CREATE SERVICE PROGRAM IN YOCTO PROJECT AND USE SYSTEMD

SystemD

It helps to provide advanced functionality and performance in your custom linux distributions.
It will initialize process during booting time, managing system process and provides services for the system becomes ready.
SystemD Functionality are ::-
Initialize system Booting process
Monitoring file system
Managing system hardware resources.
check Start, restart, stop , status process of every service.
Process for system reboot and shutdown, poweroff
systemd.bbclass file are also available in this directory to check what is in.
$ vi /home/arjun/Yocto-project/poky/meta/classes/systemd.bbclass

SystemV

Yocto project defaults use systemV in the Build image.
SystemV is basically a small process manager then comparison to SystemD.
It is provides only few service than comparison to SystemD.
It uses the Busybox process to initialize the system Booting process and other functionality process in the build image.
SystemV is not applicable to start the process is paralley.
SystemD is applicable to start the process is paralley.
systemv.bbclass file are also available in this directory to check what is in.
$ vi /home/arjun/Yocto-project/poky/meta/classes/systemv.bbclass

How to add system is yocto project.

First set the build environment in yocto project in poky folder.
$ source oe-init-build-env ../build
Enter in the conf folder.
$ cd conf
Open the local.conf file.
$ vi local.conf
Add this lines at the last of this file.
DISTRO_FEATURES:append = " systemd"
VIRTUAL-RUNTIME_init_manager = "systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED += "sysvinit"
VIRTUAL-RUNTIME_initscripts = " systemd-compat-units"

Save and exit the file.

DISTRO_FEATURES:append = This variable is used to enable the systemd features in my distro policy or provide the systemd in poky or my custom linux distribution.
It is a variable defined in the configuration file which lists the software features you want to distribute or shipped with your custom linux image.

VIRTUAL-RUNTIME_init_manager = It is a variable which is used in the yocto-project. It controls the init system , which is the first program that runs after the linux kernel boots up. The init system is responsible for starting other essential services and bringing the system up to a usable state**.**

DISTRO_FEATURES_BACKFILL_CONSIDERED = It is a variable which is used in yocto-project recipes to manage the transition between different init systems.
This variable helps to manage compatibility during init system changes.

VIRTUAL-RUNTIME_initscripts = It is a variable which is used in yocto project recipes alongside VIRTUAL-RUNTIME_init_manager. It works together with the init system to control how startup scripts are handled.
This variable determines which init scripts are included in your yocto project image.

System-compat-units = It refers to a package containing compatibility scripts or unit files that bridge the gap between the traditional sysvinit init system and the newer systemd include.
A package containing compatibility units.

Now basic details are completed .
Now to time build the build with include systemd features in your image.

$ bitbake core-image-minimal

It takes around 2-3 hours to build all things. If build successfully complete then you are run your qemu image.
Run the image with using qemu.
$ runqemu qemux86-64 nographic

And boot will start then you check the booting logs are show for your image and it include all the systemd booting logs are also show.
Now login your image name is root

Check the your systemd command is work or not.
$ systemctl

This command will know your image works or include with systemd part.

****

WRITE THE SIMPLE HELLO-WORLD SERVICE PROGRAM IN YOCTO

Enter first your creating layer.
$ cd /poky/meta-txt/recipes-example

Create one services based folder in recipes-example
$ mkdir helloworldservice

Enter this folder
$ cd helloworldservice

Create two file one is folder and other one is recipes(.bb) files.
$ mkdir files
$ vi helloworldservice.bb

Open first recipes files and write this format structure.
$ vi helloworldservice.bb

In this recipes(.bb) files.

```
SUMMARY = " A simple basic examples which is based on service program"
DESCRIPTION =  " Run Simple hello-world service"
LICENSE = " MIT"
LIC_FILES_CHKSUM = "file://${COREBASE}/meta/COPYING.MIT;md5=----------------------------------------"
SRC_URI = " file://helloworldservice.sh \
           file://helloworldservice.service "

Inherit systemd

S = "${WORKDIR}"

RDEPENDS:${PN} = " bash"

SYSTEMD_AUTO_ENABLE = "enable"

SYSTEMD_SERVICE:${PN} = "helloworldservice.service"

do_install() {

install   -d ${D}${bindir}
install  -m 0755 ${S}/helloworldservice.sh  ${D}${bindir}
install   -d ${D}/${sysconfdir}/systemd/system
install   -m 0644 ${S}/helloworldservice.service    ${D}/${sysconfdir}/systemd/system

}
```

Save and exit the files.

Enter in the files folder

```
$ cd files
```

Create two files in this folder

```
$ touch helloworldservice.service      helloworldservice.sh
```

* .service is a service file
* .sh is a shell script file

First Enter the shell script file.

```
$ vi helloworldservice.sh
```

Write the shell script data in this file *.sh
In helloworldservice.sh

```
#!/bin/bash
while true;
do
echo "Hello-world service based program !"
sleep1
done
```

save and exit the file.

Now open service file.
$ vi helloworldservice.service

In helloworldservice.service file data

[Unit]
Description = Continuous Hello-world service program
[Service]
ExecStart = /bin/bash /usr/bin/helloworldservice.sh
[Install]
WantedBy = multi-user target

Save and exit.

Go to the Build folder for set environment.
$ cd poky

Set the environment
$ source oe-init-build-env ../build

Enter in the conf folder
$ cd conf

Open the local.conf file
$ vi local.conf

Add this line at the end of this file.
IMAGE_INSTALL:append = " helloworldservice"

Save and exit the file

Now time to build the service program first.
$ bitbake helloworldservice

Now time to build the image.
$ bitbake core-image-minimal

Ouput also check for your program in work directory
$ tmp / work / core2-qemu----/1.0.r0 / helloworldservice/1.0.r0/

Enter ls command to show all the files
$ ls

One folder is created in this folder name is image
$ cd image

Show where is your scripts will include.

Also check output for your service program in the image.

run your image with using qemu.
$ runqemu qemux86-64 nographic

login name is **root**

first start your service in the image.
$ systemctl start helloworldservice

Now check the status of your service
$ systemctl status helloworldservice

also check your shell script files is present in the /usr/bin folder.

$ cd /usr/bin

$ ls helloworldservice

Show your service program

********** END **********