Programming issues in Real Time Systems

Gahan Saraiya
Institute of Technology
Nirma University
Ahmedabad, India
18mcec10@nirmauni.ac.in

Rushi Trivedi
Institute of Technology
Nirma University
Ahmedabad, India
18mcec08@nirmauni.ac.in

***Abstract*—The growth in industry has driven the requirement for Real Time systems over time for many applications. Majority applications are now days considered as Real Time Systems where some applications may result in catastrophic event. But Even though Response Time of every system are considered as base criteria for each and every systems from a small electronics chip or software to largely distributed applications or network containing cluster of hardware and software. Response Time is not just simple thing which can be achieved easily as the desire is to get minimum response time without loosing accuracy of the system. This is because of the various issues arises in design of real time systems due to programming, hardware architecture, operating system, network latency (if distributed) etc. This paper illustrates and focuses on programming issues arises for real time systems.**

***Index Terms*—Real Time Systems, Programming issues**

I. Introduction

Designing Real time systems is a challenging task as a growing requirements and expectations from systems. As Real time systems have to interact with real world entities most of the challenge comes from that. These interactions can get more and more complex. Typically a Real time systemmight be interacting with thousands of such entities at the same time. For example, a telecom system routinely handles calls from thousands of people at the same moment. The system has to connect each call differently. Constraint like sequence of calls, amount of noise and quality of call availability of service etc.

For applications such as in air-traffic control system, satellite launching are considered as hard Real time systembecause if deadlines which are not followed the system then it causes the catastrophic damage. For some other applications such as temperature reading response deadline is important but can be missed occasionally which is not desirable but not causing any catastrophic damage hence such applications are considered as soft Real time system. Also some deadlines can start soft and later become hard. i.e., a $3ms$ periodic reading on an aircraft pressure sensor is ideal but can be missed, however a $15ms$ interval could cause catastrophe. A firm system such as weather forecast is one in which if the deadline is missed then no value for doing it later.

Components which are displayed in Fig. 1 are to be managed by program efficiently in such a way that maintains the accuracy of the system and achieve desire performance in terms of meeting deadline and response time.

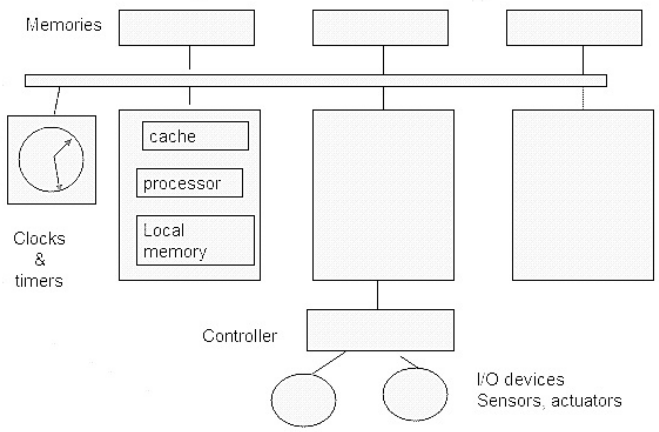

Fig. 1. Generic Diagram of Real time system [1]

II. Real time system Model

A Real time system model can be defined with below parameters:

- Workload model (task graph) is about what behaviors system will be asked to perform
  partially ordered set of tasks with timing information for each task where ordering represents dependency for the task
- Resource model (Capacity graph) is about tools and capabilities system applies
  contains all of the passive resources that the system must use along with their acceptable patterns of use system resources are divided in to:
  - Processors - active resources such as computers, database servers, etc.
  - Resources - passive resources such as memory, semaphores, locks, other data.
- Scheduling algorithm is about activity coordinated by system
  considers the dependencies and interferences among the tasks
- Resource management algorithm
  Activity coordinated by system

Each job in Real time system can be defined as a unit work and set of such jobs together called as *task set* achieves desired

activity. Each and every job of task set runs on a processor and depends on some resources. Real time system job can be characterizes as:

- temporal parameters
- functional parameters
- resource parameters
- interconnection parameters

Of course, the very important information for the real-time analysis are the timing prediction parameters. The execution time of jobs has many uncertainties due to following variable parameters:

- execution - caches, pipelines
- internal - branches, loops
- external - preemption, interference

All of these issues lead to difficulties in making a priori decisions about how much time to allow, and promote the use of simulation models to help.

III. Issues for Programming Environments

For many years, job of real time system programmer is to write a program with an execution time which is lesser than the time constrained allowed. To predict the precise execution time of a program is very difficult hence it usually takes multiple rounds of trail and error method to build a reliable Real time system. Also to port existing real-time software to a new configuration rather than building it from scratch takes the same level of effort which is not desirable since the requirements for real-time software are stronger than ever, because of ubiquity of computer systems in all application areas. And not only this but the next-gen real-time applications have more strict timing constraints.

To cope with such issues one needs to think of out-of-the box solution. Such an approach could be to make the system very flexible such that it can meet wide range of timing constraint under various system configurations. Thus the approach focuses on scalability for wide range of timing constraints rather than depending on precise execution time of computation. In many hard real-time systems, the requirement for functional correctness is not as strict as the requirement for temporal correctness. For these applications, the flexible performance approach is thus preferred [2].

To design and develop the next generation Real time system using approach described above, a programming language which has constructs for right away expressing timing constraints is required. The programmer defines the temporal constraints for the computation and also provides all possible set of codes to be executed. The runtime system along with the compiler and the scheduler must decide and/or generate the code for the execution so that the constraints are met.

Many language features are desirable for real-time systems programming. In fact, almost all features desirable in a "good" conventional (i.e., non-real-time) programming language can be considered as essential for real-time systems programming. For example, five requirements have been suggested for real-time software in [3]: *predictability, reliability, tasking, modularity, and maintainability*. Other then predictability all remaining four are always highly desirable in every system.

A. Asynchronous Communication

Remote procedure calls (RPC) are used in computer systems to simplify design of software. RPC allows a programmer to call procedures on a remote machine with the same semantics as local procedure calls. RPCs really ease the design and development of formal systems, but they are of very limited use in Real time systems. The main reason is that mainly communication in the real world is asynchronous, i.e. only few message interactions can be classified into the query response prototype that works so well using RPCs.

Thus most Real time systems support design based on state machine where multiple messages can be received in a single state. The contents of the received message determines the next state. State machines provide a much flexible mechanism to handle asynchronous message interactions. However the flexibility comes with its own complexities.

B. Working with Distributed Architectures

Most Real time systems involve processing on many different nodes. The system itself distributes the load of processing among multiple processors. This raises several challenges for programmer:

- Maintaining Consistency: to maintain consistency in data-structure is a challenge when multiple processors are involved in execution. Generally consistency is maintained by running data-structure audits.
- Initializing the System: Initializing the system with multiple processors is too much complicated than bringing up a single machine. When a node finishes initialization, it will initiate software downloads for all the child nodes directly connected to it. This process goes on in an hierarchical fashion till the complete system is initialized.
- Inter-Processor Interfaces: One of the biggest headache in Real time systems is defining and maintaining message interfaces. Defining of interfaces is complicated by ordering and padding rules for different byte in processors. Maintenance of interfaces is complicated by backward compatibility issues. For example if a cellular system changes the air interface protocol for a new breed of phones, it will still have to support interfaces with older phones.
- Load Distribution: When multiple processors and links are involved in message interactions distributing the load evenly can be a daunting task. If the system has evenly balanced load, the capacity of the system can be increased by adding more processors. Such systems are said to scale linearly with increasing processing power. But often designers find themselves in a position where a single processor or link becomes a bottle neck. This leads to costly redesign of the features to improve system scalability.
- Centralized Resource Allocation: Distributed systems may be running on multiple processors, but resources

must be allocated from a shared pool. Shared pool allocation is typically managed by a single processor allocating resources from the shared pool. If the system is not designed cautiously, the shared resource allocator can become a bottle-neck in achieving full system capacity.

C. Loop size, timer granularity, multi-programming, etc

Programmer needs to take account of loops size, timer granularity, multi-programming, imprecise timer, sleep() etc. to avoid potential timing hazards.

D. Sequential programs, parallel programs, timely programs

E. Client-server priority assignments - priority inversion

F. Verification, analysis, and testing

IV. What Programming Language to choose?

So far the paper explained the Real time system model, component and significance of programming in Real time system. There are lot of programming languages available for normal software development such as Assembly language, C, C++, Java, Python, GO, Pearl, Ruby, R etc. However not every language fits by the needs of Real-Time application. As already described the desired five requirements for Real time system in Section III not all language guarantees all the five requirements and we have to eliminate those language.

Programming languages for Real time system can be classified as below:

- Assembly Languages
- Sequential systems implementation languages – e.g. RTL/2, Coral 66, Jovial, C.
- High-level concurrent languages – e.g. Ada, Chill, Modula-2, Mesa, Java.

where in assembly languages and sequential systems implementation languages require operating systems support whereas High-level concurrent languages does not. Also High-level concurrent languages are impetus from the software crisis.

As a result below are listed popular languages available for programming of Real time system:

- Java/Real-Time Java
- C and Real-Time POSIX
- Ada 95
- Also Modula-1 for device driving

A. Some issues and fixes for Python

Python is most popular scripting language for utility development with huge number of features such that many programmer prefers development in Python however due to the feature provided by the language it makes it not usable for Real-Time application. In Real-Time computing, latency[1] from an interrupt to application code handling that interrupt being run, is both small and predictable. This means that a control process can be run repeatedly at most precise time intervals[2]. The variation observed in latency is called jitter[3]

For instance consider people building inverters for transmission of power cares about very precise performance where expected maximum jitter might be $1\mu s$[4].

TABLE I
Issues and solution for Python

Draw back	Possible Solution
Python mostly runs on desktop OS[5]. Desktop OS impose a lower limit to the maximum jitter; for windows it is observed in several seconds. The numbers for desktop Linux are almost similar.	However in desktop Linux one can apply different compile-time options and patch sets to the Linux kernel to improve the situation with $PREEMPT\_RT\_FULL$ [4].
Python's garbage collector makes latency non-deterministic. When Python runs garbage collector program has to wait until it finishes.	One may be able to avoid this through careful memory management and carefully setting the garbage collector parameters, but depending on what libraries are being used one may failed to explicitly manage memory.
Python's heap allocation. Most of the real-time application avoids it to achieve predictability as heap allocation is efficient for memory management but on the other hand amount of time is not predictable because of it.	The more libraries used the memory management and amount of time to predict becomes more and more difficult.
it is essential that real-time processes have all their memory kept in physical RAM and not paged out to swap. There is no good way of controlling this in Python, especially running on Windows	on Linux one might be able to fit a call to $mlockall()$ in somewhere. however any new allocation will upset things.

V. Timed-C: An Programming Language to Program RTS

As discussed in previous sections, there are various programing languages which can be used to program Real Time Systems, along with Ada, RTJS, Python, Java, C is also popular programming language of Real time Systems, but simple C language lacks in providing various timing constraints (constructs) as discussed in above sections,also Real time POSIX API provides more complex and erroneous code.So in this section one more RTS programing language is discussed which is Timed-C, Timed-C is an simply extension of simple C language with addition of various timing construct, concurrency construct and scheduling construct, Timed-C also provides less complex and error free code.

- Timed-C Primitives: In general Timed-C language contains various types of timing primitives like Soft primitives, Absolute primitives, Firm primitives, concurrent

[1] In general set off by a timer

[2] or consider that external events can be timed very precisely

[3] $1ms$ maximum jitter interpreted as - an interrupt arriving repeatedly will have a response latency that deviates by at most $1ms$.

[4] This maximum jitter may varies up to couple of $milliseconds$ depending upon application

primitives, scheduling primitives.Although having various timing primitives here we have discussed only soft primitives.

- Soft Timing primitives: As in Real Time Systems, timing constructs plays an vital role. So if logical time or execution time is equal to specified real time the proposed algorithm is optimal,but if logical time is not equal to specified real time i.e. does not satisfies timing constructs, then program has to react explicitly using timing primitives. Soft Timing primitives provides various functions to handle timing constraints which are sdelay, stp, fdelay, ftp. Here in this section sdealy is discussed.
  - sdelay:It is an timing point. Syntax of sdelay is sdelay(expr,n)
    ;where expr can be an integer value or any c expression and n is resolution in form of exponent.
    Primitive sdelay() specifies the relative delay from the previous timing point.Soft timing point ensures a lower bound on the specified delay.
    If time taken is more than specified delay then there will be overshoot (as prer last delay).
    Else return 0, no overshoot.

  Also it is to be noted that the starting time of any program is considered as Start timing point implicitly or by default.
  Given below example illustrates the use of sdelay() function.

```
1) int main(){
2) initialize();
3) sdelay(20, ms);
4) sense();
5) sdelay(50, ms);
6) }
```

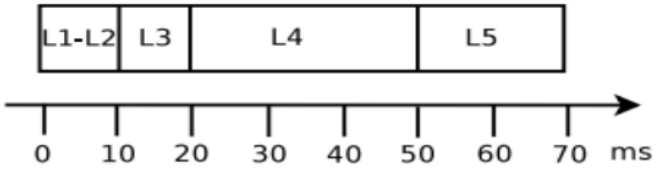

Fig. 2. demo of sdelay [5]

In above program and Fig 2, at first sdelay which is at L3 he difference is 20ms which is equal to specified current delay hence no overshoot will be there,similarly at L5 for second delay relative difference is 50ms hence no overshoot.

A. Implementation Of Timed C

KTC compiler is used to compile Timed-C which source to source compiler,like other compiler KTC compiler also consists of various phases,which are discussed below:

1) Front-End: Front end consists of two phases Initial Analysis and Static Analysis. Output of front end will be CIL (C-Intermediate Language) and AST (Abstract Syntax Tree).

- Initial Analysis:Here Timed-C input file is taken and addition of timing constructs, parsing and generation of CIL and AST will be done. After this construction of an Hash data structure is done which consists label and various timing constraints and this CIL, AST, Hash will be given to Static Analysis phase.
- Static Analysis:Here the inputs will be CIL,AST and Hash data structure. Static Analysis is used to reject statements with incorrect timing behavior. After removing statement with error,CIL and AST will send to Back end.

2) Back-End: Back end of KTC compiler consists of Transformation and Code generation steps and gives final FreeRTOS and POSIX file as output.

- Transformation: transforms the language primitives and emits a platform dependent C file and Transformation depends on the type of target platform.
- Code Generation: This is the final phase, it emits C-code and gives output Free RTOS C file and POSIX file.

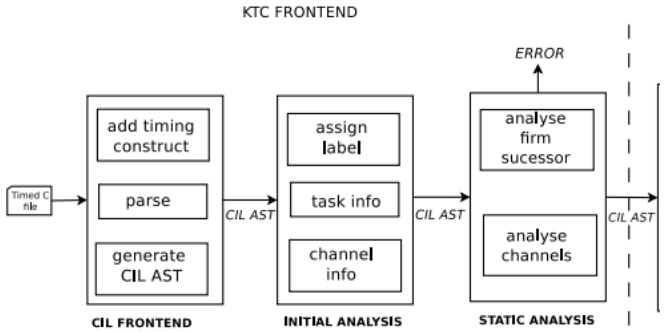

Fig. 3. Front end of KTC compiler [5]: Output CIL AST of this phase is given as input to the KTC backend phase of Fig 4

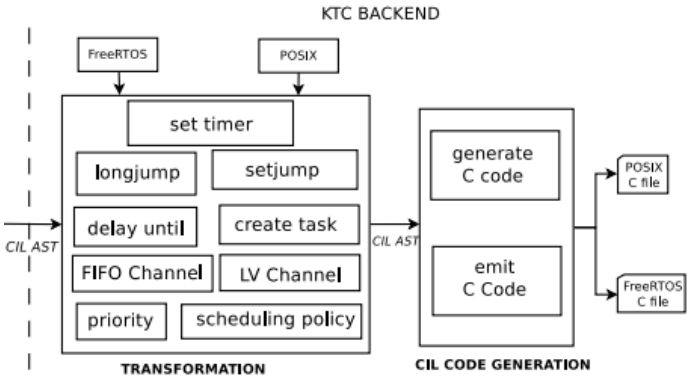

Fig. 4. Back end of KTC compiler [5]: Input CIL AST is fetched from the output of frontend phase of KTC described in Fig 3

VI. Conclusion

In this paper we have described the model of Real time system model in Section II followed by the issues for program-

ming environment and explained significance of programming in Real time system . In Section IV and the same section describes some fixes that could be done if one desires to use Python to develop Real time system. As a result we can conclude that predictability is highly concerned requirement for Real time system and programmer must manage everything in code explicitly to achieve predictability.

REFERENCES

[1] A. C. Shaw. Real-time issues. [Online]. Available: http://jcsites.juniata.edu/faculty/rhodes/smui/realtime.htm

[2] K.-J. Lin, "Advances in real-time systems," S. H. Son, Ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995, ch. Issues on Real-time Systems Programming: Language, Compiler, and Object Orientation, pp. 335–351. [Online]. Available: http://dl.acm.org/citation.cfm?id=207721.207735

[3] A. D. Stoyenko and W. A. Halang, "Extending pearl for industrial real-time applications," *IEEE Software*, vol. 10, no. 4, pp. 65–74, July 1993.

[4] Linux real-time capabilities. Toradex Developer Center. [Online]. Available: https://developer.toradex.com/knowledge-base/real-time-linux#Linux_Realtime_Capabilities

[5] S. Natarajan and D. Broman, "Timed c: An extension to the c programming language for real-time systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2018, pp. 227–239.