

# **Laporan Praktikum**

## **Sistem Operasi**

**Modul 7 & 8**



**Nama : Asep haryana saputra**

**NIM : 20230810043**

**Kelas : TINFC-2023-04**

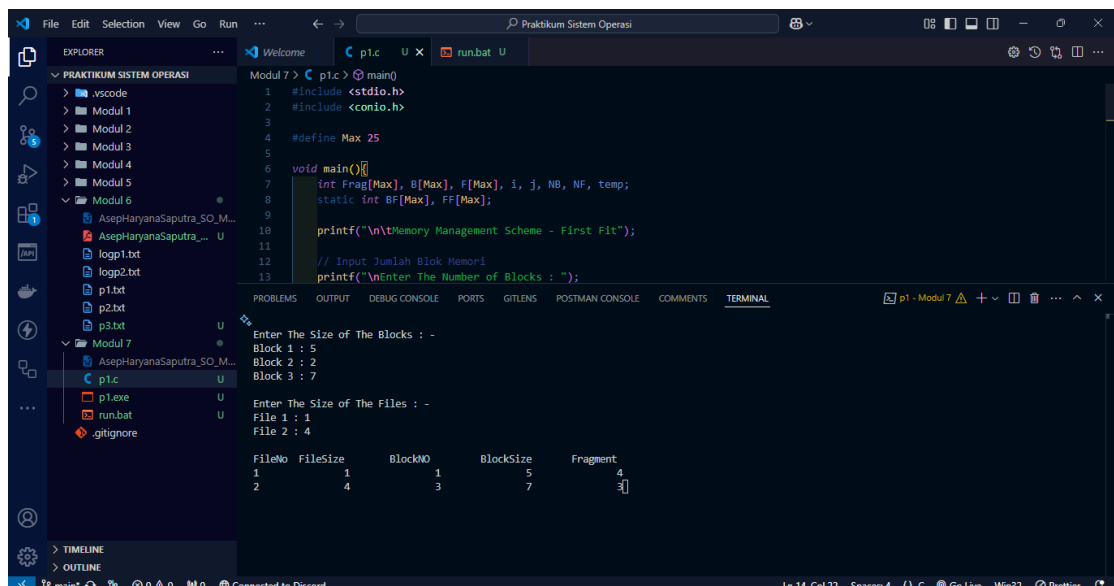
**Teknik Informatika**  
**Fakultas Ilmu Komputer**  
**Universitas Kuningan**

## Pretest

1. Fragmentasi internal terjadi ketika ruang memori yang dialokasikan lebih besar daripada kebutuhan proses, sehingga sisa ruang dalam partisi tidak terpakai tetapi tetap teralokasi, misalnya partisi 100 KB diberikan untuk proses yang hanya membutuhkan 70 KB, menyisakan 30 KB tak terpakai. Sebaliknya, fragmentasi eksternal terjadi ketika terdapat banyak ruang kosong yang tersebar di antara partisi, namun tidak ada satu ruang pun yang cukup besar untuk memenuhi kebutuhan proses, seperti saat proses membutuhkan 250 KB tetapi ruang kosong tersebar di dua partisi terpisah masing-masing 200 KB dan 50 KB.
2. Strategi alokasi memori **First-Fit** memiliki keuntungan karena proses pencarian ruang memori lebih cepat. Algoritma ini langsung mengambil blok memori pertama yang cukup besar untuk memenuhi kebutuhan proses, sehingga efisiensi waktu lebih tinggi dibandingkan **Best-Fit**. Namun, kelemahannya adalah berpotensi menyebabkan fragmentasi eksternal yang lebih besar karena blok-blok kecil tersisa tidak terorganisir secara optimal.

Sebaliknya, **Best-Fit** cenderung menghasilkan penggunaan memori yang lebih efisien karena memilih blok terkecil yang cukup untuk kebutuhan proses, sehingga mengurangi sisa ruang yang tidak terpakai. Namun, kekurangannya adalah waktu pencarian lebih lama karena harus memeriksa semua blok memori yang tersedia untuk menemukan yang terbaik, dan tetap berisiko menghasilkan fragmentasi eksternal jika blok kecil tersisa tersebar di lokasi berbeda.

## Praktikum



```
Modul 7 > C p1.c > main()
1 #include <stdio.h>
2 #include <conio.h>
3
4 #define Max 25
5
6 void main()
7 {
8     int Frag[Max], B[Max], F[Max], i, j, NB, NF, temp;
9     static int BF[Max], FF[Max];
10
11     printf("\n\tMemory Management Scheme - First Fit");
12
13     // Input Jumlah Blok Memori
14     printf("\nEnter The Number of Blocks : ");
15
16     Enter The Size of The Blocks : -
17     Block 1 : 5
18     Block 2 : 2
19     Block 3 : 7
20
21     Enter The Size of The Files : -
22     File 1 : 1
23     File 2 : 4
24
25     FileNo  FileSize  BlockNO  BlockSize  Fragment
26     1       1        1          5          4
27     2       4        3          7          3

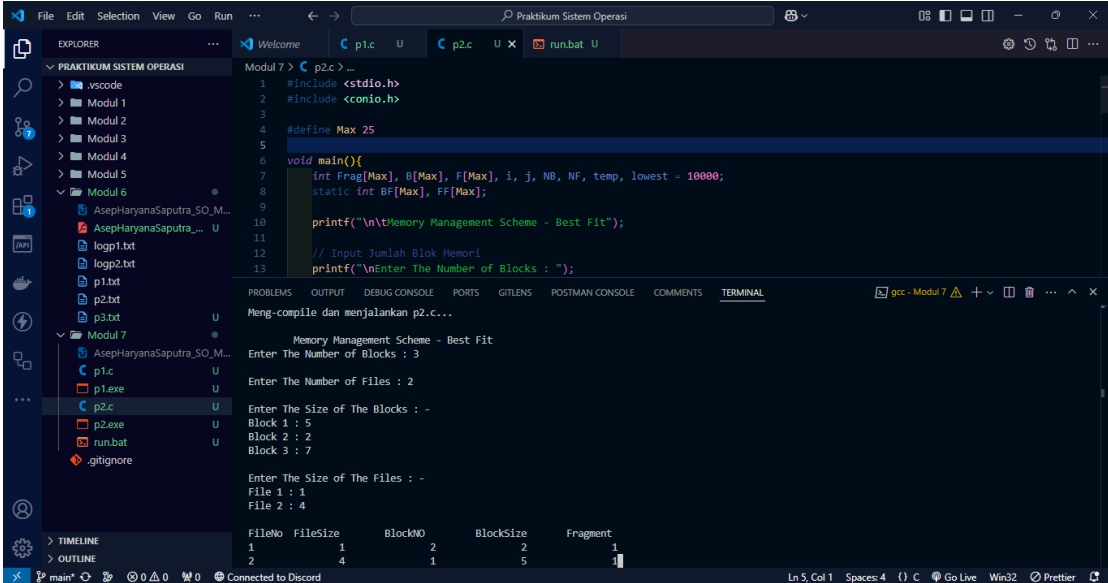
```

1.

Kode: <https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%207/p1.c>

Strategi First-Fit dalam alokasi memori dengan partisi tetap bekerja dengan mencari blok memori pertama yang cukup besar untuk file yang akan disimpan. Misalnya, terdapat tiga blok memori berukuran 5, 2, dan 7. File pertama berukuran 1 ditempatkan pada blok pertama (5), menghasilkan ruang kosong sebesar 4 yang tidak terpakai (fragmentasi internal). File kedua berukuran 4 ditempatkan pada blok ketiga (7), menyisakan ruang kosong sebesar 3. Sementara itu, blok kedua (2) tidak digunakan karena ukurannya tidak mencukupi untuk file mana pun. Strategi ini unggul dalam kecepatan pencarian ruang, tetapi seringkali menghasilkan banyak ruang kosong yang kurang efisien.

2.

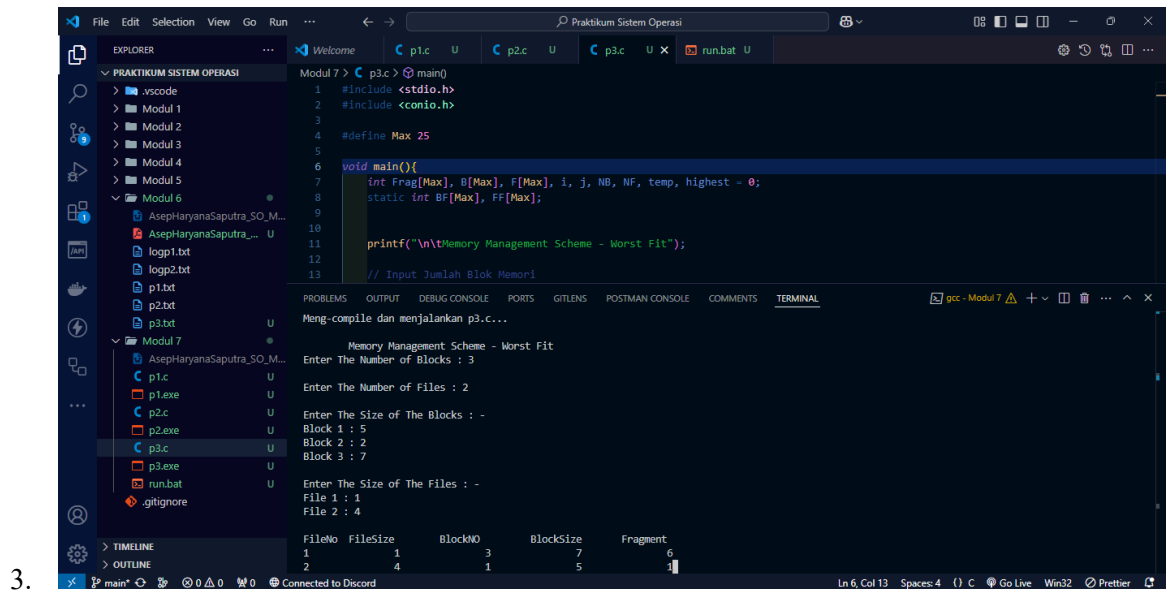


```

1 #include <stdio.h>
2 #include <conio.h>
3
4 #define Max 25
5
6 void main(){
7     int Frag[Max], B[Max], F[Max], i, j, NB, NF, temp, lowest = 10000;
8     static int BF[Max], FF[Max];
9
10    printf("\n\tMemory Management Scheme - Best Fit");
11
12    // Input Jumlah Blok Memori
13    printf("\nEnter The Number of Blocks : ");
14
15    Meng-compile dan menjalankan p2.c...
16
17    Memory Management Scheme - Best Fit
18    Enter The Number of Blocks : 3
19
20    Enter The Number of Files : 2
21
22    Enter The Size of The Blocks : -
23    Block 1 : 5
24    Block 2 : 2
25    Block 3 : 7
26
27    Enter The Size of The Files : -
28    File 1 : 1
29    File 2 : 4
30
31    FileNo  FileSize  BlockNO  BlockSize  Fragment
32    1         1         2         2         1
33    2         4         1         5         1
  
```

Kode: <https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%207/p2.c>

Metode Best-Fit bekerja dengan memilih blok memori terkecil yang cukup untuk menyimpan file. Dalam contoh, File 1 berukuran 1 KB ditempatkan di Blok 2 yang berukuran 2 KB, sehingga menyisakan ruang kosong sebesar 1 KB. File 2 berukuran 4 KB ditempatkan di Blok 1 berukuran 5 KB, dengan sisa ruang kosong 1 KB. Sementara itu, Blok 3 yang berukuran 7 KB tetap kosong karena tidak ada file yang sesuai. Metode ini lebih efektif dalam mengurangi pemborosan memori dibandingkan metode lainnya, meskipun membutuhkan waktu lebih lama untuk menemukan blok yang paling sesuai. Metode Best-Fit bekerja dengan memilih blok memori terkecil yang cukup untuk menyimpan file. Dalam contoh, File 1 berukuran 1 KB ditempatkan di Blok 2 yang berukuran 2 KB, sehingga menyisakan ruang kosong sebesar 1 KB. File 2 berukuran 4 KB ditempatkan di Blok 1 berukuran 5 KB, dengan sisa ruang kosong 1 KB. Sementara itu, Blok 3 yang berukuran 7 KB tetap kosong karena tidak ada file yang sesuai. Metode ini lebih efektif dalam mengurangi pemborosan memori dibandingkan metode lainnya, meskipun membutuhkan waktu lebih lama untuk menemukan blok yang paling sesuai.



Kode: <https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%207/p3.c>

Algoritma Worst-Fit mengelola memori dengan memilih blok memori terbesar untuk menyimpan file, sehingga menghindari ruang kosong yang terlalu kecil. Dalam skenario ini, terdapat tiga blok memori berukuran 5, 2, dan 7, serta dua file berukuran 1 dan 4. File pertama (1 KB) ditempatkan di blok terbesar, yaitu blok 3 (7 KB), menyisakan ruang kosong sebesar 6 KB. File kedua (4 KB) ditempatkan di blok terbesar yang tersisa, yaitu blok 1 (5 KB), menyisakan ruang kosong sebesar 1 KB. Blok 2 (2 KB) tidak digunakan karena ukurannya terlalu kecil untuk file mana pun. Meskipun menyisakan ruang kosong besar, metode ini bertujuan mengurangi fragmentasi eksternal yang terlalu kecil.

## Posttest

1.

Ketiga algoritma alokasi memori—First Fit, Best Fit, dan Worst Fit—menawarkan pendekatan yang berbeda untuk menempatkan file pada blok memori:

- **First Fit:** Menempatkan file pada blok pertama yang cukup besar. Metode ini cepat, tetapi sering meninggalkan ruang kosong yang tidak terpakai, sehingga menghasilkan fragmentasi internal yang signifikan.
- **Best Fit:** Memilih blok dengan sisa ruang terkecil yang cukup untuk file. Meskipun memanfaatkan memori lebih efisien, algoritma ini lebih lambat karena membutuhkan pencarian yang lebih teliti dan dapat meninggalkan fragmentasi kecil yang sulit dimanfaatkan.
- **Worst Fit:** Menggunakan blok dengan ruang terbesar untuk mengurangi fragmentasi eksternal. Metode ini cenderung mencegah terbentuknya ruang kecil yang tidak dapat digunakan, meskipun berisiko meninggalkan ruang kosong besar.

## Tugas

1. Strategi alokasi memori dinamis yang berdekatan seperti *First Fit* dan *Best Fit* cenderung mengalami fragmentasi eksternal.

Pada *First Fit*, file ditempatkan pada blok pertama yang cukup besar untuk menampungnya. Namun, ini sering meninggalkan ruang kosong yang terlalu kecil untuk file berikutnya, menyebabkan terbentuknya banyak ruang kosong kecil di antara blok memori yang tidak dapat digunakan.

*Best Fit*, meskipun lebih efisien dalam penggunaan ruang memori dengan memilih blok terkecil yang cukup, tetap dapat menghasilkan fragmentasi eksternal karena ruang kecil yang tersisa tersebar di berbagai lokasi. Hal ini membuat memori yang tersedia sulit dimanfaatkan untuk file berikutnya.

Sebaliknya, *Worst Fit* yang memilih blok terbesar cenderung mengurangi fragmentasi eksternal, meskipun kadang menyisakan ruang kosong besar yang tidak efisien.

2. Tiga solusi yang dapat diterapkan untuk mengatasi fragmentasi eksternal adalah:
  - **Penggabungan (Compaction):** Memindahkan file di memori untuk menyatukan ruang kosong yang terfragmentasi menjadi satu blok besar, sehingga memori lebih efisien digunakan.
  - **Penyusunan Kembali File (File Reorganization):** Mengatur ulang file secara berkala untuk meminimalkan fragmentasi eksternal dan mengoptimalkan penggunaan ruang memori.
  - **Penggunaan Algoritma Pengalokasian yang Efisien:** Algoritma seperti *Worst Fit* dapat digunakan untuk mengurangi fragmentasi eksternal dengan memilih blok memori terbesar, sehingga menghindari ruang kosong kecil yang tersebar.
3. **Apa itu aturan 50 persen?**

**Jawaban:**

Aturan 50 persen adalah prinsip dalam manajemen memori dinamis yang menyarankan agar blok memori tidak diisi lebih dari 50 persen kapasitasnya.

Setidaknya separuh kapasitas blok memori harus tetap kosong untuk memastikan alokasi memori berikutnya dapat dilakukan dengan lebih efisien. Prinsip ini bertujuan untuk mengurangi fragmentasi eksternal, sehingga memori yang tersedia dapat dimanfaatkan secara optimal.

4. **Apa itu pemadatan?**

**Jawaban:**

Pemadatan (*Compaction*) adalah proses dalam manajemen memori untuk mengatasi fragmentasi eksternal.

Proses ini memindahkan file atau proses di memori untuk menyatukan ruang kosong yang tersebar menjadi satu blok besar yang kontinu. Dengan demikian, ruang memori yang sebelumnya terfragmentasi dapat dimanfaatkan dengan lebih baik.

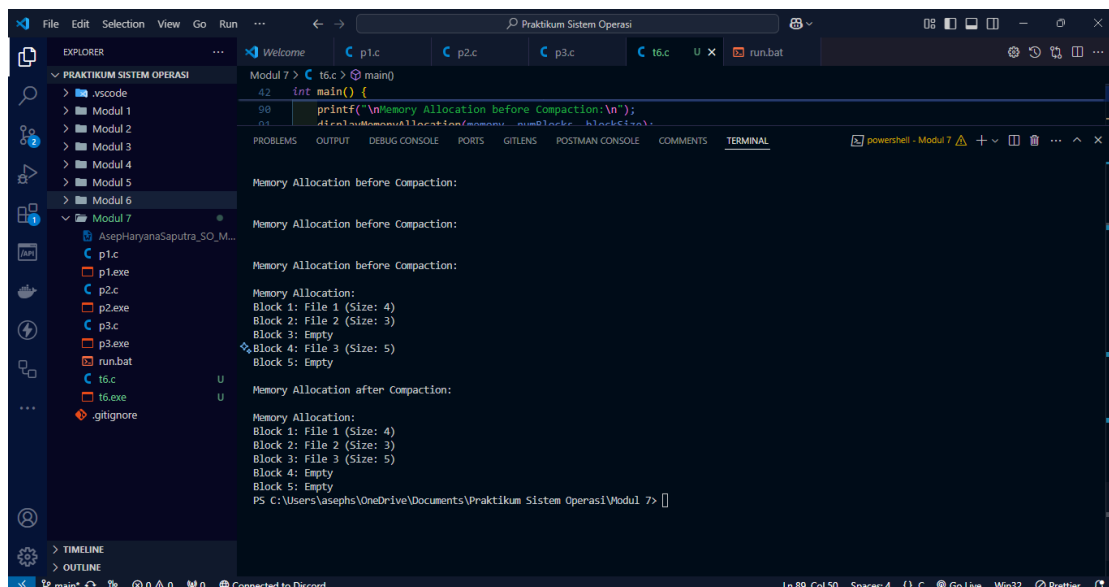
Namun, pemadatan membutuhkan waktu dan sumber daya komputasi yang signifikan, sehingga terlalu sering melakukannya dapat mengganggu kinerja sistem.

5. **Manakah teknik alokasi memori yang paling efisien, paling cocok, dan paling buruk? Mengapa?**

**Jawaban:**

- **Paling Efisien: *Best Fit***  
Teknik ini memilih blok dengan ruang kosong terkecil yang cukup untuk file, sehingga memaksimalkan penggunaan ruang memori dan mengurangi pemborosan. Namun, proses pencarian blok yang sesuai membutuhkan waktu lebih lama.
- **Paling Cocok: *Worst Fit***  
Dengan memilih blok terbesar, *Worst Fit* mengurangi risiko fragmentasi kecil yang tidak terpakai. Meskipun menyisakan ruang kosong yang besar, teknik ini dapat berguna dalam menghindari fragmentasi eksternal.
- **Paling Buruk: *First Fit***  
Teknik ini cepat karena hanya memilih blok pertama yang cukup besar. Namun, sering menyebabkan fragmentasi internal yang besar dan banyak ruang kosong yang tidak efisien, sehingga cenderung menjadi pilihan yang paling buruk.

6.



```

Modul 7 > C t6.c > main()
42 int main() {
98     printf("\nMemory Allocation before Compaction:\n");
99     int i;
100     for (i = 0; i < 5; i++) {
101         printf("Block %d: File %d (Size: %d)\n", i, i, i);
102     }
103     printf("\nMemory Allocation after Compaction:\n");
104     int j;
105     for (j = 0; j < 5; j++) {
106         printf("Block %d: File %d (Size: %d)\n", j, j, j);
107     }
108     return 0;
109 }

Memory Allocation before Compaction:

Memory Allocation before Compaction:

Memory Allocation:
Block 1: File 1 (Size: 4)
Block 2: File 2 (Size: 3)
Block 3: Empty
Block 4: File 3 (Size: 5)
Block 5: Empty

Memory Allocation after Compaction:

Memory Allocation:
Block 1: File 1 (Size: 4)
Block 2: File 2 (Size: 3)
Block 3: File 3 (Size: 5)
Block 4: Empty
Block 5: Empty
PS C:\Users\asephs\OneDrive\Documents\Praktikum Sistem Operasi\Modul 7>
  
```

Kode: <https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%207/t6.c>

Program ini dirancang untuk mengalokasikan memori secara dinamis dengan menggunakan teknik pemadatan. Pengguna diminta untuk memasukkan jumlah blok memori, ukuran setiap blok, serta ukuran file yang akan dialokasikan. Setelah file dialokasikan ke blok-blok yang tersedia, proses pemadatan dilakukan untuk menyusun ulang file, sehingga memori menjadi lebih rapat dan mengurangi ruang kosong di antara blok-blok. Program menampilkan alokasi memori sebelum dan sesudah pemadatan, memperlihatkan bagaimana file ditempatkan serta bagaimana memori disusun ulang untuk meminimalkan fragmentasi. Dengan pemadatan, ruang kosong yang tidak terpakai dapat diminimalkan, meningkatkan efisiensi penggunaan memori secara keseluruhan.

## Kesimpulan

Praktikum ini mengkaji berbagai strategi alokasi memori dinamis seperti *First Fit*, *Best Fit*, dan *Worst Fit*, serta penerapan teknik pemadatan untuk mengatasi fragmentasi. Fragmentasi internal terjadi ketika ruang kosong dalam partisi tidak dimanfaatkan, sementara fragmentasi eksternal muncul akibat ruang kosong yang tersebar sehingga tidak mencukupi untuk alokasi file baru. Strategi *First Fit* unggul dalam kecepatan pencarian tetapi sering menyebabkan fragmentasi eksternal, sedangkan *Best Fit* lebih efisien dalam memanfaatkan ruang memori meski memerlukan waktu pencarian lebih lama. Di sisi lain, *Worst Fit* efektif dalam menghindari fragmentasi kecil namun sering meninggalkan ruang kosong besar yang tidak efisien. Untuk mengatasi fragmentasi eksternal, teknik pemadatan digunakan dengan menyusun ulang file di memori, menggabungkan ruang kosong menjadi satu blok besar sehingga memori lebih optimal. Meskipun efektif, pemadatan membutuhkan waktu dan sumber daya tambahan. Keseluruhan praktikum menunjukkan bahwa pemilihan strategi alokasi memori yang tepat, disertai pemadatan jika diperlukan, dapat meningkatkan efisiensi penggunaan memori dan kinerja sistem operasi.