LAPORAN PRAKTIKUM

SISTEM OPERASI

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

MODUL 5

DISUSUN OLEH :

NAMA : MOHAMAD ABAN SY'BANA

NIM : 20230810012

KELAS : TINFC-2023-04

TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS KUNINGAN

2024

Pre Test

1. **Apa tujuan utama dari Algoritma Bankir (Banker's Algorithm) dalam pencegahan deadlock?**
   Jawab : Tujuan utama dari Algoritma Bankir adalah untuk mencegah deadlock dengan memastikan bahwa sistem hanya mengalokasikan sumber daya jika sistem berada dalam safe state. Dengan kata lain, algoritma ini memastikan bahwa semua proses dapat menyelesaikan eksekusi tanpa menemui situasi deadlock.
2. **Bagaimana cara Algoritma Bankir menentukan apakah suatu sistem berada dalam keadaan aman (safe state)?**
   Jawab :
   Algoritma Bankir menentukan keadaan aman dengan:

   - Memeriksa apakah ada urutan proses yang dapat dijalankan sepenuhnya menggunakan sumber daya yang tersedia saat ini.
   - Algoritma akan mencoba menemukan urutan proses di mana setiap proses dapat menyelesaikan tugasnya dengan kebutuhan maksimalnya dipenuhi oleh sumber daya yang tersedia dan sumber daya yang dikembalikan oleh proses lain setelah selesai.

3. **Apa yang dimaksud dengan "Need" dalam Algoritma Bankir dan bagaimana cara menghitungnya?**
   Jawab : Need adalah jumlah sumber daya tambahan yang dibutuhkan oleh suatu proses untuk menyelesaikan tugasnya.

   Cara menghitungnya adalah:

   $$\text{Need} = \text{Maximum} - \text{Allocation}$$

   Di mana:

   - Maximum adalah jumlah maksimum sumber daya yang dibutuhkan oleh suatu proses.
   - Allocation adalah jumlah sumber daya yang sudah dialokasikan ke proses tersebut.

Praktikum

```
#include <stdio.h>
struct file {
   int all[10];  // Alokasi Array
   int max[10];  // Maximum Array
   int need[10]; // Need Array
   int flag;     // Flag Untuk Tanda Proses Dikunjungi
};
int main() {
   struct file f[10];
   int fl = 0;
   int i, j, k, p, b, n, r, g, cnt = 0, id, newr;
   int avail[10], seq[10];
   // Input jumlah proses
   printf("Enter number of processes -- ");
   scanf("%d", &n);
   // Input jumlah sumber daya
   printf("Enter number of resources -- ");
   scanf("%d", &r);
   // Input alokasi dan maksimum
   for (i = 0; i < n; i++) {
      printf("\nEnter details for P%d\n", i);
      printf("Enter allocation -- ");
      for (j = 0; j < r; j++) {
         scanf("%d", &f[i].all[j]);
      }
      printf("Enter Max -- ");
      for (j = 0; j < r; j++) {
         scanf("%d", &f[i].max[j]);
      }
      f[i].flag = 0; // Set flag ke 0 (belum dikunjungi)
   }
   // Input sumber daya yang tersedia
   printf("\nEnter Available Resources -- ");
```

```
for (i = 0; i < r; i++) {
   scanf("%d", &avail[i]);
}
// Input permintaan baru
printf("\nEnter New Request Details\n");
printf("Enter process ID -- ");
scanf("%d", &id);
printf("Enter Request For Resources -- ");
for (i = 0; i < r; i++) {
   scanf("%d", &newr);
   f[id].all[i] += newr;
   avail[i] -= newr;
}
// Hitung matriks Need
for (i = 0; i < n; i++) {
   for (j = 0; j < r; j++) {
      f[i].need[j] = f[i].max[j] - f[i].all[j];
      if (f[i].need[j] < 0) {
         f[i].need[j] = 0;
      }
   }
}
// Algoritma Bankir
cnt = 0;
while (cnt != n) {
   g = 0;
   for (j = 0; j < n; j++) {
      if (f[j].flag == 0) { // Jika proses belum dikunjungi
         b = 0;
         for (p = 0; p < r; p++) {
            if (avail[p] >= f[j].need[p]) {
               b++;
            }
         }
         if (b == r) { // Semua sumber daya mencukupi
```

```
            printf("\nP%d dikunjungi", j);
            seq[fl++] = j;
            f[j].flag = 1; // Tandai proses sebagai dikunjungi
            for (k = 0; k < r; k++) {
               avail[k] += f[j].all[k];
            }
            // Cetak sumber daya yang tersedia dalam format gambar
            printf(" (%d %d %d)", avail[0], avail[1], avail[2]);

            cnt++;
            g = 1;
         }
      }
   }
   if (g == 0) {
      printf("\nREQUEST NOT GRANTED -- DEADLOCK OCCURRED");
      printf("\nSYSTEM IN UNSAFE STATE");
      return 1;
   }
}
// Sistem dalam keadaan aman
printf("\n\nSISTEM DALAM KEADAAN AMAN");
printf("\nSafe sequence adalah - ");
for (i = 0; i < fl; i++) {
   printf("P%d ", seq[i]);
}
printf("\n");
// Tampilkan detail proses
printf("\nProcess\t\tAllocation\t\tMax\t\tNeed\n");
for (i = 0; i < n; i++) {
   printf("P%d\t\t", i);
   for (j = 0; j < r; j++) {
      printf("%3d", f[i].all[j]);
   }
   printf("\t");
```

```
        for (j = 0; j < r; j++) {
            printf("%4d",f[i].max[j]);
        }
        printf("\t");
        for (j = 0; j < r; j++) {
            printf("%4d",f[i].need[j]);
        }
        printf("\n");
    }
    return 0;
}
```

Hasil RUN

```
Enter number of processes -- 5
Enter number of resources -- 3

Enter details for P0
Enter allocation -- 0 1 0
Enter Max -- 7 5 3

Enter details for P1
Enter allocation -- 2 0 0
Enter Max -- 3 2 2

Enter details for P2
Enter allocation -- 3 0 2
Enter Max -- 9 0 2

Enter details for P3
Enter allocation -- 2 1 1
Enter Max -- 2 2 2

Enter details for P4
Enter allocation -- 0 0 2
Enter Max -- 4 3 3

Enter Available Resources -- 3 3 2

Enter New Request Details
Enter process ID -- 1
Enter Request For Resources -- 1 0 2

P1 dikunjungi (5 3 2)
P3 dikunjungi (7 4 3)
P4 dikunjungi (7 4 5)
P0 dikunjungi (7 5 5)
P2 dikunjungi (10 5 7)

SISTEM DALAM KEADAAN AMAN
Safe sequence adalah - P1 P3 P4 P0 P2

Process         Allocation              Max             Need
P0                0  1  0          7   5   3       7   4   3
P1                3  0  2          3   2   2       0   2   0
P2                3  0  2          9   0   2       6   0   0
P3                2  1  1          2   2   2       0   1   1
P4                0  0  2          4   3   3       4   3   1

--------------------------------
Process exited after 46.33 seconds with return value 0
Press any key to continue . . .
```

Post-Test

1. **Apa tujuan utama dari penerapan Algoritma Bankir dalam program ini?**
   Jawab : Tujuan utama adalah untuk mencegah deadlock dengan memastikan bahwa sistem hanya memberikan sumber daya jika keadaan tetap berada dalam "safe state".

2. **Jelaskan bagaimana program di atas menentukan apakah sistem dalam keadaan aman (safe state) atau tidak.**
   Jawab : Program menentukan "safe state" dengan mengevaluasi apakah semua proses dapat dijalankan secara berurutan tanpa kehabisan sumber daya yang tersedia. Hal ini dilakukan dengan membandingkan kebutuhan setiap proses dengan sumber daya yang tersedia.
3. **Bagaimana cara program ini menghitung kebutuhan (need) untuk setiap proses?**
   Jawab : Need dihitung dengan rumus : Need[i][j] = Max[i][j] - Allocation[i][j]
   Di mana Max adalah jumlah maksimum sumber daya yang dibutuhkan suatu proses, dan Allocation adalah sumber daya yang sudah dialokasikan.
4. **Jika terjadi deadlock, apa yang ditampilkan oleh program dan apa penyebab dari deadlock tersebut?**
   Jawab : Jika terjadi deadlock, program akan menampilkan pesan bahwa tidak ada urutan eksekusi yang aman. Penyebab deadlock adalah tidak cukupnya sumber daya untuk memenuhi kebutuhan proses yang tersisa.

Tugas

1. **Bagaimana kelebihan penggunaan algoritma Bankir dibanding teknik grafik alokasi sumber daya?**
   Jawab : Algoritma Bankir lebih fleksibel karena dapat menangani sistem dinamis dengan proses dan sumber daya yang terus berubah. Teknik grafik lebih cocok untuk situasi statis.
2. **Jelaskan perbedaan deadlock avoidance dengan deadlock prevention ?**
   Jawab :
   - Deadlock Avoidance: Sistem memastikan sumber daya hanya diberikan jika tetap berada dalam "safe state".
   - Deadlock Prevention: Menghindari deadlock dengan mencegah salah satu kondisi deadlock (seperti mutual exclusion atau hold and wait) terjadi.
3. **Tulis program C untuk menerapkan teknik deteksi kebuntuan untuk skenario berikut ?**

   - a. Satu contoh dari setiap jenis sumber daya.
   - b. Banyak contoh dari setiap jenis sumber daya.

   Jawab :

   Code

```
#include <stdio.h>
struct file {
   int all[10];  // Alokasi Array
   int max[10];  // Maximum Array
   int need[10]; // Need Array
   int flag;     // Flag Untuk Tanda Proses Dikunjungi
};
int main() {
```

```
struct file f[10];
int fl = 0;
int i, j, k, p, b, n, r, g, cnt = 0, id, newr;
int avail[10], seq[10];
// Input jumlah proses
printf("Enter number of processes -- ");
scanf("%d", &n);
// Input jumlah sumber daya
printf("Enter number of resources -- ");
scanf("%d", &r);
// Input alokasi dan maksimum
for (i = 0; i < n; i++) {
   printf("\nEnter details for P%d\n", i);
   printf("Enter allocation -- ");
   for (j = 0; j < r; j++) {
      scanf("%d", &f[i].all[j]);
   }
   printf("Enter Max -- ");
   for (j = 0; j < r; j++) {
      scanf("%d", &f[i].max[j]);
   }
   f[i].flag = 0; // Set flag ke 0 (belum dikunjungi)
}
// Semua sumber daya tersedia
for (i = 0; i < r; i++) {
   avail[i] = 1;
}
// Input permintaan baru
printf("\nEnter New Request Details\n");
printf("Enter process ID -- ");
scanf("%d", &id);
printf("Enter Request For Resources -- ");
for (i = 0; i < r; i++) {
   scanf("%d", &newr);
   f[id].all[i] += newr;
```

```
    avail[i] -= newr;
}
// Hitung matriks Need
for (i = 0; i < n; i++) {
    for (j = 0; j < r; j++) {
        f[i].need[j] = f[i].max[j] - f[i].all[j];
        if (f[i].need[j] < 0) {
            f[i].need[j] = 0;
        }
    }
}
// Algoritma Bankir
cnt = 0;
while (cnt != n) {
    g = 0;
    for (j = 0; j < n; j++) {
        if (f[j].flag == 0) { // Jika proses belum dikunjungi
            b = 0;
            for (p = 0; p < r; p++) {
                if (avail[p] >= f[j].need[p]) {
                    b++;
                }
            }
            if (b == r) { // Semua sumber daya mencukupi
                printf("\nP%d dikunjungi (", j);
                for (p = 0; p < r; p++) {
                    printf("%d", f[j].all[p]);
                    if (p < r - 1) printf(", ");
                }
                printf(")");
                seq[fl++] = j;
                f[j].flag = 1; // Tandai proses sebagai dikunjungi
                for (k = 0; k < r; k++) {
                    avail[k] += f[j].all[k];
                }
```

```
            cnt++;
            g = 1;
          }
        }
    }
    if (g == 0) {
        printf("\nREQUEST NOT GRANTED -- DEADLOCK OCCURRED");
        printf("\nSYSTEM IN UNSAFE STATE");
        return 1;
    }
}
// Sistem dalam keadaan aman
printf("\nSISTEM DALAM KEADAAN AMAN");
printf("\nThe safe sequence is: ");
for (i = 0; i < fl; i++) {
    printf("P%d ", seq[i]);
}
printf("\n");
// Tampilkan detail proses
printf("\nProcess\t\tAllocation\t\tMax\t\tNeed\n");
for (i = 0; i < n; i++) {
    printf("P%d\t\t", i);
    for (j = 0; j < r; j++) {
        printf("%3d ", f[i].all[j]);
    }
    printf("\t");
    for (j = 0; j < r; j++) {
        printf("%6d ", f[i].max[j]);
    }
    printf("\t");
    for (j = 0; j < r; j++) {
        printf("%6d ", f[i].need[j]);
    }
    printf("\n");
}
```

```
    return 0;
}
```

Hasil RUN

a. Satu contoh dari setiap jenis sumber daya.

```
Enter number of processes -- 3
Enter number of resources -- 2

Enter details for P0
Enter allocation -- 0 1
Enter Max -- 1 1

Enter details for P1
Enter allocation -- 1 0
Enter Max -- 1 1

Enter details for P2
Enter allocation -- 0 0
Enter Max -- 1 1

Enter New Request Details
Enter process ID -- 1
Enter Request For Resources -- 0 1

P0 dikunjungi (0, 1)
P1 dikunjungi (1, 1)
P2 dikunjungi (0, 0)
SISTEM DALAM KEADAAN AMAN
The safe sequence is: P0 P1 P2

Process         Allocation              Max             Need
P0                0   1             1      1        1      0
P1                1   1             1      1        0      0
P2                0   0             1      1        1      1

--------------------------------
Process exited after 26.08 seconds with return value 0
Press any key to continue . . .
```

b. Banyak contoh dari setiap jenis sumber daya.

```
Enter number of processes -- 3
Enter number of resources -- 2

Enter details for P0
Enter allocation -- 1 2
Enter Max -- 3 3

Enter details for P1
Enter allocation -- 2 1
Enter Max -- 3 2

Enter details for P2
Enter allocation -- 3 3
Enter Max -- 4 4

Enter New Request Details
Enter process ID -- 1
Enter Request For Resources -- 1 1

P1 dikunjungi (3, 2)
P2 dikunjungi (3, 3)
P0 dikunjungi (1, 2)
SISTEM DALAM KEADAAN AMAN
The safe sequence is: P1 P2 P0

Process         Allocation              Max             Need
P0                1   2             3      3        2      1
P1                3   2             3      2        0      0
P2                3   3             4      4        1      1

--------------------------------
Process exited after 46.54 seconds with return value 0
Press any key to continue . . .
```