Task Management for Soft Real-Time Applications based on General Purpose Operating Systems *

Paulo Pedreiras, Luís Almeida

Departamento de Electrónica, Telecomunicações e Informática, Universidade de Aveiro
3810-193 Aveiro, Portugal

{pedreiras,lda}@det.ua.pt

Abstract. *Nowadays there is a widespread use of both real-time and general purpose Operating Systems (RTOS / GPOS) within embedded applications. GPOS are preferred in many cases that exhibit soft real-time constraints since they deliver sufficient real-time performance and even outperform RTOS in aspects like hardware support, price, availability and quality of development tools. However GPOS lack some key features commonly required in embedded applications, e.g. the support to automatic activation of recurrent tasks, phase control and precedence constraints. This fact led to the development of a simple user-space process manager library, called PMan, which delivers basic time and synchronization services, thus simplifying considerably the development of embedded applications on top of a GPOS. This paper describes the PMan library implemented in Linux and presents results that confirm the usefulness of the services provided.*

1. Introduction

Many embedded systems, such as those found in planes, trains, cars, robots and machine tools, exhibit specific timeliness, predictability and precedence constraints that must be respected. In many cases they are built on top of COTS microprocessor boards, possibly PC-compatibles, e.g. PC104, SBCs and Mini-ITX, and using multi-tasking operating systems or kernels, both real-time and general purpose (RTOS / GPOS) despite the profound architectural and functional differences exhibited by these classes of software infrastructures (Gopalan, 2001).

In fact, GPOS are typically time-shared multi-processing systems, optimized to manage heterogeneous classes of resources (CPU, disk, network interface, etc). The performance criteria for these systems are mostly associated with average throughput and fairness, the typical applications are not strictly time-constrained and may exhibit unpredictable activations, blocking and execution times. Conversely, RTOS privilege the timely execution of the activities they support. However the predictability delivered by this class of OSs comes at a price, which usually takes the form of additional information and constraints on the application tasks, e.g., bounded worst-case execution time, minimum inter-arrival time or activation period, relative phases and precedence constraints, and on operating system primitives, e.g., bounded blocking times, predictable synchronization primitives, suitable schedulers and admission control.

The architectural dichotomy presented by RTOS and GPOS leads to significant differences at the application implementation level, and thus the choice of using a

* This work was partially supported by the European Commission (NoE ARTIST2, IST-2-004527) and by IEETA - Universidade de Aveiro, Portugal.

RTOS or a GPOS may not be completely dictated by the timeliness and predictability aspects, only. For example, in soft real-time applications, which tolerate occasional failures in the time domain, GPOS may be preferred since they deliver sufficient real-time performance and generally outperform RTOS in practical aspects like hardware support, price, availability and diversity and quality of development tools.

Nevertheless, GPOS lack some features that are commonly required by embedded applications, e.g., support for automatic activation of recurrent tasks with enough precision, phase control and precedence constraints. These difficulties have been perceived in the scope of the CAMBADA middle-size robotic soccer team (Cambada), developed at the University of Aveiro, Portugal, and led to the development of a user-space simple process manager library, called PMan, which extends the native services provided by the underlying GPOS, i.e., Linux in this case. The PMan services are currently used to automatically trigger processes, adapt the Quality of Service (QoS) delivered to each process according to the operational environment, and to enforce phase and precedence constraints between distinct but related processes.

This paper describes the PMan library implemented in Linux and presents results that confirm the usefulness of the services provided. It is organized in the following way, Section 2 discusses related work, Section 3 presents the internals of the process manager layer (PMan), Section 4 analysis a case study including practical experiments and Section 5 concludes the paper.

2. Related work

Since the mid 90s that several attempts were made to achieve real-time performance with time-sharing general purpose operating systems. One approach that received substantial attention from the research community was the use of CPU reservations (Lee *etal*, 1996) (Jones *etal*, 1997) according to which a task could establish a contract with the CPU, reserving a given amount of time units every given period. These reservations would have priority over the time-sharing tasks. However, this model was not adequate to provide low jitter execution, could lead to large blocking unless a consistent system-wide reservation scheme was applied to all resources, and did not account for variable execution time. This aspect caused particular inefficiency in multimedia applications, thus Chu and Nahrstedt (1997) proposed supporting a new class of periodic but variable execution time tasks, which was completely built at user-level, thus highly portable, but required a kernel that could provide reserves. This same idea of providing real-time support to applications running in user space was also the motivation for the development of the LXRT module in RTAI/Linux (LXRT) and to the Xenomai project (Xenomai). These approaches provide good timing performance but require adequate kernel level support that is not provided by the Linux kernel alone. A different path was followed in (Chu and Nahrstedt, 1997) in which soft real-time operation was achieved with a simple user-space scheduler based on the fixed priorities defined within POSIX.4. We also follow this approach, providing a user-space scheduler that can simply be executed as a normal application in a Linux GPOS, without need for kernel patches or specialized kernels. With respect to (Chu and Nahrstedt, 1997), we take a step further adding support for off-set and precedence constraints.

3. The process manager layer - PMan

The core of our proposal is the processor manager layer (PMan), which aims at facilitating the development of soft real-time applications, extending the native services

provided by the underline GPOS in the following aspects:

- automatic activation of recurrent tasks;
- settling of relative phase control, allowing to establish temporal offsets among tasks;
- precedence constraints, conditioning the release of processes to the conclusion of a set of predecessors;
- on-line process management and QoS adaptation, allowing adding and removing processes at run-time as well as changing dynamically the temporal properties of the executing ones, without service disruption.

The time management within PMan is associated to a periodic tick whose source is user-configurable and can be generated with a timer or external event. For example, in the CAMABADA project the PMan tick is associated with the arrival of image frames, in order to minimize the latency in the image processing related processes.

The PMan operation relies on certain data concerning the processes (PMan table). A process record is shown in Table 1. The process name and process pid fields allow a proper process identification, being used to associate a table entry with a particular process and to send OS signals to the processes, respectively. The period and phase fields are used to trigger the processes at adequate instants. The period is expressed in number of system ticks, allowing each process to be triggered every n ticks. The phase and delay fields permit de-phasing the processes activation, e.g. to balance the CPU load over time, with potential benefits in terms of process jitter. The deadline field allows, when necessary, to carry out sanity checks or recovery actions in case of deadline violations. A user specified process is automatically activated upon occurrence of such an event. The following section of the PMan process record is devoted to the recollection of statistical data, which can be useful for profiling purposes. Finally, the status field keeps track of the current process state.

The library of services associated to the PMan layer is summarized in Table 2. The layer is initialized via the **PMAN_init** service and terminated with **PMAN_close**. The process registration in the PMan table is carried out with **PMAN_procadd**. After registering it is necessary to bind the process OS pid using **PMAN_attach**. This separation allows having a process registering itself autonomously or having a third party managing the registration and properties of other processes. Process entries may be removed from the PMan table by calling **PMAN_procdel**. **PMan_detach** dissociates a process from a PMan table entry. Attaching/detaching processes can be carried out online to allow, for example, selecting one from a set of processes to carry out a given action according a desired cost function, i.e. implementing functional alternatives that can be useful during CPU overloads.

A specific feature of PMan is the support for precedence constraints among processes. For example, in a classical sampling-controller-actuation loop it is necessary to guarantee that these functions are executed strictly in this order to have the end-to-end latency minimized. With the recent advances in computing hardware, e.g., hyper-threading and multi-core CPUs, simpler techniques, e.g., based on fixed priorities, are no longer enough to enforce the right sequencing, being necessary to use explicit synchronization primitives. These ones become hard to use in non-trivial situations, with many-to-many dependencies between processes or when the set of processes (and consequently their precedence relations) varies dynamically. To cope with this difficulty the PMan library manages automatically the precedence constraints among processes. Applications declare precedence relationships using **PMAN_precadd** and,

conversely, **PMAN_precdel** to remove previously established relationships. PMan checks all related precedences before actually releasing a process. The status of precedence constraints is updated whenever processes terminate.

Table 1. PMan process record

Process identification		
	PROC_name	Process ID string
	PROC_pid	OS process ID
Generic temporal properties		
	PROC_period	Period (frames)
	PROC_phase	Phase (frames)
	PROC_delay	Execution delay (ms)
	PROC_deadline	Deadline (μs)
Precedence constraints		
	PROC_pred_name	Predecessor list
QoS management		
	PROC_qosdata	QoS attributes
	PROC_qosupdateflag	QoS change flag
Statistical data		
	PROC_laststart	Activation instant of last instance
	PROC_lastfinish	Finish instant of last instance
	PROC_nact	Number of activations
	PROC_ndm	Number of deadline misses
Process status		
	PROC_status	Process status

Table 2. PMan services

PMAN_init	allocates resources (shared memory, semaphores, etc) and initializes the PMan data structures
PMAN_close	releases resources used by PMan
PMAN_procadd	register a process in the PMan table
PMAN_procdel	removes a process from the PMan table
PMAN_attach	attaches the OS process id to an already registered process, completing the registration phase
PMAN_detach	clears the process id field from a PMan record
PMAN_prec_add	specifies a precedence constraint between to processes
PMAN_prec_rem	removes the precedence constraint between two processes
PMAN_QoSupd	changes the QoS attributes of a process already registered in the PMan table
PMAN_TPupd	changes the temporal properties (period, phase, deadline or delay) of a process already registered in the PMan table
PMAN_epilogue	signals that a process has terminated the execution of one instance
PMAN_query	allows to retrieve statistical information about one process
PMAN_tick	called upon the availability of every new frame, triggering the activation of processes

The **PMAN_QoSupd** call allows changing the QoS allocated to each process at runtime. The QoS attributes depend on the underlying OS. The current implementation over Linux considers the OS priority as a QoS parameter. The application processes are assigned real-time priorities, with SCHED_FIFO scheduler, via the **sched_setscheduler** system call. The process priority, supplied as argument to **PMAN_QoSupd**, must be

within the range [sched_get_priority_min, sched_get_priority_max]. Similarly, the temporal properties of one process can also be updated dynamically using **PMAN_TPupd**. The ability to change the QoS of processes at runtime is particularly useful when the environment is highly variable and/or hard to characterize, allowing the application to dynamically adapt itself, allocating more resources to the processes that, in each instant, have higher impact on the global performance.

The **PMAN_epilogue** call must be issued by every process managed by PMan, just before termination. This service is required for internal PMan management, namely verification of deadline violations and updating precedence constraints. **PMAN_query**, on the other hand, allows accessing the statistical data of each registered process, which can be useful, for example, for profiling and load management. Finally, **PMAN_tick** carries out temporal management in PMan, incrementing the tick count, activating processes, checking task deadlines, etc.. This service must be requested periodically either with a system timer or with an external event. The latter mode, which is not commonly found, supports a transparent synchronization of the application execution with an external event stream, such as the arrival of image frames from a camera with automatic image capture.

4. Process synchronization with PMan: a case study

4.1. The CAMBADA vision subsystem architecture

As stated above, the PMan library was developed to address some difficulties perceived in the scope of the CAMBADA RoboCup middle-size robotic soccer team (Cambada).

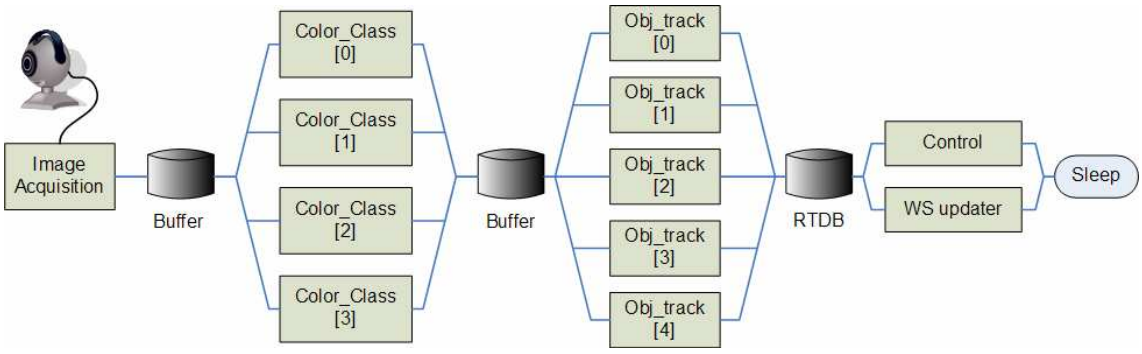

Figure 1. CAMBADA vision subsystem architecture

Currently, the vision subsystem architecture (Figure 1) uses one catadioptric configuration implemented with a low cost Fire-wire web-camera (BCL 1.2 Unibrain camera) and a hyperbolic mirror. The camera delivers 640x480 YUV images at 30 frames per second. When a new frame becomes available, the image handling process is automatically triggered and the frame is placed in a shared memory buffer. A set of processes (Color_Class[i] in Figure 1) will then analyze the acquired image for color classification, creating a new one with "color labels" (an 8 bit per pixel image). This image is also placed in a shared image buffer, which is afterwards analyzed by the object detection processes (Obj_track[i] in Figure 1). The output of the detection processes is placed in the real-time database (RTDB) which can be accessed by the other processes on the system, such as the control action and world state update.

4.2. Motivation for explicit precedences and relative offsets

The activation of the image-handling processes is carried out by the PMan manager right after the arrival of each new image frame. In earlier versions (Pedreiras *etal.* 2006; 2007) the PMan triggered sets of related tasks simultaneously, using priorities to enforce precedence constraints. This approach worked well with a single CPU and in the absence of hyper-threading but it failed to enforce such constraints when the computing platform of the robots was updated to Intel™ Core2Duo™ processors with two CPUs and hyper-threading. This is a common problem in real-time applications, which depend on specific features of the underlying hardware platform. When this is replaced, previous assumptions may fail leading to a poor application performance and possibly to a system failure. Therefore, the **PMAN_precadd** and **PMAN_precdel** primitives (Table 2) were added to the PMan library to deal with precedence constraints explicitly, as referred in Section 3.

Another identified problem was the need to adjust the control parameters individually, for each robot, due to differences in CPU processing power and thus differences in end-to-end image handling latencies. Moreover, the highly variable nature of the execution time of image processing activities further complicated the controller tuning. To solve these problems the actual release instant of the control process was decoupled from the termination of the preceding object tracking processes and it was set with a predetermined offset with respect to the image reception. This technique substantially reduces the dependence of the control performance on the underlying hardware used and it was added to the current version of PMan.

4.3. Experimental results

To experimentally verify the implementation, and assess the performance, of the PMan library several experiments were carried out, using the CAMBADA architecture depicted in Figure 1. Table 3 shows the process set used in the experiments.

Table 3: experimental process set

Process	Period	Precedence list	Delay (ms)	Description
readFireWire	1	{}	0	Interface with camera. Calls the tick event.
Color_Class[0..3]	1	readFireWire	0	Color classification. Each process scans 1/4th of the image.
Obj_Track[0..4]	1	Color_Class[0..3]	0	Object tracking (ball, obstacles, blue goals, yellow goal, lines).
Control	1	Obj_Track[0..4]	20	Issues control commands.
WSUpdater	1	Obj_Track[0..4]	20	Updates the RTDB with the processed data.

Two hardware platforms were used, one based on an Intel Pentium M Processor at 1.6GHz (Asus A3N notebook PC) and another based on an Intel P4 DualCore at 2.6GHz (Asus Pundit desktop PC). Both run the Linux 2.6.18 kernel with a tick of 4ms (HZ=250). The first experiment aimed at assessing the overhead induced by the PMan layer, namely by executing the PMAN_tick service. Table 4 shows the latencies measured from the activation of the *readFireWire* process, which calls PMAN_tick, to the start of execution of the process that executes right after. Each tick eleven processes

are triggered but in two situations, without and with the relative offset (*Delay*) specified in Table 3 for the *Control* and *WSUpdater* processes. The latencies vary from 7µs to 55µs and from 18µs to 73µs, respectively, being the extra latency of the second case caused by a creation of a wake-up thread and an extra call to the nanosleep primitive.

Table 4: Upper bound on the execution time of *PMAN_tick*

Processor	*Control* and *WSUpdater* activ.	Max. (µs)	Min. (µs)	Avg. (µs)	St.dev. (µs)
P M, 1.6GHz	Immediate	36	7	17.2	2.75
	Deferred	72	18	44.4	4.39
P4 Dual Core, 2.6GHz	Immediate	55	8	11.3	1.72
	Deferred	73	19	25.3	2.04

Table 5 shows the results of a second experiment to verify the effectiveness of the deferred activations, namely those of the *Control* process. When the activation is immediate, i.e. with *Delay=0*, the activation latency is highly variable (with the notebook) and hardware dependent, reaching a jitter around 20ms in the worst-case, which is a non-negligible value from the control performance point of view. With deferred activation, i.e. *Delay=20ms*, the jitter is substantially reduced, being around 4ms (the OS tick) for both the notebook and desktop PC. Furthermore, as desired, the activation latency value and jitter become nearly hardware independent. The measured activation delay presents a systematic offset of plus 4ms to 8ms in excess to the specified *Delay* value (20ms), which represents between one and two Linux OS ticks. This can be reduced compiling the Linux kernel with a shorter tick duration (e.g., 1ms, HZ=1000) or with high resolution clock patches.

Table 5: Control process activation delay

Processor	*Control* process activation	Max. (ms)	Min. (ms)	Avg. (ms)	St.dev. (ms)
P M, 1.6GHz	Immediate	24.15	5.28	19.15	2.842
	Deferred	27.98	23.89	24.22	0.841
P4 Dual Core, 2.6GHz	Immediate	5.21	2.14	3.69	0.534
	Deferred	28.13	24.01	26.04	2.002

Figure 2 shows an excerpt of an execution timeline in the notebook PC. Process activations are indicated by small circles. The four *Color classification* processes and the five *Object tracking* processes execute in sequence, inheriting the execution jitter of their predecessors. However, the *Control* and *World State update* processes, on top, have an activation delay that absorbs the execution jitter of the predecessors, resulting in a higher activation regularity. Notice, nevertheless, that these processes also have precedence constraints, which are always enforced, even if the execution of the predecessors takes longer than the specified activation delay.

5. Conclusions

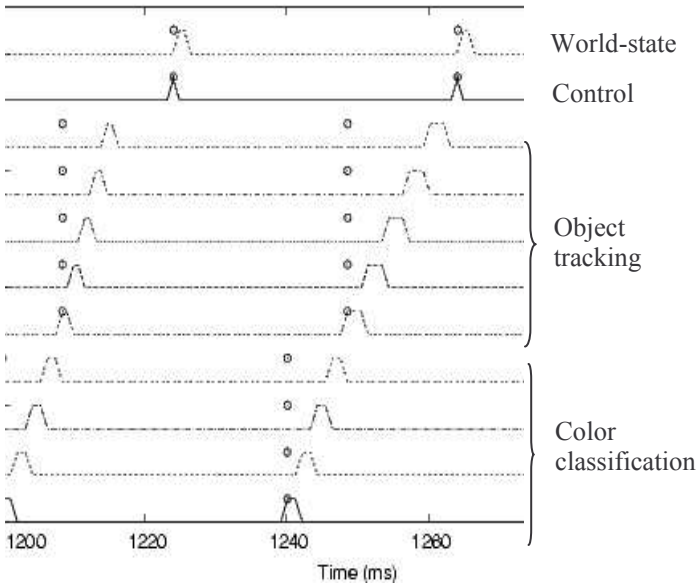

Figure 2. Visualizing activation delays

Using general purpose operating systems for soft real time applications has several advantages related with the abundance of device drivers and software tools. However, such applications still require adequate timing services, for process activation and synchronization. In this paper we presented a process management library that provides such services with substantial hardware independence and executes completely within user-space, being thus very flexible to deploy and use. In particular, this library provides support for periodic activation possibly with relative offsets and explicit precedence constraints, and also dynamic adaptation of temporal parameters and QoS attributes. The library was developed within the scope of the CAMBADA RoboCup middle-size soccer robots. Using this application as a case study, the paper presents several practical experiments that show the low overhead induced by the process management structure and the effectiveness of its support for precedence constraints and relative offsets with hardware independence.

6. References

Gopalan, Kartik (2001). "Real-Time Support in General Purpose Operating Systems". *ECSL Technical Report TR92, Experimental Computer Systems Lab, Computer Science Dept, Stony Brook University, Stony Brook, NY - 11794-4400*.

Lee,C., R. Rajkumar and C. Mercer (1996). "Experience with CPU reservation and dynamic QoS in real-time Mach". *Multimedia Japan*, March 1996.

Jones, M.B., D. Rosu and M. Rosu (1997). "CPU reservation and time constraints: Efficient, predictable scheduling of independent activities". Proc. of 16th ACM Symp. on Operating Systems Principles (SOSP'97), St. Malo, France, October 1997.

Chu, H. and K. Nahrstedt (1997). "A soft real time scheduling server in UNIX operating system". Proc. of European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services. Darmstadt, Germany, September 1997.

Chu, H. and K. Nahrstedt (1999). CPU service classes for multimedia applications. Proc. of IEEE Conf. on Multimedia Computing and Systems (MCS'99). Florence, Italy, June 1999.

Pedreiras, P., F. Teixeira, N. Ferreira, L. Almeida, A. Pinho, F. Santos (2006). "Enhancing the reactivity of the vision subsystem in autonomous mobile robots using real-time techniques". RoboCup Symposium: Papers and Team Description Papers, RoboCup-2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence, Springer, 2006.

Pedreiras, P., F. Teixeira, N. Ferreira, L. Almeida, A. Pinho, F. Santos (2007). "A Real-Time Framework for the Vision Subsystem in Autonomous Mobile Robots". in *Vision Systems*. G. Obinata and A. Dutta (eds.). ARS. To be published in April 2007.

Cambada. "Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture". University of Aveiro, Portugal. http://www.ieeta.pt/atri/cambada/

LXRT. http://www.fdn.fr/~brouchou/rtai/rtai-doc-prj/doxyapi/group__lxrt.html

Xenomai. http://www.xenomai.org/index.php/Main_Page