DISCRETE EVENT MODELS

1. **What Is a Discrete Event Model?**
2. **Discrete Event Simulation**
3. **Efficiency of Discrete Event Simulation**
4. **Potential Ambiguities in Discrete Event Simulation**

ما هي "نماذج الأحداث المتقطعة "؟

في عالم الأنظمة المدمجة ، نحن نتعامل مع أنظمة يتغير سلوكها استجابةً لأحداث (Events) معينة تحدث في أزمنة متفرقة ومتقطعة (Discrete)، وليس بشكل مستمر.

لنتخيل ذراع روبوتية (Robotic Arm) مهمتها التقاط أجسام من على حزام ناقل.

- **النظام (System)** : هو الذراع الآلية مع وحدة التحكم (Microcontroller) الخاصة بها.
- **الحالة (State)** : هي الوضع الحالي للنظام في أي لحظة. مثلاً: "الذراع متوقفة"، "الذراع تتحرك نحو الجسم"، "القابض (Gripper) مفتوح"، "القابض مغلق".
- **الحدث (Event)** : هو أي واقعة لحظية قد تُغير حالة النظام. مثلاً :
  - وصول إشارة من حساس الأشعة تحت الحمراء تفيد بوجود جسم أمامه.
  - انتهاء حركة الذراع ووصولها للموقع المطلوب.
  - ضغط المستخدم على زر البدء.

إن هذه الأحداث تقع في نقاط زمنية محددة. بين حدثين، قد لا يتغير شيء جوهري في حالة النظام. هذا هو جوهر "الأحداث المتقطعة". نحن نهتم باللحظات التي يحدث فيها شيء ما.

ما سنغطيه بالتفصيل في هذه المحاضرة:

1- **ما هو نموذج الحدث المتقطع؟**

النموذج هو تبسيط رياضي أو منطقي لنظامنا الحقيقي. سنقوم بتمثيل سلوك النظام كمجموعة من الحالات (States) والانتقالات (Transitions)بينها، والتي تسببها الأحداث(Events) .

مثال تطبيقي : لنمذجة الذراع الآلية، يمكن أن تكون الحالات:

- IDLE: في انتظار جسم.
- MOVING_TO_OBJECT: تتحرك نحو الجسم.
- GRASPING: تغلق القابض.
- MOVING_TO_DROP_ZONE: تتحرك لمكان التفريغ.
- RELEASING: تفتح القابض.

الحدث الذي ينقلنا من IDLE إلى MOVING_TO_OBJECT هو object_detected_event.

2- محاكاة الأحداث المتقطعة

بمجرد أن يكون لدينا "نموذج"، يمكننا عمل "محاكاة" له على الحاسوب. بدلاً من بناء الروبوت وتشغيله لمعرفة كيف سيتصرف، يمكننا تشغيل النموذج البرمجي آلاف المرات في ثوانٍ.

لماذا هذا مفيد؟ لأننا نستطيع اختبار سيناريوهات معقدة. مثلاً: ماذا لو وصل جسم جديد على الحزام الناقل بينما الذراع لا تزال في طريقها لوضع الجسم القديم؟ هل سيتعامل نظامنا مع هذا الموقف بشكل صحيح أم سيحدث خطأ؟ المحاكاة تسمح لنا باكتشاف هذه المشاكل وتصحيحها قبل كتابة الكود النهائي للمتحكم.

3- كفاءة محاكاة الأحداث المتقطعة(Efficiency of Discrete Event Simulation)

عندما يصبح نظامنا معقداً جداً (روبوت فيه عشرات الحساسات والمحركات)، قد تصبح المحاكاة بطيئة جداً. في هذا الجزء، سندرس تقنيات لجعل المحاكاة تعمل بكفاءة وسرعة. سنناقش كيف يمكن إدارة "طابور الأحداث (Event Queue) "وجدولتها زمنياً بأفضل طريقة ممكنة، وهذا أمر حيوي في تصميم أنظمة الزمن الحقيقي(Real-Time Systems) .

4- الغموض المحتمل في المحاكاة(Potential Ambiguities)

هذه نقطة متقدمة ومهمة جداً. ماذا لو حدث حدثان في **نفس اللحظة الزمنية بالضبط**؟

مثال :في نفس الميلي ثانية t=1.54s، وردت إشارتان لوحدة التحكم:

1. إشارة من حساس يفيد بوصول جسم جديد(object_detected_event) .
2. إشارة من مفتاح نهاية الشوط (Limit Switch) تفيد بأن الذراع وصلت لنقطة النهاية(arm_reached_destination_event) .

بأي ترتيب يجب على النظام معالجة هذين الحدثين؟ ترتيب المعالجة قد يغير سلوك الروبوت بالكامل. هل يتجاهل الجسم الجديد لأنه لم يكمل مهمته بعد، أم يغير مساره فوراً؟ هذا "الغموض" يجب حله على مستوى التصميم لضمان عمل النظام بشكل موثوق ومتوقع دائماً.

Discrete Event Models

- **The system is a collection of processes that respond to events.**
- ***Each event carries a time-stamp indicating the time at which the event occurs.***
- ***Time-stamps are totally ordered.***
- **A Discrete Event (DE) simulator maintains a *global event queue sorted by the time-stamps*. The simulator also keeps a single global time.**

هذه الشريحة تشرح "ميكانيكية" عمل نماذج الأحداث المتقطعة.

- **The system is a collection of processes that respond to events**

بدلاً من رؤية النظام المدمج كقطعة واحدة، نقسمه وظيفياً إلى "عمليات (Processes) "مستقلة جزئياً. يمكن التفكير في "العملية" على أنها مهمة أو وظيفة متخصصة داخل الكود البرمجي للنظام.

مثال تطبيقي على روبوت الذراع الآلية :لدينا على الأقل ثلاث عمليات (مهام) رئيسية تعمل داخل المتحكم:

1. **عملية المراقبة (Monitoring Process)** : مهمتها الوحيدة هي قراءة الإشارة من الحساس. عندما تكتشف جسماً، لا تقوم بتحريك الذراع بنفسها، بل تكتفي بتوليد حدث OBJECT_DETECTED_EVENT.
2. **عملية التحكم بالحركة (Motion Control Process)** : هذه العملية تكون في حالة سبات أو انتظار معظم الوقت. لكن عندما تستقبل حدث OBJECT_DETECTED_EVENT، تستيقظ وتبدأ في حساب مسار حركة الذراع وإرسال إشارات PWM للمحركات. وعندما تصل الذراع إلى وجهتها، تقوم بتوليد حدث آخر مثل ARM_IN_POSITION_EVENT.
3. **عملية التحكم بالقابض (Gripper Control Process)** : تنتظر حدثاً مثل ARM_IN_POSITION_EVENT. عندما يصلها، تقوم بتفعيل المحرك الصغير (Servo) الخاص بالقابض لإغلاقه، ثم تولد حدث GRASP_COMPLETE_EVENT.

إذاً، النظام بأكمله هو أوركسترا تعزف فيها هذه العمليات المتخصصة، وتتواصل فيما بينها عن طريق "الأحداث".

- **Each event carries a time-stamp indicating the time at which the event occurs**

الحدث ليس فقط "ماذا" حدث، بل "متى" حدث. الختم الزمني (Time-stamp) هو رقم دقيق يمثل لحظة وقوع الحدث، ويتم قياسه بواسطة ساعة النظام (System Clock) أو مؤقت (Timer) في المتحكم.

مثال:

- `EVENT { type: OBJECT_DETECTED, timestamp: 134.527 ms }`
- `EVENT { type: ARM_IN_POSITION, timestamp: 136.811 ms }`

هذا الختم الزمني هو ما يعطينا القدرة على فهم تسلسل الأحداث بدقة متناهية، وهو أمر لا غنى عنه لتصحيح الأخطاء (Debugging) وضمان عمل النظام بشكل صحيح.

- **Time-stamps are totally ordered**

هذه نقطة تضمن عدم وجود فوضى. معناها بسيط جداً: لأي ختمين زمنيين tA و tB، هناك ثلاث احتمالات فقط لا غير:

1. tA<tB(الحدث A وقع قبل B)
2. tA>tB(الحدث A وقع بعد B)
3. tA=tB(الحدثان وقعا في نفس اللحظة بالضبط - وهذه هي الحالة التي تسبب "الغموض" الذي ذكرناه في الشريحة السابقة وسنتطرق إليه لاحقاً).

لا يمكن أن يكون tA قبل tB وبعده في نفس الوقت. هذا الترتيب الصارم والواضح للزمن هو أساس عمل المحاكاة.

- **A Discrete Event (DE) simulator maintains a global event queue sorted by the time-stamps**

هذه هي آلية عمل المحاكي:

1. **الساعة العامة (Global Time)** : المحاكي لديه متغير واحد اسمه currentTime . هذا المتغير لا يمشي بشكل مستمر مثل ساعة الحائط، بل **يقفز** من زمن حدث إلى زمن الحدث الذي يليه.
2. **طابور الأحداث العام (Global Event Queue)** : هو قائمة تحتوي على **كل الأحداث المستقبلية** التي نعرف أنها ستحدث. هذه القائمة تكون دائماً **مرتبة**، بحيث يكون الحدث ذو الختم الزمني الأبكر في مقدمة الطابور.

سيناريو المحاكاة خطوة بخطوة:

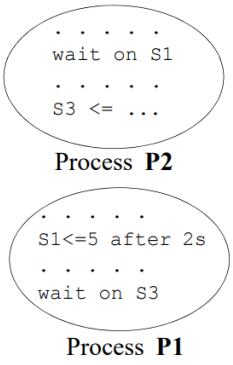
1. **البداية** .currentTime = 0 : الطابور يحتوي على حدث واحد: { type: START_SIMULATION, timestamp: 0 }
2. **الخطوة 1** :المحاكي يسحب أول حدث من الطابور .(START_SIMULATION) يضبط ساعته إلى زمن الحدث currentTime : يصبح 0 . تنفيذ هذا الحدث قد يولد حدثاً مستقبلياً، مثلاً وصول أول جسم بعد 5 ثوانٍ. فيقوم المحاكي بإضافة الحدث { type: OBJECT_DETECTED, timestamp: 5 } إلى الطابور.

3. **الخطوة 2 :** الطابور الآن فيه حدث واحد. المحاكي يسحبه. يقفز بساعته إلى زمن الحدث currentTime :يصبح 5. تنفيذ OBJECT_DETECTED يجعل عملية التحكم بالحركة تقرر أن الحركة ستستغرق 1.5 ثانية. فتضيف حدثاً جديداً للطابور : { type: ARM_IN_POSITION, timestamp: 6.5 }

4. يستمر المحاكي في سحب الحدث التالي، القفز بالزمن إليه، وتنفيذه (مما قد يولد أحداثاً مستقبلية جديدة)، وهكذا ...

المحاكي لا يضيع وقتاً في محاكاة الفترات التي لا يحدث فيها شيء. إنه يقفز مباشرة من حدث مهم إلى الحدث المهم التالي، وهذا ما يجعله فعالاً جداً.

Discrete Event Simulator



هذا المخطط يظهر نظام بسيط جداً يتكون من عمليتين فقط، P1 و P2. يمكن التفكير فيهما كبرنامجين صغيرين أو مهمتين (Tasks) تعملان بالتوازي داخل المتحكم الصغري للروبوت. تتواصل العمليتان عبر إشارتين، هما S1 و S3.

العملية P1 :

لديها سلوكان محددان:

1. **S1 <= 5 after 2s** : عندما يتم تنفيذ هذا السطر، لا يتم تغيير قيمة S1 إلى 5 في الحال، بل تقوم العملية P1 **بجدولة حدث مستقبلي** .إنها تطلب من المحاكي إضافة حدث جديد في طابور الأحداث العام لديه. هذا الحدث سيقوم بجعل قيمة S1 تساوي 5، ولكن لن ينفذه الآن، بل سينفذه بعد ثانيتين (2s) من الزمن الحالي.

2. **wait on S3:** هذا يعني أن العملية P1، بعد أن جدولت حدثها المستقبلي، ستدخل في حالة **انتظار(أو سبات)** . لن تفعل أي شيء آخر حتى تستقبل إشارة أو "تنبيه" على القناة S3.

العملية P2 :

لديها أيضاً سلوكان:

1. **wait on S1** : هذه العملية تبدأ بالانتظار. هي في حالة سبات حتى يحدث تغيير أو تأتيها إشارة على القناة S1.

2. **S3 <= ...** : عندما تستيقظ P2 (بسبب حدوث شيء على S1)، ستقوم هي بدورها بإرسال إشارة أو تغيير قيمة S3

سيناريو محاكاة خطوة بخطوة

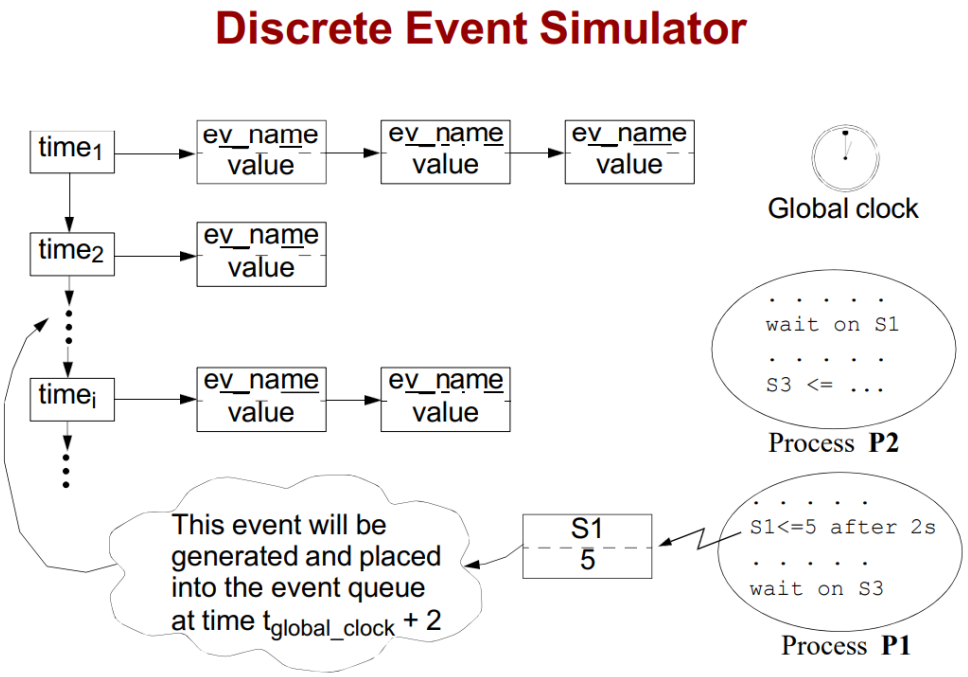
لنتخيل أن المحاكي (Simulator) سيقوم بتشغيل هذا النظام من الزمن t=0

1. **الزمن t = 0 :**
   - لنفترض أن العملية P1 تبدأ أولاً.
   - تنفذ السطر S1 <= 5 after 2s
   - **الإجراء:** P1 تضيف الحدث التالي إلى **طابور الأحداث العام** :
     - Event { target: S1, value: 5, timestamp: 2s }
   - تنتقل P1 للسطر التالي wait on S3، فتتحول حالتها إلى **"منتظرة/محجوبة(Blocked) "**
   - الآن يأتي دور P2 تنفذ أول سطر لديها وهو wait on S1. فتتحول حالتها فوراً إلى **"منتظرة/محجوبة(Blocked) "**
2. **ما بين الزمن t=0 و t=2s**
   - **الساعة العامة للمحاكي=0s**
   - **طابور الأحداث** يحتوي على: [Event(S1=5, timestamp: 2s)]
   - كلتا العمليتين P1 و P2 في حالة سبات. لا يوجد شيء لفعله.
   - المحاكي يرى أن الحدث التالي في الطابور سيقع عند الزمن 2s .لذا، يقوم **بالقفز بالزمن مباشرة** من 0s إلى 2s هذا يوفر كل وقت المعالجة الذي كان سيُهدر في محاكاة "اللاشيء".
3. **الزمن t = 2s:**
   - **الساعة العامة للمحاكي=2s**
   - المحاكي يسحب الحدث Event(S1=5, timestamp: 2s) من الطابور وينفذه.
   - **الإجراء** :قيمة الإشارة S1 تصبح 5
   - المحاكي الآن يسأل: "هل هناك أي عملية كانت تنتظر حدوث شيء على S1؟"
   - **الجواب** :نعم. العملية P2 كانت في حالة wait on S1
   - **الإجراء** :المحاكي يقوم **بإيقاظ** العملية P2
   - تستأنف P2 عملها من حيث توقفت. تنفذ السطر التالي .... => S3 لنفترض أنها تجعل S3 تساوي 10 **فوراً**.
   - **الإجراء** :قيمة الإشارة S3 تصبح 10

- المحاكي يسأل مرة أخرى: "هل هناك أي عملية كانت تنتظر حدوث شيء على S3؟"
- **الجواب** :نعم. العملية P1 كانت في حالة wait on S3
- **الإجراء** :المحاكي يقوم **بإيقاظ** العملية P1

وهكذا تدور الدورة .بعد أن استيقظت P1 قد تعود لتجدول حدثاً جديداً لـ S1 بعد ثانيتين إضافيتين (أي عند t=4s) وتعود للسبات، مما يضمن استمرارية عمل النظام.

الخلاصة :هذا المخطط يوضح كيف أن التفاعلات المعقدة المعتمدة على الزمن في الأنظمة المدمجة يمكن نمذجتها وتنفيذها بكفاءة عن طريق عمليات مستقلة تتواصل عبر طابور أحداث مركزي.



"محاكي الأحداث المتقطعة(Discrete Event Simulator - DES) "

لنتخيل أننا نصمم روبوتاً يجب عليه تنفيذ سلسلة من المهام: ربما يحرك ذراعه، ثم ينتظر أن يستشعر حساس ما جسماً معيناً، ثم يمسك بالجسم، وبعدها يرسل إشارة إلى جزء آخر من النظام. كيف يمكننا اختبار صحة منطق تصميمنا قبل بناء الروبوت الفعلي أو تشغيله في الزمن الحقيقي حيث يمكن للأخطاء أن تكون مكلفة أو حتى خطيرة؟ هنا يأتي دور **محاكي الأحداث المتقطعة**.

يمكن التفكير فيه كبيئة **افتراضية** نستطيع من خلالها "تشغيل" سلوك نظامنا عبر الزمن. الكلمة المفتاح هنا هي" **متقطعة** (discrete) " فبدلاً من التدفق المستمر للزمن الحقيقي، يقفز محاكي الأحداث المتقطعة من "حدث" مهم إلى الحدث الذي يليه مباشرة، مما يجعله فعالاً جداً.

المكونات الرئيسية:

1. **الساعة العالمية(Global Clock) :**

- هي ضابط الوقت الرئيسي للمحاكاة. لا تسير بمعدل ثابت كساعات الحائط. بدلاً من ذلك، تتقدم مباشرة إلى زمن الحدث التالي المجدول. إذا كان الحدث التالي سيقع بعد 5 مللي ثانية، تقفز الساعة 5 مللي ثانية. وإذا كان الحدث الذي يليه سيقع بعد ثانيتين، تقفز الساعة ثانيتين. هذا ما يجعلها "متقطعة".

2. **قائمة الأحداث(Event Queue) :**

- البلوكات المسماة time_1، time_2، ...، time_i، وكل منها يشير إلى كتلة أو أكثر من ev_name value تمثل **قائمة الأحداث** .إنها في جوهرها قائمة مرتبة زمنياً بكل الأحداث المستقبلية المجدولة للحدوث.
- **الحدث (Event)** بحد ذاته هو إشعار بأن شيئاً هاماً قد وقع أو يجب أن يقع. وعادة ما يكون له :
  - **بصمة زمنية** (timestamp) (مثل time_1): متى يقع الحدث.
  - **اسم/نوع الحدث** (event name/type) (مثل ev_name) : ما الذي يحدث (مثلاً:"تم_تنشيط_الحساس"، "توقف_المحرك"، "تغير_الإشارة _S1"
  - اختيارياً، **قيمة** أو بيانات مصاحبة (مثل: القيمة الجديدة لإشارة ما، بيانات الحساس).
- يقوم المحاكي دائماً باختيار الحدث ذي **البصمة الزمنية الأبكر** من هذه القائمة لمعالجته تالياً.

3. **العمليات(P1, P2) :**

- تمثل هذه الأجزاء النشطة المختلفة في نظامنا المدمج. في روبوت ما، يمكن أن تكون P1 هي وحدة التحكم في المحرك، و P2 هي وحدة معالجة بيانات الحساسات.
- العمليات يمكنها :
  - **توليد الأحداث** :التسبب في أشياء جديدة تحدث في المستقبل.
  - **التفاعل مع الأحداث** :تغيير حالتها أو سلوكها عندما يقع حدث تهتم به.
  - **الانتظار لأحداث/شروط معينة** :إيقاف تنفيذها مؤقتاً حتى يحدث شيء محدد (مثلاً "wait on S1" :أي انتظر الإشارةS1).

كيف يعمل النظام:

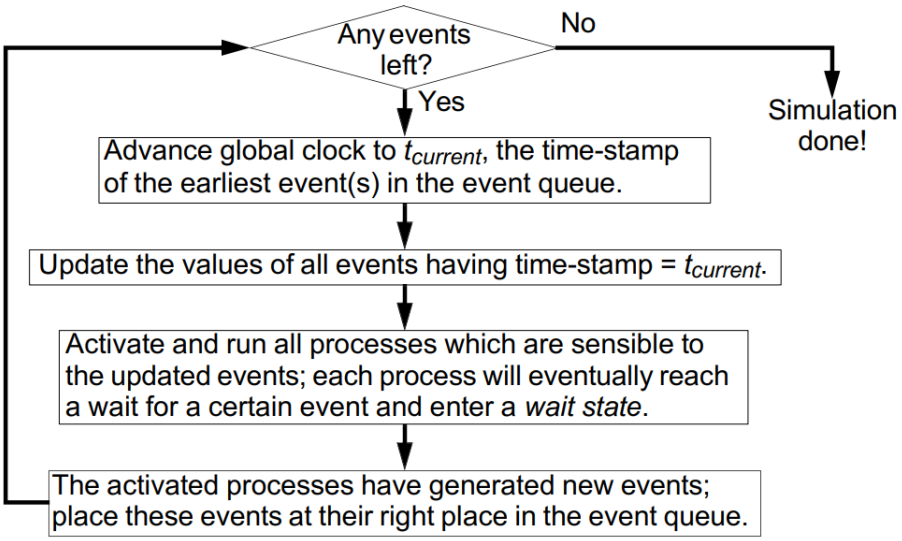
- **العملية P1** تقوم بعمل ما. في لحظة معينة، تقرر تغيير إشارة S1 حيث سيتم إنشاء هذا الحدث ووضعه في قائمة الانتظار في الوقت t_global_clock + 2".
  - لنفترض أن global_clock (الساعة العالمية) تشير حالياً إلى t = 10 ثوانٍ.
  - تقوم P1 بجدولة حدث: "عند الزمن t = 10 + 2 = 12 ثانية، ستأخذ الإشارة S1القيمة 5".
  - هذا الحدث الجديد (5، S1، بصمة زمنية 12s) يتم إدراجه في قائمة الأحداث بالترتيب الزمني الصحيح.
- **العملية P2** تظهر أنها في حالة "wait on S1" (انتظار الإشارة S1). هذا يعني أن P2 متوقفة مؤقتاً، تنتظر حدوث شيء ما مع الإشارة S1.

- عندما تصل الساعة العالمية في النهاية إلى 12ثانية، سيقوم المحاكي بمعالجة الحدث "S1 تصبح 5".
- هذا الحدث على الأرجح "سيوقظ" العملية P2 ، التي يمكنها بعد ذلك قراءة القيمة الجديدة لـ S1 ومتابعة تنفيذها (ربما تقوم بتعديل الإشارة S3، كما هو مُشار إليه بـ ... => S3).

لماذا يعتبر نهج "الأحداث المتقطعة" هذا مفيداً جداً؟

- **الكفاءة** :نحن نحسب الأشياء فقط عند وقوع الأحداث. إذا لم يكن هناك شيء مجدول للحدوث لفترة زمنية افتراضية طويلة، فإن المحاكي ببساطة يقدم الساعة إلى الأمام. هذا أسرع بكثير من محاكاة كل جزء صغير جداً من الثانية في الزمن الحقيقي (المحاكاة المستمرة).
- **تنقيح الأخطاء (Debugging)** : يمكننا التنقل بين الأحداث خطوة بخطوة، وفحص حالة النظام عند كل حدث، وفهم التفاعلات المعقدة التي قد يكون من الصعب اكتشافها في نظام يعمل في الزمن الحقيقي.
- **قابلية إعادة الإنتاج (Reproducibility)** : بما أن المحاكاة مدفوعة بتسلسل محدد من الأحداث، يمكننا إعادة تشغيلها والحصول على نفس النتائج، وهو أمر بالغ الأهمية لتنقيح الأخطاء.
- **نمذجة التزامن (Modeling Concurrency)** : يسمح لنا بنمذجة كيفية تفاعل وتزامن عدة عمليات أو مكونات مادية تعمل بالتوازي (كما في الروبوت).

Discrete Event Simulator



آلية عمل محاكي الأحداث المتقطعة :

سوف نتتبع خطوات هذه الخوارزمية كما هي موضحة في المخطط الانسيابي:

---

1. **"Any events left?" (هل توجد أحداث متبقية؟)**

- هذا هو السؤال الأول والمفصلي الذي يطرحه المحاكي. إنه يتحقق من **قائمة الأحداث** (event queue) التي ناقشناها في الشريحة السابقة.
- **إذا كانت الإجابة "No"** : فهذا يعني أن قائمة الأحداث فارغة، ولا توجد أي أنشطة مستقبلية مجدولة. وبالتالي، **"Simulation done!"(انتهت المحاكاة)** .لقد أدى النظام الافتراضي كل ما كان عليه فعله.
- **إذا كانت الإجابة "Yes"** : فهذا يعني أنه لا يزال هناك عمل يجب القيام به. ننتقل إلى الخطوة التالية.

2. **"Advance global clock to t_current, the time-stamp of the earliest event(s) in the event queue"**
(تقديم الساعة العالمية إلى t_current، وهو الختم الزمني لأبكر حدث (أحداث) في قائمة الأحداث)

- هنا تظهر الطبيعة "المتقطعة" للمحاكاة. يقوم المحاكي بالنظر في قائمة الأحداث ويحدد الحدث (أو الأحداث، إذا كان هناك عدة أحداث مجدولة في نفس الوقت بالضبط) الذي يحمل **أبكر ختم زمني(time-stamp)** .
- ثم، يقفز **بالساعة العالمية (Global Clock)** مباشرة إلى هذا الزمن، t_current. لا يتم إضاعة أي وقت في محاكاة الفترات التي لا تقع فيها أحداث.

3. **"Update the values of all events having time-stamp = t_current"**
(تحديث قيم جميع الأحداث التي لها ختم زمني =t_current).

- بمجرد وصول الساعة العالمية إلى t_current، يقوم المحاكي بمعالجة جميع الأحداث المجدولة في هذا الوقت المحدد.
- "معالجة الحدث" تعني عادةً **تغيير حالة النظام** بناءً على ما يمثله هذا الحدث. على سبيل المثال، إذا كان الحدث هو "وصول بيانات من حساس"، فقد يتم تحديث متغير يحمل قيمة هذا الحساس. أو إذا كان الحدث "تغيير قيمة الإشارة S1 إلى 5" (كما في مثالنا السابق)، يتم الآن تطبيق هذا التغيير فعلياً في النظام.

4. **"Activate and run all processes which are sensible to the updated events; each process will eventually reach a wait for a certain event and enter a *wait state*."**
(تنشيط وتشغيل جميع العمليات الحساسة للأحداث المحدثة؛ كل عملية ستصل في النهاية إلى انتظار لحدث معين وتدخل في حالة انتظار) .

- الآن بعد أن تم تحديث حالة النظام بناءً على الأحداث الحالية، يتم "إيقاظ" أو **تنشيط جميع العمليات (Processes)** التي كانت تنتظر هذه الأحداث أو تتأثر بها.
- تبدأ هذه العمليات المنشطة في تنفيذ منطقها البرمجي. قد تقوم بحسابات، اتخاذ قرارات، أو التفاعل مع أجزاء أخرى من النظام.
- كل عملية، بعد أن تنهي عملها الحالي، عادة ما تصل إلى نقطة تحتاج فيها إلى **انتظار حدث مستقبلي آخر** (مثلاً، انتظار وصول رسالة، أو انتهاء مؤقت زمني، أو تغير حالة حساس). عند هذه النقطة، تدخل العملية في **حالة انتظار (wait state)**، وتتوقف عن التنفيذ مؤقتاً. هذا يسمح لعمليات أخرى بالحصول على فرصة للتنفيذ.

5. **"The activated processes have generated new events; place these events at their right place in the event queue."**

(العمليات المنشطة قد ولّدت أحداثاً جديدة؛ ضع هذه الأحداث في مكانها الصحيح في قائمة الأحداث) .

- أثناء تنفيذ العمليات النشطة (في الخطوة السابقة)، من المحتمل جداً أنها قامت **بجدولة أحداث جديدة** لتحدث في المستقبل.
- على سبيل المثال، قد تقرر عملية ما أنها تريد تشغيل محرك بعد 100 ملي ثانية. هذا القرار يُترجم إلى حدث جديد ("تشغيل المحرك") مع بصمة زمنية (t_current + 100ms) .
- يتم أخذ كل هذه الأحداث الجديدة التي تم إنشاؤها وإدراجها في **قائمة الأحداث (event queue)** في مكانها الصحيح، أي يتم ترتيبها حسب بصمتها الزمنية.

6. **العودة إلى البداية**

- بعد وضع الأحداث الجديدة في القائمة، يعود المحاكي إلى الخطوة الأولى ("Any events left?") ويكرر الدورة بأكملها. تستمر هذه الحلقة حتى تفرغ قائمة الأحداث.

هذه الدورة من التقاط الحدث الأبكر، تقديم الزمن، تحديث الحالة، تشغيل العمليات، وجدولة أحداث جديدة هي **قلب محاكي الأحداث المتقطعة النابض** .إنها طريقة منظمة وفعالة للغاية لنمذجة سلوكيات الأنظمة المعقدة بمرور الزمن.

Discrete Event Simulation

- **The discrete event model has been mainly used for system simulation.**
  - **Several languages have been developed for system modeling based on the discrete event model. Most well known:**
    - **VHDL, Verilog (both used for hardware modeling), SystemC**
- **Efficient way to simulate distributed systems.**

  In general, efficient for large systems with autonomous components, with relatively large idle times. Systems with non-regular, possibly long times between different activities.

  Why is this the case?

  Because DE simulation will only consider the particular times when a change in the system (an event) occurs. This is opposed to, for example, cycle-based models, where *all clock-ticks* are considered.

- **"The discrete event model has been mainly used for system simulation."**

- عندما نريد أن نفهم كيف سيتصرف نظام معقد (مثل روبوت، أو شبكة اتصالات، أو حتى نظام اقتصادي) بمرور الوقت، فإن محاكاة الأحداث المتقطعة هي أداة قوية جداً. نحن لا نبني النظام فعلياً في البداية، بل نبني "نموذجاً" له ونشغله في بيئة محاكاة لنرى سلوكه.

- **"Several languages have been developed for system modeling based on the discrete event model"**

- بما أننا نتحدث عن بناء "نماذج"، فنحن بحاجة إلى لغات لوصف هذه النماذج. تم تطوير العديد من اللغات لنمذجة الأنظمة بناءً على نموذج الحدث المتقطع. أشهرها :

  - **VHDL, Verilog كلاهما يُستخدم لنمذجة العتاد الصلب:**

  - هاتان اللغتان هما عماد تصميم العتاد الرقمي(digital hardware) ، مثل المعالجات الصغرية والدارات المتكاملة (ICs) . عندما يصمم مهندس العتاد شريحة إلكترونية، فإنه يكتب وصفاً لسلوكها ودارتها باستخدام VHDL أوVerilog .

  - المحاكاة في هذه اللغات غالباً ما تكون مدفوعة بالأحداث (event-driven) على مستوى دقيق جداً (تغير الإشارات على البوابات المنطقية مثلاً). هي تطبق مبادئ محاكاة الأحداث المتقطعة ولكن على مستوى تفصيلي جداً يخص العتاد الصلب.

  - **SystemC** :

  - هذه لغة مهمة جداً في تصميم الأنظمة المدمجة. SystemC هي في الأساس مكتبة للغة++C ، وهي مصممة خصيصاً لـ **نمذجة الأنظمة على مستوى أعلى(system-level modeling) .**

  - **لماذا هي مهمة ومميزة؟** لأنها تسمح لنا بنمذجة أنظمة تحتوي على **كل من العتاد الصلب (hardware) والبرمجيات (software)** والتفاعلات بينهما، كل ذلك ضمن إطار عمل واحد يعتمد بشكل كبير على محاكاة الأحداث المتقطعة. يمكننا إنشاء نماذج لمكونات برمجية، ومكونات عتادية، وواجهات بينهما، ثم محاكاة النظام بأكمله لنرى كيف يتصرف.

  - إنها تسمح بتجريد التفاصيل الدقيقة للعتاد الصلب (التي تهتم بها VHDL/Verilog) والتركيز أكثر على هيكل النظام، التفاعلات، والأداء العام.

- **" Efficient way to simulate distributed systems"**

- الأنظمة الموزعة هي تلك التي تتكون من عدة مكونات أو عقد تعمل بشكل مستقل نسبياً وتتواصل فيما بينها (مثل شبكة من أجهزة الكمبيوتر، أو مجموعة من الروبوتات تتعاون لإنجاز مهمة).

- محاكاة الأحداث المتقطعة فعالة هنا لأن كل مكون (أو عقدة) يمكن أن يكون له جدوله الزمني الخاص للأحداث، والمحاكاة تركز فقط على لحظات التفاعل (إرسال/استقبال رسائل) أو التغييرات الداخلية الهامة داخل كل مكون.

- **"In general, efficient for large systems with autonomous components, with relatively large idle times. Systems with non-regular, possibly long times between different activities."**

- هذه النقطة تشرح "متى" تكون محاكاة الأحداث المتقطعة فعالة بشكل خاص.

- لنتخيل روبوتاً ينتظر أوامر، أو حساساً لا يرسل بيانات إلا عند حدوث تغيير كبير. معظم الوقت، هذه المكونات خاملة " (idle) ". إذا كنا نحاول محاكاة كل جزء صغير من الثانية (كل نبضة ساعة)، سنضيع الكثير من القدرة الحاسوبية في محاكاة اللاشيء.
- محاكاة الأحداث المتقطعة تتجاهل فترات الخمول هذه وتقفز مباشرة إلى "النشاط" أو "الحدث" التالي المهم.

- **لماذا هي فعالة في هذه الحالة؟ لأن محاكاة الأحداث المتقطعة تأخذ في الاعتبار فقط الأوقات المحددة التي يحدث فيها تغيير في النظام (حدث). هذا على عكس، على سبيل المثال، النماذج القائمة على الدورة(cycle-based) ، حيث يتم أخذ جميع نبضات الساعة في الاعتبار.**
  - **للتوضيح بمثال :**لنتخيل أننا نراقب شخصاً يقرأ كتاباً .
    - **النموذج القائم على الدورة (Cycle-based) :**كأننا نلتقط صورة لهذا الشخص كل ثانية، سواء قلب الصفحة أم لم يقلبها، سواء تحرك أم لم يتحرك. سنحصل على آلاف الصور المتشابهة جداً.
    - **نموذج الحدث المتقطع (Discrete Event)** : كأننا نلتقط صورة فقط عندما يقلب الشخص صفحة، أو عندما ينهض، أو عندما يبدأ في التحدث. نحن نركز فقط على "الأحداث" الهامة. هذا يعطينا فهماً جيداً لما يفعله الشخص دون الحاجة لمعالجة كم هائل من البيانات غير الضرورية.
  - بنفس الطريقة، محاكي الأحداث المتقطعة يتجاهل "الثواني الفارغة" في حياة النظام ويقفز بساعته العالمية (Global Clock) مباشرة من حدث إلى الحدث التالي في قائمة الأحداث (Event Queue) . أما المحاكي القائم على الدورة، فيقوم بتحديث حالة النظام عند كل نبضة ساعة افتراضية، حتى لو لم يتغير شيء مهم. هذا يجعله أبطأ بكثير للأنظمة التي بها فترات خمول طويلة أو أنشطة متباعدة زمنياً.

هذه الشريحة تعزز فهمنا لأهمية محاكاة الأحداث المتقطعة، وتوضح لنا الأدوات (اللغات) التي تمكننا من تطبيقها، وتفسر سر كفاءتها في سياقات معينة.

Discrete Event Simulation

- **Event driven models are primarily employed for simulation.**
  - **Functional verification**
  - **Performance evaluation**

- **Both synthesis and formal verification are very complex with DE models.**
  - **The classical trade-off between expressive power and the possibility of formal reasoning and efficient synthesis.**

هذه الشريحة تغوص في استخدامات محاكاة الأحداث المتقطعة وتلقي الضوء على بعض التحديات المتعلقة بها.

- **"Event driven models are primarily employed for simulation."**
  - هذه النقطة تعزز ما ناقشناه سابقاً. عندما نصمم نظاماً كنموذج مدفوع بالحدث (حيث التغييرات تحدث نتيجة لأحداث متقطعة)، فإن الهدف الأساسي غالباً ما يكون تشغيل هذا النموذج عبر الزمن (أي محاكاته) لفهم سلوكه.
  - يمكن تحديد هدفين رئيسيين لهذه المحاكاة:
    - **"Functional verification"**
      - **التحقق الوظيفي :** يعني التأكد من أن النظام الذي صممناه (أو نموذجه) يؤدي الوظائف المطلوبة منه بشكل صحيح ويتصرف وفقاً للمواصفات المحددة. هل يقوم بالمهام الصحيحة؟ هل يستجيب للمدخلات كما هو متوقع؟
      - **كيف تساعد المحاكاة؟** من خلال تشغيل النموذج تحت سيناريوهات مختلفة (حالات اختبار) ومراقبة سلوكه ومخرجاته، يمكننا التحقق مما إذا كان يعمل "وظيفياً" بشكل صحيح. مثلاً، هل يستجيب الروبوت للأوامر بشكل صحيح؟ هل يتجنب العوائق كما هو مصمم؟
    - **"Performance evaluation"**
      - **تقييم الأداء :** يعني قياس "مدى جودة" أداء النظام. هذا يشمل مقاييس مثل سرعة الاستجابة (latency)، الإنتاجية(throughput) ، مدى استخدام الموارد (مثل المعالج أو الذاكرة)، استهلاك الطاقة، إلخ.
      - **كيف تساعد المحاكاة؟** أثناء تشغيل المحاكاة، يمكننا جمع إحصائيات وبيانات حول هذه المقاييس. مثلاً، كم من الوقت يستغرق الروبوت لإكمال مهمة معينة تحت ظروف مختلفة؟ ما هو أقصى عدد من الطلبات يمكن لنظام مدمج معالجتها في الثانية؟
  - **مثال في سياق الروبوت:**
    - **التحقق الوظيفي :**نصمم نموذجاً لذراع روبوت ونحاكيه للتأكد من أنه يصل إلى الإحداثيات المطلوبة بدقة، وأن تتابع حركاته صحيح لتجنب الاصطدام بنفسه أو بالبيئة المحيطة.
    - **تقييم الأداء :**نحاكي نظام التحكم في الروبوت تحت أحمال عمل مختلفة لنرى مدى سرعة استجابته للأوامر الجديدة، أو كم من الوقت يستغرقه لتنفيذ سلسلة من المهام، أو هل يمكنه الحفاظ على استقراره عند الحركة بسرعات عالية.
- **"Both synthesis and formal verification are very complex with DE models."**
  - هذه نقطة مهمة جداً تلقي الضوء على بعض القيود أو التحديات.
    - **Synthesis**

- في سياق تصميم الأنظمة،Synthesis يعني عادةً **التحويل التلقائي للنموذج عالي المستوى إلى وصف أكثر تفصيلاً يمكن تصنيعه أو تنفيذه مباشرة** .مثلاً، توليد تصميم عتادي(كود VHDL/Verilog قابل للتصنيع) من نموذج SystemC ، أو حتى توليد كود برمجي من مخطط سلوكي.

- **لماذا هو معقد مع نماذج الأحداث المتقطعة؟** لأن نماذج الأحداث المتقطعة، خاصة تلك المكتوبة بلغات مثل SystemC ، يمكن أن تكون غنية جداً بالتفاصيل وتصف سلوكيات معقدة وتفاعلات زمنية دقيقة. تحويل كل هذه التفاصيل بشكل تلقائي إلى تصميم عتادي أو برمجي أمثل وفعال هو تحدٍ كبير لأدوات الـSynthesis .

- **Formal Verification**

- **التحقق الرسمي:** هو استخدام **أساليب رياضية ومنطقية لإثبات أو نفي صحة خصائص معينة للنموذج بشكل قاطع لجميع الحالات والسلوكيات الممكنة** .هذا يختلف عن المحاكاة التي تختبر فقط مجموعة محدودة من السيناريوهات. مثلاً، إثبات أن النظام "لن يصل أبداً إلى حالة جمود (deadlock)" أو أن "الاستجابة لطلب معين ستحدث دائماً خلال X ميللي ثانية".

- **لماذا هو معقد مع نماذج الأحداث المتقطعة؟** بسبب "القوة التعبيرية" العالية لهذه النماذج، فإن عدد الحالات الممكنة (state space) التي يمكن للنظام أن يمر بها يمكن أن يكون هائلاً جداً أو حتى لا نهائياً. تحليل كل هذه الحالات رياضياً يتطلب قدرات حسابية ضخمة وقد يكون غير ممكن عملياً.

- **"The classical trade-off between expressive power and the possibility of formal reasoning and efficient synthesis."**

- المقايضة الكلاسيكية بين القوة التعبيرية وإمكانية الاستدلال الرسمي والـsynthesis هي مقايضة أساسية في هندسة النظم والبرمجيات.

  - **القوة التعبيرية (Expressive Power)** : قدرة النموذج على وصف سلوكيات معقدة ودقيقة وتفاصيل متنوعة. نماذج الأحداث المتقطعة قوية جداً من هذه الناحية، مما يجعلها رائعة للمحاكاة الدقيقة.

  - **الاستدلال الرسمي (Formal Reasoning) والـ Synthesis** هاتان الخاصيتان تستفيدان من النماذج الأكثر بساطة وتقييداً.

- **المقايضة تعني:**

  - كلما زادت **القوة التعبيرية** للنموذج (أي سمحت له بوصف تفاصيل أكثر تعقيداً ومرونة)، أصبح من **الأصعب** إجراء **تحقق رسمي** شامل عليه، وأصبح من **الأصعب تحويله** بكفاءة إلى تصميم عملي.

- والعكس صحيح: النماذج الأكثر تجريداً وبساطة (مثل بعض أنواع آلات الحالة المحدودة أو شبكات بتري البسيطة) قد تكون أسهل للتحقق الرسمي والSynthesis ، لكنها قد لا تستطيع التعبير عن كل تفاصيل النظام الدقيقة التي تحتاجها للمحاكاة الواقعية.

- **مثال للمقايضة:**

- لنتخيل أننا نصف مهمة ما لشخص .

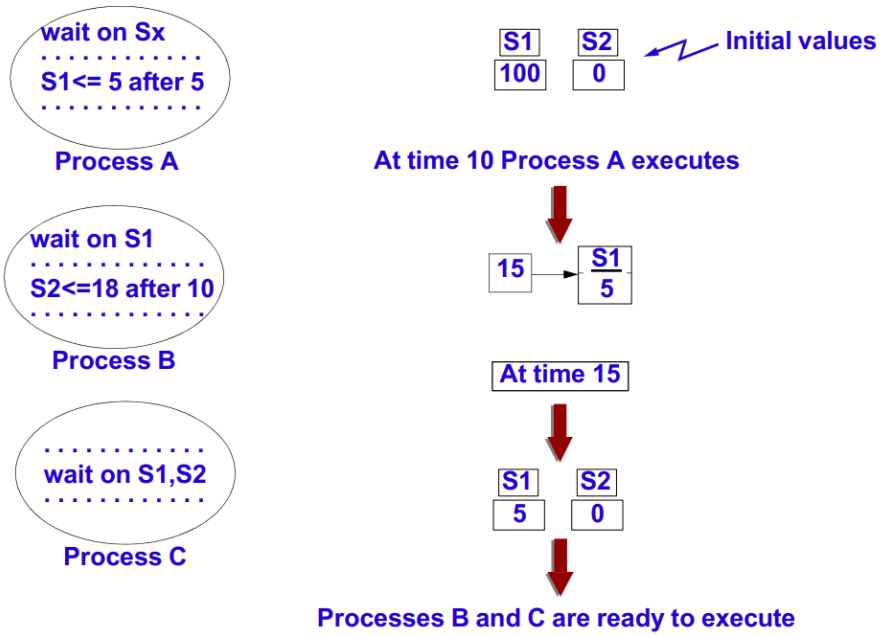
- **الوصف التفصيلي باللغة الطبيعية (قوة تعبيرية عالية) :**يمكننا وصف كل الفروق الدقيقة والمشاعر والحالات الاستثنائية. لكن سيكون من الصعب جداً على الكمبيوتر "فهم" هذا الوصف بشكل كامل للتحقق من كل جوانبه أو تحويله تلقائياً إلى برنامج خالٍ من الأخطاء.

- **مخطط انسيابي بسيط (قوة تعبيرية أقل) :**يوضح الخطوات الأساسية بشكل منظم. سيكون أسهل لتحليله (هل كل المسارات تؤدي إلى نهاية؟) وربما تحويله إلى كود، لكنه سيفقد الكثير من التفاصيل الدقيقة التي قد تكون مهمة.

- إذاً، عند اختيار مستوى التفصيل في نموذج الحدث المتقطع، يجب على المصمم أن يوازن بين الحاجة إلى دقة المحاكاة (التي تتطلب قوة تعبيرية عالية) وبين إمكانية (أو صعوبة) التحقق الرسمي من صحته أو تحويله إلى نظام فعلي لاحقاً.

هذه الشريحة تبرز قوة محاكاة الأحداث المتقطعة في التحقق الوظيفي وتقييم الأداء، ولكن أيضاً تشير إلى أن القوة التعبيرية التي تجعلها جيدة للمحاكاة قد تجعل مهام أخرى مثل الSynthesis والتحقق الرسمي أكثر تحدياً.

An Example



سنعرض مثالاً عملياً يوضح كيف تسير الأمور داخل محاكي الأحداث المتقطعة.

مثال توضيحي:

لدينا في هذا المثال نظام صغير يتكون من ثلاث عمليات (Process A, Process B, Process C) وإشارتين(Signal S1, Signal S2) .

1. **الوضع الأولي(Initial State) :**

- **العمليات:**

  - **Process A:**

    - wait on Sx تنتظر هذه العملية إشارة ما اسمها Sx هذه الإشارة ليست جزءاً من S1 أوS2 ، و تعتبر كشرط خارجي لبدء العملية A .
    - عندما يتم تنشيطها، ستقوم بتنفيذ الأمر S1 <= 5 after 5 هذا يعني: "اجعل قيمة الإشارة S1 تساوي 5، ولكن ليس الآن، بل **بعد مرور5 وحدات زمنية** من لحظة تنشيطي"

  - **Process B:**

    - wait on S1 تنتظر حدوث تغيير في الإشارة S1
    - عندما يتم تنشيطها (بسبب تغيرS1) ، ستقوم بتنفيذ الأمر S2 <= 18 after 10 :هذا يعني: "اجعل قيمة الإشارة S2 تساوي 18، **بعد مرور10 وحدات زمنية** من لحظة تنشيطي".

  - **Process C :**

- wait on S1, S2 تنتظر هذه العملية حدوث تغيير في الإشارة S1 **أو** الإشارة S2 (في معظم أنظمة المحاكاة، هذا يعني أن العملية "حساسة" لكلتا الإشارتين، وأي تغيير في أي منهما سيؤدي إلى إيقاظها لتعيد تقييم حالتها أو تنفيذ منطقها). ماذا ستفعل عند تنشيطها؟ هنا لم يتم تحديد إجراء معين لها بعد الانتظار، فقط أنها تنتظر.

- **القيم الابتدائية للإشارات(Initial Values) :**
  - S1 = 100
  - S2 = 0

2- "At time 10 Process A executes"

- نفترض أن الشرط Sx قد تحقق في الزمن 10 أو قبله بقليل، والمحاكي قرر تنفيذ A الآن(أو أن المحاكاة بدأت بتنشيط العملية A في هذا الزمن).
- عندما تبدأ العملية A في التنفيذ، فإنها تقرأ أمرها S1 <= 5 after 5
- **ماذا يحدث الآن؟** لا تتغير قيمة S1 فوراً. بدلاً من ذلك، تقوم العملية A **بجدولة حدث مستقبلي** .
  - الزمن الحالي هو 10.
  - التأخير هو 5 وحدات زمنية.
  - إذاً، يتم إضافة حدث جديد إلى **قائمة الأحداث (event queue)** للمحاكي: "عند الزمن 15 = 5 + 10، يجب أن تتغير قيمة S1إلى 5".

3- "At time 15"

- بعد أن عالج المحاكي ما حدث في الزمن 10 (وهو فقط جدولة الحدث من قبل A)، ينظر المحاكي في قائمة الأحداث. يجد أن أبكر حدث مجدول هو "تغيير S1 إلى 5 عند الزمن 15"
- لذا، **تقفز الساعة العالمية للمحاكاة (Global Clock)** مباشرة إلى الزمن 15.
- عند الزمن 15، يتم **معالجة الحدث** :
  - قيمة الإشارة S1 تتغير بالفعل من 100إلى 5.
  - تظهر القيم الجديدة S1 = 5 و S2 = 0 (لم تتغير S2 بعد)

4- "Processes B and C are ready to execute"

- بمجرد أن تغيرت قيمة S1 في الزمن 15، يقوم المحاكي بالتحقق من العمليات التي كانت تنتظر هذا التغيير :
  - **Process B**: كانت في حالة wait on S1 بما أن S1 قد تغيرت للتو، يتم **إيقاظ** العملية B وتصبح **جاهزة للتنفيذ**

- **Process C**: كانت في حالة wait on S1, S2 بما أن S1 (إحدى الإشارتين اللتين تنتظرهما) قد تغيرت، يتم أيضاً **إيقاظ** العملية C وتصبح **جاهزة للتنفيذ**.

ماذا نتوقع أن يحدث بعد ذلك ؟

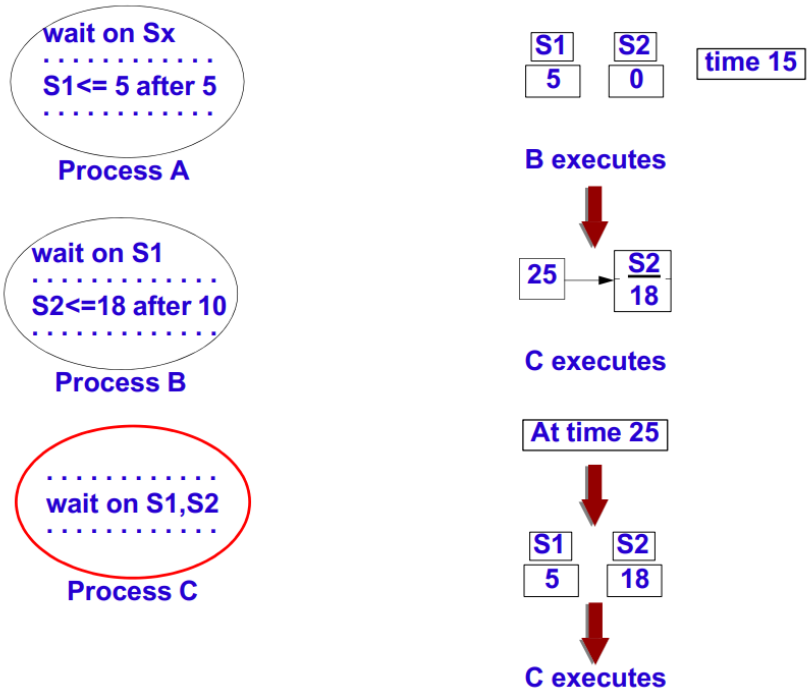
- الآن بما أن B و C جاهزتان، سيقوم المحاكي بتنفيذهما (ترتيب التنفيذ بينهما قد يعتمد على أولويات محددة مسبقاً أو قواعد أخرى في المحاكي، ولكن لنفترض أنه سينفذهما).
  - **عندما تُنفذ Process B في الزمن 15 :**(ستقرأ أمرها S2 <= 18 after 10)
    - ستقوم Process B بجدولة حدث جديد: "عند الزمن 15(الزمن الحالي) + 10 = 25، يجب أن تتغير قيمة S2إلى 18".
    - سيتم إضافة هذا الحدث (S2=18 at t=25) إلى قائمة الأحداث.
  - **عندما تُنفذ Process C في الزمن 15 :**ستقوم بتنفيذ منطقها الداخلي (الذي لم يتم توضيحه هنا). قد تقوم هي الأخرى بجدولة أحداث، أو قد تعود ببساطة إلى حالة انتظار أخرى.
- بعد تنفيذ B وربما C ، سينظر المحاكي مرة أخرى في قائمة الأحداث. إذا كان أبكر حدث هو "تغيير S2 إلى 18 عند الزمن 25"، فستقفز الساعة العالمية إلى الزمن 25، وهكذا.

هذا المثال يوضح دورة حياة المحاكاة:

1. العمليات تنتظر أحداثاً معينة.
2. عند وقوع حدث ما (أو عند بدء المحاكاة)، يتم تنشيط العمليات المعنية.
3. العمليات المنشطة قد تقوم بتغيير قيم إشارات فوراً، أو (وهو الشائع جداً كما في المثال) تقوم **بجدولة تغييرات مستقبلية** (أحداث جديدة) في قائمة الأحداث.
4. المحاكي يتقدم بالزمن دائماً إلى وقت أبكر حدث مجدول تالياً.

يوضح هذا المثال كيف أن النظام "يتحرك" فقط عند نقاط زمنية محددة (الأحداث)، وكيف تتفاعل العمليات مع بعضها البعض بشكل غير مباشر من خلال تغيير الإشارات وجدولة الأحداث.

An Example



متابعة المثال :

نبدأ من النقطة التي انتهينا عندها في الشريحة السابقة:

- **الزمن الحالي للمحاكاة (Global Clock)** 15 :
- **قيم الإشارات** S1 = 5 (تغيرت للتو)، S2 = 0.
- **حالة العمليات:** Process B و Process C أصبحتا "جاهزتين للتنفيذ " لأنهما كانتا تنتظران تغيراً في S1 .

1. **" العملية B تُنفذ عند الزمن 15:**

   - بما أن Process B أصبحت جاهزة عند الزمن 15 بسبب تغير S1(كانت wait on S1) ، يقوم المحاكي الآن بتنفيذ منطقها.
   - منطق Process B بعد تنشيطها هو S2 <= 18 after 10
   - هذا يعني أن Process B ستقوم **بجدولة حدث مستقبلي** .
     - الزمن الحالي 15.
     - التأخير المحدد: 10 وحدات زمنية.
     - الإجراء: تغيير قيمة S2إلى 18.

- إذاً، يُضاف حدث جديد إلى **قائمة الأحداث (event queue)** الخاصة بالمحاكي: "عند الزمن 25 = 10 + 15، يجب تحديث قيمة S2 لتصبح 18"
- الشريحة توضح هذا بوضع 25(زمن الحدث) مشيراً إلى S2 وقيمتها الجديدة 18

2. **العملية C تُنفذ أيضاً عند الزمن 15:**

- Process C أيضاً أصبحت جاهزة عند الزمن 15 لأنها كانت wait on S1, S2 وتغيرت S1
- يقوم المحاكي بتنفيذ منطق Process C
- الشريحة لا تحدد الإجراءات التي تقوم بها Process C عند تنفيذها في هذه اللحظة. عادةً، عندما يتم تنشيط عملية تنتظر عدة إشارات بسبب تغير إحداها، فإنها تعيد تقييم شروط انتظارها أو تنفذ جزءاً من منطقها. قد تكون Process C الآن تتحقق من قيم S1 و S2 الحالية وتقرر ما إذا كانت ستستمر في الانتظار (ربما لشرط أكثر تحديداً يتعلق بكليهما) أو تقوم بعمل ما.

3. **"At time 25"**

- بعد انتهاء تنفيذ العمليات التي أصبحت جاهزة عند الزمن 15 (مثل B وC) ، ينظر المحاكي إلى قائمة الأحداث لتحديد الحدث التالي الأبكر.
- يجد المحاكي أن أبكر حدث مجدول هو: "تحديث S2إلى 18عند الزمن 25(هذا الحدث تمت جدولته بواسطة Process B).
- يقوم المحاكي **بتقديم ساعته العالمية (Global Clock)** إلى الزمن 25
- يتم استرداد الحدث (S2 = 18) من قائمة الأحداث **ومعالجته** :
  - قيمة الإشارة S2 تتغير من 0إلى 18
  - قيمة الإشارة S1تبقى كما هي 5
- القيم الجديدة: S1 = 5 ، S2 = 18

4. **العملية C تُنفذ مرة أخرى عند الزمن 25:**

- Process C كانت في حالة wait on S1, S2
- بما أن الإشارة S2 قد تغيرت للتو (في الزمن 25)، يتم **تنشيط Process C مرة أخرى**.
- يقوم المحاكي بتنفيذ منطق ProcessC استجابةً لهذا التغيير الجديد في S2 (مجدداً، تفاصيل الإجراءات التي تقوم بها Cغير موضحة، لكنها تتفاعل مع الحدث الأخير).

ملخص ما حدث في هذه المرحلة من المحاكاة:

- **عند الزمن 15 :**
  - S1كانت قد تغيرت إلى 5
  - Process B نُفذت وقامت بجدولة تغيير S2 إلى 18 ليحدث عند الزمن 25
  - Process Cنفذت (استجابة لتغير S1) وقامت بإعادة تقييم حالتها.
- **القفزة الزمنية :**انتقل المحاكي مباشرة من الزمن 15 إلى الزمن 25 لأنه لم تكن هناك أحداث مجدولة بين هذين الوقتين.
- **عند الزمن 25 :**
  - S2 تغيرت إلى 18
  - Process C نفذت مرة أخرى (استجابة لتغير S2).

هذا يوضح كيف يتقدم المحاكي عبر الزمن بناءً على الأحداث المجدولة، وكيف يتم تنشيط العمليات المختلفة استجابة لهذه الأحداث، وكيف يمكن لهذه العمليات بدورها أن تجدول أحداثاً جديدة تؤثر على سلوك النظام في المستقبل.

Delta Delay/Delta Cycles

- **A zero delay event will be registered at a time which is infinitesimally delayed relative to the current time.**

A *delta delay* will be introduced on the event ⇒ the new event will be consumed in the following simulation cycle and not the current one.

هذه الشريحة تقدم لنا مفهوماً دقيقاً ولكنه جوهري جداً في عالم محاكاة الأنظمة، وخاصة عند نمذجة العتاد (hardware) أو الأنظمة التي تتطلب ترتيباً دقيقاً للأحداث التي تبدو وكأنها تحدث "في نفس اللحظة". هذا المفهوم هو **"تأخير دلتا (Delta Delay) "**أو **"دورات دلتا " (Delta Cycles)**.

تأخير دلتا / دورات دلتا(Delta Delay/Delta Cycles)

في بعض الأحيان، أثناء المحاكاة، قد تقوم عملية ما بجدولة حدث ليحدث **بتأخير صفري (zero delay)** قد نتساءل: "إذا كان التأخير صفراً، ألا يعني هذا أن الحدث يقع فوراً في نفس اللحظة تماماً؟" الأمر أعقد قليلاً من ذلك، وهنا يأتي دور "تأخير دلتا".

1. **"A zero delay event will be registered at a time which is infinitesimally delayed relative to the current time." (سيتم تسجيل حدث ذي تأخير صفري في وقت متأخر بشكل متناهٍ في الصغر بالنسبة للوقت الحالي)**

- هذا هو المفتاح الأول. عندما نقول "تأخير صفري"، لا يعني ذلك دائماً "في نفس هذه النانوثانية وبدون أي فاصل على الإطلاق لدرجة أن كل شيء يراه فوراً في نفس خطوة المعالجة الحالية"

- بدلاً من ذلك، يفهم المحاكي هذا "التأخير الصفري" على أنه تأخير صغير جداً جداً، أصغر من أي وحدة زمنية يمكن قياسها بواسطة **الساعة العالمية الرئيسية للمحاكاة** (تلك التي تعد الثواني، أو النانوثانية، إلخ). هذا التأخير المتناهي في الصغر هو ما نسميه "تأخير دلتا"

2. **"A delta delay will be introduced on the event ⇒ the new event will be consumed in the following simulation cycle and not the current one."**

سيتم إدخال تأخير دلتا على الحدث --> سيتم استهلاك الحدث الجديد في دورة المحاكاة التالية وليس في الدورة الحالية.

- لنتخيل أن الساعة العالمية للمحاكاة واقفة عند زمن معين، لنقل T = 20 نانوثانية.

- في هذا الزمن T، قد تحدث عدة أنشطة. لنسمي كل جولة من هذه الأنشطة داخل نفس الزمن " Tدورة محاكاة " **(simulation cycle)** أو بشكل أدق "**دورة دلتا(delta cycle)** "

- إذا قامت عملية ما، أثناء دورة دلتا الحالية (مثلاً، دورة دلتا رقم n)عند الزمن T، بجدولة حدث بتأخير صفري (أي بتأخير دلتا)، فإن هذا الحدث الجديد **لن يُعالج في دورة دلتا الحالية(n)**

- بدلاً من ذلك، سيتم وضعه جانباً ليُعالج في **دورة الدلتا التالية (دورة دلتا رقم n+1)**، مع بقاء الساعة العالمية للمحاكاة عند نفس الزمن T

- تستمر هذه الدورات (دلتا 1، دلتا 2، دلتا 3، ...) عند نفس الزمن Tحتى لا تعود هناك أي أحداث مجدولة بتأخير دلتا. عندها فقط، يمكن للساعة العالمية للمحاكاة أن تتقدم إلى الزمن التالي الذي يوجد فيه حدث مجدول (مثلاً، T + 5 نانوثانية).

لماذا هذا المفهوم مهم جداً؟

1. **الحتمية(Determinism) :**

- يضمن تأخير دلتا أن هناك ترتيباً محدداً وواضحاً لمعالجة الأحداث حتى لو كانت كلها مجدولة "في نفس الوقت" (نفس الزمن على الساعة العالمية). هذا يمنع الغموض ويجعل سلوك المحاكاة قابلاً للتنبؤ.

2. **نمذجة انتشار الإشارة في العتاد(Modeling Signal Propagation) :**

- في تصميمات العتاد الرقمي (مثلاً باستخدام VHDL أوVerilog)، عندما تتغير قيمة إشارة ما، فإن المكونات الأخرى المتصلة بهذه الإشارة لا نراها "فوراً" في نفس اللحظة التي تم فيها التغيير. حتى لو كان الانتشار سريعاً جداً، هناك فاصل منطقي. تأخير دلتا يمثل هذا الفاصل الدقيق. التغيير يحدث في دورة دلتا، ويُرى تأثيره في دورة الدلتا التالية.

3. **تجنب الحلقات اللانهائية الفورية(Breaking Combinatorial Loops)**

- إذا كان لدينا نظام فيه حلقة تغذية راجعة فورية (مثلاً، إشارة A تؤثر على B ، و B تؤثر فوراً على A ، وكلاهما بتأخير صفري)، بدون آلية مثل تأخير دلتا، قد يدخل المحاكي في حلقة لانهائية وهو يحاول تحديث A و B مراراً وتكراراً في نفس اللحظة الزمنية. تأخير دلتا "يكسر" هذه الحلقة بجعل تأثير B على A يحدث في دورة الدلتا التالية، مما يعطي فرصة للنظام للاستقرار أو لإظهار التذبذب بشكل يمكن تتبعه.

4. **ضمان قراءة القيم "القديمة" قبل رؤية "الجديدة"**

- يضمن أن العمليات الأخرى تقرأ القيمة الحالية (القديمة) للإشارة قبل أن يتم تحديثها بالقيمة الجديدة الناتجة عن حدث ذي تأخير صفري في نفس الزمن. القيمة الجديدة تصبح مرئية في دورة الدلتا التالية.

مثال تصوري بسيط:

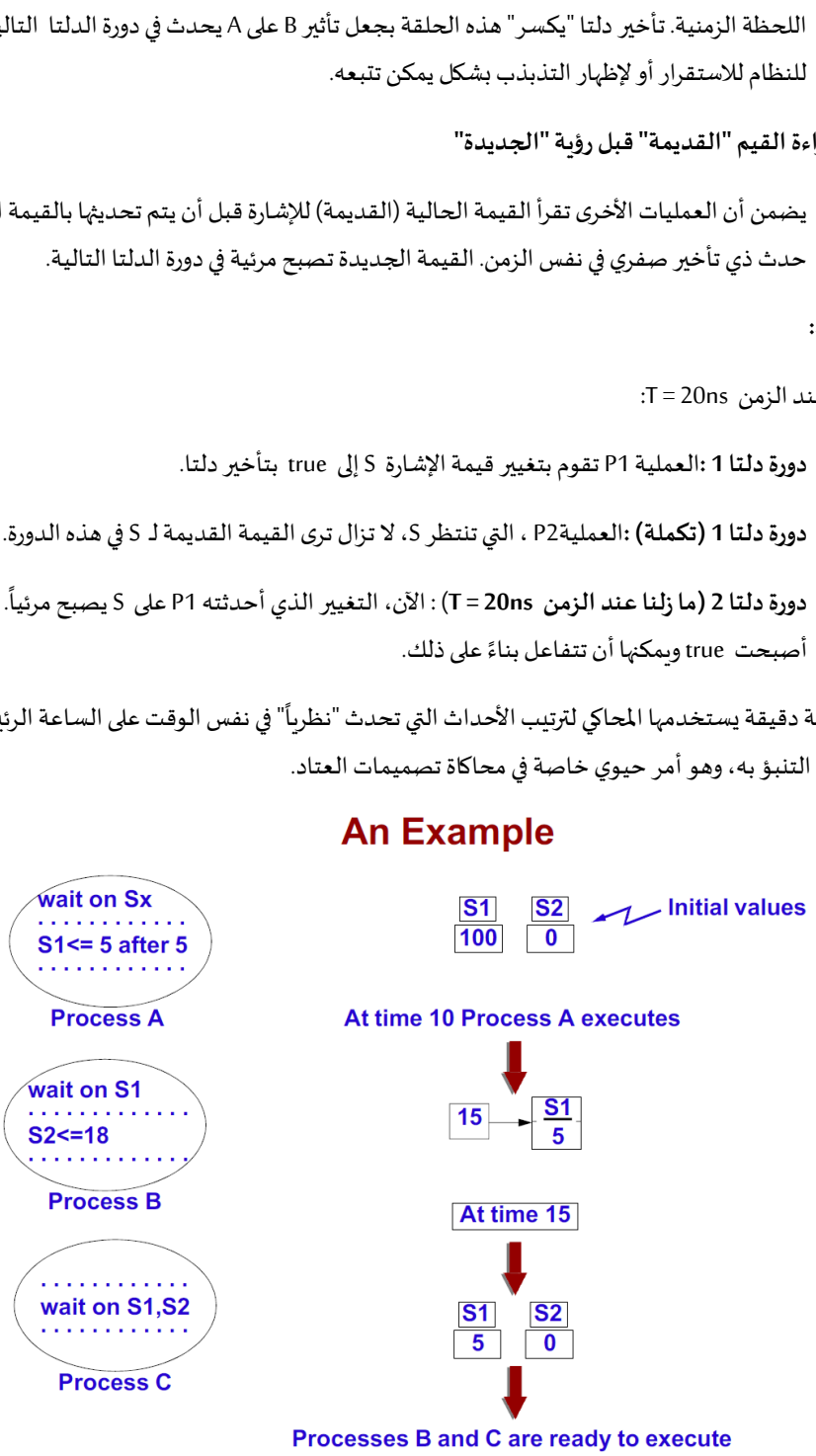
- لنفترض عند الزمن T = 20ns:

- **دورة دلتا 1 :**العملية P1 تقوم بتغيير قيمة الإشارة S إلى true بتأخير دلتا.

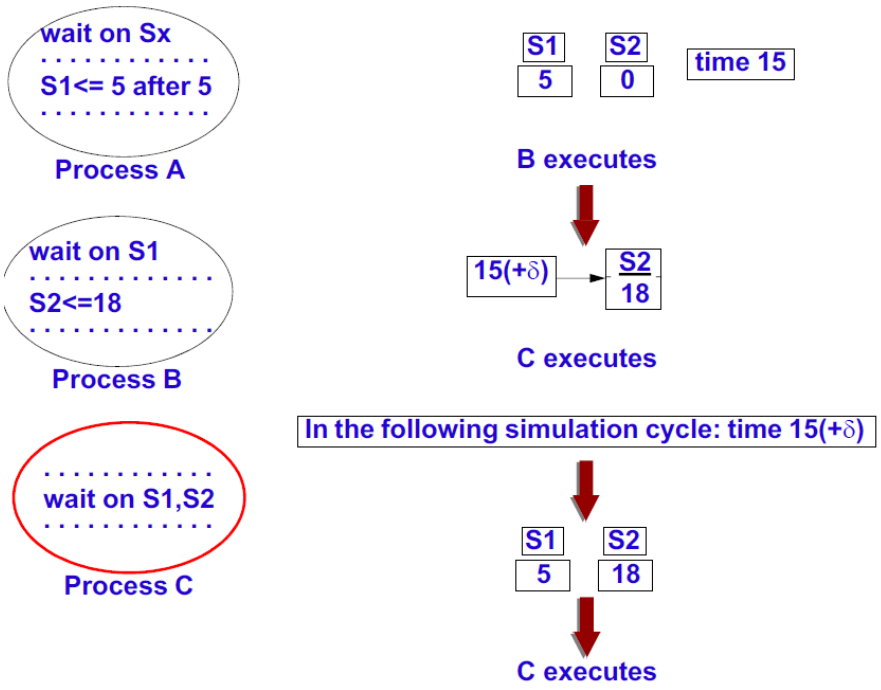
- **دورة دلتا 1 (تكملة):** العملية P2 ، التي تنتظر S، لا تزال ترى القيمة القديمة لـ S في هذه الدورة.

- **دورة دلتا 2 (ما زلنا عند الزمن T = 20ns)** : الآن، التغيير الذي أحدثته P1 على S يصبح مرئياً. العملية P2 ترى أن S أصبحت true ويمكنها أن تتفاعل بناءً على ذلك.

إذاً، تأخير دلتا هو آلية دقيقة يستخدمها المحاكي لترتيب الأحداث التي تحدث "نظرياً" في نفس الوقت على الساعة الرئيسية، مما يضمن سلوكاً منطقياً ويمكن التنبؤ به، وهو أمر حيوي خاصة في محاكاة تصميمات العتاد.



An Example


wait on Sx
S1<= 5 after 5
Process A
wait on S1
S2<=18
Process B
wait on S1,S2
Process C
S1
5
S2
0
time 15
B executes
15(+δ)
S2
18
C executes
In the following simulation cycle: time 15(+δ)
S1
5
S2
18
C executes