|  |
| --- |
| **LAPORAN PRAKTIKUM SISTEM OPERASI**  **SINKRONISASI PROSES/THREAD** |
|  |
| **13323033 Agus Pranata Marpaung 13323004 Aldi Kurniawan Tampubolon 13323019 Rahel Apriyanka Napitupulu**  **DIII TEKNOLOGI KOMPUTER** |
| **INSTITUT TEKNOLOGI DEL**  **FAKULTAS VOKASI** |

**Judul Praktikum**

|  |  |  |
| --- | --- | --- |
| **Minggu/Sesi** | P | IV/2 |
| **Kode Mata Kuliah** | : | 1031202/1041202 |
| **Nama Mata Kuliah** | : | SISTEM OPERASI |
| **Setoran** | : | *Softcopy* |
| **Batas Waktu Setoran** | : | *28 Februari 2024 jam ??:??* |
| **Tujuan** | : | 1. *Mampu menulis program untuk sinkronisasi thread pada Linux menggunakan Semaphore, Conditional Variable, dan Mutex.* |

**Petunjuk:**

1. Sebelum memberikan pertanyaan silahkan baca buku yang ada pada referensi.
2. Mencontoh pekerjaan dari orang lain akan dianggap plagiarisme dan anda akan ditindak sesuai dengan sanksi akademik yang berlaku di IT Del atau sesuai dengan kebijakan saya dengan memberikan nilai 0.
3. Jawaban diketik menggunakan word processor kemudian dikonversi ke file berekstensi .pdf
4. Penamaan file HARUS sesuai dengan format NIM\_Tugas- X\_NamaTugas.pdf (contoh: 01\_Tugas-2\_Struktur\_Sistem\_Operasi.pdf).
5. Keterlambatan menyerahkan laporan tidak ditolerir dengan alasan apapun. Oleh karena itu, laporan harus dikumpul tepat waktu.
6. Gunakan Sistem Operasi Linux boleh menggunakan Distro apapun namun disarankan untuk mempermudah praktikum gunakan Ubuntu.

**Referensi:**

1. A. Silberschatz, P.B. Galvin, and G. Gagne, Operating System Concepts, 9th edition, Chapter 1 and 2, John Wiley & Sons, Inc., 2013.
2. M. Neil, S. Richard, Beginning Linux Programming, 4th edition, Wiley, 2008.
3. **Teori [35 poin]**
4. [10 poin] Jelaskanlah latar belakang perlunya sinkronisasi proses/ thread.
5. [10 poin] Apa yang dimaksud dengan Race Condition? Berikan contonya!
6. [15 poin] Sebutkan dan jelaskan masalah klasik pada sinkronisasi!

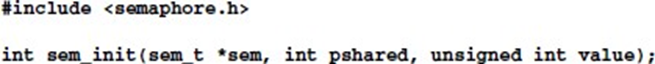
**Penjelasan:**

1. Sinkronisasi proses atau thread adalah konsep yang penting dalam pemrograman paralel dan konkurensi.Latar belakang perlunya sinkronisasi proses/ thread adalah:
2. Akses Bersama ke Sumber Daya: Ketika dua atau lebih proses atau thread berusaha mengakses sumber daya bersama, seperti variabel global, file, atau basis data, sinkronisasi diperlukan untuk mencegah konflik dan kerusakan data yang tidak terduga. Tanpa sinkronisasi, mungkin terjadi situasi di mana satu proses menulis data sementara proses lain mencoba membacanya, yang dapat menyebabkan ketidaksesuaian atau kehilangan data.
3. Keamanan Data: Sinkronisasi membantu menjaga integritas data. Dengan menggunakan teknik sinkronisasi yang tepat, Anda dapat memastikan bahwa data hanya dimodifikasi oleh satu proses atau thread pada satu waktu, mencegah kemungkinan perubahan yang tidak terduga dan memastikan konsistensi data.
4. Deadlock dan Starvation: Sinkronisasi juga membantu mencegah deadlock, di mana dua atau lebih proses saling menunggu sumber daya yang dipegang oleh yang lain, sehingga tidak ada yang bisa melanjutkan. Selain itu, sinkronisasi dapat membantu mencegah starvation, yaitu situasi di mana satu atau lebih proses tidak pernah mendapatkan akses ke sumber daya yang mereka butuhkan karena disisihkan oleh proses lain secara terus menerus.
5. Konsistensi Program: Dalam beberapa kasus, konsistensi program memerlukan langkah-langkah tertentu untuk dipastikan. Misalnya, dalam aplikasi perbankan, transaksi yang dilakukan oleh satu pengguna harus sinkron dengan transaksi yang dilakukan oleh pengguna lain untuk memastikan konsistensi data.
6. Kecepatan dan Kinerja: Sinkronisasi yang efisien juga dapat meningkatkan kinerja program. Dengan menggunakan teknik sinkronisasi yang tepat, Anda dapat mengurangi overhead yang terkait dengan pengaturan dan koordinasi proses atau thread.
7. Kerja sama dan Komunikasi: Sinkronisasi juga memungkinkan proses atau thread untuk berkomunikasi dan berkoordinasi satu sama lain. Ini penting dalam skenario di mana tugas-tugas yang berbeda perlu berbagi informasi atau saling bergantung satu sama lain untuk menyelesaikan pekerjaan.
8. Race condition adalah situasi di mana hasil akhir dari suatu program atau sistem tergantung pada urutan atau waktu pelaksanaan dari beberapa proses atau thread. Ini terjadi ketika dua atau lebih proses atau thread berbagi sumber daya bersama dan melakukan operasi read-modify-write secara bersamaan tanpa sinkronisasi yang memadaidan melakukan operasi read-modify-write secara bersamaan tanpa sinkronisasi yang memadai.
9. Beberapa masalah klasik yang sering muncul dalam sinkronisasi antara lain:
10. Deadlock: Deadlock terjadi ketika dua atau lebih proses atau thread saling menunggu untuk mendapatkan sumber daya yang dipegang oleh yang lain. Akibatnya, tidak ada proses yang dapat melanjutkan eksekusi, dan mereka semua terjebak dalam siklus tunggu yang tidak berujung. Contoh: Proses A menahan Sumber Daya 1 dan menunggu untuk Sumber Daya 2 yang dipegang oleh Proses B, sedangkan Proses B menahan Sumber Daya 2 dan menunggu untuk Sumber Daya 1 yang dipegang oleh Proses A.
11. Starvation: Starvation terjadi ketika satu atau lebih proses atau thread tidak pernah mendapatkan akses ke sumber daya yang mereka butuhkan karena terus-menerus disisihkan oleh proses atau thread lain yang lebih prioritas. Akibatnya, proses tersebut tidak pernah selesai atau sangat lambat dalam menyelesaikan tugasnya.
12. Race Condition: Race condition terjadi ketika hasil akhir dari suatu program bergantung pada urutan atau waktu pelaksanaan dari beberapa proses atau thread. Ini terjadi ketika dua atau lebih proses atau thread berbagi sumber daya bersama dan melakukan operasi read-modify-write secara bersamaan tanpa sinkronisasi yang memadai. Akibatnya, output program menjadi tidak konsisten atau tidak terduga.
13. Priority Inversion: Priority inversion terjadi ketika proses dengan prioritas yang lebih rendah menahan sumber daya yang dibutuhkan oleh proses dengan prioritas yang lebih tinggi. Ini mengakibatkan proses dengan prioritas yang lebih tinggi terhenti atau tertunda karena menunggu sumber daya yang dipegang oleh proses dengan prioritas yang lebih rendah. Hal ini dapat mengganggu jadwal dan kinerja sistem secara keseluruhan.
14. Livelock: Livelock terjadi ketika dua atau lebih proses atau thread saling berinteraksi dan saling memblokir satu sama lain tanpa membuat kemajuan dalam menyelesaikan tugas mereka. Meskipun tidak seperti deadlock yang benar-benar terhenti, livelock membuat proses terus berputar dalam lingkaran tanpa akhir, menyebabkan kegagalan dalam menyelesaikan tugas.
15. **Pemrograman**

Pada bagian ini anda akan menggunakan tiga metode dasar dalam sinkronisasi yaitu Semaphore, Conditional Variable, dan Mutual Exclusion (Mutex).

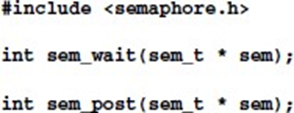
1. **Sinkronisasi dengan Semaphore**

Fungsi semaphore tidak dimulai dengan pthread\_, seperti kebanyakan fungsi spesifik thread, tetapi dengan sem\_. Empat fungsi dasar semaphore digunakan dalam thread diantaranya adalah sem\_init, sem\_wait, sem\_post, dan sem\_destroy. Semaphore dibuat dengan fungsi sem\_init dinyatakan sebagai berikut:



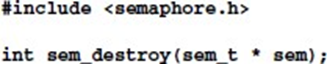
Parameter pertama berfungsi untuk menginisialisasi objek semaphore yang ditunjuk oleh sem. Parameter kedua menentukan bagaimana semaphore akan di-*sharing*. Jika bernilai 0, maka semaphore dapat di-share antar thread dalam satu proses. Tapi jika bernilai 1 maka semaphore dapat di-share antar proses. Parameter terakhir berfungsi untuk memberikan nilai awal ke objek semaphore yang diinisialisasi pada parameter pertama. Hasil dari sem\_ini merupakan integer bernilai 0 jika inisialisasi sempahore berhasil dan bernilai 1 jika terdapat error.

Pasangan fungsi berikut berguna untuk mengontrol nilai semaphore dan dideklarasikan sebagai berikut:



Keduanya fungsi diatas mengambil pointer ke objek semaphore yang diinisialisasi pada sem\_init. Fungsi sem\_post secara atomik meningkatkan nilai semaphore sebesar 1. Sedangkan fungsi sem\_wait secara atomik mengurangi nilai semaphore sebesar satu. Jika Anda memanggil sem\_wait pada semaphore dengan nilai 2, maka thread akan dieksekusi sehingga nail semaphore menjadi 1. Jika sem\_wait dipanggil lagi pada semaphore yang sama dan akan mengakibatkan nilainya menjadi 0, maka thread tersebut akan menunggu untuk dieksekusi sampai ada thread lain yang meningkatkan nilai semaphore tersebut sehingga tidak lagi bernilai 0 melalui sem\_post.

Fungsi semaphore terakhir adalah sem\_destroy. Fungsi ini digunakan saat Anda semaphore yang anda buat selesai yang dinyatakan sebagai berikut:

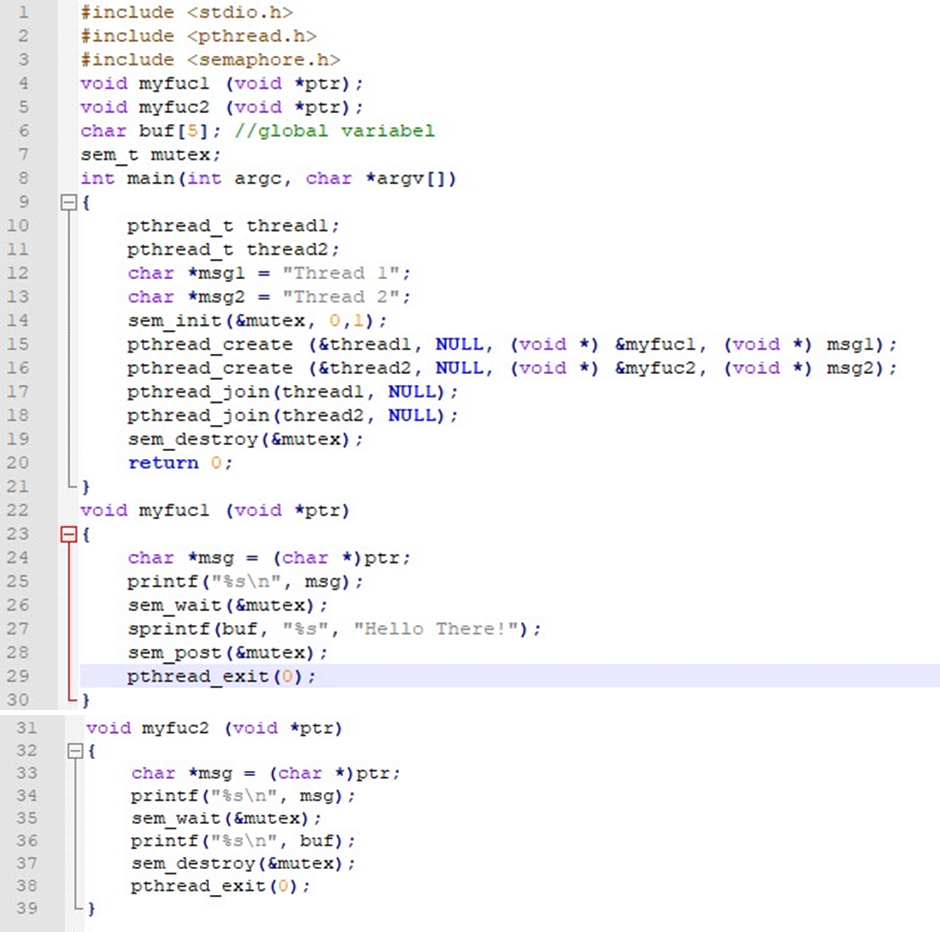


Fungsi ini mengambil pointer ke semaphore dan merapikan sumber daya yang mungkin dimiliki. Seperti kebanyakan fungsi Linux, semua fungsi ini mengembalikan nilai 0 jika berhasil dan selain 0 jika terdapat error.

Selanjutnya anda diminta untuk membuat program menggunakan metode sempahore dengan mengikuti prosedur di bawah ini.

Prosedur:

1. Buatlah file dengan nama semaphore\_basic.c lalu tambahkan code di bawah ini pada file tersebut.



1. Pada program semaphore\_basic.c di atas, terdapat suatu global variabel yaitu buf[4] yang akan di akses oleh thread1 dan thread2 melalui fungsi berikut:



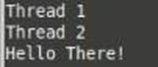
1. Thread1 akan memanggil fungsi myfuc1 dimana fungsi tersebut akan mengisi buf dengan “Hello There!” melalui pernyataan berikut:



1. Sedangkan thread2 akan memanggil funsgi myfuc2 dimana fungsi tersbut akan menampilkan isi dari buf memalui pernyataan berikut:

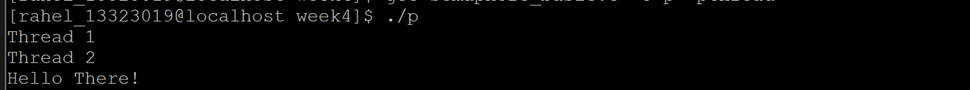


1. Berikut adalah hasil setelah program semaphore\_basic.c di jalankan:



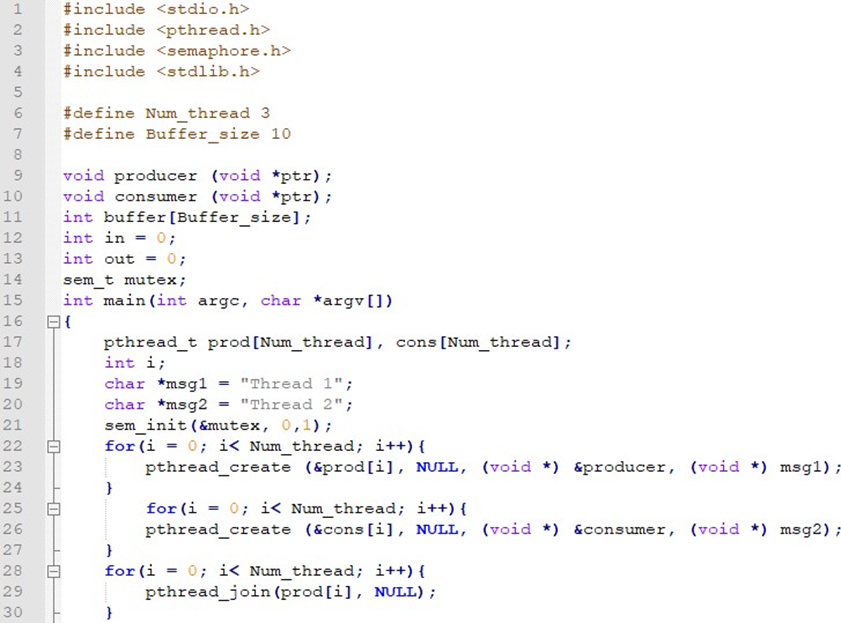
Selanjutnya anda diminta untuk membuat program menggunakan metode sempahore dengan kasus producer consumer dengan mengikuti prosedur di bawah ini.

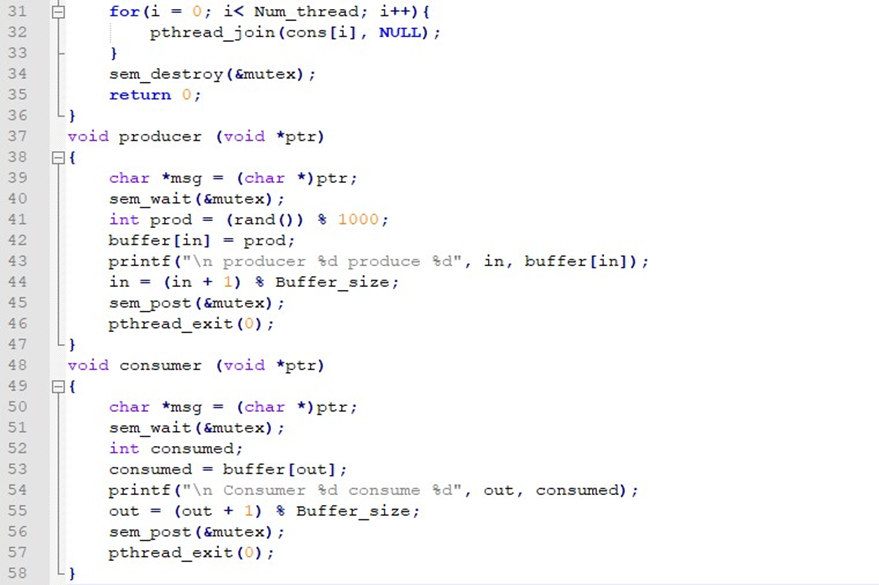
**Output:**

****

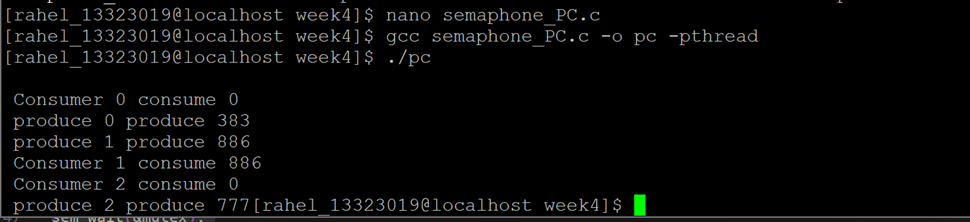
Prosedur:

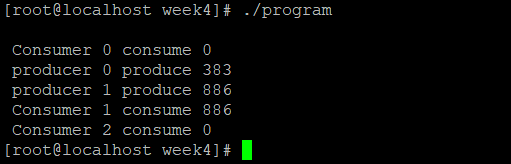
1. Buatlah file dengan nama semaphore\_PC.c lalu tambahkan code di bawah ini pada file tersebut.





**Output:**

****

****

1. Program di atas terdapat beberapa thread yang akan dibuat untuk mengakses fungsi producer dan consumer. fungsi producer yang akan memproduksi suatu nilai pada buffer dan fungsi consumer yang akan mengkonsumsi nilai pada buffer yang telah terisi.

**2. Sinkronisasi dengan Conditional Variable**

Metode dengan Conditional Variable memungkinkan sejumlah thread menunggu hingga thread lain selesai. Anda dapat mendeklarasikan variabel kondisi menggunakan :



dan deklarasi variabel mutex menggunakan:



Variabel kondisi selalu digunakan bersama dengan mutex untuk mengontrol aksesnya. Berikut adalah fungsi yang anda gunakan jika ingin membuat variabel kondisi dalam keadaan wait (menunggu).



Anda perlu mengunci mutex sebelum memanggil fungsi di atas. Semua thread yang menunggu pada variabel kondisi akan ditunda hingga thread lain menggunakan fungsi sinyal yang dinyatakan sebagai berikut:

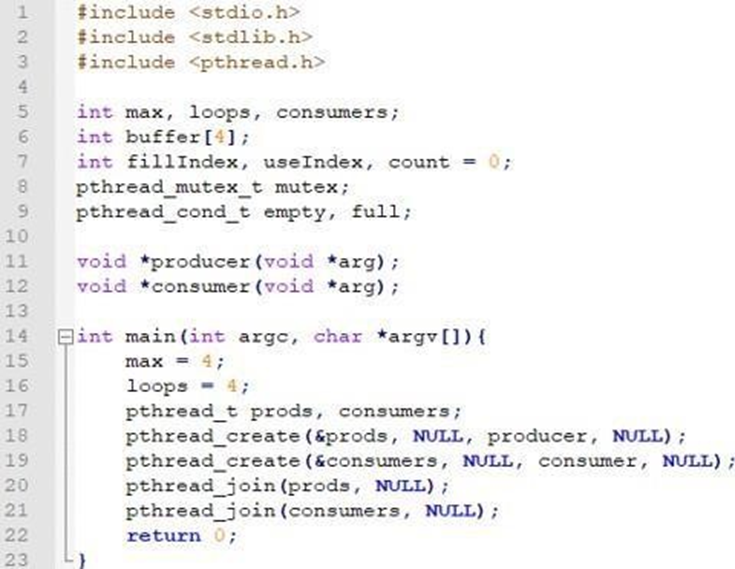


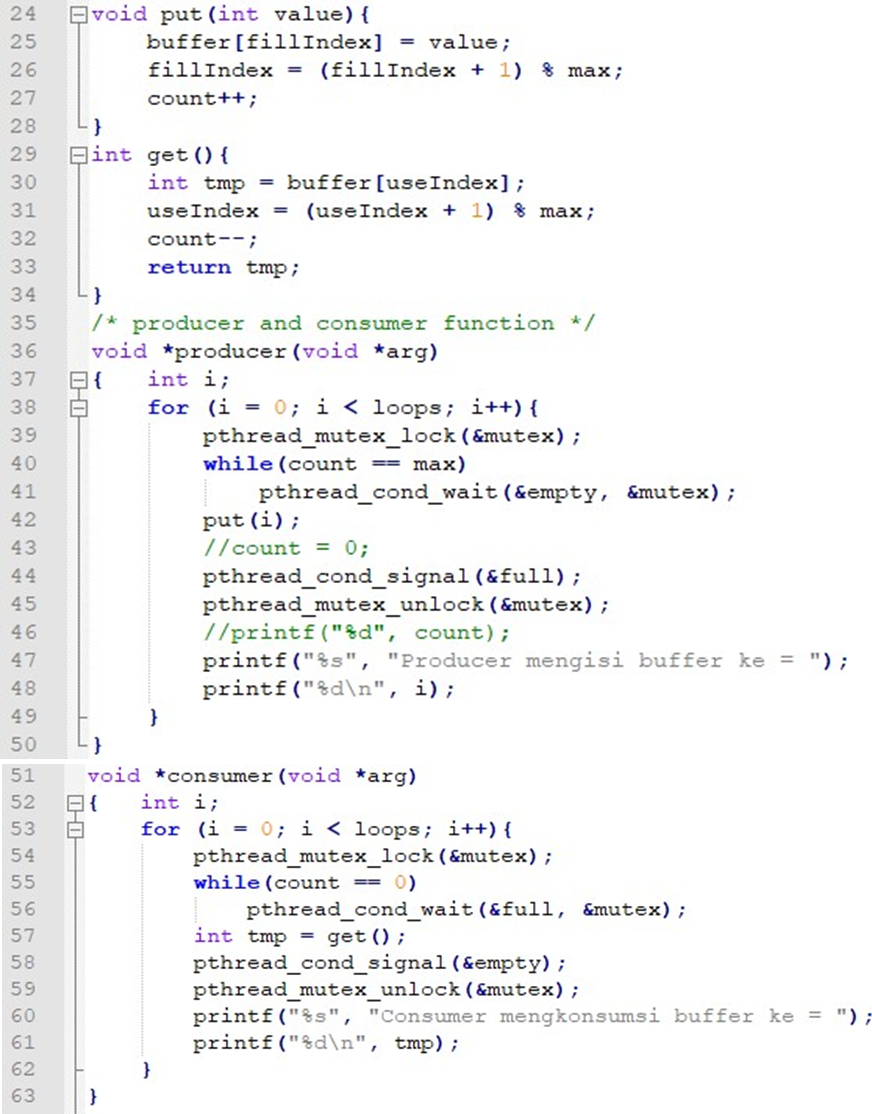
Sama seperti sebelumnya, mutex harus dikunci sebelum memanggil fungsi di atas dan dibuka setelahnya. Fungsi sinyal menyebabkan setidaknya satu utas yang menunggu untuk dimulai kembali (restarted).

Selanjutnya anda diminta untuk membuat program menggunakan metode conditional variable dengan kasus producer consumer dengan mengikuti prosedur di bawah ini.

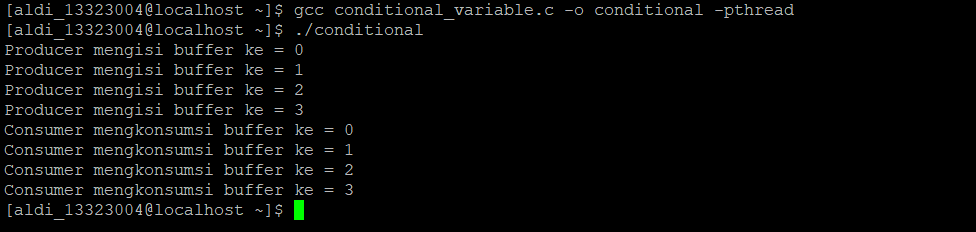
Prosedur:

1. Buatlah file dengan nama conditional\_variable.c lalu tambahkan code di bawah ini pada file tersebut.





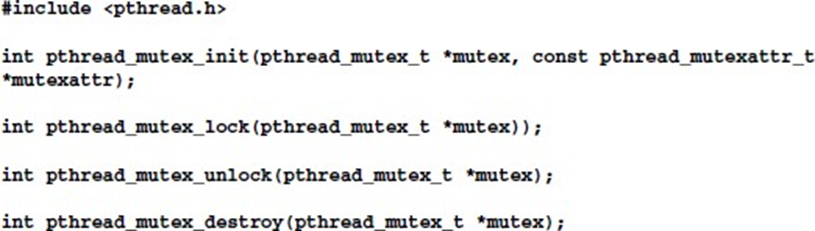
**Output:**



2. Program di atas terdapat beberapa thread yang akan dibuat untuk mengakses fungsi producer dan consumer. fungsi producer yang akan memproduksi suatu nilai pada buffer hingga penuh lalu fungsi consumer yang akan mengkonsumsi nilai pada buffer yang telah terisi penuh hingga kosong.

**3.** **Sinkronisasi dengan MUTEX.**

Cara lain untuk menyinkronkan akses dalam program multithreaded adalah dengan mutex (Mutula Exclusion), yang bertindak dengan memungkinkan pemrogram untuk "mengunci" objek sehingga hanya satu thread saja yang dapat mengakses critical secction. Untuk mengontrol akses ke bagian critical section, anda perlu mengunci mutex terlebih dahulu dan kemudian buka kunci itu setelah selesai. Fungsi dasar yang diperlukan untuk menggunakan mutex sangat mirip dengan yang dibutuhkan untuk semaphores yang dinyatakan sebagai berikut:



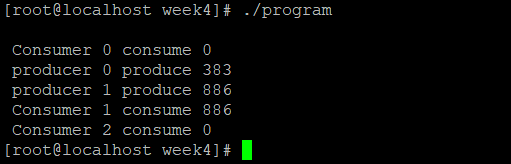
Seperti biasa, akan mengembalikan nilai 0 jika sukses, dan selain 0 jika terjadi error. Sama seperti semaphores, mutex mengambil pointer ke objek yang dinyatakan sebelumnya, dalam hal ini pthread\_mutex\_t. Parameter atribut ekstra pthread\_mutex\_init memungkinkan Anda untuk menyediakan atribut untuk mutex, yang mengontrol perilakunya.

**C. Tugas Pemrograman [65 Poin]**

1. **[20 poin]** Pada program semaphore\_PC.c, jawablah poin-poin berikut:
2. Jelaskan fungsi dari For loop pada line 22-24!
3. Jelaskan fungsi dari For loop pada line 28-30!
4. Jelaskan cara kerja fungsi producer pada line 37 dan consumer pada line 48!
5. Jalankan program semaphore\_PC.c lalu *screenshot* hasilnya!

**Penjelasan:**

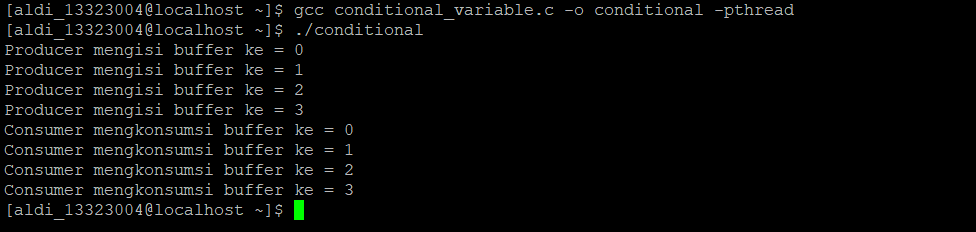
1. **For loop pada line 22-24:** Loop ini digunakan untuk membuat thread-producer. Dalam loop tersebut, pthread\_create() dipanggil sebanyak Num\_thread kali, yang mana Num\_thread merupakan jumlah thread yang ingin dibuat. Dalam contoh ini, Num\_thread memiliki nilai 3. Setiap iterasi dari loop membuat satu thread-producer baru dengan memanggil fungsi pthread\_create(), kemudian thread tersebut akan menjalankan fungsi producer().
2. **For loop pada line 28-30:** Loop ini digunakan untuk membuat thread-consumer. Cara kerjanya sama dengan loop sebelumnya, namun kali ini loop membuat thread-consumer dengan memanggil fungsi pthread\_create() dan menjalankan fungsi consumer().
3. **Fungsi producer() (line 37):**  Menggunakan semaphore mutex untuk melakukan sinkronisasi antar thread. Menunggu semaphore mutex untuk diambil (menjalankan sem\_wait(&mutex)).Kemudian, menghasilkan angka acak (prod) untuk dimasukkan ke dalam buffer.Memasukkan nilai prod ke dalam buffer pada indeks in, kemudian menaikkan nilai in untuk menunjukkan posisi selanjutnya di buffer.Setelah selesai memodifikasi buffer, melepaskan semaphore mutex untuk memberi sinyal kepada thread lain bahwa akses ke bagian kritis telah selesai. Sedangkan **Fungsi consumer() (line 48)**:Menggunakan semaphore **mutex** untuk melakukan sinkronisasi antar thread.Menunggu semaphore **mutex** untuk diambil (menjalankan **sem\_wait(&mutex)**).Mengambil nilai dari buffer pada indeks **out**, kemudian menaikkan nilai **out** untuk menunjukkan posisi selanjutnya yang akan dikonsumsi di buffer.Setelah selesai membaca dari buffer, melepaskan semaphore **mutex** untuk memberi sinyal kepada thread lain bahwa akses ke bagian kritis telah selesai.
4. Screenshot dari program **Semaphore\_PC.c**

****

1. **[15 poin]**Pada program conditional\_variable.c, jawablah poin-poin berikut:
2. Jelaskan fungsi main pada line 14-22
3. Jelaskan cara kerja fungsi producer pada line 36 dan consumer pada line 51!
4. Jalankan conditional\_variable.c lalu *screenshot* hasilnya!

**Jawab:**

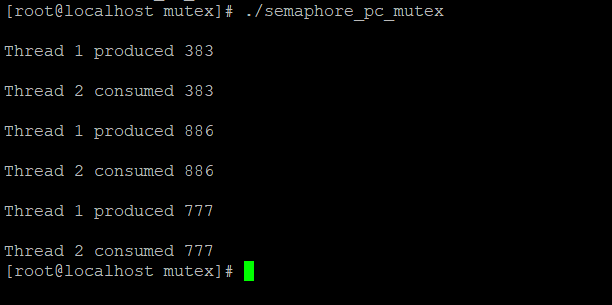
1. Fungsi **`main` pada line 14-22** untuk melakukan inisialisasi beberapa variabel seperti `max`, `loops`, dan `consumers`. Kemudian, dua thread dibuat menggunakan `pthread\_create`, satu untuk produsen (`producer`) dan satu lagi untuk konsumen (`consumer`). Setelah kedua thread selesai dieksekusi, program menunggu hingga keduanya selesai dengan menggunakan `pthread\_join`.
2. **Fungsi `producer` pada line 36** bertanggung jawab untuk menghasilkan nilai yang dimasukkan ke dalam buffer. Dalam loop `for`, jika buffer sudah penuh (`count == max`), produsen menunggu dengan menggunakan `pthread\_cond\_wait` pada kondisi `empty`, yang menunggu hingga ada slot kosong dalam buffer. Setelah mendapat kesempatan, nilai dimasukkan ke dalam buffer dengan memanggil `put`, kemudian sinyal dikirimkan menggunakan `pthread\_cond\_signal` pada kondisi `full` agar konsumen mengetahui bahwa buffer telah terisi. Fungsi `consumer` pada line 51 memiliki pola kerja yang mirip, tetapi berlawanan, yaitu menunggu hingga buffer tidak kosong (`count == 0`) sebelum mengkonsumsi nilai dari buffer dengan memanggil `get`.
3. Hasilnya:



1. **[30 poin]** Anda telah menjalankan program semaphore\_PC.c dan conditional\_variable.c. Sekarang buatlah 2 program menggunakan metode Mutex dimana satu programnya digunakan untuk menggantikan metode semaphore pada program semaphore\_PC.c dan program lainnya digunakan untuk menggantikan metode conditional variable pada program conditional\_variable.c!

**Jawab:**

1. semaphore\_PC.c



1. conditional\_variable.c

