Laporan Praktikum
Sistem Operasi

Modul 4

Nama : Asep haryana saputra

NIM : 20230810043

Kelas : TINFC-2023-04

Teknik Informatika

Fakultas Ilmu Komputer

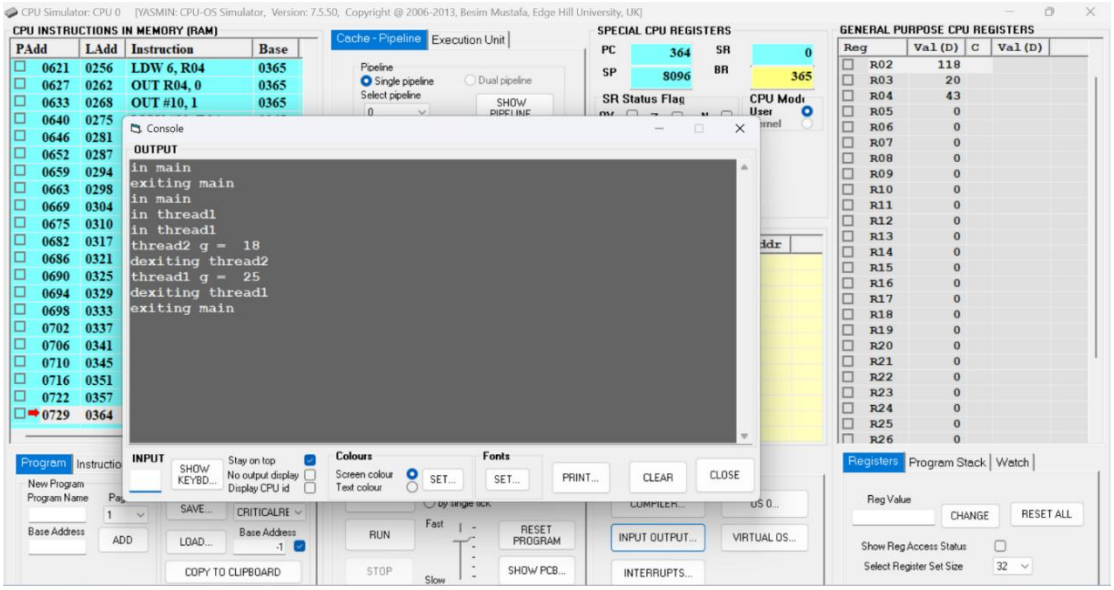
Universitas Kuningan

Pretest

1. Tujuan dari sinkronisasi proses dalam sistem operasi adalah untuk mengatur akses ke sumber daya bersama secara terkoordinasi sehingga mencegah terjadinya konflik atau inkonsistensi data. Sinkronisasi memastikan bahwa proses atau thread yang berbagi data atau sumber daya dapat beroperasi secara aman dan menghasilkan hasil yang sesuai dengan harapan.
2. Masalah yang bisa terjadi ketika beberapa thread mengakses data bersama tanpa sinkronisasi adalah terjadinya kondisi balapan (race condition) yang dapat menyebabkan inkonsistensi data, korupsi data, atau perilaku sistem yang tidak diinginkan. Selain itu, masalah seperti deadlock, livelock, dan starvation juga dapat muncul, yang menghambat kelancaran eksekusi program.

Praktikum

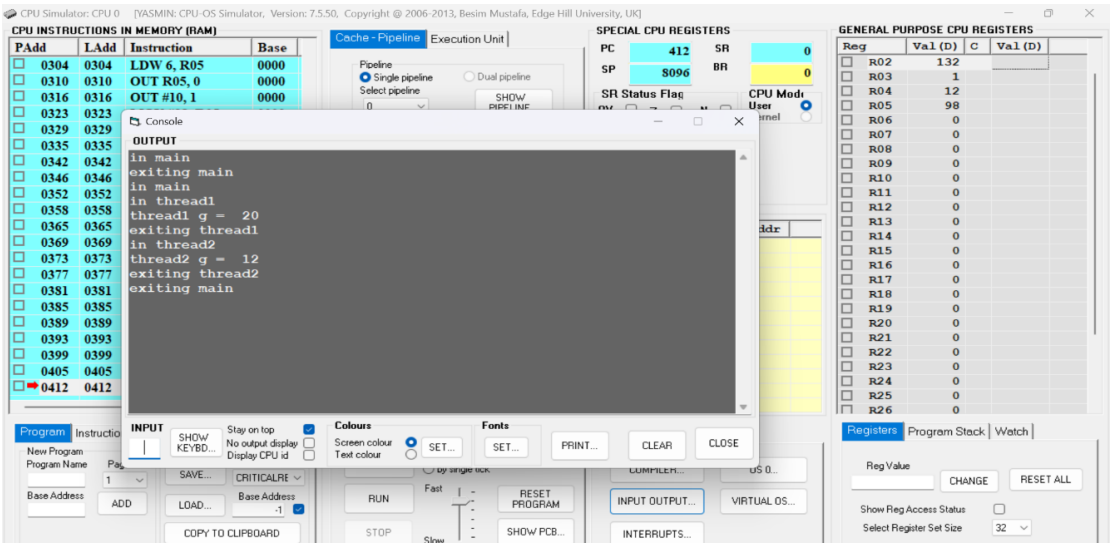
1.



Kode:
https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%204/Praktikum1.txt

Penjelasan:
program **CriticalRegion** menunjukkan bagaimana variabel global g digunakan secara tidak sinkron oleh dua thread, yaitu thread1 dan thread2. Dalam program ini, thread1 dan thread2 secara bergantian mengatur dan memodifikasi nilai g. Ketika thread1 dijalankan, variabel g diatur menjadi 0 dan kemudian ditambah sebanyak 20 kali, tetapi hasil akhirnya menunjukkan nilai g = 25, yang mengindikasikan bahwa ada gangguan dari eksekusi thread2. Hal serupa terjadi saat thread2 berjalan, di mana g diatur kembali menjadi 0 dan ditambah sebanyak 12 kali, tetapi hasil akhirnya menjadi g = 18, yang juga dipengaruhi oleh aktivitas thread1. Ini terjadi karena kedua thread mengakses variabel g tanpa mekanisme sinkronisasi, sehingga perubahan pada g yang dilakukan oleh satu thread dapat saling tumpang tindih dengan thread lainnya.

2.

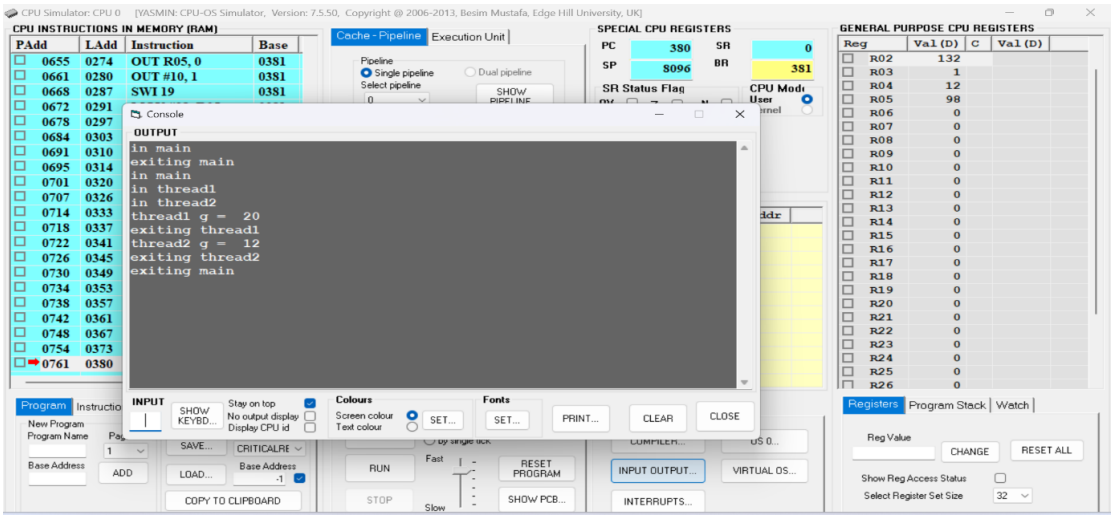


Kode:
https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%204/Praktikum2.txt

Penjelasan:
program memiliki dua thread (thread1 dan thread2) yang masing-masing melakukan perhitungan sederhana, kemudian mencetak nilai variabel g. Fungsi thread1 memulai dengan mencetak pesan bahwa eksekusi telah dimulai, lalu menginisialisasi g dengan nilai nol, menjalankan perulangan sebanyak 20 iterasi, dan meningkatkan nilai g di setiap iterasi hingga nilainya mencapai 20. Setelah selesai, thread ini mencetak nilai akhir g dan pesan bahwa eksekusi thread telah selesai. Hal yang serupa terjadi pada thread2, tetapi perulangan hanya dilakukan sebanyak 12 iterasi, sehingga nilai akhirnya adalah 12.

Dalam simulasi ini, thread pertama dan kedua dipanggil secara berurutan dari fungsi utama, namun dieksekusi sebagai thread terpisah dengan menggunakan instruksi call thread1 dan call thread2. Perintah wait memastikan bahwa fungsi utama menunggu semua thread selesai sebelum melanjutkan eksekusi dan mencetak pesan bahwa eksekusi utama telah selesai. Hasil keluaran menunjukkan alur eksekusi ini dengan jelas, termasuk nilai-nilai akhir g untuk masing-masing thread, diikuti dengan pesan akhir dari fungsi utama

3.



Kode:
https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%204/Praktikum3.txt

Penjelasan:
Program dimulai dengan thread utama yang mencetak "in main", kemudian memanggil dua thread, yaitu thread1 dan thread2, secara berurutan. Thread1 masuk ke critical region menggunakan perintah enter, menginisialisasi g ke 0, dan menambah nilainya sebanyak 20 kali dalam sebuah loop, sehingga menghasilkan nilai akhir g = 20. Setelah mencetak hasilnya, thread1 keluar dari critical region dengan perintah leave dan mencetak "exiting thread1". Proses serupa dilakukan oleh thread2, di mana ia juga masuk ke critical region, menginisialisasi ulang g ke 0, dan menambah nilainya sebanyak 12 kali dalam loop, menghasilkan nilai akhir g = 12. Setelah mencetak hasilnya, thread2 keluar dari critical region dan mencetak "exiting thread2". Pada akhirnya, setelah kedua thread selesai dieksekusi, thread utama mencetak "exiting main" sebelum program berakhir. Mekanisme critical region memastikan bahwa kedua thread mengakses variabel g secara eksklusif tanpa adanya konflik, sehingga menghindari race condition dan menjamin konsistensi hasil.

Jawablah pertanyaan berikut ini:

1. **Apa tujuan utama dari praktikum ini?**
    - **Jawab:** Tujuan utama praktikum ini adalah untuk memahami bagaimana cara kerja sinkronisasi dalam program komputer dan masalah yang bisa muncul ketika beberapa bagian program mencoba mengakses data yang sama secara bersamaan (yang disebut deadlocks).
2. **Mengapa kita menggunakan variabel global yang sama pada dua thread?**
    - **Jawab:** Kita menggunakan variabel yang sama untuk melihat apa yang terjadi ketika dua bagian program (thread) mencoba mengakses data yang sama. Jika tidak ada pengaturan yang baik, bisa terjadi masalah yang disebut race condition, di mana hasilnya tidak bisa diprediksi.
3. **Apakah kita sudah menggunakan variabel lokal, dan apakah hasilnya berbeda?**

- **Jawab:** Ya, jika setiap thread menggunakan variabelnya sendiri, maka tidak akan ada masalah karena masing-masing thread tidak akan saling mengganggu.

4. **Apa tujuan dari menambahkan kata kunci 'synchronized'?**

    - **Jawab:** Tujuannya adalah untuk mengatur agar hanya satu thread yang bisa mengakses variabel pada satu waktu. Contohnya, dalam bahasa pemrograman Java, kita menggunakan kata kunci 'synchronized' untuk tujuan ini.

5. **Apa fungsi dari kata kunci 'enter' dan 'leave'?**

    - **Jawab:** Kata kunci ini digunakan untuk menandai bagian dari kode yang hanya boleh diakses oleh satu thread pada satu waktu. Ini berbeda dari 'synchronized' yang lebih otomatis dalam mengatur akses.

6. **Apa itu semaphore dan mutex, dan apa perbedaannya?**

    - **Jawab:** Semaphore adalah alat untuk mengontrol akses ke sumber daya dengan memberikan izin kepada beberapa thread sekaligus. Sedangkan mutex adalah kunci yang hanya membolehkan satu thread untuk mengakses suatu bagian dari kode pada satu waktu.

7. **Berikan contoh nyata untuk critical region dan mutex.**

    - **Jawab:** Contoh critical region adalah saat beberapa pengguna mengakses database yang sama di server. Contoh mutex adalah saat kita mengamankan akses ke file yang sama di sistem operasi agar tidak ada dua program yang membukanya sekaligus.

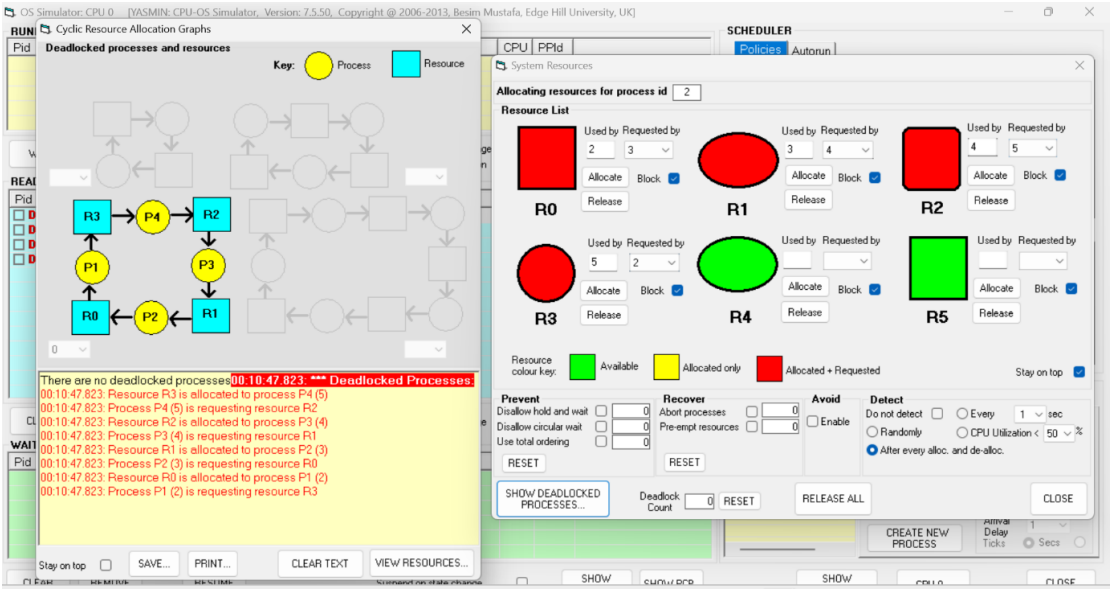
8. **Apa itu instruksi "test-and-set" dan bagaimana cara kerjanya?**

    - **Jawab:** Instruksi ini adalah cara di tingkat perangkat keras untuk memastikan hanya satu thread yang bisa mengubah nilai variabel pada satu waktu, sehingga menghindari masalah race condition. Jika tidak ada dukungan dari perangkat keras atau sistem operasi, kita bisa menggunakan teknik lain seperti menggunakan flag untuk menandai apakah suatu bagian dari kode sedang digunakan.

9. **Apa saran Anda dan bagaimana perbedaannya dengan metode yang sudah dicoba sebelumnya?**

    - **Jawab:** Saya sarankan menggunakan flag untuk menandai bagian yang sedang diakses, tetapi metode ini kurang efisien dan tidak selalu aman jika ada banyak thread yang beroperasi sekaligus. Metode sebelumnya lebih terstruktur dan aman dalam mengelola akses data.

4.



Kode:
https://github.com/MythEclipse/Praktikum-Sistem-Operasi/blob/main/Modul%204/Praktikum4.txt

Penjelasan:
Terdapat siklus ketergantungan dimana proses P1 membutuhkan sumber daya R3, yang sedang dipegang oleh proses P4. Proses P4 membutuhkan R2, yang dipegang oleh P3. P3 membutuhkan R1, yang dipegang P2. Dan P2 membutuhkan R0, yang dipegang oleh P1. Siklus ini menciptakan kebuntuan, dimana tidak ada proses yang dapat melanjutkan karena menunggu sumber daya yang dipegang oleh proses lain dalam siklus tersebut. Status deadlock ini ditampilkan pada log di bagian bawah simulator, dan grafik alokasi sumber daya secara visual menggambarkan ketergantungan siklik antara proses dan sumber daya

PostTest

1. Perangkat keras menyediakan instruksi atomik (misalnya, test-and-set, compare-and-swap) dan mekanisme locking (misalnya, mutex, semaphore) yang memungkinkan sinkronisasi antar thread.

2. Sinkronisasi mencegah *race condition* saat beberapa thread mengakses dan memodifikasi data global bersamaan, memastikan konsistensi dan integritas data.

Tugas

1. Simulasi deadlock menunjukkan bahwa ketika beberapa proses bersaing dan saling menahan sumber daya yang dibutuhkan oleh satu sama lain, terjadi kebuntuan (deadlock) yang menghentikan eksekusi. Kesimpulannya, sinkronisasi yang tidak tepat dapat menyebabkan deadlock, sehingga penting untuk menerapkan strategi pencegahan atau penanganan deadlock untuk menjamin stabilitas sistem.

Kesimpulan

Praktikum ini mengeksplorasi pentingnya sinkronisasi dalam sistem operasi, khususnya saat beberapa thread mengakses sumber daya bersama. Tanpa sinkronisasi yang tepat, kondisi balapan (race condition) dapat terjadi, yang mengarah pada hasil yang tidak konsisten dan tidak dapat diprediksi, seperti yang ditunjukkan dalam praktikum ketika beberapa thread memanipulasi variabel global yang sama secara bersamaan. Untuk mencegah masalah ini, berbagai mekanisme sinkronisasi seperti critical region, mutex, semaphore, dan instruksi atomik dapat diterapkan. Critical region, yang disimulasikan menggunakan 'enter' dan 'leave' dalam praktikum, membatasi akses ke bagian kode kritis hanya untuk satu thread pada satu waktu. Mutex berfungsi serupa, bertindak sebagai kunci yang hanya dapat dipegang oleh satu thread. Semaphore memungkinkan akses bersamaan yang lebih terkontrol, sedangkan instruksi atomik menyediakan operasi tingkat perangkat keras yang tidak dapat diinterupsi. Selain race condition, sinkronisasi yang tidak tepat juga dapat menyebabkan deadlock, di mana dua atau lebih proses saling menunggu satu sama lain untuk melepaskan sumber daya yang diperlukan, sehingga menciptakan kebuntuan. Simulasi deadlock dalam praktikum mengilustrasikan skenario ini, menekankan pentingnya perencanaan dan implementasi sinkronisasi yang cermat. Praktikum juga menyoroti perbedaan antara variabel lokal, yang aman dari race condition karena dimiliki oleh masing-masing thread, dan variabel global, yang memerlukan sinkronisasi untuk akses yang aman. Kesimpulannya, praktikum menunjukkan bahwa sinkronisasi yang efektif sangat penting untuk membangun sistem multithreaded yang andal dan efisien, mencegah masalah seperti race condition dan deadlock, serta memastikan integritas data.