Laporan Praktikum

Sistem Operasi

Dosen pengampu : (*Iwan Lesmana, S.Kom., M.Kom.*)

Modul 3

Nama : Muhammad Rizal Nurfirdaus

NIM : 20230810088

Kelas : TINFC – 2023 – 04

Teknik Informatika

Fakultas Ilmu Komputer

Universitas Kuningan

Pre test

1. Jelaskan apa yang dimaksud dengan algoritma penjadwalan non-preemptive.

   Penjadwalan non-preemptive adalah metode penjadwalan di mana proses yang sedang berjalan tidak dapat dihentikan oleh proses lain hingga proses tersebut selesai. Dalam penjadwalan ini, ketika suatu proses mendapatkan kendali CPU, proses tersebut akan terus berjalan hingga selesai sebelum CPU diberikan kepada proses lainnya. Metode ini memastikan bahwa setiap proses dijalankan sampai selesai tanpa adanya interupsi dari proses lain.

2. Sebutkan dan jelaskan dua jenis algoritma penjadwalan non-preemptive yang umum digunakan.
   - First-Come, First-Served (FCFS)

     Dalam algoritma FCFS, proses yang tiba terlebih dahulu akan dilayani terlebih dahulu. CPU akan dialokasikan kepada proses berdasarkan urutan kedatangan mereka. Proses yang masuk lebih awal akan mendapatkan prioritas untuk dieksekusi hingga selesai, kemudian diikuti oleh proses yang tiba berikutnya.

   - Shortest Job First (SJF)

     Dalam algoritma SJF, proses dengan waktu eksekusi (burst time) terpendek akan dieksekusi terlebih dahulu. Jika ada beberapa proses yang tiba bersamaan, maka proses dengan waktu eksekusi terpendek di antara mereka akan diprioritaskan untuk menggunakan CPU hingga selesai.

Praktikum

A. ALGORITMA PENJADWALAN FCFS CPU

```
#include <stdio.h>
#include <conio.h>

int main(){
   int bt[20], wt[20], tat[20], i, n;
   float wtavg, tatavg;

   // Input jumlah proses
   printf("\nEnter the number of processes -- ");
   scanf("%d", &n);

   // Input Burst Time untuk setiap proses
   for (int i=0; i<n; i++){
```

```
        printf("\nEnter Burst Time for Process %d -- ", i);
        scanf("%d", &bt[i]);
    }

    // Hitung waktu tunggu dan turnaround
    wt[0] = 0;
    wtavg = 0;
    tat[0] = bt[0];
    tatavg = tat[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 += wtavg + wt[i];
        tatavg += tatavg + tat[i];
    }

    // Tampilakn hasil
    printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND
TIME\n");
    for(i = 0; i<n; i++){
        printf("\n\t P%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("\nAverage Waiting Time -- %f", wtavg/n);
    printf("\nAverage Turnaround Time -- %f", tatavg/n);

    getch();
    return 0;
}
```

```
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              0               24
         P1              3               24              27
         P2              3               27              30
Average Waiting Time -- 25.000000
Average Turnaround Time -- 60.000000
```

Analisis :

Program ini menghitung Waiting Time (WT) dan Turnaround Time (TAT) untuk proses dengan algoritma FCFS. Pengguna memasukkan jumlah proses dan waktu burst untuk setiap proses. Program menghitung waktu tunggu dan turnaround, lalu menampilkan hasilnya. Rata-rata WT dan TAT dihitung dan ditampilkan di akhir. Namun, ada kesalahan dalam perhitungan rata-rata karena nilai sebelumnya ditambahkan dua kali, yang perlu diperbaiki.

B. **ALGORITMA PENJADWALAN SJF CPU**

```
#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

   // Minta 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; // Inisialisasi array p dengan nomor proses (P0, P1, P2, dst.)
      printf("Enter Burst Time for Process %d.. ", i);
      scanf("%d", &bt[i]); // Input burst time untuk proses ke-i
   }

   // Sorting burst time dengan metode bubble sort (ascending) agar sesuai dengan
algoritme SJF
   // Sorting ini juga menukar nomor proses agar sesuai dengan burst time
   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
            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] = wt[i] + bt[i]; // Turnaround time = waiting time + burst 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("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIMME\n");
    for(i = 0; i< n; i++)
    printf("\n\t P&d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);

// Output rata-rata waktu tunggu dan turnaround time
printf("\nAverage Waiting Time --%f", wtavg/n);
printf("\nAverage Turnaround Time --%f", tatavg/n);
getch();
}
```

```
Enter the number of processes .. 4
Enter Burst Time for Process 0.. 6
Enter Burst Time for Process 1.. 8
Enter Burst Time for Process 2.. 7
Enter Burst Time for Process 3.. 3

         PROCESS        BURST TIME       WAITING TIME    TURNAROUND TIMME

         P&d             3               3               0
         P&d             0               6               3
         P&d             2               7               9
         P&d             1               8               16
Average Waiting Time --7.000000
Average Turnaround Time --13.000000
```

Analisis :

Program ini menghitung **Waiting Time** dan **Turnaround Time** menggunakan **Shortest Job First (SJF)** dengan cara mengurutkan proses berdasarkan **Burst Time**. Program menghitung rata-rata kedua waktu tersebut dan menampilkan hasil dalam bentuk tabel.

C. ALGORITMA PENJADWALAN ROUND ROBIN CPU

```
#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;
}
```

```
odul3\" ; if ($?) { gcc ROUND_ROBIN.c -o ROUND_ROBIN } ; if ($?) { .\ROUND_ROBIN }
Masukkan jumlah proses -- 3

Masukkan Burst Time untuk proses 1 -- 24

Masukkan Burst Time untuk proses 2 -- 3

Masukkan Burst Time untuk proses 3 -- 3

Masukkan ukuran potongan waktu (time slice)-3

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
```

Analisis :

Program ini mengimplementasikan algoritma **Round Robin** untuk menghitung **Waiting Time (WT)** dan **Turnaround Time (TAT)** dari serangkaian proses dengan menggunakan

time slice yang ditentukan. Proses dijalankan bergiliran dan waktu eksekusi dihitung sesuai dengan **burst time** yang tersisa. Program kemudian menghitung dan menampilkan rata-rata **TAT** dan **WT**, serta menampilkan tabel rinci untuk setiap proses.

Post Test

1. Sebutkan dan jelaskan dua jenis algoritma penjadwalan non-preemptive yang umum digunakan.
    - First-Come, First-Served (FCFS)

      Dalam algoritma FCFS, proses yang tiba terlebih dahulu akan dilayani terlebih dahulu. CPU akan dialokasikan kepada proses berdasarkan urutan kedatangan mereka. Proses yang masuk lebih awal akan mendapatkan prioritas untuk dieksekusi hingga selesai, kemudian diikuti oleh proses yang tiba berikutnya.

    - Shortest Job First (SJF)

      Dalam algoritma SJF, proses dengan waktu eksekusi (burst time) terpendek akan dieksekusi terlebih dahulu. Jika ada beberapa proses yang tiba bersamaan, maka proses dengan waktu eksekusi terpendek di antara mereka akan diprioritaskan untuk menggunakan CPU hingga selesai.
2. Apa kesulitan utama dalam menggunakan algoritma penjadwalan Shortest Job First [SJF]?
   Kesulitan utama dalam algoritma SJF adalah menentukan waktu eksekusi (burst time) dari setiap proses sebelum proses tersebut dieksekusi. Dalam banyak kasus, burst time tidak dapat diprediksi dengan akurat, sehingga penerapan algoritma SJF bisa sulit dilakukan. Hal ini juga dapat menyebabkan starvation pada proses dengan burst time yang lebih panjang.

Tugas

1. Buatlah dalam Bahasa C untuk algoritma priorty, screen shoot kode program dan hasilnya

Priorty.c

```c
#include <stdio.h>

struct Process {
    int id;
    int burst_time;
    int priority;
};

void priorityScheduling(struct Process proc[], int n) {
    // Short proses dari priorty
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (proc[i].priority > proc[j].priority) {
                struct Process temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }

    printf("Process ID\tBurst Time\tPriority\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\n", proc[i].id, proc[i].burst_time, proc[i].priority);
    }
}

int main() {
    struct Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};
    int n = sizeof(proc) / sizeof(proc[0]);

    printf("Penjadwalan Prioritas (Non-Preemptive):\n");
    priorityScheduling(proc, n);

    return 0;
}
```

Ln 35, Col 2 Spaces: 4 UTF-8 CRLF C Go Live Prettier

```
PS C:\Users\Muhammad Rizal Nur F\Semester 3\Sistem Operasi\Modul3> cd "c:\Users\Muhammad Rizal Nur F\Semester 3\Sistem Operasi\Modul3\" ; if ($?) { gcc Priorty.c -o Priorty } ; if ($?) { .\Priorty }
Penjadwalan Prioritas (Non-Preemptive):
Process ID      Burst Time      Priority
2               5               0
3               8               1
1               10              2
PS C:\Users\Muhammad Rizal Nur F\Semester 3\Sistem Operasi\Modul3>
```

Ln 35, Col 2 Spaces: 4 UTF-8 CRLF C Go Live Prettier

Analisis :

Program ini mengimplementasikan algoritma penjadwalan prioritas non-preemptive, di mana proses disortir berdasarkan prioritas. Proses dengan prioritas lebih rendah (angka prioritas lebih kecil) akan dijalankan lebih dahulu.