UNIT V: REAL-TIME OPERATING SYSTEM TOOLS AND CASE STUDIES

Use of µC/OS-II - Case study of coding for an Automatic Chocolate Vending Machine using MUCOS RTOS - Case study of an Embedded system for an Adaptive Cruise Control Systems in a Car - Case study of an Embedded Systems for a Smart Card.

STUDY OF CODING FOR AN AUTOMATIC CHOCOLATE VENDING MACHINE USING MUCOS RTOS:

1. ACVM Specifications:
   - Alphanumeric keypad and Display.
   - Alphanumeric keypad on the top of the machine.
   - A child interaction with it when buying a chocolate.
   - Owner commands and interaction with the machine.
   - Three line LCD display unit on the top of the machine.
   - Displays menus, entered text, pictograms, and welcome, thank and other messages, and time and date.
   - Child as well as the ACVM owner GUIs with the machine using keypad and display.
• Coin insertion and delivery slots, and Internet port.
• Coin insertion slot so that the child can insert the coins to buy a chocolate.
• Delivery slot to collect the chocolate, and coins if refunded.
• Internet connection port so that owner can interact with ACVM from remote.

**Basic system in ACVM**

---

**Reprogram of the codes and relocation of the codes**

The system ROM or flash or EPROM whenever the following happens:

(i) the price of chocolate increases,

(ii) the message lines or menus or advertisement graphics needs to be changed or

(iii) machine features change.
RTOS
An RTOS has to schedule the buying tasks from start to finish.
Let μC/OS-II be the RTOS used in ACVM.

ACVM Requirements:

Purpose:
- To sell chocolate through an ACVM from which children can automatically purchase the chocolates.
- The payment is by inserting the coins of appropriate amount into a coin-slot.

Inputs:
- Coins of different denominations through a coin slot.
- User commands.

Signals, Events and Notifications:
- A mechanical system directs each coin to its appropriate port– Port_1, Port_2 or Port_5.
- Each port generates an interrupt on receiving the coin at input.
- Each port interrupt starts an ISR, which increases the value of amount collected by 1 or 2 or 5 and posts an IPC to a waiting task the system.
- Each selected menu choice gives a notification to the system.

Outputs:
- Chocolate and signal (IPC) to the system that subtracts the cost from the value of amount collected.
- Display of the menus for GUIs, time and date, advertisements, welcome and thank messages
Functions of the system:

- A child sends commands to the system using a GUI (graphic user interface).
- GUI consists of the LCD display and keypad units.
- The child inserts the coins for cost of chocolate and the machine delivers the chocolate.
- If the coins are not inserted as per the cost of chocolate in reasonable times then all coins are refunded.
- If the coins are inserted of amount more than the cost of chocolate, the excess amount is refunded along with chocolate.
- The coins for the chocolates purchased collect inside the machine in a collector channel, so that owner can get the money, again through appropriate commands using the GUI.
- USB wireless modem enables communication through Internet to the ACVM owner.

Design metrics:

- **Power Dissipation**: As required by mechanical units, display units and computer system
- **Performance**: One chocolate in two minutes and 256 chocolates before next filling of chocolates into the machine.[Assumed]
- **Process Deadlines**: Machine waits for maximum 30 s for the coins and machine should deliver the chocolate within 60 s.
- **User Interfaces**: Graphic at LCD or touch screen display on LCD and commands by children or machine owner through fingers on keypad or touch screen
- **Engineering Cost**: US$ 50000 (assumed)
- **Manufacturing Cost**: US$ 1000 (assumed)

Test and validation conditions:

- All user commands must function correctly.
- All graphic displays and menus should appear as per the program.
- Each task should be tested with test inputs.
- Tested for 60 users per hour.
Specifications Modeling Using UML:

Class diagram for ACVM Part-1

Class diagram for ACVM Part-2
Classes Display_Output and User Keypad_Input and Objects task_Display and taskUser_Keypad_Input

Objects Task_Display and TaskUser_KeypadInput
State Diagram:

State Diagram Part-1

State Diagram Part-2
Hardware Architecture:

ACVM hardware:
ACVM specific hardware to sort the coins of different denomination and each denomination.
Main Power supply 220 V 50 Hz or 110 V 60 Hz. Internal circuits drive by supply of 5 V 50mA for electronic and 12 V 2 A for mechanical systems.
A TCP/IP port.
A 1 s resolution timer is obtained by programming 8051 timer.
Flash memory part of ROM and RAM for storing temporary variables and stack.
8 MB ROM for application codes and RTOS codes.
Microcontroller 8051MX.
Software Architecture:

Software architecture (ISRs and Tasks) for ACVM

![Diagram of ACVM software architecture](image)

Fig. 11.6 Software architecture of ACVM

Multiple tasks and their synchronization model:

Multiple tasks and their synchronization model using semaphores and mailbox messages

![Diagram showing task synchronization](image)
Tasks and their priority, action and IPCs:

Task_ReadPorts:
- Priority — 9.
- Action— Waits for the coins and action as per coins collected.
- IPC pending: Event signal (s) from Port_1, Port_2 and Port_5; SemStimeout,
- IPC posted: *MboxAmount.

Task_Collect:
- Priority — 11.
- Action— Waits for coins = or > cost till timeout and act accordingly.
- IPC pending: SemFlag1;*MboxAmount.

Task_Deliver
- Priority — 12.
- Action— Waits for SemFlag2, delivers chocolate, and decreases coins’ amount after delivery.
- IPC pending: SemFlag2.

Task_Refund:
- Priority — 17.
- Action— Waits for refund event and refunds the amount.
- IPC pending: SemFlag3.
- IPC posted: *Str3.

Task_ExcessRefund
- Priority — 13.
- Action— Refunds the Excess excess amount
- IPC pending: SemFlag2,*MboxAmount.
- IPC posted: *Str4.

Task_Display:
- Priority — 15.
• Action—Waits for the message mails and display as per message.
• IPC pending: *collect, *delivered,*refund, *excessRefund, Str2, Str3, Str4 and *TimeDate.
• IPC posted.

CASE STUDY OF AN EMBEDDED SYSTEM FOR AN ADAPTIVE CRUISE CONTROL (ACC) SYSTEM IN A CAR:

Embedded Systems in a car

• Cruise control a system that takes charge of controlling the throttle from the driver and cruising the vehicle at preset and constant speed.
• may also maintain string stability in case of multiple cars streaming through highway and in case of VIP convoy.
Adaptive Cruise Control (ACC):

- Using an adaptive algorithm, ACC system maintains constant speed and can be added string stability feature in case of multiple cars streaming on highway.
- String stability— maintaining inter-car distances constant.
- Cruise control relieves the driver from that duty and the driver hands over the charge to the ACC.

When (1) road conditions are suitable (not wet or icy, or
(2) there are no strong winds or fog), or
(3) car is cruising at high speed and when there is no heavy traffic.

- The driver resumes the charge in adverse conditions.

Control front-end panel:

- Switch cum Display for 'ON', for 'OFF', 'COAST', RESUME', SET/ACCELERATE.
• The driver activates or deactivates, the ACC system by pressing ON or OFF, respectively.

Adaptive Control:
• An adaptive control—algorithm used to adapt to the current status of control inputs.
• Parameters adapt dynamically.
• In place of a constant set of mathematical parameters in the algorithm equations, the parameter are continuously adapted to the status at an instance.

**ACC System Requirements:**

**Purpose:**

Controlled cruising of car using adaptive control algorithm for continuous maintaining the car speed and inter-car distances.

**Inputs:**

• Present alignment of radar (or laser) beam emitter.
• Delay interval in reflected pulse with respect to transmitted pulse from emitter.
• Throttle position from a stepper motor position sensor.
• Speed from a speedometer.
• Brake status for brake activities from brake switch and pedal.

ACC System Ports and tasks:

• **Port_Align**— for a motor control for steps up clockwise or anticlockwise on a signal from task_Align, aligns radar or UVHF transmitting device in the lane of the front-end car.

• **Port_ReadRange**— for measuring front end-car range. Time difference deltaT is read on a signal from task_Signal to port device.

• **task_ReadRange** to read using the Port device_ReadRange circuit for the computations of deltaT between the transmission and reception instances. deltaT $1.5 \times 10^5$ measures the range rangeNow (= present range or front-car distance $d$) of the front-end car.

• **task_ReadRange** to send message for speedNow ($= velocity v$) to task_RangeRate and transmits same to all other streaming behind cars.

• **Port_Speed** — to send speed to port control function routine on receiving a signal from task_Speed.

• **Port_Brake**— to apply the brakes or emergency brakes on an interrupt signal, which runs service routine ISR_BrakeControl.

Signals, Events and Notifications:

• User commands given as signals from switches/buttons. User control inputs for ACC ON, OFF, Coast, resume, set/accelerate buttons.

• Brake event (Brake taping to disable the ACC system, as alternative to "cancel" button at front panel).

• Safe/Unsafe distance notification.

Outputs:

• Transmitted pulses at regular intervals.
• Alarms.
• Flashed Messages.
• Range and speed messages for other cars (in case of string stability mode).
• Throttle-valve and Brake control.
• Output to pedal system for applying emergency brakes and driver nonintervention for taking charge of cruising from the ACC system.

Control front-end panel:
• Switch with display—‘ON’, for ‘OFF’, ‘COAST’, RESUME’, and SET/ACCELERATE.
• COAST or RESUME switch to enable driver handover or resumes the ACC system charge.
• SET/ACCELERATE switch to set cruise speed upwards or downward.
• Switch with display glows to show green or red as per the status when the ACC activation
• Alarms and message flashing unit issues appropriate alarms and message flashing pictograms.

Functions of the system:
• Cruise control system takes charge of controlling the throttle position from the driver and enables the cruising of the vehicle at the preset constant speed. A radar system helps in maintaining intercar distance and warns of emergency situations.
• An alignment circuit aligns the radar emitter. When driving in a hilly area, the emitter alignment is must. A stepper motor aligns the attachment so that transmitter beam of radar emits with the
• required beam alignment for the given driving lane and divergence so maintain the in-lane line of
• sight of the front-end car. task_Align does this function.
• Transmit modulated pulses emit at periodic intervals and the delay period in receiving its reflection from front-end vehicle.
• The measured delay deltaT at periodic intervals.
• deltaT multiplied by 1.5 × 10^8 m/s (half of light velocity) gives the computed distance \( d \) (= Range Now) of front end car at that instance.
• The differences of $d$ with respect to safe distances $d_{safe}$ and preset distances $d_{set}$ (in case of maintaining string stability) are cyclically estimated.

• The speedometer measures the speed and $task_{Speed}$ compute error in preset speed and measured speed.

All estimated differences are cyclically sent as input to an adaptive algorithm, which adapts the control parameters and sends computed output to vacuum actuator of the throttle valve in car.

✓ $task_{Algorithm}$ for computations.

✓ $task_{Throttle}$ initiates the control output functions for this action.

✓ Interrupt service routine $ISR_{ThrottleControl}$ does the critical functions of throttle control.

✓ The car decelerates and accelerates as per setting of throttle valve orifice at an instance.

The brake is controlled when the safe distance is not maintained and warning message is flashed on the screen.

$task_{Brake}$ initiates the critical functions of brake control.

Interrupt service routine $ISR_{BrakeControl}$ performs the brake critical functions.

When battery power becomes low, the ACC system deactivates after issuing alarm and flashing messages (notifications).

**Design metrics:**

✓ **Power Source and Dissipation:** Car Battery operation.

✓ **Resolution:** 2 m inter-car distance.

✓ **Performance:** Safe distance setting 75 m to 200 m. No overshooting of controlled output for the throttle.

✓ **Process Deadlines:** Less than 1 s response on observation of unsafe distance of front-end car.

✓ **User Interfaces:** Graphic at LCD or touch screen display.

✓ **Extendibility:** The system is extendable to maintain string stability of multiple cars in a row.

✓ **Engineering Cost:** US$ 50000 (assumed).

✓ **Manufacturing Cost:** US$ 600 (assumed).
Test and validation conditions:

- Tested in dense as well light traffic conditions.
- Tested on plains, hills and valley roads.
- All user commands must function correctly.

3. Detailed functioning in Adaptive Cruise Control (ACC) System

- Retrieve the front end-car distance information from a radar or UVHF attachment at the front string wheel.
- A stepper motor aligns the attachment so that transmitter of radar maintains the line of sight to front-end car. The radar system maintains string stability and warns of emergency situations.
- Get road speed from Speedo-metric section of DAS unit.
- Get acceleration from engine section.
- Run adaptive algorithm to calculate and send control signals to stepper motor actuator at vacuum-valve.
- Orifice opening of vacuum valve controls electro-pneumatic throttle valve.
- Receive new throttle position by stepper motor position sensor.
- Get inputs of brake switch status for monitoring brake activities.
- Send output to pedal system when applying emergency brakes.
- Driver intervenes on taking charge of cruising from ACC.

Functioning of system by synchronization of tasks in ACC

![Diagram of Adaptive Cruise Control System]
Classes and class diagram:

Class diagram for ACVM Part-1

Hardware Architecture:

Hardware
ACC hardware:

A hardware system in automotive electronics has to provide functional safety. Important hardware standards and guidance—at present are following:

(a) TTP (Time Triggered Protocol),
(b) CAN (Controller Area Network),
(c) MOST (Media Oriented System Transport),
(d) IEE (Institute of Electrical Engineers) guidance standard exists for EMC (Electromagnetic Magnetic Control) and functional safety guidance.

- A microcontroller runs the tasks and ISRs except task_Algorithm.
- Internal RAM/ROM, ROM/Flash for RTOS codes for scheduling the tasks.
- CAN port interfaces with the CAN bus at the car.
- A separate processor with RAM and ROM for the task_Algorithm executes the adaptive control algorithm.
- Speedometer.
- Stepper motor based alignment unit.
- Stepper motor based throttle control unit.
- Transceiver for transmitting pulses through an antenna hidden under the plastic plates.
- LCD dot matrix display controller, display panel with buttons.
- Port devices—Port_Align, Port_Speed, Port_ReadRange, Port_Throttle and Port_Brake.

Software Architecture:

RTOS VxWorks used as alternative to OSEK-OS.

- OSEK OS standard is reliable compared to VxWorks or MUCOS.
- To demonstrate RTOS use in the ACC application, let us adapt VxWorks alternative for coding instead of OSEK-OS by adhering to the OSEK guidelines.
- Use BCC 1 type of tasks, as done in VxWorks application.
- Define each task of different priority and activate it only once in the codes.
- Use no message queues, mutex or counting semaphore.
- No in-between creation and deletion of tasks.
- Semaphores as event flags only with no run-time deletion or creation of these.
• Task can consist of three types of objects, event (semaphore), resource (statements and functions) and devices including port devices.

• Use MISRA C rules in coding.

• Use disable interrupts when a task or function enters critical section and enable interrupts when leaving critical section.

Tasks and their class, priority, action and IPCs:

Class BCC1 task_Align:
• Priority — 101.
• Action— Waits for the Reset cycle to start and send signal to Port_Align.
• IPC pending: Event signal (s) Reset.
• IPC posted: Align.
• Input: deltaStep, Step.
• Output: Step to Port_Align.

Class BCC1 task_Read-Range:
• Priority — 103.
• Action— Disable interrupts, get signal from Port, activate a radar flashing, records activation time, gets time of sensing the reflected radar signal and finds time difference, timeDiff. and Enable interrupts.
• IPC pending: Align.
• IPC posted: Range.
• Output: deltaT.

Class BCC1 task_Speed:
• Priority — 105.
• Action— Event Port_Speed starts a timer, counter start message and wait for the 10 counts for the number of wheel rotations.
• IPC pending:
• IPC posted: Speed.
• Output: speedNow.
Class BCC1task_Range-Rate:

- Priority — 107.
- Action— calculates rangeNow, get preset front car range and string Range from memory and compare. Get vset set cruising speed and compare it with current speed speedNow.
- IPC pending: Speed.
- IPC posted: ACC.
- Input: avgTireCircum, time-Diff, deltaT, stringRange, CruiseSpeed, and N_rotation
- Output: range-Error, speed-Error, range-Now, speed-Now.

Class BCC1task_Algorithm

- Priority — 109.
- Action—
  (i) Get errors of speed and range and execute adaptive control algorithm.
  (ii) Get errors of other vehicles through Port_RangeRate.
- Get other vehicles Port_Brake status.
- Get present throttle position.
- Send output, throttleAdjust to Port_Throttle.
- Send signal to Port_Brake in case of emergency braking action needed.
- Port_Brake transmits the action needed to other vehicles also.
- IPC pending: ACC.
- IPC posted: Reset.
- Inputs: range-Error, speed-Error, All Port_RangeRate values and Port_Brake statuses and VehicleID.
- Outputs: throttle- adjust and emergency for brake and throttle respectively.
Multiple tasks and their synchronization model:

Semaphores taken and given in cyclic order...
- Task_Alignment takes SemReset at cycle start and gives SemAlign.
- Task_Read Range takes SemAlign at start and gives SemRange.
- Task_Speed gives SemSpeed.
- Task_RangeRate takes SemSpeed taken at start and gives SemACC
- Task_Algorithm takes SemACC taken at start by and gives SemReset.

**ACC software for use in automobile**

- First be certified from organization authorized to issue that certification. We have seen that OSEK OS standard is required.
- Only those VxWorks or MUCOS functions which are adhering to OSEK must be used.
- Software coding IEC 61508 part 3 and MISRA C version 2 (2004) specifications of safety standards and coding language *must be used.*

**MISRA C:**

- MISRA stands for Motor Industry Reliability Association.
- MISRA C specifies a collection of rules to be used while coding in C.
MISRA-C is a standard for C language software and defines the guidelines for automotive systems for using C.


Details at http://www.misra.org.uk.

**CASE STUDY OF AN EMBEDDED SYSTEM FOR SMART CARD**

**Smart Card System Requirements:**

**Purpose:**

- Enabling authentication and verification of card and card holder by a host.
- Enabling GUI at host machine to interact with the card holder/user for the required transactions, for example, financial transactions with a bank or credit card transactions.

**Inputs:**

Received header and messages at IO port **Port_IO** from host through the antenna.
Internal Signals, Events and Notifications:

- On power up, radiation-powered charge pump supply of the card activated and a signal to start the system boot program at resetTask.
- Card start requestHeader message to task_ReadPort from resetTask.
- Host authentication request requestStart message to task_ReadPort from resetTask to enable requests for Port_IO.
- UserPW verification message (notification) through Port_IO from host.
- Card application close request requestApplClose message to Port_IO.

Outputs:

- Transmitted headers and messages at Port_IO through antenna.

Control panel:

- No control panel is at the card. The control panel and GUIs activate at the host machine (for example, at ATM or credit card reader)

Functions of the system:

- The card inserts at a host machine.
- The radiations from the host activate a charge pump at the card.
- The charge pump powers the SoC circuit consisting of card processor, memory, timer, interrupt handler and IO port, Port_IO.
- On power up, system reset signals resetTask to start.
- The resetTask sends the messages requestHeader and requestStart for waiting task task_ReadPort.
- task_ReadPort sends requests for host identification and reads through the Port_IO the host-identification message and request for card identification.
- task_PW sends through Port_IO the requested card identification after system receives the host identity through Port_IO.
- task_Appl then runs required API. The requestApplClose message closes the application.
- The card can now be withdrawn.
• All transactions between cardholder/ user now takes place through GUIs using at the host control panel (screen or touch screen or LCD display panel).

Design metrics:

✓ **Power Source and Dissipation:** Radiation powered contact less
✓ **Code size:** optimum. card system memory needs should not exceed 64 kB memory.
✓ **Limited use of data types:** multidimensional arrays, long 64-bit integer and floating points and very limited use of the error handlers, exceptions, signals, serialization, debugging and profiling.
✓ **File system(s):** Three-layered file system for data.
✓ **File management:** There is either a fixed length file management or a variable file length management with each file with a predefined offset.
✓ **Microcontroller hardware:** Generates distinct coded physical addresses for the program and data logical addresses. Protected once writable memory space.
✓ **Validity:** System is embedded with expiry date, after which the card authorization through the hosts disables.
✓ **Extendibility:** The system expiry date is extendable by transactions and authorization of master control unit (for example, bank servee).
✓ **Performance:** Less than 1s for transferring control from the card to host machine.
✓ **Process Deadlines:** None.
✓ **User Interfaces:** At host machine, graphic at LCD or touch screen display on LCD and commands for card holder (card user) transactions.
✓ **Engineering Cost:** US$ 50000 (assumed).
✓ **Manufacturing Cost:** US$ 1 (assumed).

Test and validation conditions:

✓ Tested on different host machine versions for fail proof card-host communication.
Classes and class diagram:

Class diagram

Classes:
- Task_CardCommunication is an abstract class from which extended to class (es) derive to read port and authenticate.
- The tasks (objects) are the instances of the classes Task_Appl, Task_Reset, Task_ReadPort and Task_PW.
- ISR1_Port_IO, ISR2_Port_IO and ISR3_Port_IO are interfaces to the tasks.

Other Classes:
Classes for the network, sockets, connections, datagrams, character-input output and streams, security management, digital-certification, symmetric and asymmetric keys-based cryptography and digital signatures.
Hardware Architecture:

Smart Card Hardware:

- A plastic card in ISO standard dimensions, 85.60 mm x 53.98 x 0.80 mm. It is an embedded SoC (System-On-Chip). [ISO standards - ISO7816 (1 to 4) for host-machine contact based card and ISO14443 (Part A or B) for the contactless cards.]
- Microcontroller MC68HC11D0 or PIC16C84 or a smart card processor Philips Smart XA or an ASIP Processor. Needs 8 kB+ internal RAM and 32 kB EPROM and 2/3 wire protected memory.
- CPU special features, for example, a security lock.
  - CPU locks certain section of memory - protect 1 kB or more data from modification and access by any external source or instruction outside that memory.
  - Other way of protecting - CPU access through the physical addresses, which are different from logical address used in the program.
- Standard ROM 8 kB for usual or 64 kB when using advanced cryptographic features.
- Full or part of ROM bus activates take place after a security check only.

**ROM Contains:**

  i. Fabrication key and Personalisation key (after insertion of this key, RTOS and application use only the logical addresses).
ii. RTOS codes.

iii. Application codes.

iv. Utilisation lock.

EEPROM or Flash scalable – only needed part unlocks when storing P.I.N., unlocking P.I.N., access condition, card-user data, post activation application run generated non-volatile data, invalidation lock to invalidate card after the expiry date or server instruction.

RAM – run time temporary variables.

• Chip-supply system using charge pump.
• I/O system.

Software Architecture:

Smart Card Software:

• Needs cryptographic software, needs special features in its operating system over and above the MS DOS or UNIX system features.

Protected environment -OS stored in the protected part of ROM.

• A restricted run-time environment.
• OS, every method, class and run time library should be scalable.

Optimum Code-size

• Limited use of data types; multidimensional arrays, long 64-bit integer and floating points and very limited use of the error handlers, exceptions, signals, serialisation, debugging and profiling.

• Three-layered file system for the data.

master file to store all file headers (file status, access conditions and the file lock).

• A header means file status, access conditions and the file lock.
• Dedicated file—second file to hold a file grouping and headers of the immediate successor.
• Elementary file—third file to hold the file header and its file data.
• Either a fixed length file management or a variable file length management with each file with a predefined offset.

Smart Card Software in Java:

• Java CardTM, EmbeddedJava or J2ME (Java 2 Micro Edition) JVM has thread scheduler built in.
• Java provides the features to support
(i) security using class (java.lang.SecurityManager),
(ii) cryptographic needs (package java.security*).

5. SmartOS RTOS used as alternative to MUCOS:

Smart Card OS:
• SmartOS—assumed hypothetical OS in this example, as RTOS in the card.
• Use for understanding purposes identical to MUCOS but actual SmartOS has to be
different from MUCOS.
• Its file structure is different, though it has MUCOS like IPCs and ECBs.
• function unsigned char [ ]
SmartOSEncrypt (unsigned char
*applStr, EnType type) encrypts as per encryption method, EnType = "RSA" or "DES" algorithm
chosen and returns the encrypted string.
• function unsigned char [ ] SmartOSDecrypt (unsigned char *Str, DeType type) encrypts as per
deciphering method, DeType = "RSA" or "DES" algorithm chosen and returns the deciphered
string.
• SmartOSEncrypt and SmartOSDecrypt execute after verifying the access conditions from the
data files that store the keys, PIN (Personal Identification Number) and password.

Tasks and their priority, action and IPCs:
resetTask
• Priority — 1.
• Action—Initiates system timer ticks, creates tasks, sends initial messages and suspends
• itself.
• IPC pending:
• IPC posted: SigReset, MsgQStart
• String Output: request-Header; request-
• Start

task_ReadPort:
• Priority — 2.
• Action—Wait for resetTask suspension, sends the queue messages and receives the messages. Starts the application and seeks closure permission for closing the application
• IPC pending: SigReset, MsgQStart, MsgQPW.
• MsgQAppl, MsgQAppl-Close.
• IPC posted: SemPW.
• Output: request-password, request-Appl, request-ApplClose.

**task_PW:**
• Priority — 3.
• Action—Sends request for password on verification of host when SemPW = 1.
• IPC pending: SemPW.
• IPC posted: *MsgQPW*.
• Input: request-Password.

**task_Appl:**
• Priority — 8.
• Action—when SemPW = 1, runs the application program.
• IPC pending: SemAppl.
• IPC posted: *MsgQAappl*.

**Multiple tasks and their synchronization model:**
Summary:

✓ Tasks, ISRs and IPCs required for the system are
  • Task_ReadPorts
  • Task_Collect
  • Task_Deliver
  • Task_Refund
  • Task_ExcessRefund
  • Task_Display

✓ Application of semaphores in µC/OSII as
  • event flag,
  • resource key
  • counter.

✓ Application of µC/OSII are
  • mailboxes
  • system clock.

✓ Required functions for ACC are:
  • Task_Align,
  • task_Read-Range,
  • task_Speed,
  • task_Range-Rate
  • task_Algorithm

✓ Code design given using the a hypothetical RTOS, SmartOS, which has MUCOS features plus the embedded system required cryptographic features and file security, access conditions and restricted access permissions during code run.