SIMULASI PRODUSEN – KONSUMEN UNTUK MENYELESAIKAN PROBLEM MUTUAL EXCLUSION PADA MARKET

Khairul Ummi[1], Zulfrida Andayani [2]
[1,2]STMIK Potensi Utama,
Jl.K.L Yos Sudarso Km. 6,5 No.3A Tanjung Mulia Medan
Email : kha_ummi@yahoo.com

Abstrak

Kasus producer-consumer digunakan sebagai ilustrasi pembahasan sinkronisasi. Kasus producer-consumer berisi masalah mutual-exclusion dan sinkronisasi. Kasus ini sering juga disebut sebagai bounded-buffer problem (masalah buffer dengan jumlah terbatas). Agar mencapai tujuan secara benar, proses-proses harus mensinkronkan kegiatan-kegiatannya untuk menghindari kondisi deadlock (malapetaka). Proses-proses yang berinteraksi memerlukan sinkronisasi agar terkendali dengan baik. Deadlock terjadi ketika proses-proses mengakses sumber daya secara eksklusif. Semua deadlock yang terjadi melibatkan persaingan untuk memperoleh sumber daya eksklusif oleh dua proses atau lebih. Gagasan dasar penghindaran deadlock adalah hanya memberi akses ke permintaan sumber daya yang tidak mungkin menimbulkan deadlock. Strategi ini biasanya diimplementasikan dengan pengalokasi sumber daya memeriksa dampak-dampak pemberian akses ke suatu permintaan. Agar dapat mengevaluasi safe-nya system individu, penghindaran deadlock mengharuskan semua proses menyatakan jumlah kebutuhan sumber daya maksimum sebelum eksekusi. Begitu eksekusi dimulai, tiap proses meminta sumber daya saat diperlukan sampai batas maksimum. Proses-proses yang menyatakan kebutuhan sumber daya melebihi kapasitas total system tidak dapat dieksekusi.shared-memory merupakan solusi ke masalah bounded-buffer yang mengijinkan paling banyak n-1 materi dalam buffer pada waktu yang sama. Suatu solusi, jika semua N buffer digunakan tidaklah sederhana.Pada producer-consumer, dimulai dari 0 dan masing-masing waktu tambahan dari suatu item baru diberikan kepada buffer.

***Kata Kunci** : Simulasi, producer, consumer, market, Sinkronisasi, deadlock.*

Abstract

The case of the producer-consumer synchronization is used to illustrate the discussion . Producer-consumer case contains issues of mutual -exclusion and synchronization . This case is often referred to as the bounded - buffer problem (the problem of buffer with a limited number) . Correctly in order to achieve the objectives , processes must synchronize their activities to avoid a deadlock condition (catastrophe) . Interacting processes that require synchronization so well controlled . Deadlock occurs when processes access resources exclusively . All deadlocks that occur involve competition for resources exclusively by two or more processes . Deadlock avoidance basic idea is simply to give access to the resource requests may not cause a deadlock . This strategy is usually implemented by the resource allocator examine the effects of granting access to a query . In order to evaluate the system of individual safe , deadlock avoidance requires all states the maximum amount of resource requirements prior to execution . Once execution begins , each process requesting resources as needed up to the limit . The processes stated resource requirements exceed the total capacity of the system can not dieksekusi.shared - memory is a solution to the bounded - buffer problem which allow at most n - 1 material in the buffer at the same time . One solution , if all N buffers are used is not sederhana.Pada producer-consumer , starting from 0 and each additional time a new item is given to the buffer .

Keywords : Simulation , producer , consumer , market , Synchronization , deadlocks.

1. Pendahuluan

Kasus *producer-consumer* digunakan sebagai ilustrasi pembahasan sinkronisasi. Kasus *producer-consumer* berisi masalah *mutual-exclusion* dan sinkronisasi. Kasus ini sering juga disebut sebagai *bounded-buffer problem* (masalah *buffer* dengan jumlah terbatas). Kasus ini dapat diilustrasikan sebagai berikut, produsen menghasilkan barang dan konsumen yang akan menggunakannya. Keduanya mempunyai *market* (ilustrasi dari *buffer*) bersama dan berukuran tetap. Karena ukuran *market* terbatas, petaka (bencana) dapat terjadi untuk *producer* dan *consumer*. Petaka bagi *producer* terjadi ketika *market* telah penuh, sementara *producer* ingin meletakkan barang (ilustrasi dari informasi) ke *market* yang telah penuh itu. Sedangkan petaka bagi *consumer* terjadi ketika *consumer* ingin mengambil barang sementara *market* telah / sedang kosong. Agar mencapai tujuan secara benar, proses-proses harus mensinkronkan kegiatan-kegiatannya untuk menghindari kondisi *deadlock* (malapetaka). Proses-proses yang berinteraksi memerlukan sinkronisasi agar terkendali dengan baik.

Perangkat lunak simulasi *Producer-Consumer Problem* ini untuk membantu pemahaman terhadap proses kerja dari *Producer-Consumer Problem*. Perangkat lunak juga dapat digunakan sebagai fasilitas pendukung dalam proses belajar mengajar, terutama mengenai Sistem Operasi yang tujuan tertinggi adalah agar dapat merancang sendiri atau memodifikasi system operasi yang telah ada sesuai kebutuhan khusus. Dan hubungannya dengan metode *semaphore* untuk pengendalian sinkronisasi proses-proses.

2. Analisa Sistem

Perangkat lunak simulasi *Producer-Consumer Problem* dimulai dengan tampilan *splash screen*. *Form splash screen* berisi nama / judul perangkat lunak dan nama pembuat perangkat lunak. Beberapa saat kemudian, *form input* akan tampil. *Form input* berfungsi untuk mengatur kondisi simulasi. Komponen yang dapat diatur adalah sebagai berikut:

1. Pengaturan pada *Producer*, yaitu jumlah *producer*, batas maksimum dan minimum bagi *producer* dalam satu kali produksi.
2. Pengaturan pada *Consumer*, yaitu jumlah *consumer*, batas maksimum dan minimum bagi *consumer* dalam satu kali konsumsi.
3. Pengaturan pada *Market*, yaitu batas ukuran maksimum dan minimum *market*.
4. Pengaturan lain, yaitu jenis item dan kecepatan proses simulasi.

Setelah pengaturan pada *form input*, proses simulasi dapat dimulai dengan menekan tombol 'Simulasi'. Selanjutnya, *form* simulasi akan muncul. Ketika proses simulasi sedang berjalan, *user* dapat menghentikan untuk sementara proses simulasi dengan menekan tombol 'Hentikan'. *User* juga dapat melihat laporan proses yang terjadi dalam simulasi dengan menekan tombol 'Laporan'. Laporan juga disediakan dalam bentuk tabel. Untuk melihat lebih lanjut input dari simulasi producer-consumer problem dapat dilihat dari tabel 1 berikut :

Tabel 1.

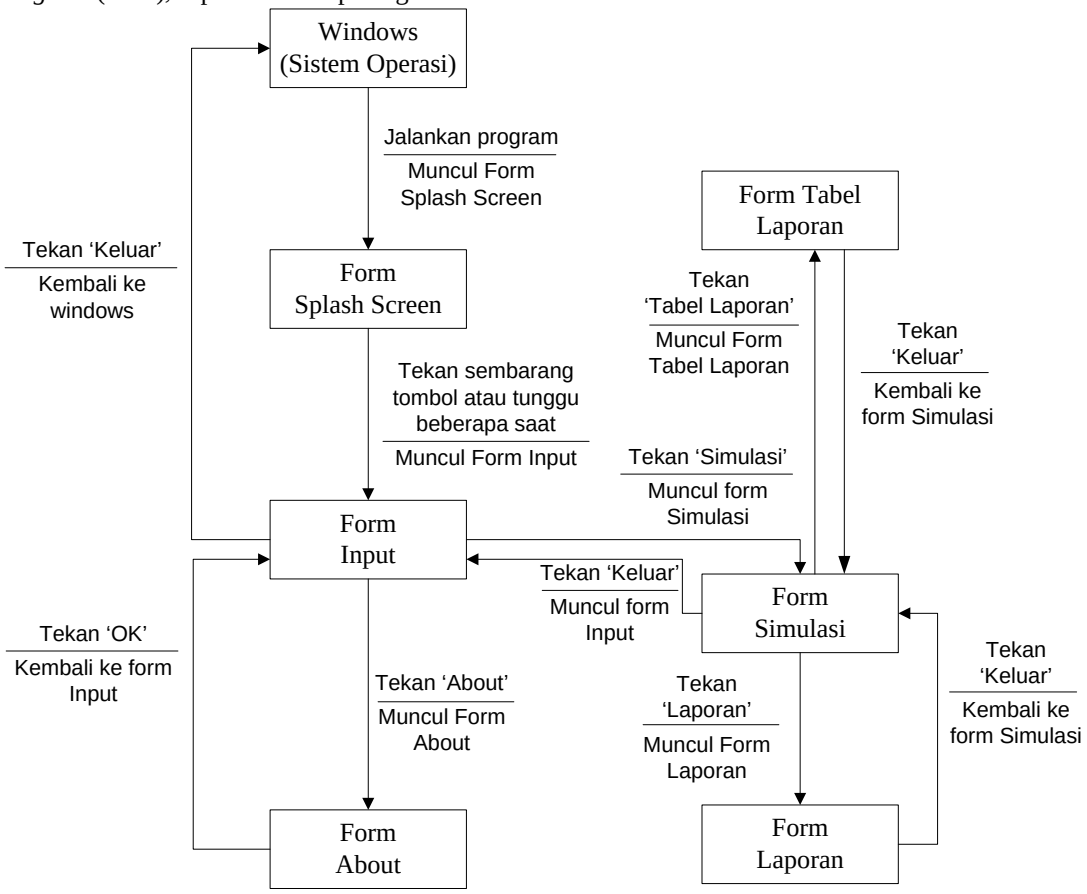
Nama : Tabel Inputan
Sumber : Input Perangkat lunak Simulasi producer-consumer Problem

PRODUCER			CONSUMER			MARKET		SETTING		
BATAS 1 X PRODUKSI			BATAS 1 X KONSUMSI			BATAS UKURAN				
LH	AK	IN	LH	AK	IN	AK	IN	ENIS ITEM	ATAS WKT SIMULASI (DETIK)	ETIK DLM PROG (MILI DETIK)
	3			4	0	3	2	JENIS	5	00
	2			5		8		JENIS	00	00
	2			4		8	1	JENIS	0	00

Tabel 2. Nama Item

Nama : Tabel Jenis Item. Sumber : Jumlah item dalam market	
JENIS ITEM	**NAMA ITEM**
JENIS 1	WORTEL
JENIS 2	TOMAT
JENIS 3	JAMUR
JENIS 4	KACANG
JENIS 5	JUS

Alur kerja perangkat lunak simulasi ini dapat digambarkan dalam bentuk *State Transition Diagram* (STD), seperti terlihat pada gambar 1. berikut.



Gambar 1. *State Transition Diagram* (STD)

Output

Hasil dari output dapat dilihat pada tabel-tabel berikut ini :

Keterangan 1 :

Tabel 3. output sistem

Nama : Laporan Hasil.
Sumber : Tabel simulasi producer-consumer

Variabel	Waktu Mulai Prod/Kons	Waktu Selesai Prod/Kons	Waktu Tiba Di Market	Aksi	Market (Tomat)
Prod 2	1	5	-	-	-
Prod	1	5	-	-	-

Keterangan 2 :
Tabel 4.

Nama : Laporan Hasil.
Sumber : Tabel simulasi producer-consumer

Variabel	Waktu Mulai Prod/Kons	Waktu Selesai Prod/Kons	Waktu Tiba Di Market	Aksi	Market (Tomat, Wortel)
Prod 5	1	27	41	+(17,18)	(60,52)
Prod 4	1	19	35	+(22,7)	(43,34)
Prod 3	1	-	-	-	-
Prod 2	1	7	23	+(8,9)	(8,9)
Prod 1	1	13	28	+(13,18)	(21,27)
Prod 2	38	50	65	+(8,4)	(11,7)
Prod 1	43	63	78	+(13,14)	(16,17)
Prod 4	50	59	74	+(6,16)	(9,19)
Prod 5	57	79	95	+(14,19)	(28,34)
Prod 2	69	-	-	-	-
Cons 3	78	-	-	-	-
Cons 2	80	-	-	-	-
Prod 1	87	90	-	-	-
Cons 4	89	90	-	-	-
Prod 1	93	90	-	-	-

Keterangan 3 :
Tabel 5.

Nama : Laporan Hasil.
Sumber : Tabel simulasi producer-consumer

Variabel	Waktu Mulai Prod/Kons	Waktu Selesai Prod/Kons	Waktu Tiba Di Market	Aksi	Market (Tomat, Wortel, Jamur, Kacang, Jus)
Prod 6	1	13	29	+(12,4,6,11,14)	(69,68,62,70,41)
Prod 5	1	7	21	+(14,4,8,13,7)	(14,4,8,13,7)
Prod 4	1	13	28	+(16,20,20,7,8)	(57,64,56,59,27)
Prod 3	1	9	25	+(16,22,9,17,5)	(30,26,17,30,12)
Prod 2	1	9	25	+(11,18,19,22,7)	(41,44,36,52,19)
Prod 1	1	13	29	+(14,19,16,11,4)	(83,87,78,81,45)
Prod 5	37	-	-	-	-

Proses simulasi *Producer-Consumer Problem* adalah sebagai berikut:

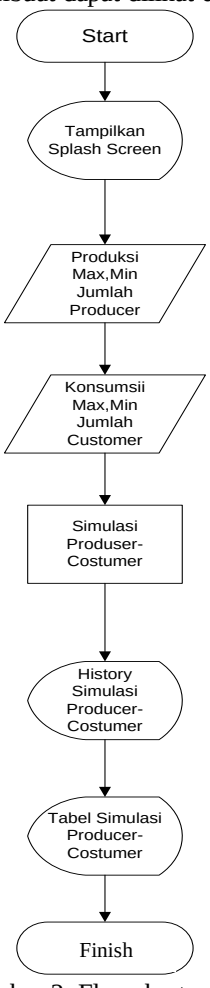
1. *Producer* aktif memproduksi dan meletakkan *item* ke *market* (*buffer*). Aksi ini akan menambah jumlah *item* di dalam *market*.

2. *Consumer* aktif mengambil *item* dari *market* (*buffer*) dan mengonsumsi *item*. Aksi ini akan mengurangi jumlah *item* di dalam *market*.
3. Apabila *market* telah penuh atau jumlah *item* di dalam *market* telah mencapai batas maksimum, maka *producer* akan tidur (memanggil aksi *sleep*).
4. Apabila *consumer* mengambil *item* dari *market* dan *producer* dalam keadaan *sleep*, maka *consumer* akan membangunkan (*wake-up*) *producer*.
5. Apabila *market* kosong atau jumlah *item* di dalam *market* telah mencapai batas minimum, maka *consumer* akan tidur (memanggil aksi *sleep*).
6. Apabila *producer* meletakkan *item* ke *market* dan *consumer* dalam keadaan *sleep*, maka *producer* akan membangunkan (*wake-up*) *consumer*.

Keenam poin di atas merupakan inti dari simulasi *Producer-Consumer Problem*. Pencegahan kondisi *deadlock* (*producer* ingin meletakkan *item* ke *market* sedangkan *market* telah penuh atau *consumer* ingin mengambil *item* dari *market* sedangkan *market* telah kosong) dihindari dengan metode *sleep* dan *wake up*. Masing-masing variabel akan memanggil aksi *sleep* untuk menghindari kondisi *deadlock* dan akan dibangunkan variabel lainnya, ketika keadaan sudah tidak menyebabkan *deadlock*.

Critical section dalam perangkat lunak simulasi ini adalah ketika *producer* ingin meletakkan *item* ke dalam *market* dan *consumer* ingin mengambil item dari *market*. Dalam hal ini, *semaphore* digunakan untuk mengatur *producer* dan *consumer* supaya tidak meletakkan atau mengambil *item* dari *market* atau mengakses *critical section* secara bersamaan. Ketika salah satu *producer* atau *consumer* meletakkan atau mengambil *item*, maka variabel lain di-*blocked* agar tidak memasuki *critical section*, sehingga jumlah *item* di dalam *market* tidak diakses secara bersamaan, nilainya tetap konsisten dan terjaga kebenarannya.

Flowchart yang akan digambarkan dibawah ini merupakan flowchart dari program yang dibuat. Mulai dari awal *system* dijalankan sampai menghasilkan History Simulasi dan Tabel Simulasi Producer-Costumer. Flowchart dari program yang dibuat dapat dilihat dari gambar 3 berikut

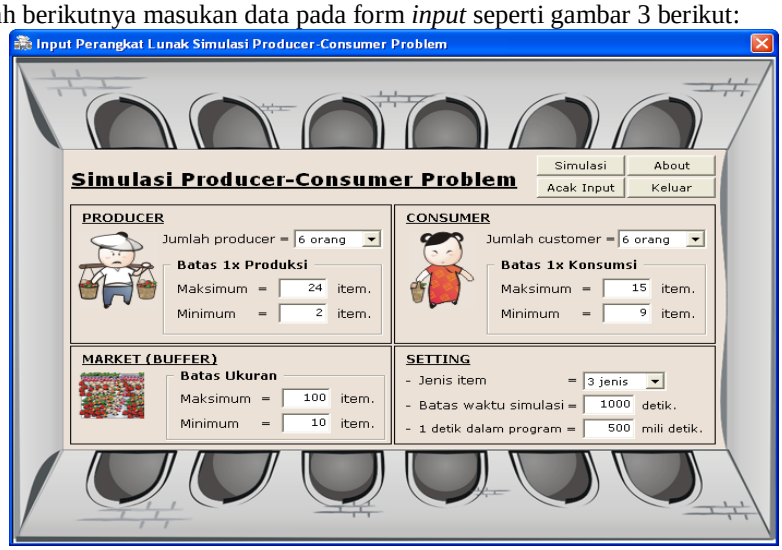


Gambar 2. Flowchart program

3. Hasil Dan Uji Coba

Form Input

Langkah berikutnya masukan data pada form *input* seperti gambar 3 berikut:



Gambar 3. input Simulasi Producer-Sonsumer Problem

Keterangan gambar

1. Masukan Jumlah *producer* = 6 orang
2. Batas satu kali produksi, maksimum= 24 *item* dan minimum = 2 *item*.
3. Jumlah *consumer* = 6 orang.
4. Batas maksimum satu kali konsumsi = 15 *item* dan minimum = 9 *item*.
5. Batas ukuran maksimum market = 100 *item*.
6. Batas ukuran minimum market = 10 *item*.
7. Banyak jenis *item* = 3 jenis.
8. Batas waktu simulasi = 1000 detik.

Form Simulasi

Tampilan awal *form* Simulasi dapat dilihat pada gambar 4 berikut.

Gambar 4. Tampilan *form* Simulasi sebelum proses simulasi dimulai

Pada saat t = 202 detik, *producer* berada dalam keadaan tertidur (*sleep*). Tampilan prosesnya dapat dilihat pada gambar 5 berikut.

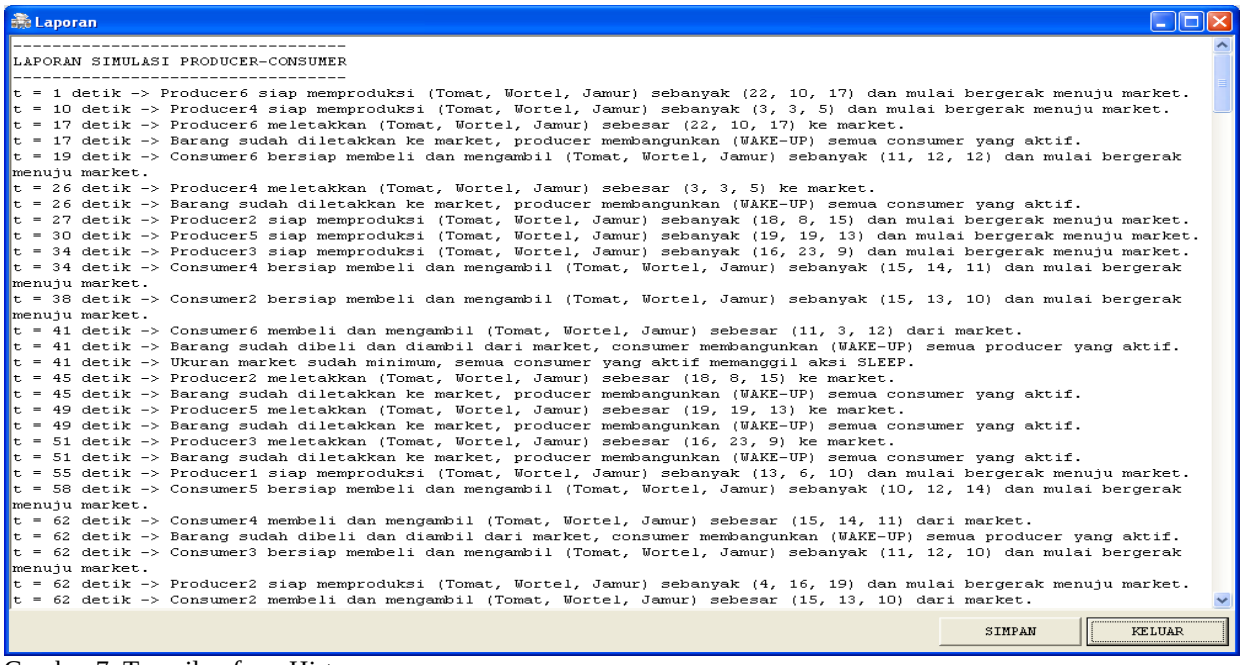
Gambar 5. Tampilan simulasi ketika *producer* berada dalam keadaan tertidur (*sleep*)

Pada saat t = 452 detik, *consumer* berada dalam keadaan tertidur (*sleep*). Tampilan prosesnya dapat dilihat pada gambar 6 berikut.

Gambar 6. Tampilan simulasi ketika *consumer* berada dalam keadaan tertidur (*sleep*)

Form History

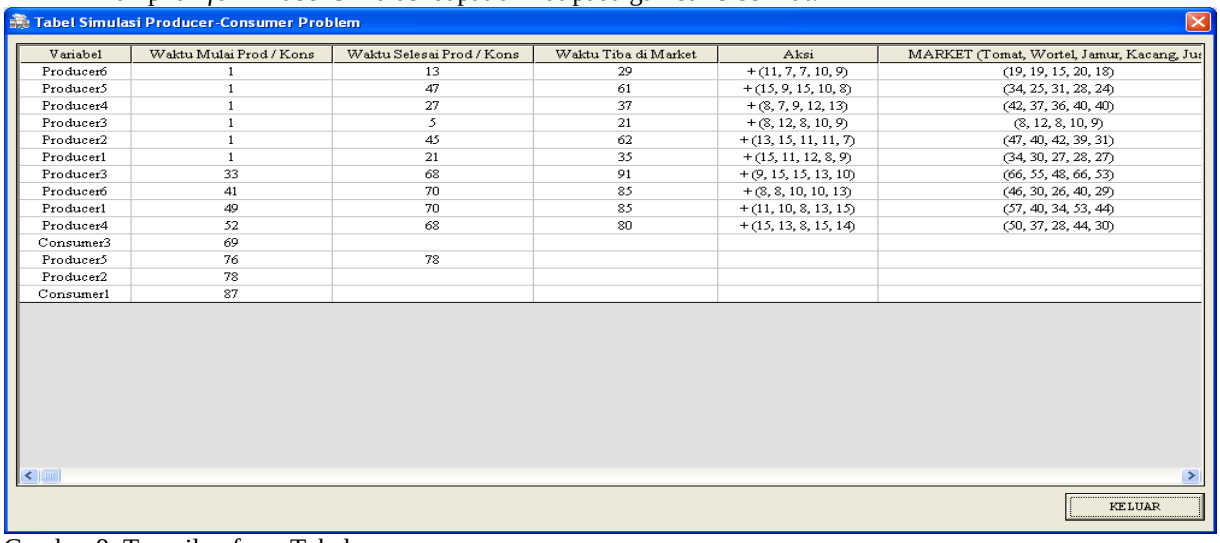
Laporan proses-proses yang terjadi dari *t* = 0 detik sampai t = 100 detik adalah sebagai berikut: Tampilan *form* History dapat dilihat pada gambar 7. berikut.



Gambar 7. Tampilan *form* History

Form Tabel Simulasi

Tampilan *form* Tabel Simulasi dapat dilihat pada gambar 8 berikut.



Variabel	Waktu Mulai Prod / Kons	Waktu Selesai Prod / Kons	Waktu Tiba di Market	Aksi	MARKET (Tomat, Wortel, Jamur, Kacang, Jus
Producer6	1	13	29	+(11, 7, 7, 10, 9)	(19, 19, 15, 20, 18)
Producer5	1	47	61	+(15, 9, 15, 10, 8)	(34, 25, 31, 28, 24)
Producer4	1	27	37	+(8, 7, 9, 12, 13)	(42, 37, 36, 40, 40)
Producer3	1	5	21	+(8, 12, 8, 10, 9)	(8, 12, 8, 10, 9)
Producer2	1	45	62	+(13, 15, 11, 11, 7)	(47, 40, 42, 39, 31)
Producer1	1	21	35	+(15, 11, 12, 8, 9)	(34, 30, 27, 28, 27)
Producer3	33	68	91	+(9, 15, 15, 13, 10)	(66, 55, 48, 66, 53)
Producer6	41	70	85	+(8, 8, 10, 10, 13)	(46, 30, 26, 40, 29)
Producer1	49	70	85	+(11, 10, 8, 13, 15)	(57, 40, 34, 53, 44)
Producer4	52	68	80	+(15, 13, 8, 15, 14)	(50, 37, 28, 44, 30)
Consumer3	69				
Producer5	76	78			
Producer2	78				
Consumer1	87				

Gambar 8. Tampilan form Tabel

4. Pembahasan

Pengaturan pergerakan Producer

Dalam perancangan perangkat lunak, pergerakan seorang *producer* ditulis di dalam sebuah *timer*. Apabila terdapat enam orang *producer* yang aktif, maka ini berarti terdapat enam buah *timer* untuk mengatur pergerakan semua *producer*. Dalam perangkat lunak ini, pergerakan *producer* antara lain adalah:

1. Berada di dalam rumah.
   Producer diasumsikan sedang memproduksi *item*. Gambar *producer* tidak terlihat pada daerah simulasi.
2. Berjalan menuju *market*.
   Producer diasumsikan telah siap memproduksi *item* dan membawa *item* hasil produksi ke *market*.
3. Meletakkan *item* hasil produksi ke *market*

Producer telah tiba di market dan akan meletakkan semua *item* hasil produksinya ke *market*. Dalam aksi ini digunakan *semaphore*, untuk menghindari pengaksesan *critical section* secara bersamaan. Di sini, *critical section* adalah keadaan ketika seorang *producer* masuk ke dalam *market* dan meletakkan *item* hasil produksinya ke *market*. Penggunaan *sempahore* akan mem-*blocked* semua variabel (*producer* ataupun *consumer*) yang ingin masuk ke dalam *critical section* ketika salah seorang *producer* ataupun *consumer* sedang berada di dalam *market* dan sedang meng-*update* jumlah *item* market dan jumlah *item*-nya sendiri. Penggunaan *semaphore* akan menjamin hanya terdapat satu buah variabel (*producer* ataupun *consumer*) yang berada di dalam *market* pada satu waktu. Cara menerapkan *sempahore* di dalam perangkat lunak adalah lakukan operasi *Down* (s = s -1) ketika *producer* akan masuk ke dalam *market*, dan lakukan operasi *Up* (s = s +1) ketika *producer* telah selesai dan keluar dari *market*. Jika nilai s bernilai -1 (s < 0) ketika sebuah variabel (*producer* ataupun *consumer*) akan masuk ke dalam *market*, maka variabel di-*blocked* atau tidak dapat masuk ke dalam *market* dan menunggu hingga nilai s = 0. Apabila *market* telah mencapai ukuran maksimum, maka panggil aksi *sleep* (tidur) untuk semua *producer*. Ketika *producer* telah selesai meletakkan *item* hasil produksinya ke *market*, panggil aksi *wake-up* (bangun) untuk semua *customer* apabila *customer* tertidur.

4. Berjalan kembali ke rumah.
   Producer telah selesai meletakkan semua *item* hasil produksinya ke *market* dan berjalan kembali ke tempat asalnya (rumah).

Pengaturan Pergerakan Consumer

Sama seperti *producer*, pergerakan seorang *consumer* ditulis di dalam sebuah *timer*. Apabila terdapat enam orang *consumer* yang aktif, maka ini berarti terdapat enam buah *timer* untuk mengatur pergerakan semua *consumer*. Dalam perangkat lunak ini, pergerakan *consumer* antara lain adalah:

1. Berada di dalam rumah.
   Consumer diasumsikan sedang mengonsumsi *item* hasil pembelian dari *market*. Gambar *consumer* tidak terlihat pada daerah simulasi.
2. Berjalan menuju *market*.
   Consumer diasumsikan telah habis mengonsumsi *item* dan mulai berjalan ke *market* untuk membeli *item*.
3. Membeli *item* dari *market*
   Consumer telah tiba di market dan akan membeli sejumlah *item* dari *market*. Dalam aksi ini digunakan *semaphore*, untuk menghindari pengaksesan *critical section* secara bersamaan. Di sini, *critical section* adalah keadaan ketika seorang *consumer* masuk ke dalam *market* dan mulai membeli dan mengambil *item* dari *market*. Penggunaan *sempahore* akan mem-*blocked* semua variabel (*producer* ataupun *consumer*) yang ingin masuk ke dalam *critical section* ketika salah seorang *producer* ataupun *consumer* sedang berada di dalam *market* dan sedang meng-*update* jumlah *item* market dan jumlah *item*-nya sendiri. Penggunaan *semaphore* akan menjamin hanya terdapat satu buah variabel (*producer* ataupun *consumer*) yang berada di dalam *market* pada satu waktu. Cara menerapkan *sempahore* adalah sama seperti pembahasan pada algoritma pergerakan *producer*. Apabila *market* telah mencapai ukuran minimum, maka panggil aksi *sleep* (tidur) untuk semua *consumer*. Ketika *consumer* telah selesai membeli dam mengambil *item* dari *market*, panggil aksi *wake-up* (bangun) untuk semua *producer* apabila *producer* tertidur.
4. Berjalan kembali ke rumah.
   Consumer telah selesai membeli dan mengambil *item* dari *market* dan berjalan kembali ke tempat asalnya (rumah).

5. Penutup

Setelah menyelesaikan perangkat lunak simulasi *Producer-Consumer Problem*, maka diambil kesimpulan sebagai berikut:

1. Perangkat lunak menggunakan metode *sleep and wake-up* untuk mencegah masalah yang terjadi ketika *buffer* penuh, sementara *producer* ingin meletakkan *item* ke *buffer* dan *consumer* ingin mengambil *item* sementara *buffer* telah kosong.
2. Perangkat lunak menggunakan *semaphore* untuk untuk mem-*blocked producer* atau *consumer* lain ketika salah satu *producer* atau *consumer* sedang berada dalam *buffer*..

3. Perangkat lunak simulasi *Producer-Consumer* ini merupakan illustrasi dari proses sinkronisasi, yaitu bagaimana cara mengatur beberapa proses yang mengakses beberapa variabel secara bersamaan.

Daftar Pustaka

[1.] Thomas J. Kakiay, 2004, Pengantar Sistem Simulasi, Andi, Yogyakarta
[2.] Abas Ali Pangera, Dony Ariyus. - 1, Cet.1.2005, Sistem Operasi, Andi Offset, Yogyakarta
[3.] Hariyanto, Bambang. 2003. *Sistem Operasi*. Bandung : Informatika, hal. 157
[4.] Abdul Kadir, 2010, *Database* MySql, Penerbit: Andi, Yogyakarta
[5.] Abdul Kadir, 2008, Tuntunan Praktis Belajar Database Menggunakan MySQL, C.V ,Andi Offset. Yogyakarta.