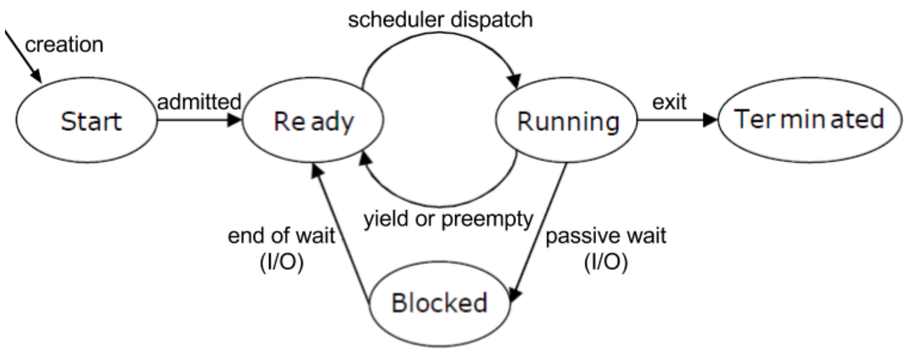
Real-Time Concurrent Programming

Summary by Victor Winberg

3 Principles

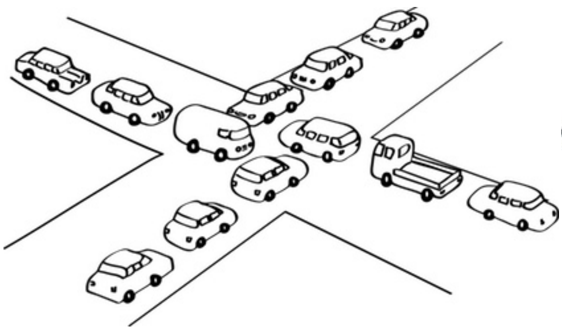
- **Parallelism -** the execution of processes are carried out simultaneously, where programs often gets divided into smaller ones.
- **Process -** an instance of a computer program that is being executed.
- **Process state -** in a multitasking system, process may have various states, not recognized by the operating system kernel. But an useful abstraction.
- **Process Life Cycle -** the context of processes initial, maturing and final stages of evolution and growth.



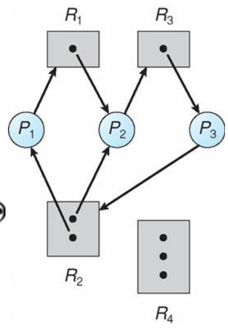
- **Process State Diagram -** nodes represent states and arcs represent transitions (e.g. above).
  - **Ready -** the thread has told the scheduler its ready and waits, until the sceduler decides to execute the thread by initialize a *context switch* and move the thread from *ready* to *running*.
  - **Running -** the thread is executing its instructions on the CPU, until the scheduler decide to move the thread from *running* to *ready*.
  - **Blocked -** the thread (itself) has decided to yield the processor (e.g. due to not being able to lock a resource, wait for I/O operations or having to sleep) or preemption (temporarily interrupt) decided by the scheduler.
- **Context switch -** the process of storing and restoring the state of a process or thread so that execution can be resumed later, by saving the current threads call stack and registers, and restoring another.

- **Multithreading - hardware** - the ability of a CPU or a single core in a multi-core processor to execute multiple processes or threads concurrently.
- **Multithreading - software** - allows multiple threads to exist within the context of one process.
- **Volatile** - a value may change between different accesses, even if it does not appear to be modified.
- **Transient** - a property of any element in the system that is temporary.

4 Deadlock



(a) A deadlock



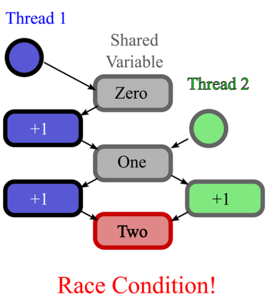
(b) RAG

- **Deadlock** - occurs when two competing actions wait for the other to finish, and thus neither ever does (e.g. above (a)).
- **Four conditions** - deadlock arise if all of the conditions are held.
  - **Mutual exclusion** - resources involved are unshareable (only one process can use the resource at any given time).
  - **Hold and Wait** (aka resource holding) - a process is currently holding at least one resource and requesting additional resources which are being held by other processes.
  - **Nonpreemption** - a resource can be released only voluntarily by the process holding it.
  - **Circular wait** - processes in the system form a circular chain where each process in the list is waiting for a resource held by the next process in the chain.
- **Resource Allocation Graph** - tracks which resource is held by which process and which process is waiting for a resource of a particular type (e.g. above (b))
- **Deadlock Prevention** - preventing one of the four conditions.
  - **Mutual exclusion** - no process will have exclusive access to a resource.

- **Hold and Wait** - requiring processes to request all the resources they will need before starting up.
- **Nonpreemption** - difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time. However, algorithms that allow preemption include lock-free and wait-free algorithms.
- **Circular wait** - disabling interrupts during critical sections and using a hierarchy to determine a partial ordering of resources.

- **Deadlock Avoidance** - establish system does not enter an unsafe state.
  - **Resource Allocation Graph (RAG)**
  - **Banker's algorithm** - tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources.
- **Deadlock Detection and Recovery**
  - **Wait-for graph** - a directed graph used for deadlock detection.
  - **Modified Banker's algorithm** - see if further allocations can be made on not based on current allocations.
- **Livelock** - is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing.

5 Interprocess communication: Shared Variables



(a) A race condition

```
bool flag[2] = {false, false};
int turn;
```

```
P0:      flag[0] = true;
P0_gate:
turn = 1;
while (flag[1] && turn == 1)
{
    // busy wait
}
// critical section
...
// end of critical section
flag[0] = false;
```

```
P1:      flag[1] = true;
P1_gate:
turn = 0;
while (flag[0] && turn == 0)
{
    // busy wait
}
// critical section
...
// end of critical section
flag[1] = false;
```

(b) Peterson's algorithm

- **Race Conditions** - when the output is dependent on the sequence or timing of other uncontrollable events (e.g above (a)).
- **Critical region** - a section that cannot be executed by more than one process, which otherwise lead to unexpected or erroneous behavior.

- **Starvation** - a process is perpetually denied necessary resources to process its work, may be caused by errors in a scheduling or mutual exclusion algorithm, resource leaks or denial-of-service attack.
- **Bounded waiting** - (or bounded bypass) means that there exist a limit of times a process is bypassed by other processes before its request to enter a critical region is granted, preventing starvation.
- **Hardware-Assisted Lock Variables** - the processor provides a test-and-set instruction.
  - **Test-and-set** - instruction that writes to a memory location and returns its old value as a single atomic (non-interruptible) operation.
- **Software-Based Mutual Exclusion** - a synchronization mechanism that limits access to a resource, e.g. Dekker's or Peterson's algorithm.
  - **Dekker's algorithm** - first known correct solution to the mutual exclusion problem.
  - **Peterson's algorithm** - allows two or more processes to share a single-use resource without conflict, using shared memory for communication (e.g. above (b)).
- **Active/Busy Wait** - a process repeatedly checks to see if a condition is true, wasting process cycles.
- **Passive Wait** - a process that cannot proceed (e.g. waiting for an I/O operation) are placed in the *Blocked* state.
  - **Blocked state** - a state where processes does not proceed execution, without wasting processor cycles, saving processor power.
- **Semaphore** - a variable or abstract data type that controls access to common resources by multiple processes.
- **Monitor** - a synchronization construct that allows threads to have both mutual exclusion and the ability to wait (block) for a certain condition.

6 Interprocess communication: Message Passing

- **Send and Receive** - the basic two primitives message-passing relies on.
- **Message passing** - sends a message to a process and relies on the process with its infrastructure to select and invoke correct segment code to run, which fall into three main areas:
  - **Name** - (or the identity) of the senders and recipients.
  - **Synchronization** - the sync constraints the primitives enforce.
  - **Buffer** - the buffer space provided.

9 Network Communication

- **Ethernet Protocol** - a communication protocol restricted to the physical and data link layer.
- **TCP/IP** - a connection oriented stream over an IP network with guarantees that all sent packets will reach the destination correctly.
- **UDP** - a connection-less protocol, instead datagram oriented, with integrity guarantees on the single datagram often used in real-time applications.

11 Real-Time Cyclic-based Scheduling

- **Scheduling model** - produce the expected output in all cases and correct with respect to *timings*, which must include two main elements:
  - **Scheduling algorithm** - a set of rules for ordering the use of system resources, in particular the processors.
  - **Worst-case behavior** - how much time might be needed to guarantee that the algorithm will always finish on time.
- **Cyclic executive** - (aka timeline scheduling or cyclic scheduling) a fixed set of periodic tasks that is repeatedly executed in their correct order.

12 Real-Time Task-based Scheduling

- **Rate-monotonic scheduling (RMS)** - static priorities are assigned according to the cycle duration of the job, the shorter cycle duration the higher priority, called fixed-priority assignment.
- **Earliest deadline first (EDF)** - dynamic algorithm that places processes in priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline.
- **RMS > EDF** - the advantages RMS has over EDF:
  - **Fixed priority** - is easier to implement when priority is static.
  - **Run-time** - is less complexity and overhead.
  - **Overload** - is easier to predict assignment (lower-priority will miss their deadline).
  - **EDF has a domino-effect** - due to its less predictable a large amount of tasks can unnecessarily miss their deadline.
- **EDF > RMS** - the advantages EDF has over RMS, is that EDF is always able to exploit the full processor capacity.

14 Response Time based Schedulability Analysis

- **Response Time Analysis (RTA)** - an exact (necessary and sufficient) prediction of the worst-case response time of each fixed-priority task on single-processor system.
- **Worst-case execution time** - obtained by measurement and analysis.
- **Aperiodic and sporadic tasks** - an infinite sequence of identical jobs, that finishes at an irregular rate (e.g. user or external event interaction).

15 Task Interactions and Blocking

- **Priority inversion** - a problematic scenario when a high priority task is indirectly preempted by a lower priority task effectively "inverting" the relative priorities of the two tasks.
- **Priority inheritance protocol** - a method for eliminating priority inversion, by increasing the priority of a process (A) to the maximum priority of any other process waiting for any resource on which A has a resource lock.
- **Direct blocking** - a high-priority task is blocked when trying to acquire a resource held by a lower-priority task.
- **Push-through blocking** - a consequence of the priority inheritance protocol when an intermediate-priority task cannot run because a lower-priority task has temporarily inherited a higher priority.
- **Priority ceiling protocol** - a synchronization protocol to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections. In this protocol each resource is assigned a priority ceiling, which is a priority equal to the highest priority of any task which may lock the resource. The protocol works by temporarily raising the priorities of tasks in certain situations, thus it requires a scheduler that supports dynamic priority scheduling.
- **Worst-case response time R_i of task τ_i**

$$R_i^{(k+1)} = C_i + B_i + \sum_{j \in hp} \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j,$$

C_i	worst-case execution time,
B_i	worst-case blocking time,
T_i	task's period.

Bibliography

1. Wikipedia: `https://en.wikipedia.org/wiki/Concurrent_computing`
2. I. C. Bertolotti and G. Manduchi: Real-Time Embedded Systems: Open-Source Operating Systems Perspective