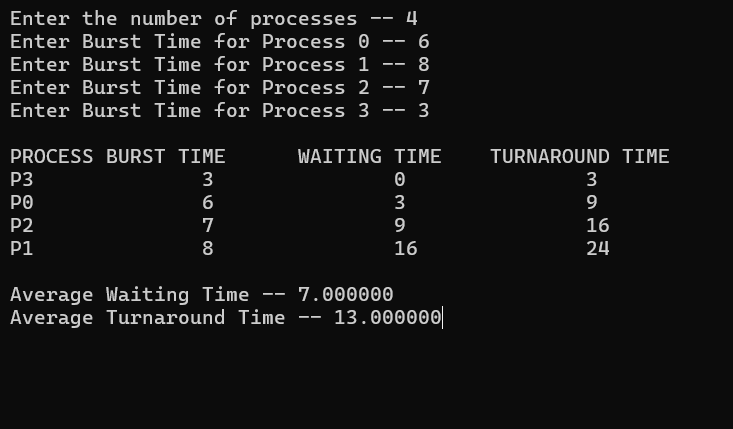
printf("\n\nAverage Waiting Time -- %f", wtavg / n);

printf("\nAverage Turnaround Time -- %f", tatavg / n);

getch();

}

**Hasil RUN Praktikum 2**



Penjelasan :

Gambar tersebut menunjukkan hasil penjadwalan CPU dengan algoritma Shortest Job Next (SJN) untuk empat proses dengan *burst time* sebagai berikut: P0 = 6, P1 = 8, P2 = 7, dan P3 = 3. Pada algoritma SJN, proses dengan waktu eksekusi terpendek akan dieksekusi terlebih dahulu. Berdasarkan urutan eksekusi, *waiting time* (waktu tunggu) dihitung untuk setiap proses: P3 = 0, P0 = 3, P2 = 9, dan P1 = 16. *Turnaround time* (waktu penyelesaian) dihitung dengan menjumlahkan *burst time* dan *waiting time*: P3 = 3, P0 = 9, P2 = 16, dan P1 = 24. Rata-rata waktu tunggu menjadi 7, dan rata-rata waktu penyelesaian menjadi 13. Algoritma SJN menghasilkan waktu tunggu dan waktu penyelesaian yang lebih efisien dengan mendahulukan proses yang memiliki waktu eksekusi terpendek terlebih dahulu.

**Praktikum 3**

#include <stdio.h>

int main() {

int i, j, n, bu[10], wa[10], tat[10], t, ct[10], max;

float awt = 0, att = 0, temp = 0;

// Meminta input jumlah proses

printf("Enter the number of processes: ");

scanf("%d", &n);

// Meminta input burst time untuk setiap proses

for (i = 0; i < n; i++) {

printf("\nEnter Burst Time for process %d: ", i + 1);

scanf("%d", &bu[i]);

ct[i] = bu[i]; // Salin burst time ke array ct untuk nanti

}

// Meminta input time slice

printf("\nEnter the size of time slice: ");

scanf("%d", &t);

// Mencari burst time maksimum

max = bu[0];

for (i = 1; i < n; i++) {

if (max < bu[i]) {

max = bu[i];

}

}

// Proses Round Robin

while (1) {

int done = 1; // Penanda apakah semua proses selesai

for (i = 0; i < n; i++) {

if (bu[i] > 0) { // Jika ada proses yang masih memiliki burst time, loop harus lanjut

done = 0;

if (bu[i] > t) { // Jika burst time lebih besar dari time slice

temp += t; // Tambah waktu dengan time slice

bu[i] -= t; // Kurangi burst time

} else { // Jika burst time lebih kecil atau sama dengan time slice

temp += bu[i]; // Tambah waktu dengan sisa burst time

tat[i] = temp; // Hitung turnaround time

bu[i] = 0; // Proses selesai

}

}

}

if (done == 1) // Jika semua proses selesai, keluar dari loop

break;

}

// Menghitung waiting time dan turnaround time untuk masing-masing proses

for (i = 0; i < n; i++) {

wa[i] = tat[i] - ct[i]; // Waktu tunggu = turnaround time - burst time asli

att += tat[i]; // Total turnaround time

awt += wa[i]; // Total waiting time

}

// Output hasil

printf("\nThe Average Turnaround time is: %f", att / n);

printf("\nThe Average Waiting time is: %f", awt / n);

printf("\n\nPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");

// Tampilkan detail setiap proses

for (i = 0; i < n; i++) {

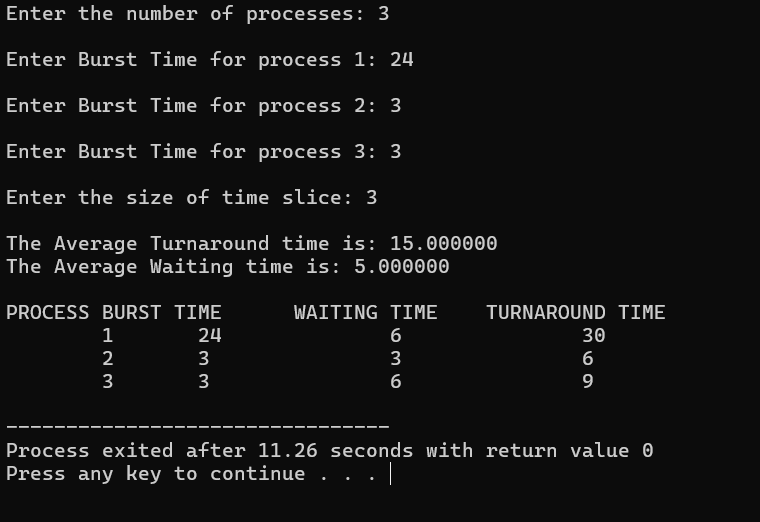
printf("\t%d\t%d\t\t%d\t\t%d\n", i + 1, ct[i], wa[i], tat[i]);

}

return 0;

}

**Hasil RUN Praktikum 3**

****

Penjelasan :

Gambar ini menunjukkan hasil penjadwalan proses menggunakan algoritma *Round Robin* dengan tiga proses yang memiliki *burst time* masing-masing: Proses 1 = 24, Proses 2 = 3, dan Proses 3 = 3, serta *time slice* sebesar 3 satuan waktu. Algoritma ini bekerja dengan memberi setiap proses giliran eksekusi sebesar 3 satuan waktu secara bergantian. Proses pertama mendapat waktu eksekusi awal 3 satuan, diikuti oleh proses kedua dan ketiga, dan kemudian kembali lagi ke proses pertama hingga semua proses selesai. Hasilnya, waktu tunggu (*waiting time*) untuk setiap proses adalah 6, 3, dan 6, sedangkan waktu penyelesaian (*turnaround time*) masing-masing adalah 30, 6, dan 9. Rata-rata waktu tunggu adalah 5 satuan waktu, dan rata-rata waktu penyelesaian adalah 15 satuan waktu. Algoritma Round Robin ini membantu membagi waktu eksekusi secara adil antar proses, sehingga setiap proses mendapat giliran eksekusi secara merata.

**PosTest**

1. Sebutkan dan jelaskan dua jenis algoritma penjadwalan non-preemptive yang umum di gunakan ?

**Jawab :**

1. First-Come, First-Served (FCFS) Algoritma FCFS adalah algoritma penjadwalan yang sederhana di mana proses yang datang pertama kali akan dilayani terlebih dahulu. FCFS dianggap adil, tetapi memiliki kelemahan yaitu *convoy effect*, di mana proses yang membutuhkan waktu lama dapat menyebabkan proses lain yang lebih singkat harus menunggu lama, sehingga meningkatkan waktu tunggu rata-rata.
2. Shortest Job First (SJF) Algoritma SJF memilih proses dengan waktu eksekusi atau *burst time* terpendek terlebih dahulu. Algoritma ini mengurangi waktu tunggu rata-rata secara signifikan dibandingkan dengan FCFS, karena proses yang lebih singkat tidak harus menunggu proses yang lebih panjang. Namun, kelemahan SJF adalah bahwa ia membutuhkan estimasi atau pengetahuan awal tentang durasi setiap proses, dan bisa mengakibatkan ketidakadilan terhadap proses yang lebih panjang.
3. Apa kesulitan utama dalam menggunakan algoritma penjadwalan Shortest Job First (SJF) ?

**Jawab :**

* Estimasi atau Pengetahuan Waktu Eksekusi (Burst Time) yang Akurat.
* Ketidakadilan terhadap Proses yang Lebih Panjang (Starvation).
* Implementasi yang Kompleks dalam Sistem Dinamis.
* Keterbatasan Penggunaan dalam Sistem Non-Preemptive.

**Tugas**

1. Buatlah dalam Bahasa C untuk algoritma priorty, screenshoot kode program dan hasilnya ?

Code

#include <stdio.h>

#include <conio.h>

int main(){

int bt[25], wt[25], tat[25], i, n;

float wtavg, tatavg;

// Input jumlah proses

printf("\Enter the number of processess -- ");

scanf("%d", &n);

// Input burst time untuk setiap proses

for(i = 0; i < n; i++){

printf("\nEnter burst time for process %d -- ", i);

scanf("%d", &bt[i]);

}

// Inisialisasi waktu tunggu dan turnaround

wt[0] = 0;

wtavg = 0;

tat[0] = bt[0];

tatavg = bt[0];

// Hitung waktu tunggu dan turnaround untuk setiap proses

for(i = 1; i < n; i++){

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg += wt[i];

tatavg += tat[i];

}

// Tampilan hasil

printf("\n\tProcess\tBurst Time\tWaiting Time\tTurnAround Time\n");

for(i = 0; i < n; i++){

printf("\n\tP%d\t\t%d\t\t%d\t\t%d", i, bt[i], wt[i], tat[i]);

}

// Hitung dan tampilkan rata rata waktu tunggu dan turnaround

printf("\n\nAverage Waiting Time -- %f", wtavg/n);

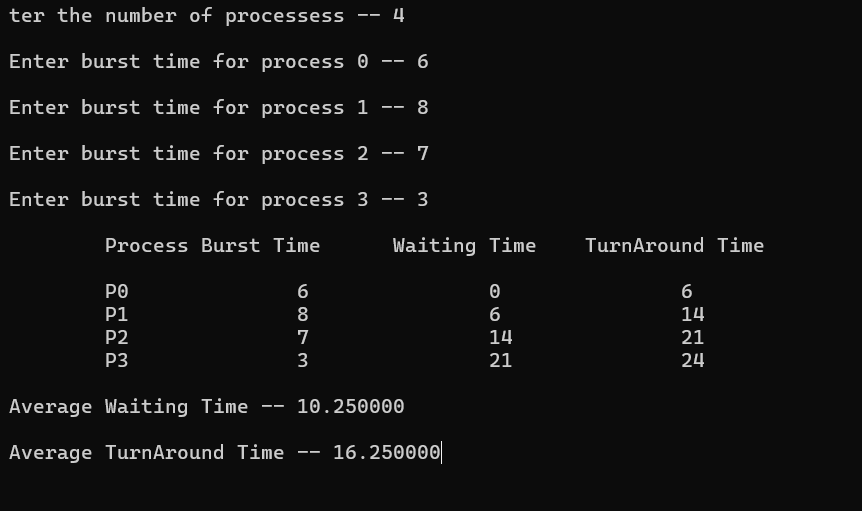
printf("\n\nAverage TurnAround Time -- %f", tatavg/n);

getch();

return 0;

}

**Hasil RUN**

****

Kesimpulan :

Dengan algoritma FCFS, proses yang datang lebih awal akan diproses terlebih dahulu, tanpa mempertimbangkan *burst time*. Hal ini menyebabkan proses dengan waktu eksekusi kecil (P1 dan P2) harus menunggu lama jika ada proses dengan *burst time* besar yang datang lebih awal (seperti P0), sehingga rata-rata waktu tunggu dan waktu penyelesaian (turnaround) menjadi tinggi.