

Laporan Praktikum

Sistem Operasi

Modul 5



Nama : Asep haryana saputra

NIM : 20230810043

Kelas : TINFC-2023-04

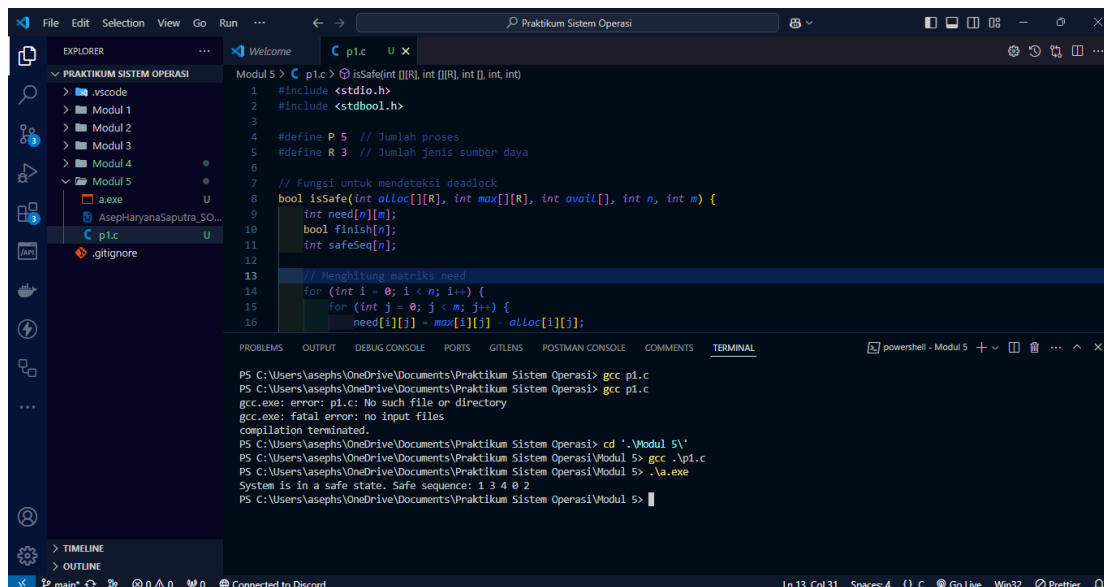
Teknik Informatika
Fakultas Ilmu Komputer
Universitas Kuningan

Pretest

1. Tujuan utama Algoritma Bankir adalah mencegah deadlock dengan memastikan sistem tetap berada dalam kondisi aman (*safe state*). Algoritma ini bekerja dengan memeriksa apakah permintaan sumber daya oleh suatu proses dapat dipenuhi tanpa menyebabkan potensi deadlock. Sebelum alokasi dilakukan, algoritma memprediksi dampak dari pemberian sumber daya terhadap ketersediaan dan keamanan sistem. Jika sistem tetap aman setelah permintaan dipenuhi, alokasi dilakukan; jika tidak, permintaan ditolak atau ditunda hingga sumber daya mencukupi. Dengan pendekatan ini, algoritma memastikan alokasi sumber daya dilakukan secara efisien tanpa mengorbankan kestabilan sistem.
2. Algoritma Bankir menentukan keadaan aman (*safe state*) dengan mensimulasikan alokasi sumber daya untuk memastikan semua proses dapat menyelesaikan eksekusinya secara berurutan tanpa deadlock. Algoritma memeriksa apakah sumber daya yang tersedia cukup untuk memenuhi kebutuhan proses tertentu, lalu secara hipotetis mengalokasikannya. Setelah proses selesai, sumber daya dikembalikan ke sistem untuk digunakan oleh proses lain. Jika ditemukan urutan di mana semua proses dapat selesai, sistem dianggap aman; jika tidak, sistem berada dalam kondisi tidak aman (*unsafe state*).
3. Dalam Algoritma Bankir, **Need** adalah jumlah sumber daya tambahan yang dibutuhkan oleh suatu proses untuk menyelesaikan eksekusinya. Nilai ini dihitung dengan mengurangi jumlah sumber daya yang telah dialokasikan dari jumlah maksimum yang diperlukan oleh proses tersebut.

Praktikum

1.



```
Modul 5 > C p1.c isSafe(int [[R], int [[R], int [], int, int)
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 #define P 5 // Jumlah proses
5 #define R 3 // Jumlah jenis sumber daya
6
7 // Fungsi untuk mendeteksi deadlock
8 bool isSafe(int alloc[R][R], int max[R][R], int avail[], int n, int m) {
9     int need[n][m];
10    bool finish[n];
11    int safeSeq[n];
12
13    // Menhitung matriks need
14    for (int i = 0; i < n; i++) {
15        for (int j = 0; j < m; j++) {
16            need[i][j] = max[i][j] - alloc[i][j];
17        }
18    }
19
20    // Algoritma Banker
21    int count = 0;
22    while (count < n) {
23        for (int i = 0; i < n; i++) {
24            if (!finish[i] && allNeedLessThanAvail(i, need, avail)) {
25                finish[i] = true;
26                count++;
27                // Update availability
28                for (int j = 0; j < m; j++) {
29                    avail[j] += alloc[i][j];
30                }
31            }
32        }
33    }
34
35    // Cek apakah semua proses selesai
36    for (int i = 0; i < n; i++) {
37        if (!finish[i]) return false;
38    }
39    return true;
40 }
41
42 int main() {
43     // Inisialisasi data
44     int max[R][R] = {{3, 2, 2}, {4, 3, 1}, {2, 2, 2}, {4, 1, 3}, {3, 1, 1}};
45     int alloc[R][R] = {{0, 1, 0}, {1, 0, 0}, {0, 0, 1}, {1, 1, 0}, {0, 0, 0}};
46     int avail[R] = {3, 3, 2};
47     int n = P;
48     int m = R;
49
50     if (isSafe(alloc, max, avail, n, m)) {
51         printf("System is in a safe state. Safe sequence: ");
52         // Output safe sequence
53     } else {
54         printf("System is not in a safe state.");
55     }
56
57     return 0;
58 }
```

```
PS C:\Users\jasephs\OneDrive\Documents\Praktikum Sistem Operasi> gcc p1.c
gcc.exe: error: p1.c: No such file or directory
gcc.exe: fatal error: no input files
compilation terminated.
PS C:\Users\jasephs\OneDrive\Documents\Praktikum Sistem Operasi> cd ".\Modul 5\"
PS C:\Users\jasephs\OneDrive\Documents\Praktikum Sistem Operasi\Modul 5> gcc .\p1.c
PS C:\Users\jasephs\OneDrive\Documents\Praktikum Sistem Operasi\Modul 5> .\a.exe
System is in a safe state. Safe sequence: 1 3 4 0 2
PS C:\Users\jasephs\OneDrive\Documents\Praktikum Sistem Operasi\Modul 5>
```

Kode:

<https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%205/p1.c>

Penjelasan:

Fungsi `isSafe` memeriksa apakah sistem dalam keadaan aman dengan menghitung matriks kebutuhan (*need*), memeriksa apakah ada urutan proses yang dapat dieksekusi tanpa deadlock, dan mengembalikan `true` jika aman atau `false` jika deadlock terdeteksi.

Fungsi `main` mendefinisikan contoh matriks alokasi, matriks maksimum, dan array sumber daya yang tersedia, kemudian memanggil `isSafe` untuk memeriksa keamanan sistem.

PostTest

1

Tujuan utama penerapan Algoritma Bankir dalam program ini adalah memastikan sistem berada dalam keadaan aman (*safe state*) guna mencegah deadlock. Algoritma ini mengevaluasi kebutuhan sumber daya setiap proses dan memeriksa apakah sumber daya yang tersedia cukup untuk memenuhinya. Dengan demikian, algoritma memastikan semua proses dapat diselesaikan tanpa menyebabkan deadlock, menjaga urutan eksekusi tetap aman.

2

Untuk menentukan apakah sistem dalam keadaan aman (*safe state*), program mencari urutan proses yang dapat dieksekusi berdasarkan sumber daya yang tersedia. Jika semua proses dapat dijalankan tanpa ada yang terjebak menunggu, sistem dianggap aman. Sebaliknya, jika tidak ditemukan urutan tersebut, sistem berada dalam keadaan tidak aman, yang berpotensi menyebabkan deadlock.

3

Program menghitung kebutuhan (*need*) setiap proses dengan mengurangi jumlah sumber daya yang telah dialokasikan dari jumlah maksimum yang diperlukan. Sebagai contoh, jika sebuah proses membutuhkan 5 unit sumber daya dan saat ini telah dialokasikan 3 unit, maka sisa kebutuhannya adalah 2 unit. Rumusnya adalah **Need = Max - Allocation**.

4.

Jika terjadi deadlock, program akan menampilkan pesan bahwa permintaan sumber daya tidak dapat dipenuhi karena sistem berada dalam keadaan tidak aman. Deadlock terjadi ketika proses-proses dalam sistem saling menunggu sumber daya yang tidak tersedia, sehingga tidak ada proses yang dapat melanjutkan eksekusi.

Tugas

1

Algoritma Bankir lebih fleksibel karena memeriksa setiap permintaan sumber daya sebelum dialokasikan, memastikan sistem tetap dalam keadaan aman dan menghindari deadlock. Algoritma ini cocok untuk sistem yang dinamis dengan perubahan kebutuhan sumber daya. Sebaliknya, Grafik Alokasi lebih sederhana karena hanya mendeteksi deadlock jika ada siklus dalam grafik, tetapi kurang efektif untuk menangani sistem dengan banyak perubahan permintaan sumber daya.

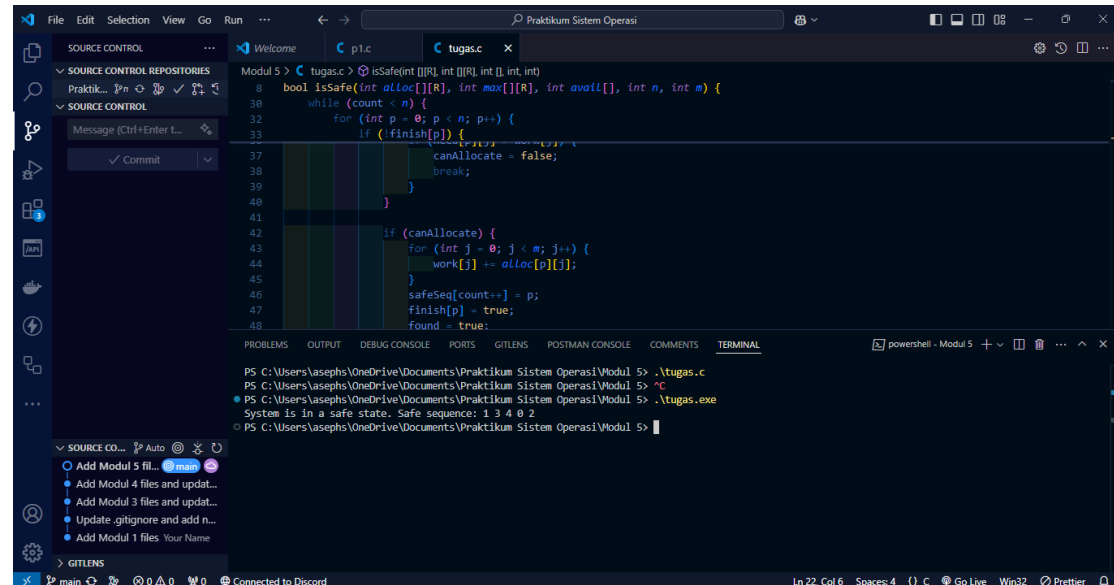
2

Deadlock Avoidance mencegah deadlock dengan menganalisis setiap permintaan sumber daya secara hati-hati untuk memastikan tidak akan menimbulkan masalah di masa depan, menjaga sistem tetap aman. Sementara itu, Deadlock Prevention berfokus pada pengubahan aturan

penggunaan sumber daya, seperti melarang kondisi yang dapat memicu deadlock, tetapi pendekatan ini sering kali mengurangi efisiensi sistem.

3.

A



B

Untuk mendeteksi deadlock dalam skenario dengan banyak contoh dari setiap jenis sumber daya, program dapat dimodifikasi agar mendukung lebih banyak proses dan jenis sumber daya. Meskipun fungsi dan logika dasarnya tetap sama, ukuran matriks untuk alokasi dan sumber daya perlu diperbesar sesuai kebutuhan.

Contoh:

Misalnya, program diubah untuk menangani 10 proses dan 5 jenis sumber daya. Penyesuaian dilakukan seperti berikut:

```
#define P 10
```

```
#define R 5
```

```
int alloc[10][5] = { /* Data alokasi untuk 10 proses dan 5 sumber daya */ };
```

```
int max[10][5] = { /* Data maksimum untuk 10 proses dan 5 sumber daya */ };
```

```
int avail[5] = { /* Data sumber daya yang tersedia */ };
```

Kesimpulan