6

Deadlock

Modul 6

Deadlock

6.1. Tujuan Praktikum

Materi praktikum pada bab ini bertujuan untuk memperkenalkan situasi deadlock yang mungkin terjadi pada system operasi. Sebelum terjadi deadlock, terdapat kondisi-kondisi yang dapat menjadi penyebab situasi deadlock.

6.2. Capaian Praktikum

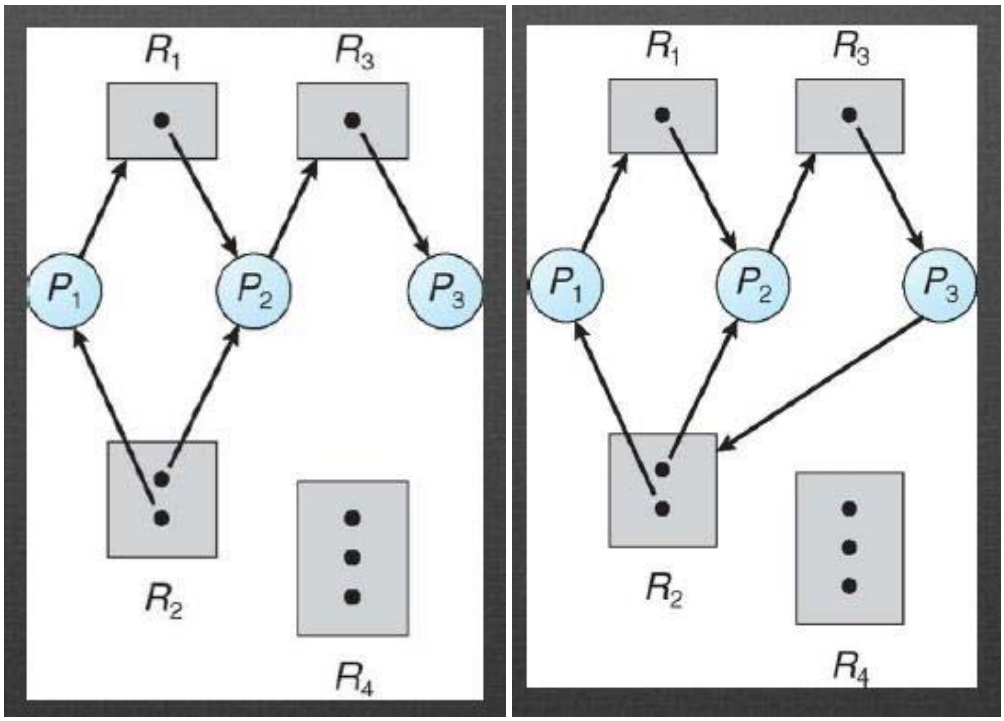
1. Mahasiswa mampu memahami konsep deadlock.
2. Mahasiswa mampu mempraktekkan metode deadlock prevention menggunakan algoritma banker

6.3. Dasar Teori

Permasalahan deadlock terjadi karena sekumpulan proses-proses yang di-blok dimana setiap proses membawa sebuah sumber daya dan menunggu mendapatkan sumber daya yang dibawa oleh proses lain. Misalnya sistem mempunyai 2 tape drive dan terdapat dua proses P1 dan P2 yang masing masing membawa satu tape drive dan masing-masing memerlukan tape drive yang dibawa proses lain sehingga terjadi keadaan saling menunggu resource dan sistem di-blok. Contoh lain misalnya terdapat semaphore A dan B yang diinisialisasi 1 dan terdapat dua proses P0 dan P1 masing-masing membawa semaphore A dan B. Kemudian P0 dan P1 meminta semaphore B dan A dengan menjalankan operasi wait. Hal ini mengakibatkan proses di-blok dan terjadi deadlock.

6.3.1. Graph Allocation Resource

Graph Alocation Resource merupakan salah satu metode yang dipakai untuk mengidentifikasi terjadi deadlock. Sebuah graph disusun untuk menggambarkan hubungan hubungan antara setiap proses dan resource yang tersedia. Ilustrasi Graph Allocation Resource ditunjukkan dalam gambar 6.1.



(a) (b)

Gambar 6.1 (a) Resource Alocation Graph Tanpa *Deadlock* (b) Resource Alocation Graph Dengan *Deadlock*

Hal penting yang perlu diperhatikan dalam penggunaan algoritma banker adalah tentang penyusunan matriks, terdapat beberapa matriks yang perlu dikuasi berkaitan dengan algoritma banker :

1. Jumlah *resource* (sumber daya) dari setiap proses yang mungkin diminta disebut dengan ***request***
2. Jumlah *resource* (sumber daya) dari setiap proses yang dibutuhkan disebut matriks ***max***
3. Jumlah sisa *resource* yang masih dimiliki oleh sistem, disebut dengan matriks ***available***
4. Jumlah *resource* yang telah dialokasikan disebut dengan matriks ***allocation***

Resource hanya boleh/bisa diberikan untuk dipakai pada suatu proses jika:

1. ***request ≤ max***, jumlah permintaan (request) tidak lebih banyak dari jumlah maksimum resource.
2. ***request ≤ available***, jumlah request tidak boleh lebih banyak dari jumlah resource yang tersedia. jika tidak proses harus menunggu hingga resource yang diminta ada.

6.4. Langkah Praktikum

Praktikum ini dilakukan dengan terlebih dahulu terhubung dengan layanan aws educate dengan cara mengaktifkan instance dari halaman instance summary. Pilih action dan Start untuk mengaktifkan instance. Lakukan koneksi SSH dengan cara yang sama seperti pada Bab 1.

1. Login ke sistem GNU/Linux kemudian buka terminal
2. Lakukan percobaan terhadap tiap kode program pada masing-masing sub bab berikut.
3. Lakukan kompilasi kode program dengan menggunakan perintah

```
gcc -o <Nama_Output> <Nama_Source.c>
```

4. Jalankan hasil kompilasi menggunakan perintah ./<Nama_Output>
5. Capture / Snapshot output-nya dan simpan sebagai laporan.
6. Amati output tiap percobaan dan jawab pertanyaan pada masing-masing percobaan.

```c
#include <stdio.h>
#include <stdlib.h>

void print(int x[][10], int n, int m)
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        printf("\n");
        for (j = 0; j < m; j++)
        {
            printf("%d\t", x[i][j]);
        }
    }
}

// Resource Request algorithm
void res_request(int A[10][10], int N[10][10], int AV[10][10], int pid,
```

```
                int m)
{
    int reqmat[1][10];
    int i;
    printf("\n Enter additional request :- \n");
    for (i = 0; i < m; i++)
    {
        printf(" Request for resource %d : ", i + 1);
        scanf("%d", &reqmat[0][i]);
    }
    for (i = 0; i < m; i++)
        if (reqmat[0][i] > N[pid][i])
        {
            printf("\n Error encountered.\n");
            exit(0);
        }
    for (i = 0; i < m; i++)
        if (reqmat[0][i] > AV[0][i])
        {
            printf("\n Resources unavailable.\n");
            exit(0);
        }
    for (i = 0; i < m; i++)
    {
        AV[0][i] -= reqmat[0][i];
        A[pid][i] += reqmat[0][i];
        N[pid][i] -= reqmat[0][i];
    }
}

// Safety algorithm
int safety(int A[][10], int N[][10], int AV[1][10], int n, int m, int a[])
{
    int i, j, k, x = 0;
    int F[10], W[1][10];
    int pflag = 0, flag = 0;
    for (i = 0; i < n; i++)
        F[i] = 0;
    for (i = 0; i < m; i++)
        W[0][i] = AV[0][i];
    for (k = 0; k < n; k++)
    {
        for (i = 0; i < n; i++)
        {
```

```
63              if (F[i] == 0)
64              {
65                  flag = 0;
66                  for (j = 0; j < m; j++)
67                  {
68                      if (N[i][j] > W[0][j])
69                          flag = 1;
70                  }
71                  if (flag == 0 && F[i] == 0)
72                  {
73                      for (j = 0; j < m; j++)
74                          W[0][j] += A[i][j];
75                      F[i] = 1;
76                      pflag++;
77                      a[x++] = i;
78                  }
79              }
80          }
81          if (pflag == n)
82              return 1;
83      }
84      return 0;
85  }
86
87  // Banker's Algorithm
88  void accept(int A[][10], int N[][10], int M[10][10], int W[1][10], int *n,
89              int *m)
90  {
91      int i, j;
92      printf("\n Enter total no. of processes : ");
93      scanf("%d", n);
94      printf("\n Enter total no. of resources : ");
95      scanf("%d", m);
96      for (i = 0; i < *n; i++)
97      {
98          printf("\n Process %d\n", i + 1);
99          for (j = 0; j < *m; j++)
100         {
101             printf(" Allocation for resource %d : ", j + 1);
102             scanf("%d", &A[i][j]);
103             printf(" Maximum for resource %d : ", j + 1);
104             scanf("%d", &M[i][j]);
105         }
106     }
```

```
107     printf("\n Available resources : \n");
108     for (i = 0; i < *m; i++)
109     {
110         printf(" Resource %d : ", i + 1);
111         scanf("%d", &W[0][i]);
112     }
113     for (i = 0; i < *n; i++)
114         for (j = 0; j < *m; j++)
115             N[i][j] = M[i][j] - A[i][j];
116     printf("\n Allocation Matrix");
117     print(A, *n, *m);
118     printf("\n Maximum Requirement Matrix");
119     print(M, *n, *m);
120     printf("\n Need Matrix");
121     print(N, *n, *m);
122 }
123
124 int banker(int A[][10], int N[][10], int W[1][10], int n, int m)
125 {
126     int j, i, a[10];
127     j = safety(A, N, W, n, m, a);
128     if (j != 0)
129     {
130         printf("\n\n");
131         for (i = 0; i < n; i++)
132             printf(" P%d ", a[i]);
133         printf("\n A safety sequence has been detected.\n");
134         return 1;
135     }
136     else
137     {
138         printf("\n Deadlock has occured.\n");
139         return 0;
140     }
141 }
142
143 int main()
144 {
145     int ret;
146     int A[10][10];
147     int M[10][10];
148     int N[10][10];
149     int W[1][10];
150     int n, m, pid, ch;
```

```
151     printf("\n DEADLOCK AVOIDANCE USING BANKER'S ALGORITHM\n");
152     accept(A, N, M, W, &n, &m);
153     ret = banker(A, N, W, n, m);
154     if (ret != 0)
155     {
156         printf("\n Do you want make an additional request ? (1=Yes|0=No)");
157         scanf("%d", &ch);
158         if (ch == 1)
159         {
160             printf("\n Enter process no. : ");
161             scanf("%d", &pid);
162             res_request(A, N, W, pid - 1, m);
163             ret = banker(A, N, W, n, m);
164             if (ret == 0)
165                 exit(0);
166         }
167     }
168     else
169         exit(0);
170     return 0;
171 }
```

6.5. Pembahasan

Setelah anda berhasil melakukan kompilasi program lakukan percobaan dengan langkahlangkah berikut :

1. Masukkan variable berikut :

   a. Jumlah proses = 5

   b. Jumlah resource = 3

   Dengan Matriks

	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

2. Amati hasilnya dan tuliskan outputnya!

3. Apakah permintaan penambahan resource dari kondisi pada no 1 diijinkan apabila P4 melakukan permintaan tambahan alokasi sebanyak R1=3, R2=3 dan R3=0
4. Amati hasilnya dan tuliskan outputnya untuk kondisi dari point 3.
5. Apakah permintaan penambahan resource dari kondisi pada no 1 diijinkan apabila P4 melakukan permintaan tambahan alokasi sebanyak R1=0, R2=2 dan R3=0
6. Amati hasilnya dan tuliskan outputnya untuk kondisi pada point 5.
7. Pada bagian program terdapat Resource Request Alghorithm (baris 14 sampai 40) dan Safety Alghoritm (baris 42- 72). Jelaskan konsep algortimanya dan penggunaannya!
8. Dapatkan kondisi pada soal no 1, 3 dan 5 di gambarkan dengan menggunakan Resource Alocation Graph? Jelaskan!

6.6. Kesimpulan

Jelaskan kesimpulan dari hasil percobaan pada Bab 6 ini

UNTUK DIPERHATIKAN

Setiap selesai melakukan praktikum, jangan lupa menonaktfikan instance agar billing kuota tidak terus berjalan. Lakukan dengan cara memilih button Action pada Instance Summary. Pilih Stop sebagai perintah yang harus dijalankan.