Parallel Processing

Parallel Processing means **Asynchronous Type** of function module.

Generally when we call a function module, it will stop the current program, execute another (called) program and then returns control to original program and again original program starts execution.

With asynchronous function module control will start parallelly without stopping the current program from which you called the function module.

We use parallel processing when we have to process large amount of data.

In parallel processing, a job step is started as usual in a background processing work process. A program that runs in a job step can be programmed to use a special variant of asynchronous RFC to have portions of the data to be processed run in parallel in other work processes. You can recognize such a program by its use of the CALL FUNCTION STARTING NEW TASK DESTINATION IN GROUP instruction to start the function modules that process the data.

While the job itself runs in a background process, the parallel processing tasks that it starts run in dialog work processes. Such dialog work processes may be located on any SAP server.

Parallel processing has been implemented in some SAP applications that have long-running reports. The parallel processing interface is also available directly to customers.

**Pre-requisites to use parallel processing :-**

- The function module that you call must be marked as externally callable. This attribute is specified in the **Remote function call supported** field in the function module definition (transaction SE37).
- The called function module may not include a function call to the destination "BACK."
- The calling program should not change to a new internal session after making an asynchronous RFC call. That is, you should not use SUBMIT or CALL TRANSACTION in such a report after using CALL FUNCTION STARTING NEW TASK.
- You cannot use the CALL FUNCTION STARTING NEW TASK DESTINATION IN GROUP keyword to start external programs.
- In calls between systems, both systems must be of Release 3.0A or higher.

**Function Modules and ABAP Keywords for Parallel Processing**

```
CALL FUNCTION RemoteFunction ( FM Name ) STARTING NEW TASK Taskname
```

With this ABAP statement, you are telling the SAP system to process function module calls in parallel. Typically, you'll place this keyword in a loop in which you divide up the data that is to be processed into work packets.

**DESTINATION IN GROUP**

```
Unknown macro: (group|DEFAULT)
```

Specifying **IN GROUP** as a destination allows you to execute multiple function modules in parallel on a predefined group of application servers in the current SAP system.

For **group**, you must specify a data object of the type **RZLLI_APCL** from the ABAP Dictionary, one that is either initial, or one that includes the name of an RFC server group created in transaction RZ12.

When specifying **DEFAULT**, or if **group** is initial, all application servers that are currently available in the current SAP system are used as a group.

```
Unknown macro: {PERFORMING subr}
```
ON END OF TASK

You can use this addition to specify either a subprogram subr or, as of Release 6.20, a method meth as a callback routine, which is executed after the asynchronously called function module has finished. The method meth must be public, and can have only one non-optional input parameter p_task of type clike.

WAIT: ABAP keyword

WAIT UNTIL <logical expression>

Required if you wish to wait for all of the asynchronous parallel tasks created with CALL FUNCTION to return. This is normally a requirement for orderly background processing. May be used only if the CALL FUNCTION includes the PERFORMING ON RETURN addition.

WAIT UNTIL log_exp [UP TO sec SECONDS].

This variant of the statement WAIT is only intended to be used after an asynchronous RFC with callback routines. It interrupts the program execution for as long as the result of the logical expression log_exp is false. Any logical expression can be specified for log_exp.

If the result of log_exp is false, the program waits until a callback routine of a function that was previously called asynchronously has been executed, and then checks the logical expression again. If the result of the logical expression is true, or if the callback routines of all functions that were previously called asynchronously have been executed, then the program execution continues with the statement following WAIT.

Specifying UP TO restricts the program interruption to a maximum number of seconds, specified in sec.

RECEIVE: ABAP keyword

§ RECEIVE RESULTS FROM FUNCTION Remotefunction

§ Required if you wish to receive the results of the processing of an asynchronous RFC. RECEIVE retrieves IMPORT and TABLE parameters as well as messages and return codes.

RECEIVE RESULTS FROM FUNCTION func	parameter list [KEEPING TASK].

This statement can be used in a callback routine specified for the asynchronous RFC, in order to receive output parameters of an asynchronously called function func in the parameter list parameter list, and to assign return values to exceptions.

With the addition KEEPING TASK, the asynchronous RFC connection is retained, as is the context of the called function module along with it. When a new call is made with the same task ID, the same global data of the function group is addressed. Without the addition KEEPING TASK, an asynchronous RFC connection is finished after the remote function is executed or the results are copied.

When we call function module in parallel processing, no values can be copied with IMPORTING and CHANGING.

Example Program -

REPORT ztest1.

parameters: p_kunnr type kna1-kunnr.

data: functioncall1(1) type c,
   cstgdetail1 type bapicustomer_kna1.
constants: done(1) type c value 'X'.

start-of-selection.

call function 'BAPI_CUSTOMER_GETDETAIL2'
  starting new task 'FUNC1'
  destination 'NONE'
  performing set_function1_done on end of task
  exporting
customerno = p_kunnr.

* Receive remaining asynchronous replies
wait until functioncall1 = done.

write: / cstgdetail1.

* ---------------------------- FORM FUNCTION1_DONE ---------------------------- *

form set_function1_done using taskname.

receive results from function 'BAPI_CUSTOMER_GETDETAIL2'
importing
customergeneraldetail = cstgdetail1.

functioncall1 = done.

endform.