INTER-PROCESS COMMUNICATION AND SYNCHRONISATION:
Lesson-12: Signal Function

1. Signal

Signal

- One way for messaging is to use an OS function signal ( ).
- Provided in Unix, Linux and several RTOSes.
- Unix and Linux OSes use *signals* profusely and have thirty-one different types of *signals* for the various events.

Signal

- A signal is the software equivalent of the flag at a register that sets on a hardware interrupt. Unless masked by a *signal* mask, the *signal* allows the execution of the *signal* handling function and allows the handler to run just as a hardware interrupt allows the execution of an ISR

Signal…

- Signal is an IPC used for signaling from a process A to OS to enable start of another process B
- Signal is a one or two byte IPC from a process to the OS.

Signal …

- Signal provides the shortest communication.
- The *signal* ( ) sends a one-bit output for a process, which unmasks a signal mask of a process or task (called signal handler)
- The handler has coding similar to ones in an ISR runs in a way similar to a highest priority ISR.

Signal …

- An ISR runs on an hardware interrupt provided that interrupt is no masked. The signal handler also runs on signal provided that signal is no masked

Signal …

- Signal ( ) forces a signaled process or task called signal handler to run.
- When there is return from the signaled or forced task or process, the process, which sent the signal, runs the codes as happens on a return from the ISR.

Signal ...

- OS connects a signal to a process or ISR *j* (called signal handler function), and resets the signal mask of *j*.
- Then the *j* runs after all higher than *j* priority processes (or ISRs) finish.

Signal ...

- An OS provision for *signal* as an IPC function means *a* provision for interrupt-message from a process or task to another process or task

Signal ( ) IPC function

1. SigHandler ( ) to create a signal handler corresponding to a signal identified by the signal number and define a pointer to signal context. . The signal context saves the registers on signal.
2. Connect an interrupt vector to a signal number, with signaled handler function and signal handler arguments. The interrupt vector provides the program counter value for the signal handler function address.

Signal ( ) IPC functions

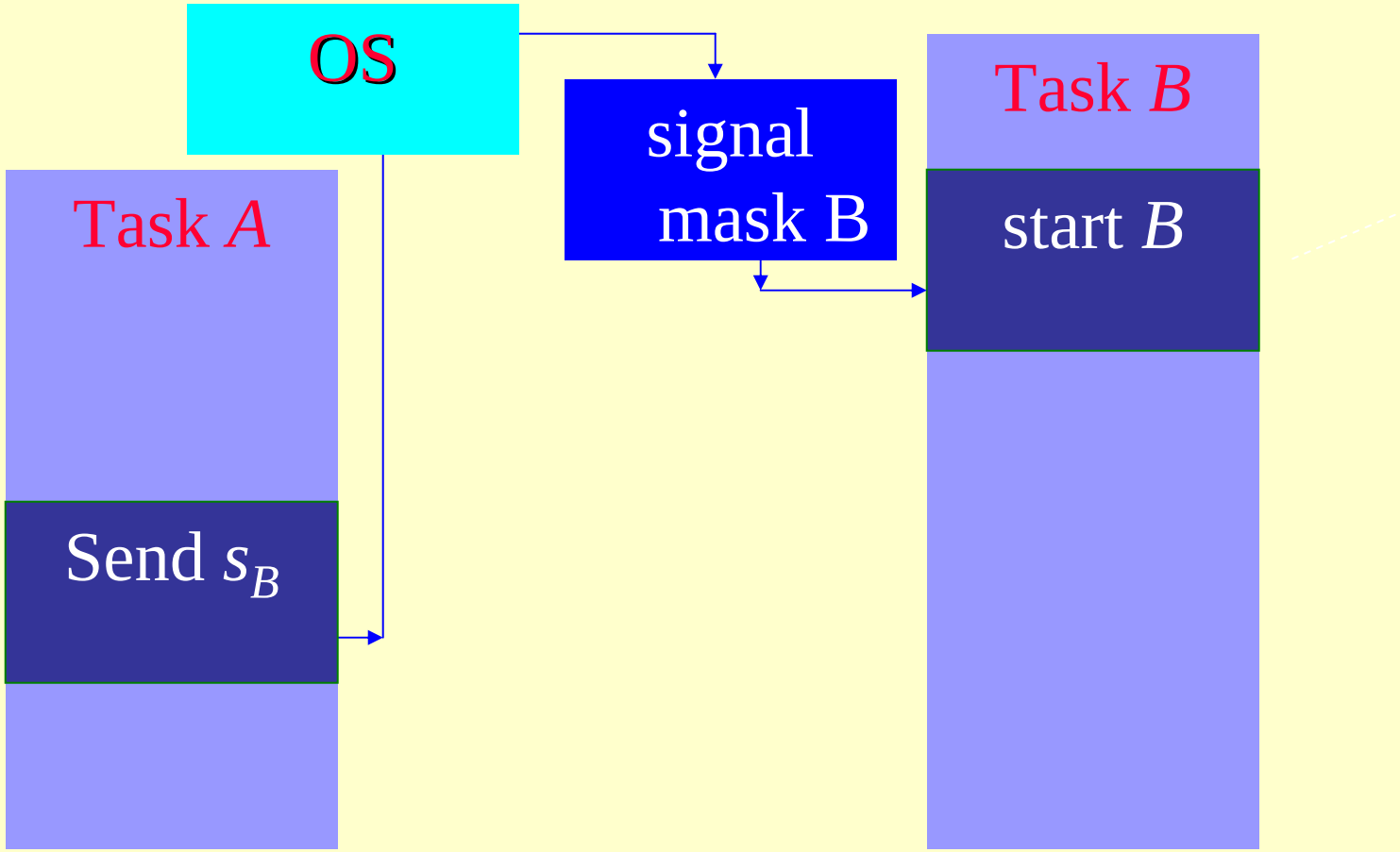
3. A function signal ( ) to send a signal identified by a number to a signal handler task
4. Mask the signal
5. Unmask the signal
6. Ignore the signal

2. Comparison between signal and semaphore

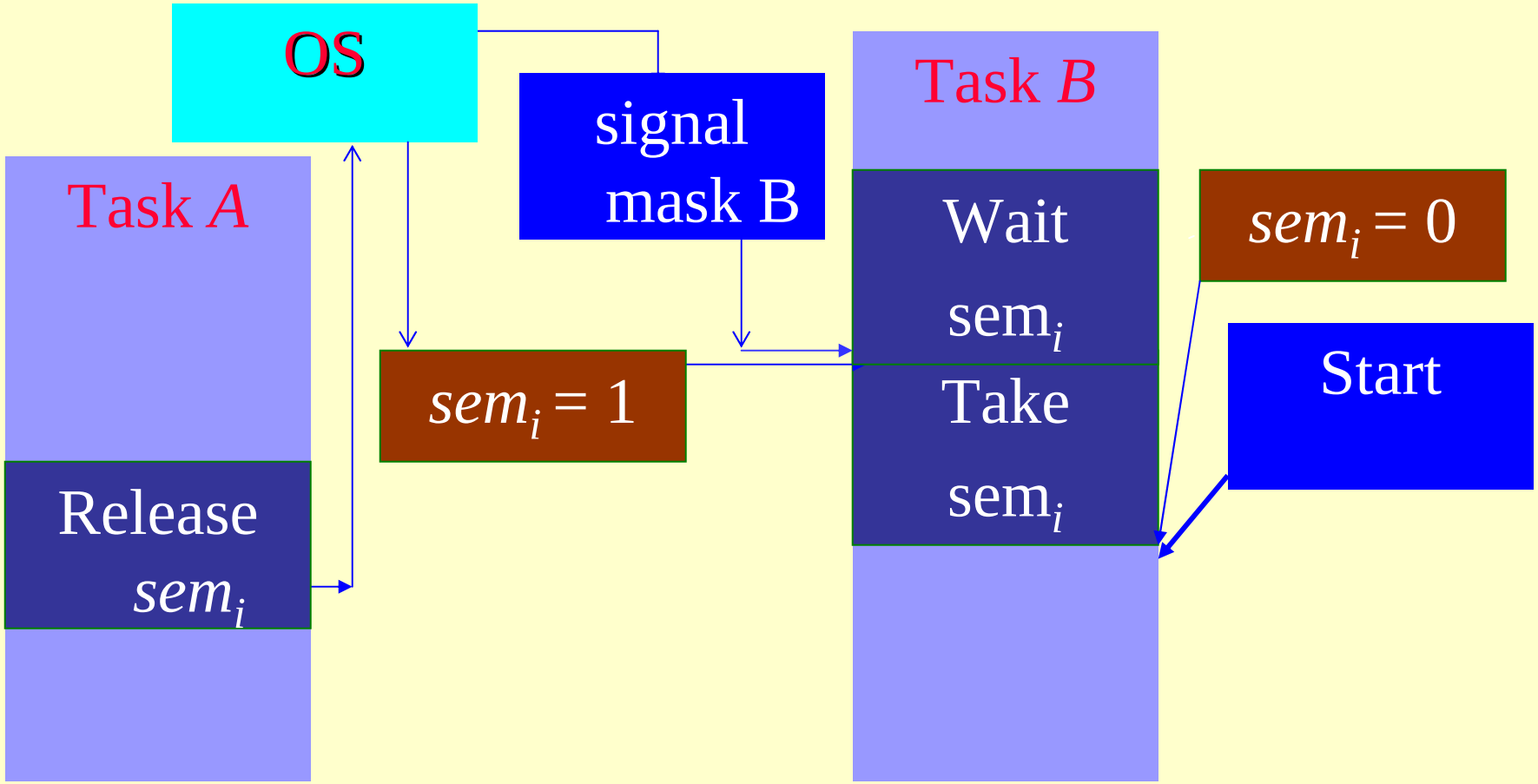
Signal and Semaphore

- Some OSes provide the signal and semaphore both IPC functions
- Every OS provides semaphore IPC functions.
- When the IPC functions for *signal* are not provided by an OS, then the OS employs semaphore for the same purpose.

Task *A* sending signal s_B to initiate Task *B* (signal handler) to run



Task *A* sending semaphore as event flag sem_i to initiate a task section waiting to take sem_i before it could run



3. Example of using signal ( )

Signal ( ) Example

- Handling of *exception*.
- An exception is a process that is executed on a specific reported run-time condition.

Signal ( ) Example

A *signal* reports an error (called 'Exception') during the running of a task and then lets the scheduler initiate an error-handling process or ISR, for example, initiate error-login task.

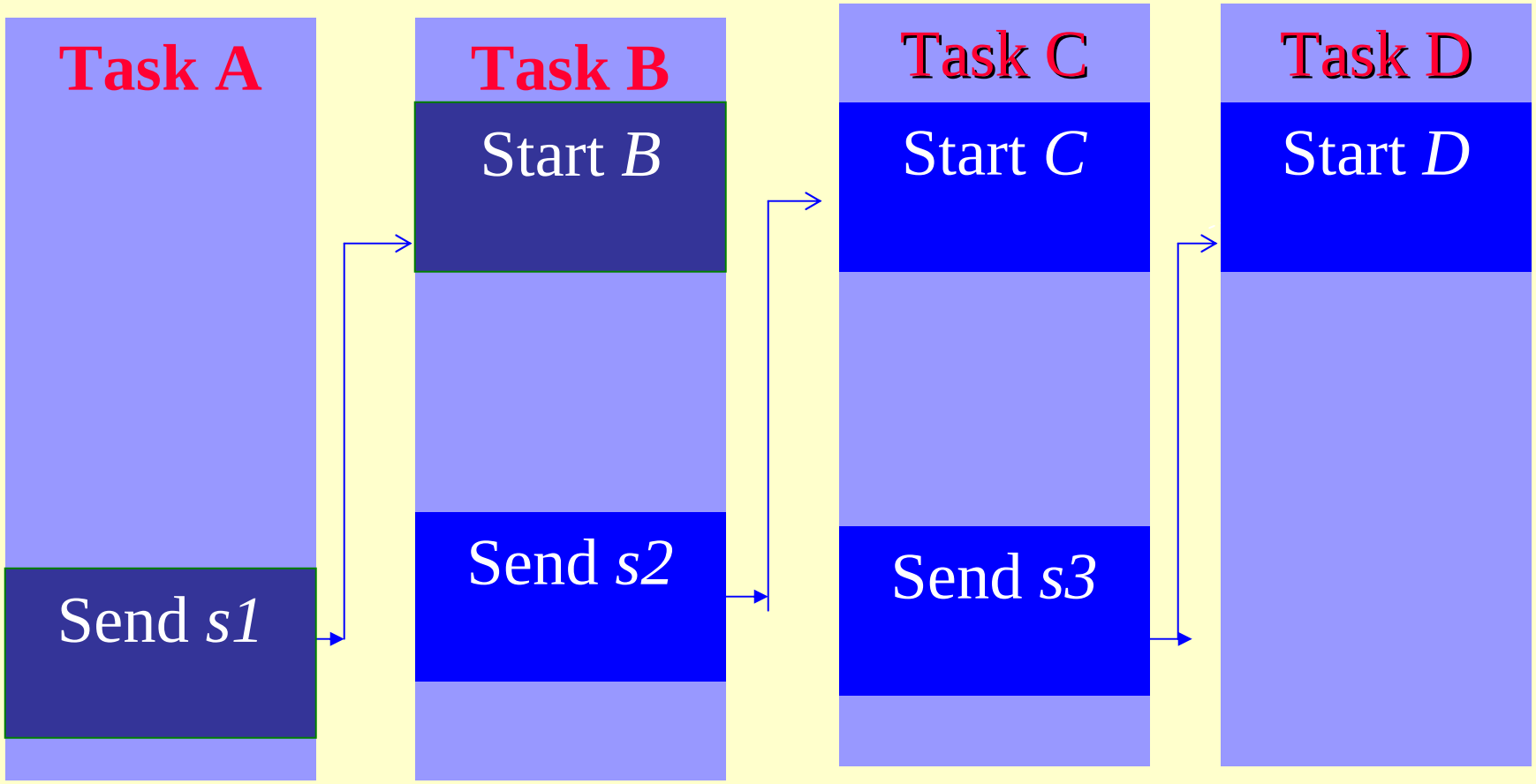
- Handling of *signal* is similar to an ISR handling function using an interrupt vector.

4. Advantage of using a Signal

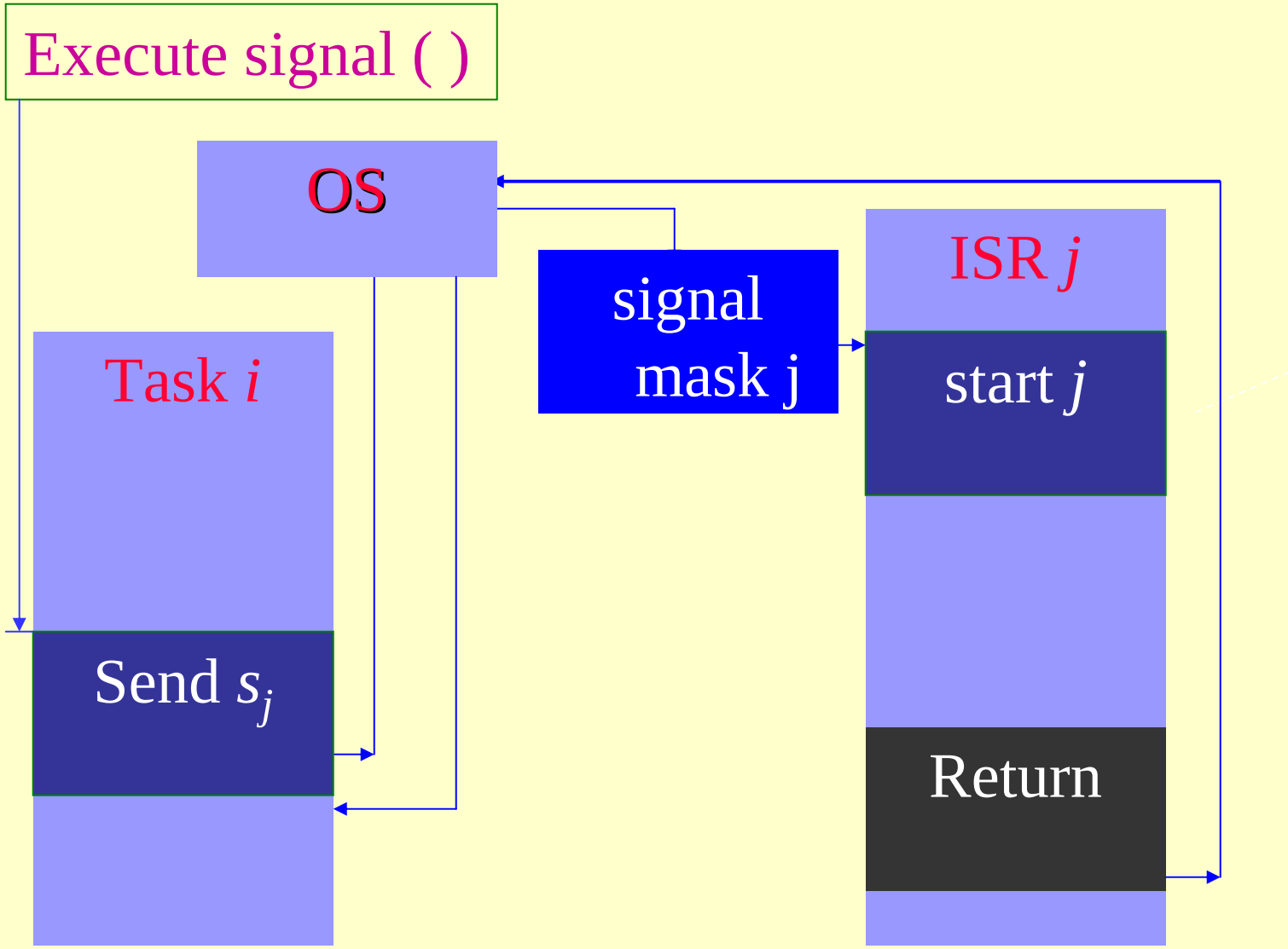
Advantage of using a Signal

- Unlike semaphores, it takes the shortest possible CPU time. The signals are the flag or one or two byte message used as the IPC functions for synchronizing the concurrent processing of the tasks.

Signal handler Tasks B, C, and D Synchronizing their execution by signals s1, s2 and s3



Task *i* sending signal *s* to initiate signal handler ISR *j*



Advantage of using a Signal

1. A signal is the software equivalent of the flag at a register that sets on a hardware interrupt.
2. It is sent on some exception or on some condition, which can set during running of a process or task or thread.
3. Sending a signal is software equivalent of throwing *exception* in C/C++ or Java program

Advantage of using a Signal

4. Unless process is masked by a *signal* mask, the *signal* allows the execution of the *signal* handling process, just as a hardware interrupt allows the execution of an ISR

Advantage of using a Signal

5. A signal is identical to setting a flag that is shared and used by another *interrupt servicing process.*
6. A *signal* raised by one process forces another process to interrupt and to catch that *signal* provided the *signal* is not masked at that process.

5. Drawbacks of using a Signal

Drawbacks of using a Signal

1. Signal is handled only by a very high priority process (service routine). That may disrupt the usual schedule and usual priority inheritance mechanism.
2. Signal may cause reentrancy problem [process not retuning to state identical to the one before signal handler process executed].

Summary

We learnt

- (i) The simplest IPC for messaging and synchronizing processes is signal. A signal provides the shortest message. Signals are used for initiating another ISR or task and error handling process called signal handler.

  (ii) A signal is equivalent throwing exception in Java/C/C++ or setting of an interrupt flag

End of Lesson-12 on Signal