LAPORAN PRAKTIKUM

SISTEM OPERASI

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

MODUL 4

DISUSUN OLEH :

NAMA : MOHAMAD ABAN SY'BANA

NIM : 20230810012

KELAS : TINFC-2023-04

TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS KUNINGAN

2024

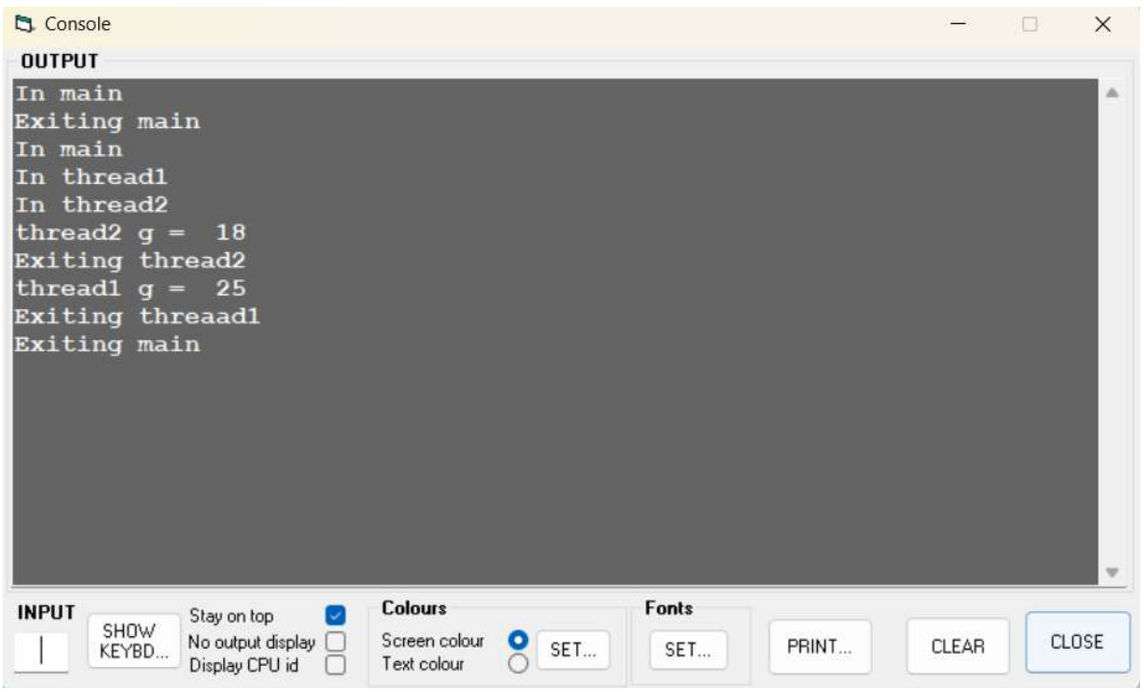
PRE-TEST

1. Apa tujuan dari sinkronisasi proses dalam sistem operasi ?
   Jawab :
   - **Menghindari Deadlock**: Memastikan bahwa proses tidak terjebak dalam keadaan di mana mereka saling menunggu sumber daya yang dipegang oleh proses lain, yang dapat menyebabkan sistem menjadi tidak responsif.
   - **Penggunaan Sumber Daya yang Efisien**: Memastikan bahwa sumber daya sistem digunakan dengan cara yang efisien dan tidak ada yang terbuang atau tidak terpakai karena konflik.

2. Masalah apa yang bisa terjadi ketika beberapa thread mengakses data bersama tanpa sinkronisasi ?
   Jawab :
   - **Deadlock**: Proses atau thread dapat saling menunggu untuk sumber daya yang dipegang oleh satu sama lain, menyebabkan sistem menjadi tidak responsif.
   - **Race Condition**: Ketika beberapa thread mengakses dan mengubah data bersama secara bersamaan, hasil akhir dapat berbeda tergantung pada urutan eksekusi. Hal ini dapat menyebabkan hasil yang tidak diinginkan atau tidak konsisten.
   - **Resource Starvation**: Beberapa thread mungkin tidak pernah mendapatkan akses ke sumber daya yang mereka butuhkan karena sumber daya tersebut selalu diambil oleh thread lain.

PRAKTIKUM 1

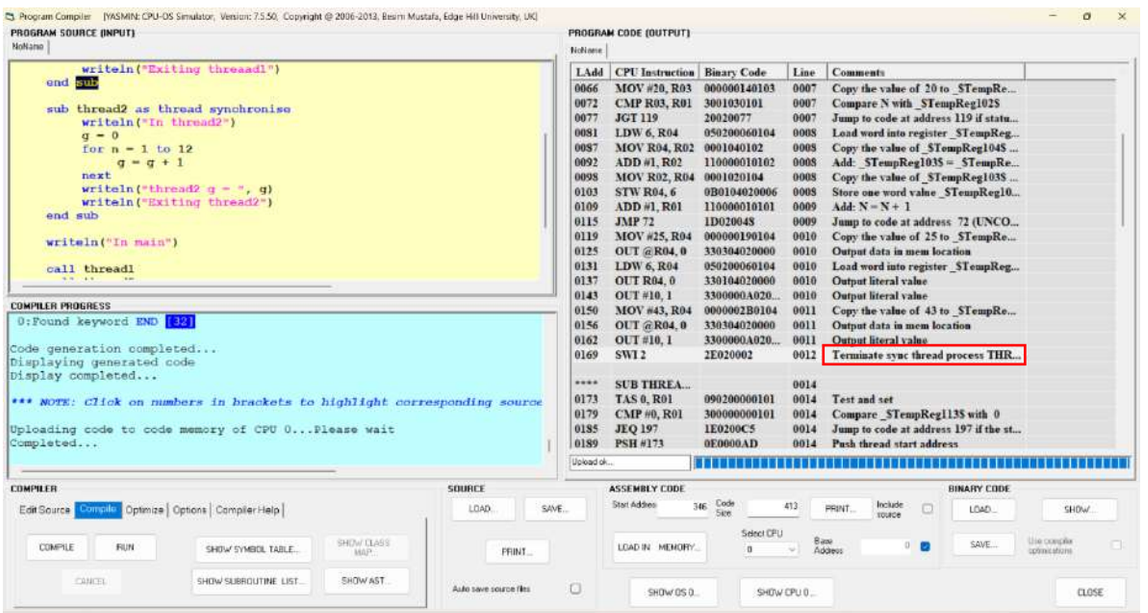
```
program CriticalRegion
	var g integer
	sub thread1 as thread
		writeln("In thread1")
		g = 0
		for n = 1 to 20
			g = g + 1
		next
		writeln("thread1 g = ", g)
		writeln("Exiting threaad1")
	end sub
	sub thread2 as thread
		writeln("In thread2")
		g = 0
		for n = 1 to 12
			g = g + 1
		next
		writeln("thread2 g = ", g)
		writeln("Exiting thread2")
	end sub
	writeln("In main")
	call thread1
	call thread2
	wait
	writeln("Exiting main")
end
```

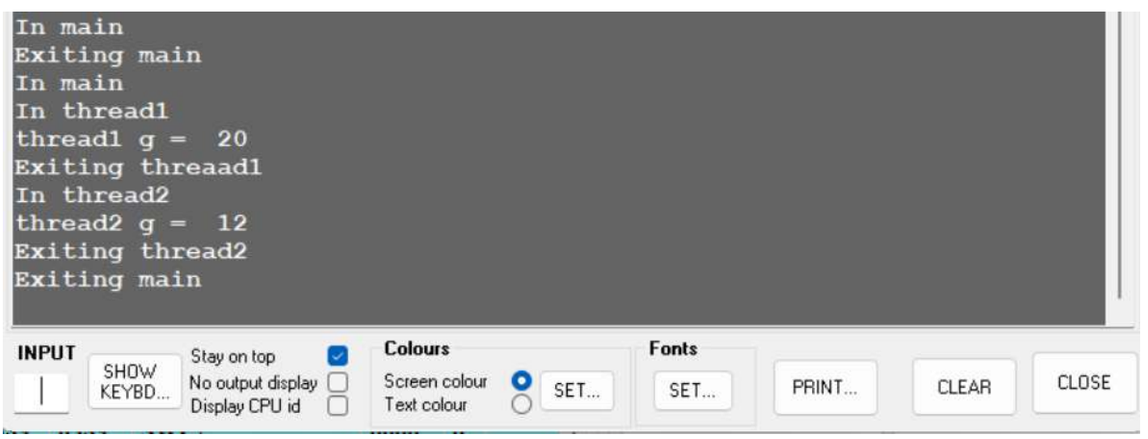


PRAKTIKUM 2

```
program CriticalRegion
    var g integer

    sub thread1 as thread synchronise
        writeln("In thread1")
        g = 0
        for n = 1 to 20
            g = g + 1
        next
        writeln("thread1 g = ", g)
        writeln("Exiting threaad1")
    end sub

    sub thread2 as thread synchronise
        writeln("In thread2")
        g = 0
        for n = 1 to 12
            g = g + 1
        next
```

```
        writeln("thread2 g = ", g)
        writeln("Exiting thread2")
    end sub

    writeln("In main")

    call thread1
    call thread2

    wait

    writeln("Exiting main")
end
```



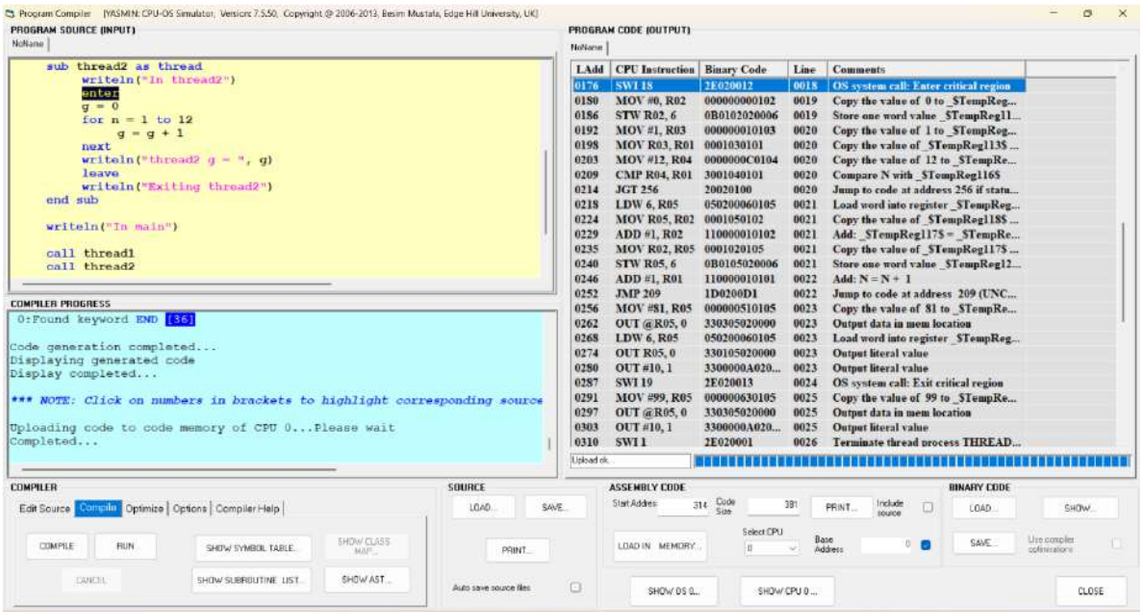


PRAKTIKUM 3

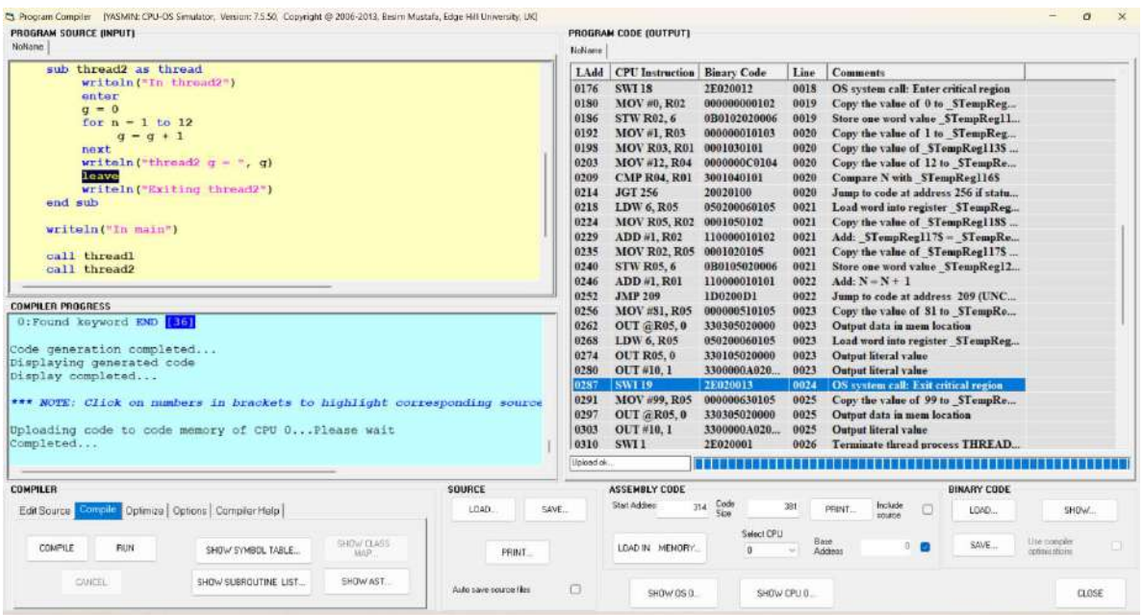
```
program CriticalRegion
	var g integer

	sub thread1 as thread
		writeln("In thread1")
		enter
		g = 0
		for n = 1 to 20
			g = g + 1
		next
		writeln("thread1 g = ", g)
		leave
		writeln("Exiting threaad1")
	end sub

	sub thread2 as thread
		writeln("In thread2")
		enter
		g = 0
		for n = 1 to 12
			g = g + 1
		next
		writeln("thread2 g = ", g)
		leave
		writeln("Exiting thread2")
	end sub

	writeln("In main")

	call thread1
	call thread2

	wait
```

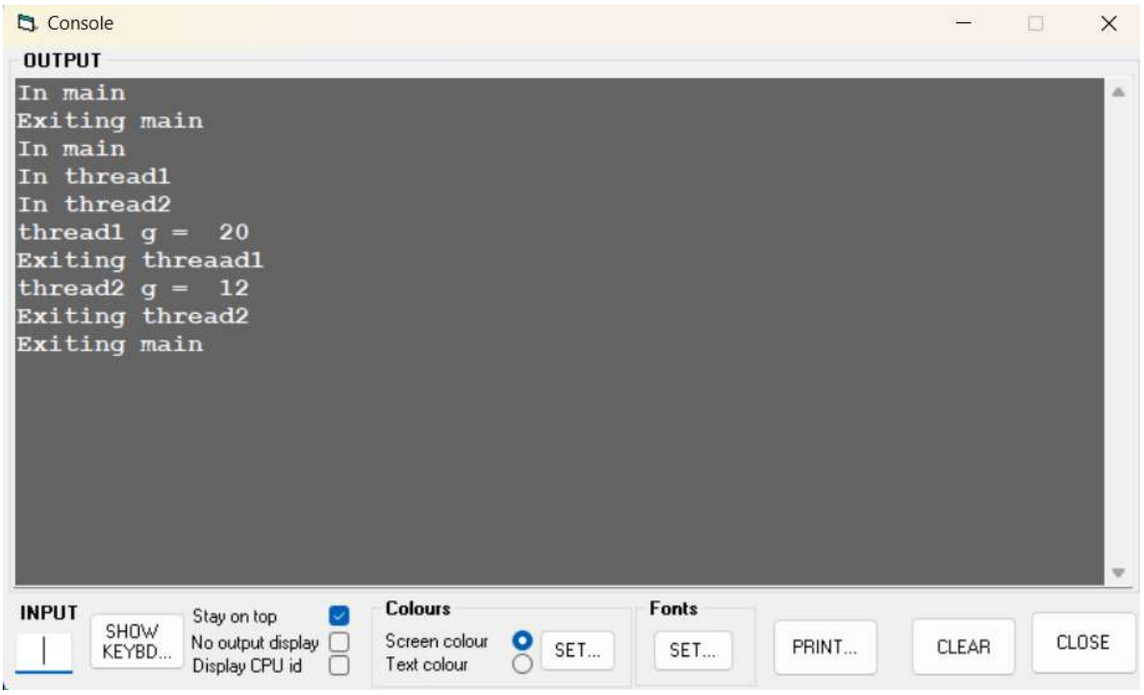
```
        writeln("Exiting main")
end
```

Kompilasi perintah enter



Kompilasi perintah leave





Jadi apa kesimpulannya? Untuk memhami percobaan di atas jawablah, pertanyaan pertanyaan berikut ini:

1. Jelaskan tujuan utama praktimum ini menurut anda.
   Jawab :
   - Memahami konsep Critical Region dalam pengelolaan variabel bersama (shared resources) pada sistem multithreading.
   - Mempelajari mekanisme sinkronisasi seperti enter dan leave, serta bagaimana menghindari race conditions.
   - Mengeksplorasi penggunaan variabel global dan lokal dalam konteks thread.
   - Memahami penerapan semaphore, mutex, dan teknik lainnya untuk melindungi critical region.

2. Mengapa kita menggunakan variabel global (g) yang sama pada dua thread?
   Jawab :
   - Mengilustrasikan potensi masalah seperti race condition, di mana dua thread mengakses variabel secara bersamaan tanpa mekanisme sinkronisasi yang tepat, sehingga menghasilkan hasil yang tidak diinginkan.

3. Sudahkan kita menggunakan variabel lokal, dan apakah hasilnya akan berbeda? Lakukan eksperimen kecil dengan sedikit modifikasi pada kode yang ada dan jalankanlah program tersebut
   Jawab :
   - Ya Hasilnya akan berbeda, karena tidak ada interaksi antara thread pada level variabel, sehingga tidak ada konflik dalam memodifikasi nilai variabel.

4. Pada modifikasi yang pertama ditambahkan kata kunci synchronise. Jelaskan tujuan modifikasi ini. Beri contoh istilah untuk synchronisedalam bahasa pemrograman nyata.
   Jawab :
   - Synchronise digunakan untuk memastikan bahwa akses ke resource yang dibagikan (dalam hal ini g) dilakukan secara aman dengan mencegah dua thread atau lebih menjalankan kode critical section secara bersamaan.
     Contohnya
     Public synchronized void incrementglobal() {

```
    G++;
}
```

5. Pada modifikasi yang kedua digunakan kata kunci enter dan leave. Jelaskan fungsi modifikasi ini, dan apa bedanya dengan (d)?
   Jawab :
   - Synchronise lebih high-level dan otomatis dalam banyak bahasa, sementara enter dan leave memerlukan kontrol manual dan lebih fleksibel untuk implementasi custom.

6. Critical regions seringkali diimplementasikan menggunakan semaphore dan mutex. Jelaskan pengertiannya dan apa perbedaannya.
   Jawab :
   - Semaphore: Mekanisme sinkronisasi yang bisa mengizinkan lebih dari satu thread untuk mengakses resource.
   - Mutex: Mekanisme sinkronisasi yang hanya mengizinkan satu thread pada satu waktu untuk mengakses resource.
     Perbedaanya
     - Semaphore dapat memiliki nilai >1 (mengizinkan akses bersama).
     - Mutex hanya untuk eksklusivitas (nilai 0 atau 1).

7. Berikan contoh nyata untuk suatu critical region (atau mutex region). Beri contoh nyata suatu mutex.
   Jawab :
   - Contoh Critical Region: Database Transaction, Dua transaksi tidak boleh membaca dan menulis ke data yang sama secara bersamaan.
   - Contoh Mutex: Penguncian printer di jaringan, Hanya satu pengguna yang dapat mengakses printer pada satu waktu.

8. Beberapa arsitektur komputer memiliki instruksi "test-and-set" untuk menerapkan critical region. Jelaskan bagaimana teknik ini bekerja dan mengapa penerapannya pada hardware.
   Jawab :
   - Instruksi test-and-set: Atomik, mengecek apakah variabel tertentu sudah diatur, lalu mengatur nilainya jika belum.

9. Jika hardware maupun OS tidak menyediakan bantuan, bagaimana anda dapat memproteksi critical region dalam kode anda? Berikan saran anda dan jelaskan dimana perbedaannya dengan metode-metode yang telah diuji coba kan diatas
   Jawab :
   - Gunakan polling dan busy-waiting loop (kurang efisien, tetapi efektif jika diperlukan).

PRAKTIKUM 4

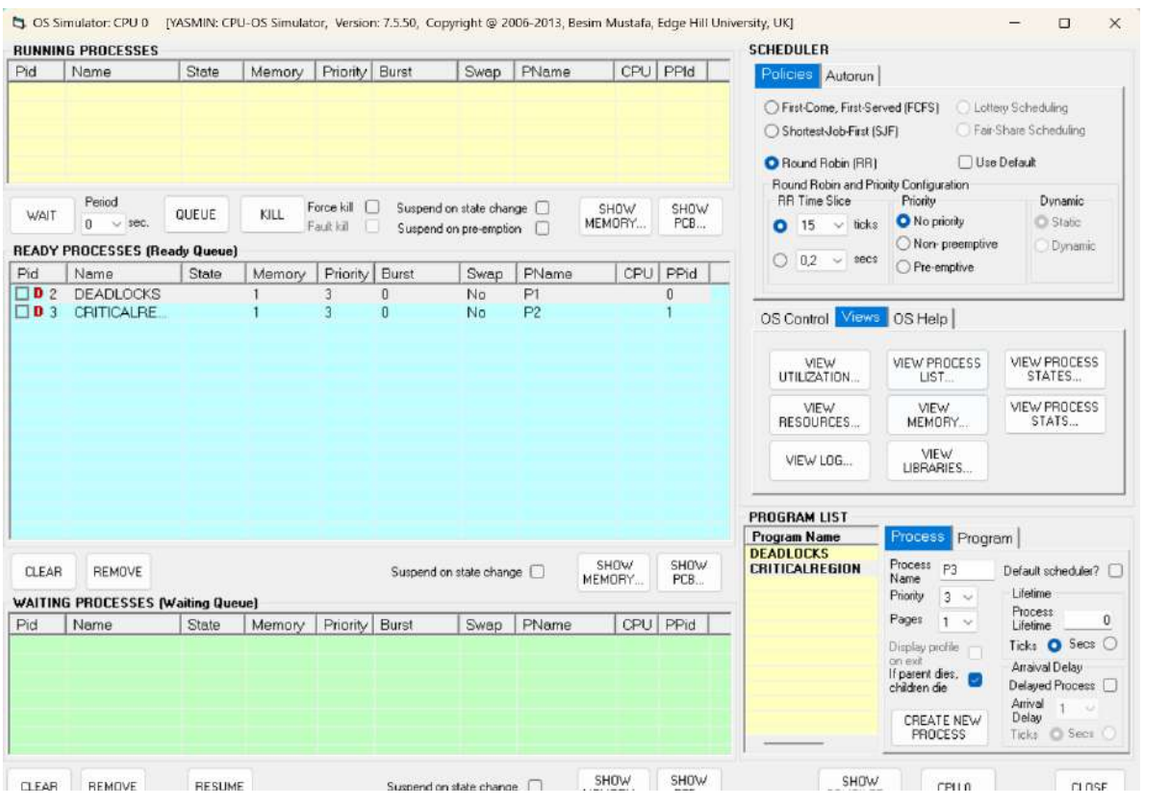
program Deadlocks

while true

n=1

wend

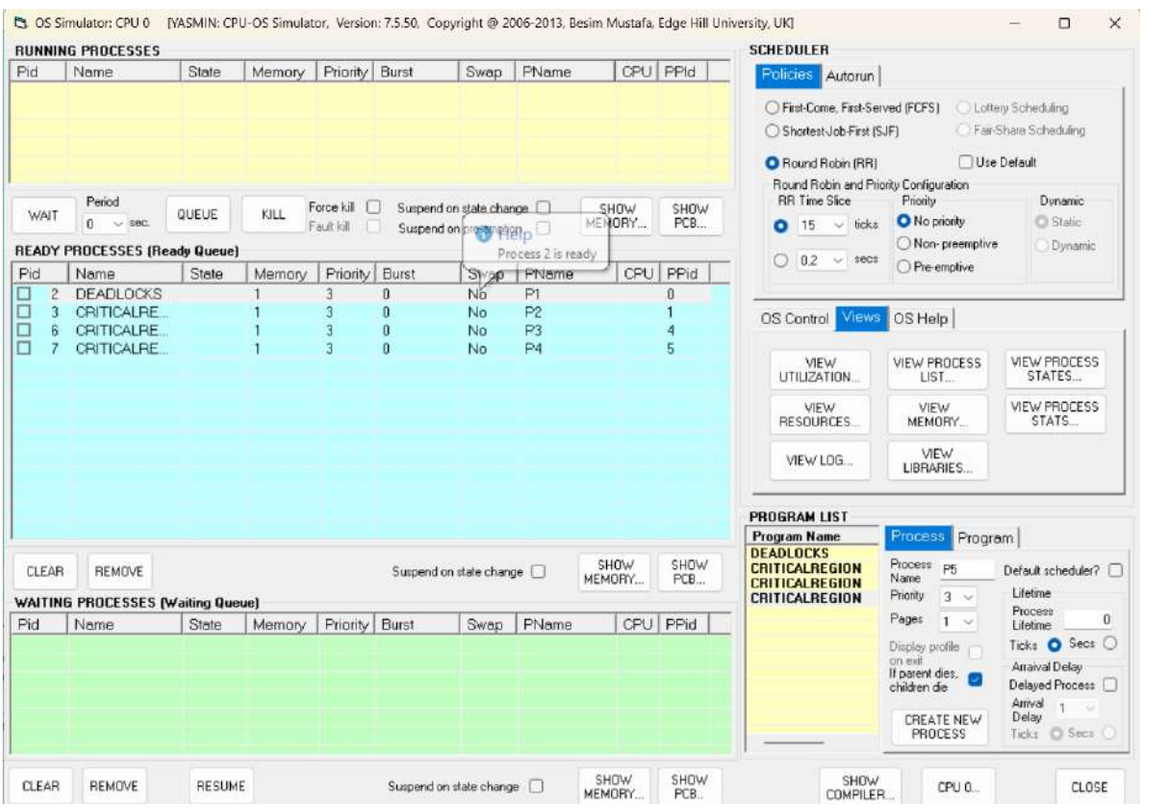
end

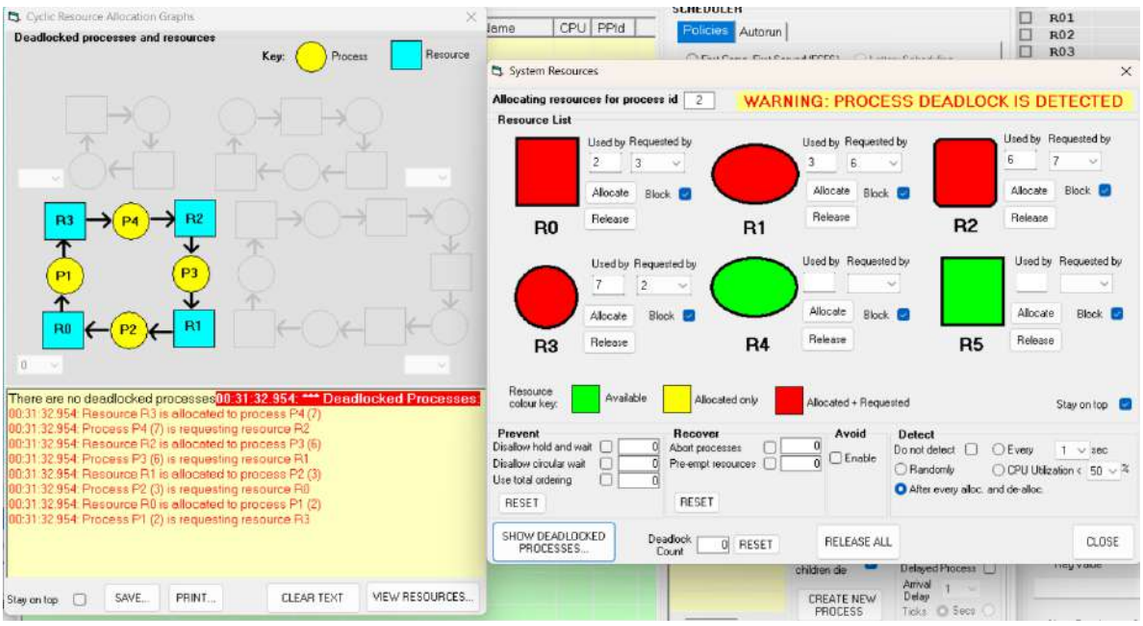


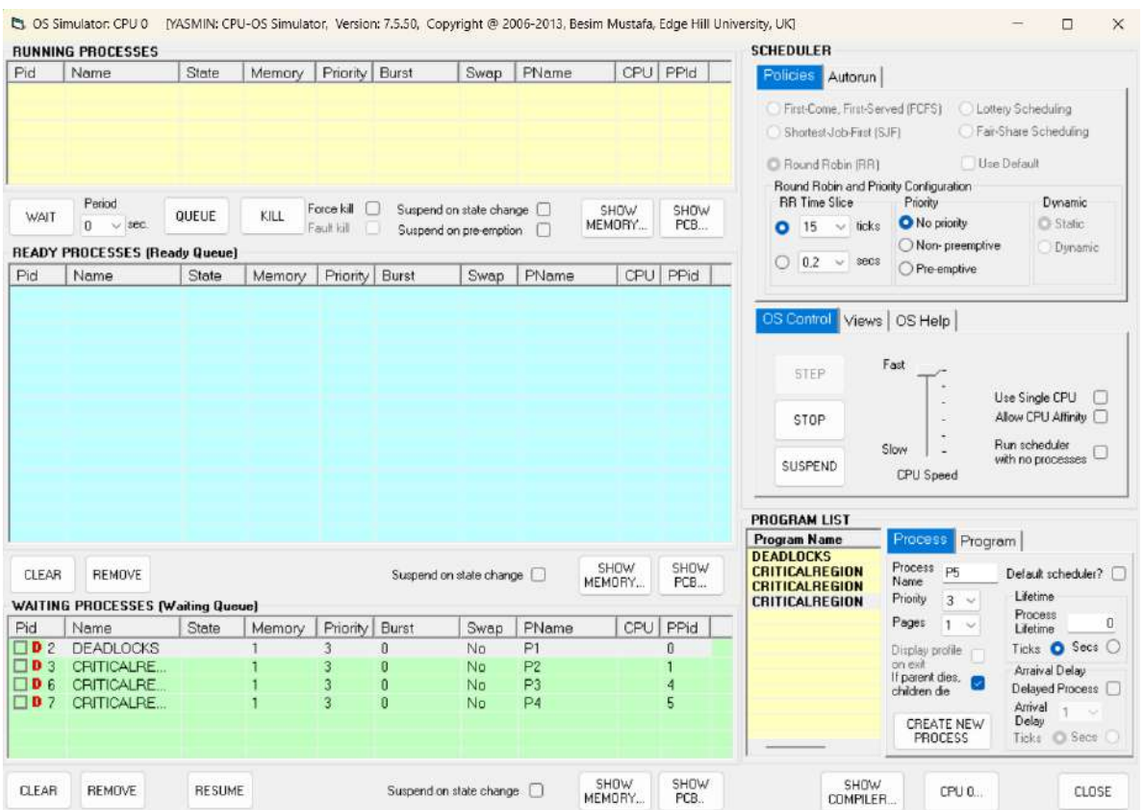
Buatlah 4 proses di sinii saya menggunakan program deadlock yang tadi dan 3 praktikum yang ada di atas

- Sekali deadlock terbentuk, cobalah menjalankan proses dengan klik pada tombol START. Jelaskan yang terjadi. Proses pada antrian ready belum mengalami deadlock, kejadian tersebut terjadi ketika proses telah berjalan ?
  Jawab :
  Setelah deadlock terbentuk, jika Anda menekan tombol START, sistem tidak dapat melanjutkan eksekusi proses yang terlibat dalam deadlock. Hal ini terjadi karena semua proses yang sedang deadlock saling menunggu sumber daya yang tidak tersedia. Pada simulasi ini, antrian proses READY mungkin tetap ada tetapi proses tidak berjalan karena masing-masing saling menunggu, dan waktu simulasi berhenti.

- Dalam kondisi proses yang deadlock, apa yang dapat anda lakukan untuk mengatasinya? Berikan dua cara untuk mengatasinya, kemudian praktikkan menggunakan simulator
  Jawab :
  - Salah satu cara untuk mengatasi deadlock adalah dengan menghentikan proses (kill di bagian ready queue)yang terlibat dalam deadlock secara paksa (abort). Dengan begitu, sumber daya yang mereka pegang dilepaskan dan dapat digunakan oleh proses lain.
  - Cara kedua adalah dengan (Klik Release) sumber daya yang dipegang oleh proses tertentu dan memberikannya kepada proses lain yang membutuhkannya.




Cyclic Resource Allocation Graphs
Deadlocked processes and resources
Key: Process Resource
R3 P4 R2 P1 P3 R0 P2 R1
There are no deadlocked processes
00:31:32.954: *** Deadlocked Processes:
00:31:32.954: Resource R3 is allocated to process P4 (7)
00:31:32.954: Process P4 (7) is requesting resource R2
00:31:32.954: Resource R2 is allocated to process P3 (6)
00:31:32.954: Process P3 (6) is requesting resource R1
00:31:32.954: Resource R1 is allocated to process P2 (3)
00:31:32.954: Process P2 (3) is requesting resource R0
00:31:32.954: Resource R0 is allocated to process P1 (2)
00:31:32.954: Process P1 (2) is requesting resource R3
Stay on top
SAVE...
PRINT...
CLEAR TEXT
VIEW RESOURCES...
SCHEDULER
Policies
Autorun
R01
R02
R03
System Resources
Allocating resources for process id
2
WARNING: PROCESS DEADLOCK IS DETECTED
Resource List
R0
R1
R2
R3
R4
R5
Used by
Requested by
Allocate
Block
Release
Resource colour key:
Available
Allocated only
Allocated + Requested
Stay on top
Prevent
Disallow hold and wait
Disallow circular wait
Use total ordering
RESET
Recover
Abort processes
Pre-empt resources
Avoid
Enable
Detect
Do not detect
Every
sec
Randomly
CPU Utilization <
After every alloc. and de-alloc.
SHOW DEADLOCKED PROCESSES...
Deadlock Count
RELEASE ALL
CLOSE
CREATE NEW PROCESS
Delayed Process
Arrival Delay
Ticks
Secs


OS Simulator: CPU 0 [YASMIN: CPU-OS Simulator, Version: 7.5.50, Copyright @ 2006-2013, Besim Mustafa, Edge Hill University, UK]
RUNNING PROCESSES
Pid Name State Memory Priority Burst Swap PName CPU PPid
WAIT
Period
sec.
QUEUE
KILL
Force kill
Fault kill
Suspend on state change
Suspend on pre-emption
SHOW MEMORY...
SHOW PCB...
READY PROCESSES (Ready Queue)
Pid Name State Memory Priority Burst Swap PName CPU PPid
CLEAR
REMOVE
Suspend on state change
SHOW MEMORY...
SHOW PCB...
WAITING PROCESSES (Waiting Queue)
Pid Name State Memory Priority Burst Swap PName CPU PPid
D 2 DEADLOCKS 1 3 0 No P1 0
D 3 CRITICALRE... 1 3 0 No P2 1
D 6 CRITICALRE... 1 3 0 No P3 4
D 7 CRITICALRE... 1 3 0 No P4 5
CLEAR
REMOVE
RESUME
Suspend on state change
SHOW MEMORY...
SHOW PCB...
SCHEDULER
Policies
Autorun
First-Come, First-Served (FCFS)
Lottery Scheduling
Shortest-Job-First (SJF)
Fair-Share Scheduling
Round Robin (RR)
Use Default
Round Robin and Priority Configuration
RR Time Slice
15
ticks
0.2
secs
Priority
No priority
Non- preemptive
Pre-emptive
Dynamic
Static
Dynamic
OS Control
Views
OS Help
STEP
STOP
SUSPEND
Fast
Slow
CPU Speed
Use Single CPU
Allow CPU Affinity
Run scheduler with no processes
PROGRAM LIST
Program Name
DEADLOCKS
CRITICALREGION
CRITICALREGION
CRITICALREGION
Process
Program
Process Name
P5
Default scheduler?
Priority
3
Pages
1
Lifetime
Process Lifetime
0
Ticks
Secs
Display profile on exit
If parent dies, children die
Arraival Delay
Delayed Process
Arrival Delay
Ticks
Secs
CREATE NEW PROCESS
SHOW COMPILER...
CPU 0...
CLOSE

POSTES

1. Jelaskan peran perangkat keras dalam mendukung mekanisme sinkronisasi ?
   Jawab :

   - Peran perangkat keras dalam mendukung mekanisme sinkronisasi adalah menyediakan instruksi dan fitur yang memungkinkan pengelolaan akses data bersama secara aman dan efisien. Misalnya, perangkat keras modern mendukung instruksi atomik seperti *compare-and-swap* dan *test-and-set*, yang memungkinkan operasi dijalankan secara penuh tanpa interupsi oleh thread lain, sehingga mencegah konflik data.
   - Koherensi cache juga diterapkan untuk memastikan bahwa setiap inti prosesor memiliki salinan data yang konsisten saat data tersebut diakses oleh beberapa thread.
   - Selain itu, perangkat keras menyediakan dukungan untuk mekanisme locking, seperti mutex di level perangkat keras, yang memungkinkan penguncian data secara eksklusif agar tidak diakses bersamaan oleh thread lain. Fitur-fitur ini menjadi dasar penting bagi mekanisme sinkronisasi, membantu menjaga integritas data dalam lingkungan komputasi multithreading.

2. Mengapa penting untuk menerapkan sinkronisasi saat thread mengakses data global ?
   Jawab :
   - Sinkronisasi penting saat thread mengakses data global untuk mencegah kondisi balapan (race condition) yang dapat menyebabkan inkonsistensi data. Tanpa sinkronisasi, beberapa thread dapat mencoba mengubah data yang sama secara bersamaan, yang mengakibatkan hasil akhir yang tidak dapat diprediksi. Sinkronisasi juga mencegah deadlock dan starvation dengan memastikan setiap thread mendapat akses yang adil ke sumber daya. Dengan sinkronisasi, aplikasi menjadi lebih andal dan konsisten karena semua thread mengakses data secara teratur dan terkendali.

Tugas

1. Beri penjelasan dan kesimpulan terhadap hasil simulasi deadlock yang telah di lakukan di atas ?
   Jawab :

   Penjelasan :

   Deadlock adalah kondisi di mana dua atau lebih proses saling menunggu sumber daya yang dimiliki oleh proses lain, sehingga tidak ada proses yang dapat melanjutkan eksekusi. Dalam simulasi CPU-OS yang dilakukan, kondisi deadlock terjadi karena proses saling menahan sumber daya (resource) tanpa melepaskannya, sambil menunggu sumber daya lain yang sedang digunakan oleh proses lain.

   Kesimpulan :

   Deadlock dapat terjadi jika semua syarat penyebabnya terpenuhi, yaitu mutual exclusion, hold and wait, no preemption, dan circular wait. Mutual exclusion terjadi ketika suatu sumber daya hanya dapat digunakan oleh satu proses dalam satu waktu. Hold and wait muncul saat proses yang sedang menahan sumber daya menunggu sumber daya lain tanpa melepas sumber daya yang sudah dimiliki. No preemption menyebabkan sumber daya yang sedang digunakan tidak dapat diambil secara paksa. Circular wait terjadi ketika terdapat siklus di mana proses-proses saling menunggu sumber daya satu sama lain, sehingga membentuk lingkaran tanpa akhir. Keempat kondisi ini harus diantisipasi untuk mencegah terjadinya deadlock.